Erschienen in: Kurt Bauknecht, Wilfried Brauer und Thomas Mück (Herausgeber). *Informatik 2001: Wirtschaft und Wissenschaft in der Network Economy – Visionen und Wirklichkeit.* Tagungsband der GI/OCG-Jahrestagung, 25.-28. September 2001, Universität Wien. Seiten 916-921. ISBN 3-85403-157-2

Industrieller Einsatz von Application Server Technologie

Jochen Rütschlin[†‡], Jürgen Sellentin[†], Bernhard Mitschang[‡]

{jochen.ruetschlin|juergen.sellentin}@DaimlerChrysler.com
bernhard.mitschang@informatik.uni-stuttgart.de

[†]DaimlerChrysler AG
Forschung und Technologie
Labor IT for Engineering
Abteilung Prozesskette
Produktentwicklung (FT3/EK)
Postfach 23 60 · D-89013 Ulm

[‡]Universität Stuttgart
Institut für Parallele und Verteilte
Höchstleistungsrechner (IPVR)
Abteilung Anwendersoftware (AS)
Breitwiesenstraße 20-22
D-70565 Stuttgart

Zusammenfassung

In diesem Beitrag wollen wir anhand einer Integrationsarchitektur aus dem EAI-Umfeld motivieren und aufzeigen, wie Application Server Technologie sinnvoll bei der Zusammenführung von Systemen in einer vernetzen Umgebung eingesetzt werden kann. Dazu stellen wir erst unsere bisherige Integrationsarchitektur vor und erläutern an dieser einige Nachteile des traditionellen Vorgehens. Ein Abschnitt über Application Server und die J2EE-Bestrebungen leiten über zu einem Neuvorschlag der Integrationsarchitektur, realisiert auf Basis eben dieser Application Server Technologie.

Schlüsselwörter: *Enterprise Application Integration (EAI), Integrationsarchitektur, Middleware, Application Server, J2EE.*

1. Einleitung

Heutzutage fordert der Markt für eine wettbewerbsfähige Positionierung immer kürzer werdende Produktlebenszyklen, denen die Unternehmen mit einer zunehmenden Vernetzung ihrer Software-Systeme begegnen. Historisch bedingt sind diese Systeme nicht für einen interoperablen Einsatz ausgelegt, da sie oft nur für die Lösung bestimmter Aufgaben ins Leben gerufen wurden bzw. auf die Optimierung einzelner Prozesse in Teilbereichen abzielen. Wünschenswert ist dagegen vielmehr eine einheitliche Sicht auf Daten und Prozesse (ein sogenanntes *Single-System-Image*), um einen reibungsloseren Informationsfluss zu gewährleisten bzw. um aus den bereits vorhandenen Daten neue Informationen zu gewinnen. Diese Zielsetzung wird häufig auch als *Enterprise Application Integration* (EAI) bezeichnet.

Der alleinige Einsatz von Datenbank-Middleware-Systemen [9] stellte sich als unzureichend heraus, da oftmals an bestimmte Informationen nur mittels einer Funktionsschnittstelle gelangt werden konnte [4]. Daher wurde in der Abteilung FT3/EK der DaimlerChrysler-Forschung eine **Integrationsarchitektur** auf Basis einer klassischen 3-Schichten-Architektur entworfen. Als Middleware-Lösung diente hierbei CORBA, wodurch plattform- und programmiersprachenunabhängig beliebige Clients und vor allem Backend-Systeme zusammengeführt werden sollten.

Gleichzeitig mit der Etablierung der Integrationsarchitekturen kam die Welle der **Application Server** auf. Verschiedene Software-Hersteller boten Produkte an, mit denen die Unternehmen zukünftig ihre Anwendungen plattformunabhängig entwickeln sollten. Unter dem Schlagwort „Java auf dem Server“ wurden Web-Technologien wie Servlets ([16], [5]) und Java Server Pages (JSP; [17], [1]) zusammen mit dem EJB-Komponentenmodell (*Enterprise JavaBeans*; [15], [8]) als Produkt vermarktet. Verschiedene, gleich mitgelieferte Konnektoren [14] ergänzten das Paket und sollten eine leichte Anbindung von Backend-Systemen garantieren.

Nachdem unsere Integrationsarchitektur trotz aller Erfolge noch einige Probleme im praktischen Einsatz zeigt, wollen wir uns im Folgenden der Frage widmen, inwieweit eine Portierung auf Application Server Technologie gewinnbringend und damit auch sinnvoll ist. Dafür beginnen wir in Abschnitt 2 zunächst mit einer kurzen Einführung in den ursprünglichen Ansatz unserer Integrationsarchitektur. Anschließend folgt eine Diskussion der Application Server Technologie in Abschnitt 3. Der Fokus liegt hierbei insbesondere auf der sogenannten J2EE-Architektur. Auf Basis dieser Grundlagen folgt dann in Abschnitt 4 die Diskussion unserer Kernfrage: Kann die

Application Server Technologie eine sinnvolle Basis für Integrationsarchitekturen bilden? In Abschnitt 5 schließen wir mit einer kurzen Zusammenfassung und einem Ausblick auf zukünftige Arbeiten unsere Betrachtungen ab.

2. Szenario Integrationsarchitekturen

Gemäß der zuvor beschriebenen Zielsetzung der Erreichung einer Interoperabilität zwischen verschiedenen, heterogenen Einzelsystemen, wurden diese – ähnlich wie bei der Verknüpfung verschiedener Datenquellen mit einem Datenbank-Middleware-System – über eine eigen entwickelte Integrations-Middleware zusammengeführt (vgl. Abbildung 1); es sei dabei noch einmal betont, dass auf Daten mancher Systeme nur über eine Funktionsschnittstelle zugegriffen werden kann.

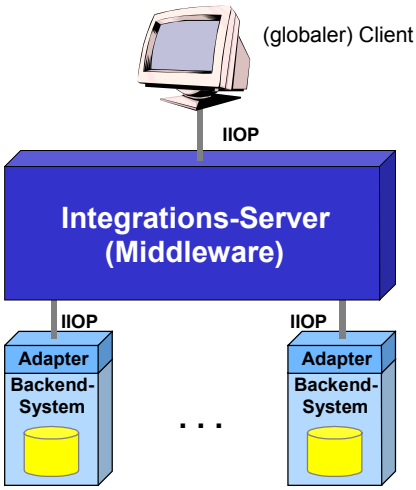

Abbildung 1: Integrationsarchitektur mit Middleware-Ansatz (CORBA).

Um bei der Analogie der Datenbanken zu bleiben, entsprach einer einheitlichen SQL-Schnittstelle aus der relationalen Welt eine von uns definierte Zugriffsschnittstelle, welche mittels eines individuell für jedes System zugeschnittenen Adapters bedient wurde. Der Integrations-Server (aus Performanzgründen in C++ implementiert) realisierte in erster Linie das Single-System-Image: der globale Client sieht sich einer einheitlichen Schnittstelle und Datenmenge gegenüber. Von ihm stammende Anfragen werden entsprechend in Teilanfragen zerlegt und über die Adapter an die Backend-Systeme weitergeleitet. Die zurückgelieferten Teilergebnisse werden schließlich im Server zu einer Antwort zusammengesetzt und an den Client zurückgegeben. Der (globale) Client war zuerst eine Java-Applikation, wurde später aber aus Distributionsgründen in ein Java-Applet umgebaut.

Bei der Konzeption der Architektur wurde versucht, soweit möglich Standards einzusetzen. Für die Kommunikation zwischen den verschiedenen Ebenen der Architektur haben wir CORBA ausgewählt. Entsprechend der in [12] beschriebenen Probleme beim Einsatz von CORBA in datenintensiven Umgebungen haben wir hierfür eine generische Schnittstellenspezifikation benutzt. Als Basis für das gemeinsame Datenmodell wurde das STEP AP 214 gewählt, ein international standardisiertes Schema für die Automobilindustrie [7]. Neben diesem Schema haben wir innerhalb des Clients weiterhin die im STEP-Standard (ISO 10303) definierte Zugriffsschnittstelle SDAI (*standard data access interface*) verwendet, um von der verwendeten Middleware-Technologie zu abstrahieren und ein umfangreiches Caching zu ermöglichen. In diesem Zusammenhang haben wir uns an der Standardisierung des SDAI-Java-Binding beteiligt [11].

Als Vorteile unseres Ansatzes sehen wir die Eingangs schon erwähnte Unabhängigkeit von den durch die eingebundenen Systeme verwendeten Programmiersprachen sowie die Unabhängigkeit der Client-Plattform (einzige Vorgabe ist ein applet-fähiger Browser). Des weiteren steht uns durch den Einsatz von Java auf dem Client die volle Funktionalität und Mächtigkeit einer Programmiersprache zur Verfügung, womit beispielsweise direkt auf dem Client CAD-Daten komfortabel bearbeitet werden können (drehen, spiegeln, *render*n, etc.). Als ebenfalls vorteilhaft erwies sich die generische Adapterschnittstelle zur Middleware, die für alle eingebundenen Systeme verwendet wurde (*stubs* und *skeletons* mussten nicht jedes mal neu erzeugt werden).

Auf der Seite der Nachteile war uns schon während der Konzeption der Architektur bewusst, dass die eingesetzten IDL-Schnittstellen der Adapter in Ermangelung geeigneter Alternativen proprietär ausfallen werden, was dementsprechend dann auch in Kauf genommen wurde. Negative praktische Erfahrungen machten wir bereits „auf dem Weg nach draußen“ an der Firewall: oft waren die Firewalls nur für den Port 80 (HTTP) freigeschaltet, nicht aber für das benötigte IIOP, welches je nach Installation einen anderen Port belegte. Es mussten Anträge für eine explizite Freischaltung gestellt werden, die neben dem zeitlichen Aspekt eines solchen Antrags häufig auch den Nachteil hatten, dass sie nur für bestimmte Maschinen wirksam waren. Hinzu kam, dass die GUI (*graphical user interface*) des Client je nach eingesetztem Browser bzw. der von diesem verwendeten JDK-Version entweder generell Probleme machte oder aber anders dargestellt wurde, womit der Aspekt der Plattformunabhängigkeit verloren ging, da explizite Vorgaben für zu verwendende Browser- und JDK-Versionen gemacht werden mussten.

Des weiteren stellte sich heraus, dass es schwierig war, server-seitig eine modulare Programmierung in C++ durchzuhalten. Das Ergebnis war eine monolithische Architektur, in der mit der Zeit Abhängigkeiten immer schwieriger nachvollziehbar wurden, was zwangsläufig eine reduzierte Wartbarkeit zur Folge hatte (die Einbindung neuer Systeme stellte sich als mehr und mehr zeitintensiv heraus).

3. Application Server Technologie: J2EE

Mit dem Begriff der Application Server Technologie wird auch sehr häufig der Begriff J2EE (*Java 2 Enterprise Edition*) [13] in Zusammenhang gebracht. Tatsache ist, dass ein Application Server mehr oder weniger den J2EE-Standard umsetzt. Dabei handelt es sich um einen Standard für die Entwicklung von mehrstufigen Unternehmensanwendungen auf Basis von Java. Dieser Standard wurde neben Sun Microsystems von einer Kollaboration aus führenden Herstellern von Unternehmens-Software geprägt. Im Wesentlichen besteht er aus der Zusammenfassung folgender (Einzel-) Technologien und einem darauf aufbauenden Applikationsmodell (in Abbildung 2 dargestellt).

- Enterprise JavaBeans™ Architecture
- JavaServer™ Pages
- Java™ Servlet
- J2EE™ Connector
- Java Naming and Directory Interface (JNDI)
- Java™ Interface Definition Language (IDL)
- JDBC™
- Java™ Message Service (JMS)
- Java™ Transaction API (JTA)
- JavaMail
- RMI-IIOP.

Auf die angeführten Technologien soll im Folgenden nur partiell und in Bezug auf das in Abbildung 2 dargestellte Applikationsmodell eingegangen werden (der interessierte Leser sei auf [13] verwiesen).

Applikationslogik wird im Wesentlichen in EJBs abgebildet. Dies sind Objekte der Java Komponentenarchitektur, die ähnlich den JavaBeans eine definierte Schnittstelle aufweisen und in einem sogenannten *Container* ablaufen (dieser stellt die Laufzeitumgebung dar). Je nach Anwendungsfall kann man bei der Programmierung mit EJBs unterschiedliche Typen von Objekten verwenden: *Entity Beans* mit *container managed persistence* (CMP) oder *bean managed persistence* (BMP) sowie *stateful* und *stateless Session Beans*. *Session Beans* enthalten nur temporäre Daten, die nach dem Beenden der Anwendung (der sog. *Session*) gelöscht werden. *Entity Beans* stellen hingegen Objekte mit einem persistenten Zustand dar. Zur Erlangung von Persistenz werden die relevanten Daten (Attribute) der EJB in einer Datenquelle abgespeichert. Diese Speicherung kann entweder der Container automatisch übernehmen (*container managed*), wobei zuvor beim sogenannten *Deployment* der Bean die Datenquelle spezifiziert werden muss, oder aber der Programmierer sieht die Speicherung explizit vor (*bean managed*). Dabei sollte beachtet werden, dass *Entity Beans* aller Regel nach langsamer sind als *Session Beans*. Dabei ist BMP noch mal langsamer als CMP [3]. Wir führen diesen Umstand auf die gleichen Ursachen zurück wie die bereits erwähnten Probleme von CORBA im Bereich datenintensiver Anwendungen [12]. Ein wesentlicher Grund liegt in der mangelhaften Fähigkeit der Application Server eine große Menge feingranularer Objekte effizient zu verwalten.

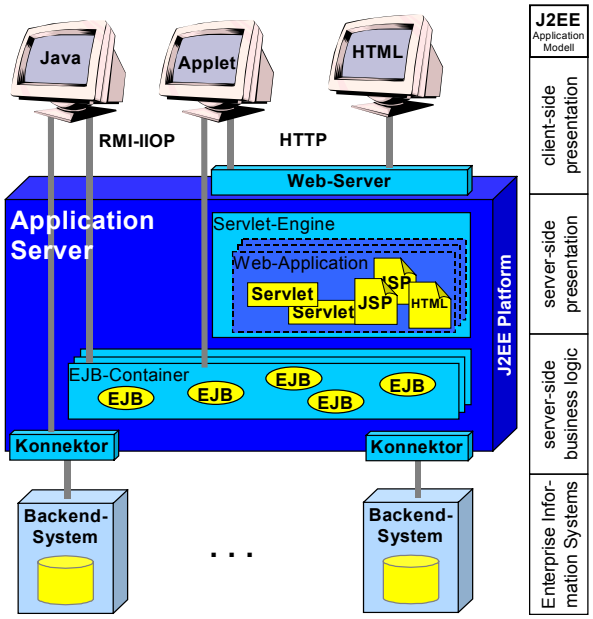

Abbildung 2: J2EE Applikationsmodell am Beispiel von IBM WebSphere [6].

Die Geschäftslogik bedient sich der in den Backend-Systemen (*Enterprise Information Systems*, EIS) gespeicherten Daten und Informationen. Ihre Anbindung erfolgt entweder direkt oder über sogenannte J2EE-Konnektoren. Ein Konnektor stellt im Wesentlichen einen Ressourcenadapter zum Backend-System zur Verfügung, welcher dessen API (*application programming interface*) durch ein Java-basiertes kapselt; die Verbindung zwischen dem Konnektor und dem Backend kann damit über ein beliebiges Protokoll erfolgen. Der Adapter wird gemäß der J2EE Connector Architecture Spezi-

fikation [14] entweder direkt von einer Java-Anwendung oder über eine EJB angesprochen.

Als Client sind Browser (HTML) oder Java-Anwendungen (Applet bzw. reine Applikationen) vorgesehen. Die Mittlerschicht zwischen dem (Web-) Client und der Geschäftslogik übernimmt ein Web-Server, erweitert um Servlet- und JSP-Funktionalität. Ein Servlet ([16], [5]) ist im Wesentlichen nichts anderes als ein Java-basiertes CGI (*common gateway interface*). Es nimmt einen HTTP-Request samt enthaltener Parameter entgegen und hat durch den Einsatz einer Programmiersprache (in dem Fall Java) die Möglichkeit, dynamische HTML-Seiten zu erzeugen (ggf. unter Zuhilfenahme von Daten, die aus den Backend-Systeme zugegriffen werden), die quasi mittels einfacher Print-Befehle auf der Standardausgabe an den Web-Server bzw. Browser zurückgegeben werden. JSPs ([17], [1]) basieren auf Servlet-Technologie und wurden eingeführt, um Layout und „Anwendungslogik" einer HTML-Seite voneinander zu trennen. Bei ihnen ist es möglich, in einer normalen HTML-Seite – durch bestimmte Tags eingeleitet – direkt Java-Code für die dynamischen Anteile unterzubringen (intern überführt der JSP-Prozessor beim Aufruf der Seite diese in ein Servlet, welches dann in der Servlet-Engine zur Ausführung kommt).

Derzeit werden Application Server typischerweise beim Online-Geschäft (Web-Shops) oder dem Aufbau von Portalen eingesetzt, wobei häufig nur die Bestandteile der Web-Applikationen (vgl. Abbildung 2) – also keine EJBs – verwendet werden.

4. Integrationsarchitekturen auf Basis von Application Server Technologie

Aufgrund des Eingangs erwähnten Aufkommens von Application Servern und den praktischen Erfahrungen mit unserer Integrationsarchitektur (vgl. Abschnitt 2), kam die Frage auf, ob man mit der Application Server Technologie nicht die EAI-Aktivitäten effektiver in den Griff bekommen könne. Eine fast direkte Umsetzung der bisherigen Integrationsarchitektur (bzgl. der drei Schichtenarchitektur) bot sich an: der ehemals mehr oder weniger monolithisch aufgebaute Integrations-Server wurde in Form modularer EJBs nachgebildet (allerdings unter ausschließlicher Verwendung von Session Beans, siehe Diskussion in Abschnitt 3); der Zugriff auf die Adapter wurde ebenfalls als EJB realisiert (vgl. Abbildung 3); der Client blieb vorerst mit kleineren Änderungen als Applet realisiert.

Im Folgenden schließt sich nun eine kurze Diskussion dieser neuen, Application Server-basierten Integrationsarchitektur an.

4.1. Fixierung auf Java

Durch Einführung der Application Server Technologie, die ja voll auf die Programmiersprache Java setzt, verlieren wir mit diesem Ansatz die zuvor durch CORBA erzielte Programmiersprachenunabhängigkeit. Bei genauerem Hinsehen stellt dies jedoch keine allzu große Einschränkung dar, weil durch Adapter bzw. Konnektoren weiterhin auf Systeme zugegriffen werden kann, die auf unterschiedlichen Plattformen laufen und in beliebigen Programmiersprachen entwickelt wurden.

Zudem soll im Application Server nur bedingt neue Funktionalität realisiert werden, da bei uns primär der Integrationsaspekt im Vordergrund steht, d.h. die Nutzung von bereits bestehender Funktionalität.

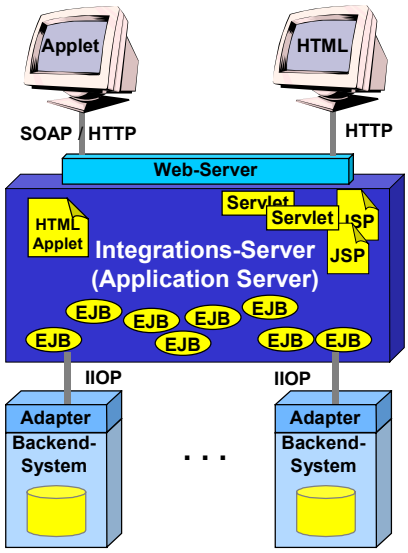

Abbildung 3: Integrationsarchitektur mit Application Server Ansatz.

4.2. Nutzung von Konnektoren

Eine noch offene Frage ist die Nutzung kommerziell verfügbarer J2EE-Konnektoren [14] für den Zugriff auf die Backend-Systeme. Vermutlich funktionieren diese nur bedingt gut, da sie primär auf Standardinstallationen von Systemen wie SAP oder Metaphase zugeschnitten sind. In unserem Umfeld sind diese Software-Pakete allerdings hochgradig angepasst (*customized)* wodurch einige Probleme und daraus resultierende Anpassungen entstehen. Dementsprechend haben wir unsere derzeitige Architektur ohne die Verwendung von Konnektoren definiert (siehe Abbildung 3). Wir werden diesen Aspekt in künftigen Arbeiten evaluieren.

4.3. Vom *fat* zum *thin* Client

Während die Portierung des Integrations-Servers offenkundig ist und bisher recht vielversprechend aussieht, sehen wir beim Client zwei, teilweise nicht unproblematische Alternativen. Der Client bestand in der ursprünglichen Architektur aus einer Java-Anwendung bzw. – um der Problematik der Software-Distribution zu entgehen – aus einem Applet. Dieses Applet haben wir im Zuge einer schnellen Portierung vorerst einmal beibehalten und lediglich die Kommunikation von IIOP auf SOAP über HTTP [2] umgestellt, wodurch die Firewall-Problematik umgangen werden konnte.

Anfängliche Anforderungen wie Portabilität und die Erfordernis einer gewissen Mächtigkeit in der Anwendung (beispielsweise zur Unterstützung von Caching bzw. allgemeiner Bearbeitungs- oder Darstellungsfunktionalität von CAD-Daten) waren ausschlaggebend für die Entscheidung, Java auf dem Client einzusetzen. Allerdings hat es sich in der Praxis gezeigt, dass Fragestellungen wie die Firewall-Problematik viel ausschlaggebender sind, als das erwähnte Caching auf dem Client oder die Bearbeitung von CAD-Daten. Gerade im CAD-Bereich hat sich herausgestellt, dass Konstruktionen und Änderungen über das Web normalerweise gar nicht notwendig sind und somit die Nutzung des Web-Viewers unseres strategischen CAD-Systems zum Betrachten und Herunterladen der Dateien völlig ausreichend ist.

Die weiterhin bestehende Problematik der vorgeschriebenen Browser- und JDK-Versionen zur korrekten Ausführung des Clients, sowie die anfänglichen langen Übertragungszeiten zum Laden des Applets liesen den Gedanken an einen *thin HTML client* aufkommen. D.h. durch Einsatz von Servlets und JSPs soll client-seitig nur noch ein Browser mit reinem HTML (ggf. unter Zuhilfenahme von JavaScript) erforderlich sein. Dies macht allerdings eine geschickte Aufteilung der bisherigen Client-Funktionalität auf Client *und* Server notwendig. Dabei müssen bisher bekannte und vor allem etablierte Konzepte moderner Benutzeroberflächen wie z.B. Strukturbäume, Register und Auswahlfelder zusammen mit der an die Applikationslogik geknüpften Abfolge von Bildschirmmasken mittels HTML nachgebildet werden. Dafür soll in einer weiteren Arbeit zuerst einmal geprüft werden, ob und wie beispielsweise aus Java bekannte Grafikelemente nachgebildet werden können und wie ein Sitzungskontext mit dem zustandslosen HTTP gehalten werden kann (Stichworte *URL-rewriting* oder *state-machine*). Dabei kann sicherlich teilweise auf bereits bekannte Konzepte und Techniken der Web-Programmierung [18] zurückgegriffen werden. In einem zweiten Schritt ist zu überlegen, wie eine so geartete GUI-Programmierung möglichst modular (also übersichtlich und gut wartbar) erfolgen kann. Das wäre beispielsweise denkbar durch den Einsatz einer abstrakten – möglicherweise in XML gehaltenen – Beschreibung der Bildschirmmaske, aus der dann mittels eines wie auch immer gearteten „Prozessors“ eine entsprechende HTML-Seite (bzw. ein Fragment von dieser) oder ein Servlet generiert werden kann. Hintergedanke ist dabei der Umstand, dass manche GUI-Elemente nicht rein objektorientiert programmiert werden können sondern dabei mehrere Faktoren einfließen. Beispielsweise müssen bei einer möglichen Realisierung eines Strukturbaumes im Dateisystem abgelegte Grafiken, HTML-Code, JavaScript-Code und Cascading-Stylesheet-Definitionen erzeugt und aufeinander abgestimmt werden. Aus der Praxis ist bekannt, dass ein Großteil des dabei anfallenden Codes einem gewissen Muster folgt, der somit auch bequem maschinell erzeugt werden könnte (bei Festlegung einiger weniger Bezeichner und Strukturen).

4.4. Einbettung in Portale

Mit dem Aufkommen des Portalgedankens (insbesondere des Portlet-Konzepts) stellt sich auch bei unserer Architektur die Frage, inwieweit sie in ein Portal integriert werden kann. Die Verwendung von Application Server Technologie steht dem erst einmal nicht im Wege, da fast ausschließlich alle Portal-Produkte auf ebensolcher Technologie aufgebaut sind (meist in Form mehrerer Servlets und JSPs). Allerdings eignen sich diese kommerziellen Produkte eigentlich nur für die Einbettung von Text bzw. reinem HTML-Content (vgl. dazu [10]). Die Unterstützung von komplexen HTML-Anwendungen – und in diese Kategorie fällt der globale Client unserer Integrationsarchitektur – ist nicht möglich (sicherlich bedingt durch das häufig ohne Frames auskommende Portlet-Konzept) bzw. kann nur in Form eines sogenannten *Launch-Links* erfolgen. Dabei läuft die Anwendung dann in einem extra, über den Link geöffnetem Browser-Fenster. Die Einbettung unserer Architektur in Portale kann somit nicht als gelöst betrachtet werden und ist dementsprechend Gegenstand weiterer Arbeiten.

5. Zusammenfassung und Ausblick

In diesem Artikel haben wir die Frage diskutiert, inwieweit sich klassische Integrationsarchitekturen auf die relativ neue Technologie der Application Server portieren lassen. Wir haben erkannt, dass sich die zugrundeliegenden Architekturkonzepte mehr oder weniger direkt auf die J2EE-Architektur übertragen lassen. Gleichzeitig

erhält man durch die Verwendung von EJBs eine modularere Architektur innerhalb des Integrations-Servers.

Eine umfassende Untersuchung muss alle in der J2EE-Architektur vorgeschlagenen (siehe Aufzählung in Abschnitt 3) Konzepte berücksichtigen und deren Nützlichkeit bewerten. Hinsichtlich der Untersuchung zu unserer Integrationsarchitektur konnten bisher noch nicht alle J2EE-Konzepte betrachtet werden. Im Wesentlichen haben wir uns auf die Verwendung von Session Beans und den Einsatz von Servlet/JSP-Techniken beschränkt. Diese können als technologisch ausgereift bezeichnet werden, so dass einem industriellen Einsatz nichts im Wege steht. Bei Entity Beans und den J2EE-Konnektoren sind noch Zweifel anzumelden, insbesondere ob diese Konzepte nicht in die gleichen Fallen wie z.B. CORBA laufen [12]. Hier sind mit Sicherheit weitere Untersuchungen nötig, die wir demnächst angehen werden.

Wirft man einen Blick auf die Details, so haben wir im Vergleich zu dem relativ geringen Aufwand der Portierung einige Vorteile erzielt. Zunächst einmal bewirkt die Beschränkung auf Java innerhalb des Integrations-Servers keine wirklichen Nachteile. Im Bezug auf Client und Backends sind wir nach wie vor unabhängig von Plattformen und Programmiersprachen. Zur Lösung der Firewall-Problematik haben wir unseren alten Java-Client zunächst beibehalten und nur die Kommunikation von CORBA (IIOP) auf SOAP über HTTP umgestellt. Langfristig ist hier wie erläutert an eine reine HTML-Lösung gedacht, die sich dann hoffentlich besser in Portale integrieren lässt. In diesem Zusammenhang werden wir untersuchen, inwieweit sich Java-Oberflächen (semi-) automatisch in HTML-Masken überführen lassen. Dabei muss natürlich die Logik auf den Integrations-Server verlagert werden (voraussichtlich mittels JSP-Technologie).

Zusammenfassend lässt sich sagen, dass die Technologie der Application Server vielversprechend ist und einem industriellen Einsatz generell nichts im Wege steht. Allerdings muss man sehr wohl aufpassen, welche Konzepte denn wirklich ausgereift und im jeweiligen Fall angemessen sind. An dieser Stelle fehlen mit Sicherheit weitere Studien und Kriterienkataloge. Gleichzeitig entwickeln sich natürlich sowohl die Technologien der J2EE als auch die darauf basierenden Produkte kontinuierlich weiter. Insofern kann dieser Beitrag nur als ein Schnappschuss der aktuellen Lage gesehen werden.

6. Quellenverzeichnis

[1] H. Bergsten: *JavaServer Pages*. O'Reilly, 2000.

[2] D. Box et.al.: *Simple Object Access Protocol (SOAP) 1.1*. W3C Note, World Wide Web Consortium, 2000. http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

[3] C. Dorda: *Verwaltung von Benutzerprofilen mit Enterprise JavaBeans*. Studienarbeit, Nr. 1799, Universität Stuttgart, 2001.

[4] K. Hergula, T. Härder: *A Middleware Approach for Combining Heterogeneous Datasources – Integration of Generic Queries and Predefined Function Access*. In: Proc. 1st International Conference on Web Information Sytems Engineering (WISE), Hongkong, 2000, S. 22-29.

[5] J. Hunter with W. Crawford: *Java Servlet Programming* (2nd edition). O'Reilly, 2001.

[6] IBM WebSphere Application Server. http://www.ibm.com/software/webservers/appserv/

[7] ISO DIS 10303 Industrial automation systems and integration: Product data representation and exchange – Part 214: *Application protocol: Core data for automotive mechanical design processes*, Draft Intl. Standard, ISO TC184/ SC4/WG3 N765, 1999.

[8] R. Monson-Haefel: *Enterprise JavaBeans*. O'Reilly, 2000.

[9] Proceedings 4. Workshop „Föderierte Datenbanken". TU-Berlin, 1999.

[10] J. Rütschlin: *Ein Portal – Was ist das eigentlich?* Workshop *E-Business Engineering* auf der gemeinsamen Jahrestagung der GI und OCG, Wien, 26.-28. Sep. 2001.

[11] J. Sellentin, B. Mitschang: *Data-Intensive Intra- and Internet Applications Based on Java, CORBA, and the World Wide Web*, Invited Paper in: E. Bertino, and S. Urban: "Object-Oriented Technology in Advanced Applications", Special Issue of Theory and Practice of Object Systems (TAPOS), Vol. 5, No. 3, John Wiley & Sons, 1999.

[12] J. Sellentin: *Konzepte und Techniken der Datenversorgung für Informationssysteme*, Informatik Forschung und Entwicklung (IFE), 15(2): 92-109, 2000.

[13] Sun Microsystems, Inc.: *Java 2 Enterprise Edition*. http://java.sun.com/j2ee/

[14] Sun Microsystems, Inc.: *J2EE Connector Architecture*. http://java.sun.com/j2ee/connector/

[15] Sun Microsystems, Inc.: *Enterprise JavaBeans*. http://java.sun.com/products/ejb/

[16] Sun Microsystems, Inc.: *Java Servlet Technology*. http://java.sun.com/products/servlet/

[17] Sun Microsystems, Inc.: *JavaServer Pages*. http://java.sun.com/products/jsp/

[18] V. Turau: *Techniken zur Realisierung Web-basierter Anwendungen*. Informatik-Spektrum 22(1): 3-12, 1999.