

abhängig von der konkreten Konfiguration in weiten Grenzen variieren, kann aber bei wirklichen *Thin Clients* bis zu über 60% betragen.

Die im Vergleich zur herkömmlichen Anwendungsentwicklung völlig andersartige Umgebung läßt die Einbindung vorhandener Applikationen als schwierig erscheinen. Entwickler müssen sich mit verschiedenen Sprachen beschäftigen, so insbesondere mit HTML und Skript-Sprachen wie z. B. *JavaScript*, *Perl* und *Tcl*. Insbesondere der Zugriff auf Datenbanken geschieht noch weitgehend über proprietäre Schnittstellen. Die Situation kann verbessert werden, indem *Applets* und *Servlets* mit einer einheitlichen Sprache (*Java*) entwickelt werden und eine neutrale Schnittstelle zum Datenbankzugriff (derzeit *JDBC*) genutzt wird.

Transaktionsorientierte Anwendungen sind nur mit Schwierigkeiten zu realisieren. Da eine Verbindung nach jeder Antwort des *Web Server* wieder terminiert wird, sind die für den konkurrierenden Mehrbenutzerbetrieb typischen Sperrverfahren (z. B. *Optimistic Locking*) nicht anwendbar. Es bedarf permanenter Verbindungen, um dieses Defizit zu beseitigen. Außerdem ist der Vorrat an Interaktionselementen (*Controls*), der zur Gestaltung der Bediener-Oberfläche eingesetzt werden kann, für anspruchsvollere Anwendungen noch unzureichend.

3.9.4.3 Nutzung eines *Web Browser* zur Ausführung objektorientierter Anwendungen

Als dritte prinzipielle Möglichkeit bietet sich die Realisierung verteilter Anwendungen auf Basis eines *Object Request Broker* (ORB), insbesondere *CORBA* oder *RMI*, an. Anwender-Arbeitsstationen werden als *Thin Clients* ausgelegt und verfügen neben der Betriebs- und Netzwerk-Software lediglich über einen *Web Browser*, der als universaler *Client* fungiert.

Dieser Ansatz ist sehr sinnvoll, da die Stärken zweier „Welten“ miteinander vereint werden können. *Java* als Sprache bietet Plattformunabhängigkeit, ein ORB Ortstransparenz. Ein weiterer Vorteil eines ORB ist die Möglichkeit, in einem verteilten Anwendungssystem unterschiedliche Programmiersprachen einzusetzen und auf diese Weise bestehende Anwendungssysteme schrittweise zu migrieren. Über Dienste wie beispielsweise *CORBA Persistent Object Services* (POS) können Zugriffe auf Datenbanksysteme erfolgen.

Die wesentliche Aufgabe besteht in diesem Szenario darin, die Bediener-Oberfläche einer Anwendung innerhalb der Anzeigefläche des *Web Browser* darzustellen und die Verbindung zur Präsentationskomponente der Anwendung herzustellen. An die Stelle des HTTP-Protokolle könnte dann das mächtigere verbindungsorientierte IOP-Protokoll (*CORBA*) treten. Um dieses Ziel zu erreichen, muß zunächst über den *Web Browser* eine Verbindung hergestellt werden. Dies geschieht indem nach Eingabe einer URL ein *Applet* vom *Web Server* in den *Web Browser* geladen und ausgeführt wird. Dieses *Applet* stellt die Verbindung zur Anwendung her, die dann über das IOP-Protokoll Methodenaufrufe austauscht.

Die andersartige Umgebung macht Schwierigkeiten

Transaktionsorientierte Anwendungen bereiten Schwierigkeiten

Der *Web Browser* fungiert als universaler *Client*

Stärken können vereint werden

An die Stelle von HTTP kann IOP treten

Das *Web-Applet* fungiert als „Sprungbrett“

Bei Aufruf einer Seite wird ein eingebettetes „*Web-Applet*“ an den *Client* (*Web Browser*) übermittelt. Es dient dazu, die Verbindung zum *Server* über das IIOP-Protokoll herzustellen und das *Frontend* (die Bediener-Oberfläche) der Anwendung zu laden. Das „*Web-Applet*“ kommuniziert mit einem CORBA *Server*-Objekt, das seinerseits auch *Client*-seitige Objekte aufrufen kann. Sobald die Verbindung hergestellt ist, können sämtliche Nachrichten zwischen *Client* und *Server* bis zur Beendigung der Anwendung über dieses Protokoll ausgetauscht werden. Insofern handelt es sich nicht mehr um eine typische Internet-/Intranet-, sondern um eine klassische *Client/Server*-Anwendung.

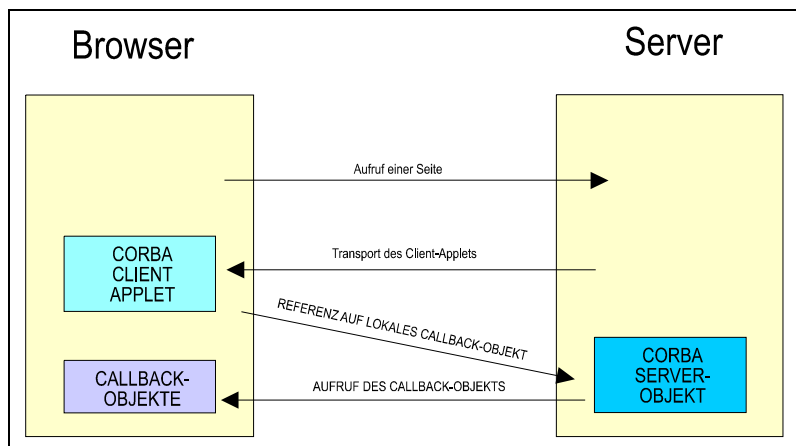


Abb. 3-65: Integration Internet-/Intranet-Anwendungen und CORBA

Heterogene Systeme werden durch klare Trennung von Schnittstellen und Implementierung unterstützt

334

Im Vergleich zu einer geschlossenen Entwicklungsumgebung muß die Trennung von Entwicklungs- und heterogener Ausführungsumgebung auf andere Weise erreicht werden. Schnittstellen-Definitionen werden mittels der sprachneutralen Interface *Definition Language* (IDL) beschrieben, um Schnittstellen und Implementierung voneinander zu trennen. Mittels des *Object Request Broker* (ORB) können Objekte Botschaften miteinander austauschen, unabhängig von der Sprache die zur Entwicklung der Klassen verwendet wurde. Vorausgesetzt die *IDL-Mappings* (z. B. C++/IDL, Smalltalk/IDL, Java/IDL usw.) sind standardisiert, ist vollständige Interoperabilität erreicht. Die Schnittstelle zur Datenhaltung wird sinnvollerweise über eine Klassen-Bibliothek oder ein *Framework* realisiert, das die Anwendung von der konkreten Schnittstelle zum Datenhaltungssystem isoliert.

Um eine vielseitige Bediener-Schnittstelle anbieten zu können, sollten sowohl konventionelle GUI-Systeme als auch *Web Browser* unterstützt werden. Die Isolation der Bediener-Schnittstelle vom plattformspezifischen GUI-System kann mit einem geeigneten Framework geschehen. Um die Bediener-Schnittstelle innerhalb eines *Web Browser* abbilden zu können, ist die Integration mit dem *Web Browser* erforderlich.

Der *Web Server* spielt in diesem Szenario nur eine untergeordnete Rolle. Im Minimalfall beherbergt der *Web Server* lediglich das *Applet*, das beim Start der Anwendung auf den *Web Browser* heruntergeladen wird. Die folgende Abbildung zeigt das Prinzip im Zusammenhang.

3.9 Strategien

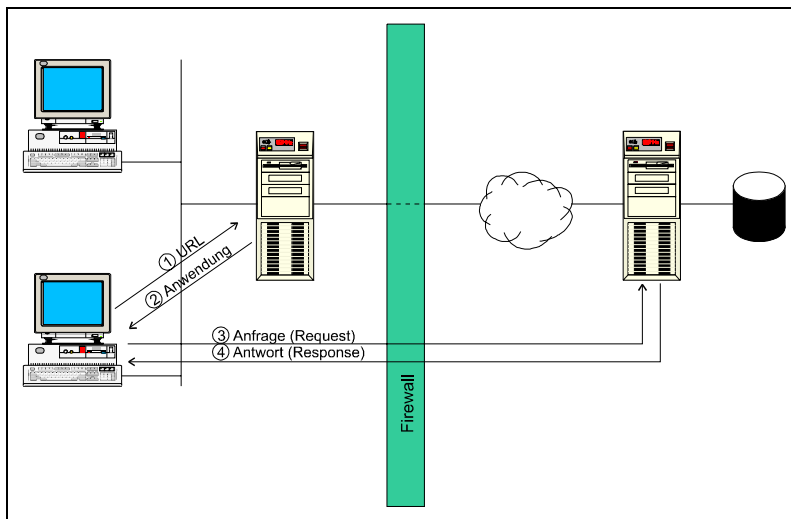


Abb. 3-66: Prinzip einer verteilten Anwendung mit Nutzung des Web Browser

Die Lastverteilung kann dynamisch gestaltet werden. Sofern Java zur Entwicklung eingesetzt wird, werden die benötigten Klassen zur Laufzeit nachgeladen. Im Vergleich zu CORBA zeigt sich eine deutlich höhere Dynamik.

Insgesamt gesehen gelingt in diesem Szenario die Zusammenführung der Technologien fast vollständig. Lediglich bei der Präsentation zeigt sich ein kleiner Bruch, der sich allerdings nicht vermeiden läßt, da die Anwendung vollständig auf einem oder mehreren *Server*-Systemen residiert und „irgendwie“ aktiviert werden muß. Die Aktivierung geschieht durch Eingabe der URL der Start-Seite der Anwendung.

Für komplexe transaktionsorientierte Anwendungen ist auch diese Möglichkeit nicht gut geeignet, da die Bediener-Schnittstelle durch das "Web Applet" abgebildet werden muß. Die eingeschränkten Möglichkeiten der Bediener-Schnittstelle wurden bereits im Zusammenhang mit Alternative 2 angesprochen. Sofern die Bediener-Schnittstelle auf mehrere *Applets* "verteilt" wird, stellt sich das Problem der Kommunikation zwischen *Applets*.

3.9.5 Trends

Ohne Zweifel ist die gleichzeitige Präsenz unterschiedlicher *Application Server*-Typen nur eine vorübergehende Erscheinung. Niemand kann daran interessiert sein, zwei *Application Server-Frameworks* für oft weitgehend deckungsgleiche Einsatzgebiete zu lizenzieren. *Web Application Server* und konventionelle *Application Server* werden mit hoher Wahrscheinlichkeit zu hybriden bzw. integrierten *Application Servers* evolvieren.

Langfristig werden sich die Anforderungen auf der Ebene des größten gemeinsamen Nenners einpendeln. Ein *Application Server* muß somit den Leistungsumfang eines TP-Monitors mit den entsprechenden Diensten wie z. B. Ressourcen-Verwaltung unterstützen. Deshalb ist zu erwarten,

Die Lastverteilung ist dynamisch einflußbar

Die Integration gelingt fast vollständig

Alternative 3 ist nicht für komplexe Anwendungen geeignet

335

Unterschiedliche Application Server-Typen sind eine vorübergehende Erscheinung

Application Server werden einen TP-Monitor integrieren

daß sich langfristig nur die Produkte behaupten können, die einen objekt-orientierten TP-Monitor realisieren.

Eine weitgehende Unabhängigkeit von der Umgebung wird möglich

Integrierte *Application Server* ermöglichen eine weitgehende Unabhängigkeit von der Umgebung. Sie unterstützen den Zugriff über unterschiedliche Frontends (statische und dynamische HTML-Frontends, Java *Applets*, konventionelles GUI, s. u.). Daten können mit unterschiedlichen Datenhaltungssystemen verwaltet werden. Die nachfolgende Abbildung zeigt das Prinzip eines integrierten *Application Server*. Als *Object Bus* wird ein CORBA-ORB unterstellt. Die Darstellung ändert sich prinzipiell nicht, falls der CORBA-ORB durch RMI oder COM/DCOM substituiert wird. Das IIOP-Protokoll muß lediglich entsprechend durch JRMP bzw. DCOM ersetzt werden.

336

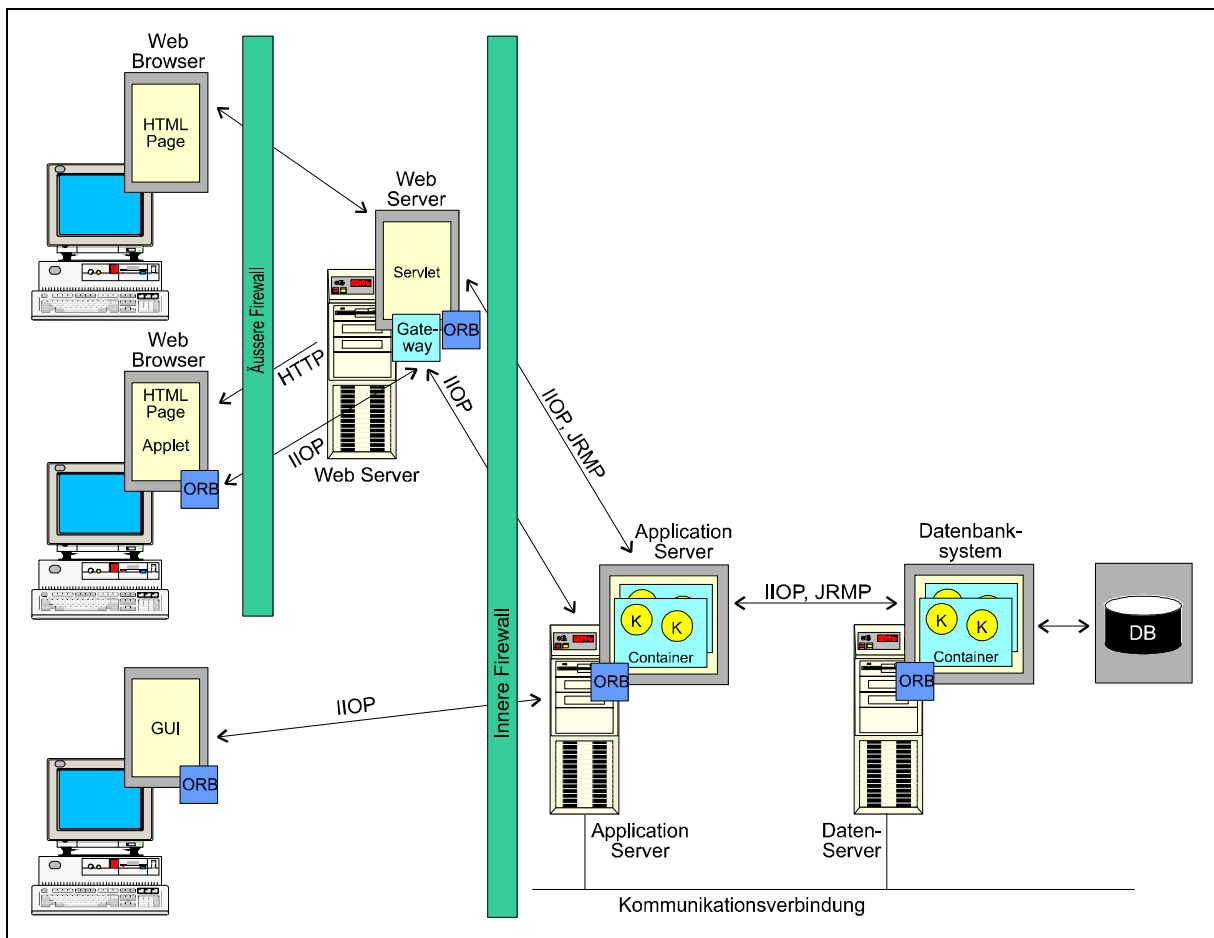


Abb. 3-67: Prinzip eines integrierten Application Server

Application Server-Frameworks sind an die Plattform gebunden

Application Server basieren auf einer bestimmten Plattform für verteilte Objekte sowie einer bestimmten Komponenten-Plattform. Deshalb ist das *Application Server*-Framework technologisch entsprechend gebunden. So kann z. B. der Microsoft *Transaction Server* (MTS) ausschließlich über COM/DCOM kommunizieren und lediglich ActiveX-Komponenten ausführen. Die meisten *Application Server Frameworks* basieren auf einem CORBA-ORB und unterstützen Enterprise JavaBeans-Komponenten.