

Der Entwickler 05/1999

Aller guten Dinge sind drei - Multi-Tierumgebungen und Datenbanken

von Andreas Tengicki und Henry Wolf

Zur Realisierung komplexer Anwendungen werden immer häufiger Multi-Tier-Architekturen eingesetzt. Begriffe wie COM/DCOM, CORBA, bzw. Transaction-Server stehen für diese Technologie. Was bedeutet dies für den Aufbau von Anwendungssystemen insbesondere welche neuen Anforderungen entstehen dabei für den Einsatz und Funktionalität von Datenbanken? Diese und angrenzende Fragen sollen in diesem Artikel besprochen werden.

1. Klassische Client-Server-Technologie

Zu Beginn wollen wir uns kurz die Unterschiede zwischen der Client-Server-Technologie und der Multi-Tier-Architektur beleuchten. Bei der klassischen Client-Server-Architektur spricht man vom sogenannten Two-Tier-Modell. Der Client, also das auf der Workstation ablaufende Programm, beinhaltet die Präsentationsebene (Darstellungsebene, Benutzerschnittstelle) und die Funktionsebene (Programmierfunktionalität). In der Datenbank werden die Daten gespeichert und Bedingungen, die zwischen den Daten gelten, überwacht (referentielle Integrität).

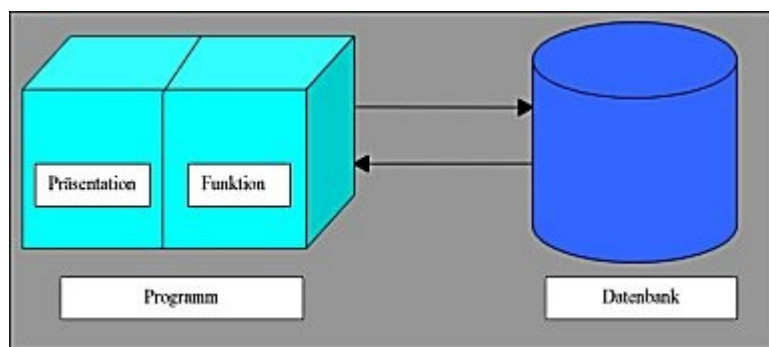


Abb. 1: Klassischer Client/Server-Aufbau

Das besondere an SQL-Datenbanken bzw. relationalen Datenbankmanagementsystemen wie z.B. Oracle, Interbase, Informix und anderen ist, daß Anwendungsfunktionalität auch in der Datenbank abgelegt werden kann. Dies betrifft sowohl die Implementierung von Unterprogrammen auf der Ebene der Datenbank (Storage Procedures, Trigger, Storage Functions, Views) aber auch sogenannte Grundfunktionalitäten, wie die Verwaltung der Benutzerzugriffe (Mehrbenutzerzugriffe) und referentielle Integrität.

Im Normalfall werden bei Client-Server-Systemen Client und Datenbank auf verschiedenen Maschinen implementiert. Die hat einerseits den Vorteil, daß man die Leistung des Anwendungssystems auf zwei unterschiedlichen Maschinen verteilen kann, bzw. in der Lage ist, den Anforderungen entsprechend auch unterschiedliche Systemplattformen einzusetzen. Zum anderen ist es relativ einfach, mehreren Nutzern von unterschiedlichen Clients und auch unterschiedlichen Systemplattformen, die Daten gemeinsam zur Verfügung zu stellen.

2. Neue Aspekte beim Einsatz der Multi-Tier-Technologie

Die Multitier-Technologie ist in vielerlei Hinsicht nur eine logische Erweiterung der klassischen Client-Server-Technologie um den Einsatz einer Middleware, also eines weiteren Computers, der die Anwendungslogik enthält. Dabei wird das ursprüngliche Client-Programm auf die Präsentationsfunktionalität reduziert und die Anwendungslogik, d. h. die Verarbeitung von Daten und die Datenbankzugriffe, auf Basis der eingesetzten Middleware realisiert.

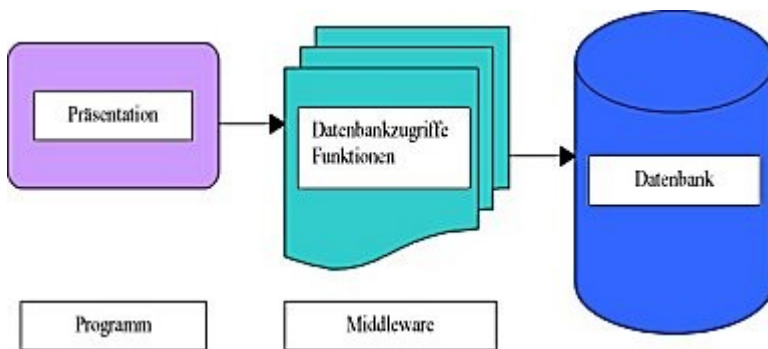


Abb. 2: Aufteilung der Anwendung in Multi-Tier-Architekturen

Das heißt, es entsteht eine neue "Sharing"-Ebene in einem Anwendungssystem. Konnten bisher Daten zwischen verschiedenen Arbeitsplätzen auf unterschiedlichen Systemplattformen verfügbar gemacht werden, können bei Einsatz der Multi-Tier-Technologie auch ganze Programmfunktionen für unterschiedliche Systemplattformen verfügbar gemacht werden.

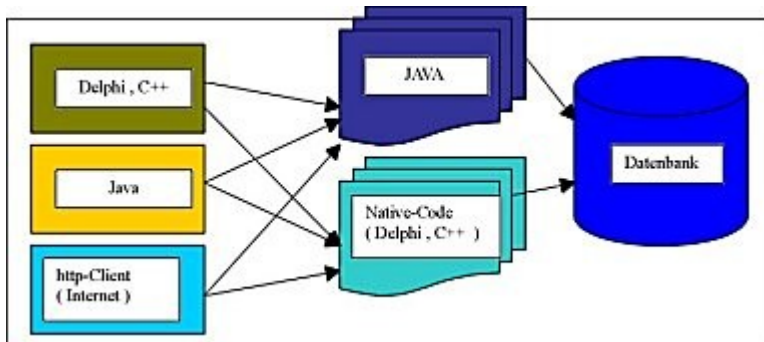


Abb. 3: Realisierungsmöglichkeiten von Multi-Tier-Anwendungssystemen

So kann die Middleware Programmfunktionalität an lokale Clients, welche in der Regel als native-Code-Anwendungen implementiert werden, ebenso zur Verfügung stellen wie über das Intranet/Internet und dessen entsprechende Middleware-Implementierungen, wie COM/DCOM oder CORBA. Diese gestatten die Nutzung von verschiedenen Programmiersprachen zur Implementierung von Anwendungsservern, wie die auf der Basis der Middleware implementierten Programme genannt werden. Dies wird durch die Objektarchitektur solcher Anwendungsserver und sprachunabhängige Schnittstellen zum Zugriff auf die "Objekte" realisiert.

3. Anforderungen an die Hardware

Was ändert sich nun aus der Sicht des Einsatzes von Datenbanken und deren Implementierung beim Einsatz dieser Technologie. Als erstes fällt auf, dass der Haupt-Datenverkehr nicht mehr zwischen den Maschinen, auf welchen die Programme implementiert sind und dem Datenbankserver stattfindet, sondern zwischen der Maschine auf welcher die Middleware installiert wurde und dem Datenbankserver stattfindet. Die Ergebnismengen von Datenbankabfragen werden in der Regel auf dem Middleware-Server zwischengespeichert. Zwischen den Clients und der Middleware werden nur noch die aktuell anzuzeigenden Daten, ähnlich der Verarbeitung mit klassischen Mainframe-Terminals, übertragen.

Bei ausreichender Hardware ist in klassischen Client-Server-Systemen oft das Netzwerk der Engpass. Um dies zu umgehen wird oft mit Subnetzen, sprich mit verschiedene Netzwerke, welche eigenständig mit dem Server verbunden werden, gearbeitet. Diese Möglichkeit besteht in der Regel, zumindest beim Einsatz einer zentralen Middleware-Maschine, in einem Multitiersystem nicht gearbeitet werden. Da der Netzwerk-Traffic zwischen Datenbank und Middleware groß und zwischen Middleware und deren Clients in der Regel gering ist sollte bei der Implementierung eines solchen Systems auf eine entsprechende Ausstattung des Netzwerkes achten. Ein Back-Bone zwischen Datenbank und Middleware ist hier in der Regel die Lösung.

Da wie erwähnt, die Ergebnismengen von Datenbankabfragen nicht mehr auf jedem Client extra zwischengespeichert, sondern zentral auf der Middlewaremaschine gespeichert werden, sollte auch hierbei auf eine entsprechende Hardware, insbesondere Hauptspeicherausstattung geachtet werden. Hinzu kommt, dass viele Middlewareprodukte zur Steigerung der Performance ihrer Systeme eine Cache-Funktionalität bieten. D. h. auf der Datenbank findet je nach Funktionalität des RDBMS eine Caching statt und auf der Seite des Middleware-Servers erneut.

Außerdem soll in diesem Zusammenhang noch erwähnt werden, dass pro Zugriff des Clients eine Instanz des

Anwendungsservers auf der Middleware gestartet wird und Ressourcen anfordert und auch eine logische Verbindung (Session) zur Datenbank eröffnet.

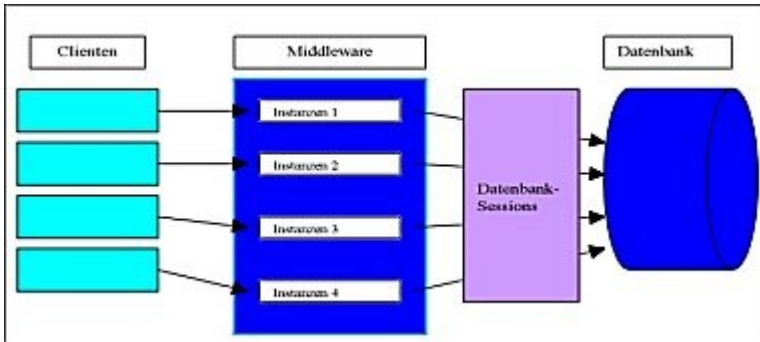


Abb. 4: Instanzen und Sessions

Der eigentliche Client ist auf Grund seiner Funktionsbeschränkung in den Hardwareanforderungen bescheidener als in klassischen Client-Server-Architekturen.

Dies alles zeigt, daß zum Beispiel die Anwendung einer solchen Architektur bei DFÜ-Zugriffen von Clienten besonders gut geeignet ist, da wie erwähnt, der Netzwerk-Traffic zwischen dem Clients und der Middleware bedeutend geringer ist als bei klassischen Client-Server-Systemen und die Ansprüche an die Client-Hardware relativ gering sind.

4. Anforderungen an die Datenbanken und deren Implementierungen

Dadurch, das pro Client-Programm eine Instanz eines Anwendungsserver-Moduls gestartet wird und eine logische Datenbankverbindung aufgebaut wird und zudem ein Client auf mehrere Anwendungsserver-Module zugreifen kann, ergeben sich hieraus natürlich spezielle Anforderungen an die Datenbank. So kann ein Client eventuell mehr als eine logische Datenbankverbindung initiieren. In klassischen Client-Server-Umgebungen initiiert ein Client in der Regel eine logische Datenbanksession.

In Multi-Tier-Umgebungen sind diese unter Umständen bedeutend zahlreicher. Hieraus ergibt sich, daß das Datenbanksystem viele parallele, teilweise konkurrierende Zugriffe (Sessions) und damit auch eine in der Regel höhere Zahl von Transaktionen verarbeiten können muß. Relationale Datenbankmanagementsysteme werden in der Regel diesen Anforderungen gerecht. Bei einem Einsatz von Desktop-Datenbanken müßten diese Mechanismen programmtechnisch implementiert werden.

Desktop-Datenbanken bieten im Gegensatz zu SQL-Datenbanken in der Regel keine Mechanismen zur Transaktionskontrolle an. Gerade in Multi-Tier-Umgebungen spielt dies jedoch eine große Rolle, um solche Systeme sicher und fehlertolerant aufzubauen. Kommt es zum Abruch der Verbindung zwischen Client und Middleware-Anwendungsserver, muß die Datenbank in der Lage sein, die noch stehenden Transaktionen abzuschließen und die logischen Verbindungen zu den von den Clients initiierten Anwendungsserver-Modulen zu beenden.

Im Prinzip werden an das Datenbanksystem gegenüber klassischen Client-Server-Systemen keine neuen Anforderungen gestellt, nur die Bedeutung von Funktionalitäten wie Transaktionskontrolle, Multi-User-Management etc. gewinnen enorm an Bedeutung.

5. Unterschiede bei der Anwendungsentwicklung

Der modulare Aufbau von Multi-Tier-Anwendungssystemen (unterschiedliche Clients, verschiedene Anwendungsserver-Module pro Anwendung) stellt an die Entwickler der Datenbanken die Anforderung, ein flexibles und relativ stabiles Datenbankmodell zu erarbeiten. Eine Änderung schlägt eventuell nicht nur auf eine Anwendung, sondern auf mehrere Anwendungsserver durch, welche dann entsprechend umprogrammiert werden müssen.

Der Aufbau einer Multi-Tier-Anwendung erfordert von der Datenbank bis hin zur Implementierung ein durchdachteres und strukturierteres Vorgehen als es in vielen Bereichen der klassischen Entwicklung in der Praxis vorherrschende Vorgehensweise ist. Wird dies beachtet, ist die Komplexität solcher Anwendungssysteme kein Schreckgespenst.

6. Aufbau einer Multi-Tier-Anwendung

Im folgenden wollen wir kurz den Aufbau einer Multi-Tier-Anwendung schematisch darstellen.

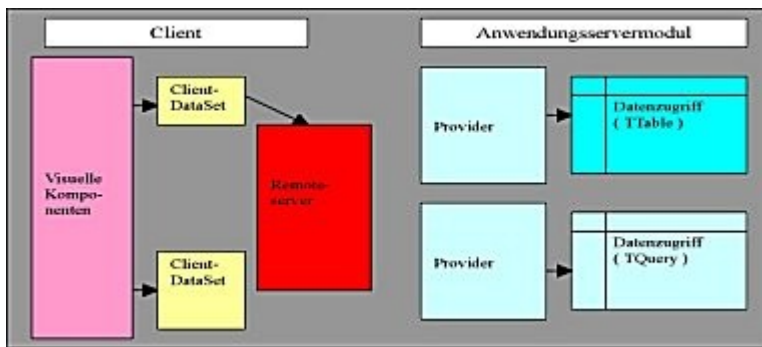


Abb. 5: Aufbau einer Multi-Tier-Anwendung

Als erstes wird man eine der Aufgabe entsprechende Datenbank entwerfen. In einer komplexen Multi-Tier-Architektur wird diese Datenbank von verschiedenen Anwendungsmodulen genutzt und es ist um so wichtiger, daß dieser Entwurf Stabilität in dem Sinne hat, daß nicht jede Erweiterung oder Änderung an einem Anwendungsmodul zu grundlegenden Änderungen an der Datenbank führt. Es gilt, wie bei Client-Server-Systemen, daß ein guter Datenbankentwurf der halbe Weg zur Lösung der Aufgabe ist.

Steht die Datenbank und hat man sich für eine Middleware entschieden, sind in der Regel als Erstes die Anwendungsserver-Module zu entwickeln. In einem Anwendungsservermodul werden, wie erwähnt, die Datenbankzugriffe realisiert und die Ergebnismengen veröffentlicht. Ein jedes Anwendungsmodul kann dabei in einer für sein Problem optimalen Programmiersprache geschrieben werden. In Richtung der Datenbank kommuniziert das Anwendungsmodul über die Datenschnittstelle (BDE, JDBC, API); in Richtung des Clients wird eine Schnittstelle veröffentlicht. Die Syntax zur Veröffentlichung dieser Schnittstelle hängt von der gewählten Middleware Plattform und dem verwendeten Protokoll ab. Die Middleware instanziiert dann die Applikationsmodule und realisiert den Betrieb des Applikationsservers.

Bei der Entwicklung in des Clienten wird via des Middleware-Protokolls (DCOM / Corba u.a.) eine Verbindung zu dem Anwendungsserver-Modul hergestellt und die Ergebnismengen via clientseitiger Komponenten der visuellen Darstellung zur Verfügung gestellt. Dabei erfolgt normalerweise nur noch die Implementierung der Oberfläche, da die eigentliche Anwendungslogik auf den Middlewareserver ausgelagert wurde.

Bei der Gestaltung der Oberfläche ist aber auf besondere Aspekte Rücksicht zu nehmen. Soll die Multi-Tier-Architektur z. B. eingesetzt werden, um die Datenbankbelastung in einem Call-Center zu skalieren, entsteht eine asynchrone Verarbeitung deren spezielle Anforderungen umzusetzen sind. Denn während bei einem Client-Server-System der Client direkt mit der Datenbank verbunden ist und Meldungen direkt an den Client gegeben werden können, z. B. "Bestellung nicht möglich, da Artikel vergriffen", wird dies bei einer Multi-Tier-Umgebung komplexer. Verbindet man die Telefonisten-Arbeitsplätze direkt mit der Datenbank, wird das Bestellsystem direkt nach der Fernsehwerbung vor Hunderten von Bestellungen, die in wenigen Minuten eingehen, einigermäßen sicher in die Knie gehen. Daher ist es sinnvoll, zwischen Client (Arbeitsplatz) und Datenbank einen oder mehrere Applikationsserver zu stellen, die eine Vorbearbeitung für die Datenbank machen und z. B. Sammelbestellungen statt Einzelbestellungen generieren. Nur was, wenn ein Artikel vergriffen ist? Was tun, wenn diese vielleicht schon mit dem nächsten Kunden telefoniert. Alles Fragen, die auf Grund der Asynchronität der Ereignisse entstehen und im Rahmen der konkreten Aufgabenstellung beantwortet werden müssen.

7. Vorteile bei der Systempflege

Diese neue Technologie bringt nicht nur Neuerungen für die Entwickler mit sich, sondern auch für die "Pfleger", d.h. die Betreiber von Multi-Tier Anwendungssystemen und insbesondere für die Datenbankadministratoren. Dies ist in der Regel eines der entscheidenden Kriterien für den Einsatz von Multi-Tier-Architekturen.

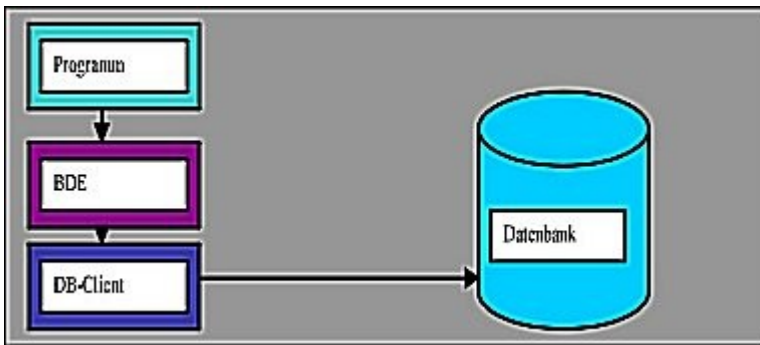


Abb. 6: Installationsschema klassische Client-Server-Umgebung

So muß in klassischen Client/Server - Systemumgebungen auf jedem Arbeitsplatz neben der Anwendung auch der Datenbankclient installiert und konfiguriert werden. Bei Installationen mit vielen Arbeitsplätzen führt das häufig zu Erscheinungen, welche Administratoren als "Turnschuhnetzwerk" bezeichnen. Bei jedem Versionswechsel der Datenbank oder durch den Anwender verursachten Problemen müssen alle entsprechenden Arbeitsplätze neu installiert bzw. die Einstellungen überprüft werden.

Einige Datenbanken, wie z.B. Oracle mit Oracle Names, stellen zwar einige Erleichterungen, insbesondere für die Konfiguration der Arbeitsplätze zur Verfügung, dies schließt jedoch nicht alle Probleme aus. Jeder Entwickler, welcher mittels BDE auf Datenbanken zugreift, kennt die Probleme insbesondere bei vielen Installationen oder bei Installationen auf Systeme, auf denen z.B. Produkte von T-Online oder D-Info installiert sind.

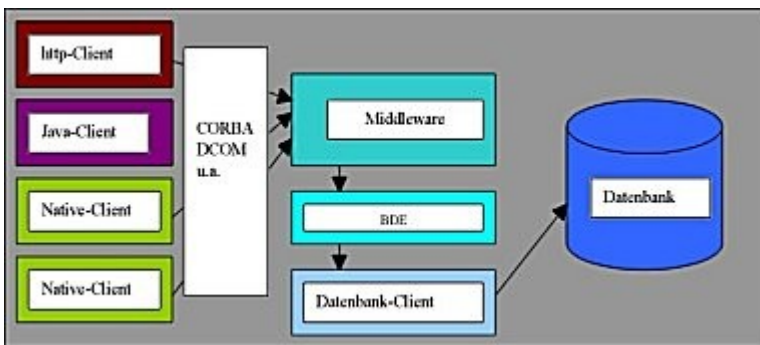


Abb. 7: Installationsschema bei Multi-Tier-Installationen

Diese Probleme werden durch eine zentrale Installation der Datenbankschnittstelle auf dem Middleware-System erheblich reduziert. Je mehr Arbeitsplätze innerhalb eines solchen Anwendungssystems betrieben werden, je effektiver wird somit auch der Betrieb / Unterhalt einer solchen Lösung.

8. Fazit

Multi-Tier-Anwendungen sind komplexer als klassische Client-Server-Systeme. Sie bieten jedoch sowohl für den Aufbau von Anwendungssystem, als auch für die Pflege des Anwendungssystems aus Entwicklersicht als auch aus der Sicht des Betreibers der Installation zahlreiche Vorteile.

Der Markt bietet mittlerweile viele Produkte im Bereich Middleware an. Hierbei sei auf den Artikel "MiddlewareMania" von Adrian Volgler in der Ausgabe 1/99 des *Entwicklers* verwiesen, in welchem eine Übersicht über verfügbare Produkte vorliegt. Ebenfalls sei an dieser Stelle das Buch "COM/DCOM mit Delphi" von Andreas Kosch verwiesen, erschienen im Software & Support Verlag, erwähnt, welches sich insbesondere mit den Produkten auf der Plattformen von Microsoft (COM/DCOM/MTS) auseinandersetzt.

Für die Integration von Datenbanken sei als kurzes Fazit erwähnt, das der Einsatz von Desktop-Datenbanken wie z.B. Paradox oder dBase auf Grund der Anforderungen an solche Systeme verbietet. Entscheidend für die Stabilität und Funktionalität ist jedoch die Qualität der eingesetzten Middleware. Stabile Datenbankmanagementsysteme bieten schon aus den Anforderungen der klassischen Client-Server-System eigentlich alles was an Funktionalität benötigt wird (Transaktionsmechanismen u.a.).