CORBA 3.0 Components Model

Eine Seminarausarbeitung

Von

Jan Gerritsen

Seminar Verteilte und Parallele Systeme I (SS 02)

Fachhochschule Bonn-Rhein-Sieg

Abgabe: Juni 2002

1. Einleitung Grundlagen

1.1. Über dieses Dokument

Mit dem Komponentenmodell des neuen CORBA 3.0 Standards stellt die Object Management Group (OMG) eine umfangreiche und offene Plattform für verteilte Anwendungen bereit. Das Komponentenmodell ist, im Gegensatz zu Konkurrierenden Konzepten, nicht auf eine festgelegte Programmiersprache oder eine Betriebssystemplattform beschränkt.
Diese Ausarbeitung versucht, einen Eindruck über die Grundlegenden Mechanismen und Elemente des Komponentenmodells zu vermitteln und aufzuzeigen, inwieweit CORBA mit dem neuen Standard die Anforderungen an eine moderne Middelware für verteilte Anwendungen erfüllen kann, sobald die momentan noch fehlenden vollständigen Implementierung seitens der Produktanbieter zur Verfügung stehen.

1.2. Warum das Komponentenmodell entwickelt wurde

Die Informatik und die damit verwandten Technologien haben in den letzten Jahren als treibende Kraft, Einzug in sehr viele Bereiche des Lebens gehalten. Und obwohl sie eine moderne Ingenieurswissenschaft ist, hat es dennoch immer wieder den Anschein das es sich um ein altes Handwerk handelt. So werden immer wieder ähnliche Produkte von unterschiedlichen Personen in mühsamer Handarbeit entwickelt, ohne das präzise Entwürfe, Fertigbauteile, Konstruktionsanleitungen oder ähnliches zum Einsatz kommen.
Es gibt unterschiedliche Ansätze, wie Modularisierung, Design Pattern und Objektorientierung, um den Entwicklungsprozess zu professionalisieren.
Die Wiederverwendbarkeit spielt dabei eine zentrale Rolle, wozu ausgefeilte Komponentenkonzepte entwickelt werden.

Gerade bei der zunehmenden Komplexität der Anwendungen, durch die zunehmende Vernetzung und Verteilung, werden gut durchdachte und sauber geplante Konzepte und Softwareentwürfe immer wichtiger. Ein zentraler Gedanke ist den Entwickler, mit Hilfe einer höhere Abstraktion durch mehrschichtige Architekturen, möglichst wenig mit den zu Grunde liegenden Systemen in Berührung kommen zu lassen. Erfahrungsgemäß verbringen

> *„...Programmierer bei der Konstruktion verteilter Systeme über 50 Prozent ihrer Zeit damit, ihre Anwendung in die bestehende Infrastruktur zu integrieren, beispielsweise in die vorhandenen Kommunikationsmechanismen, Transaktionsmonitore, Sicherheitskonzepte, Multithreading-Bibliotheken oder Datenbanksysteme“*[1]

Die Heterogenität heutiger Serverlandschaften mit unterschiedlichen Standards erschwert die Installation verteilter Dienste auf mehreren Plattformen noch zusätzlich.

[1] [Wagner_2000]

Bei der Bewältigung dieser Probleme wird heutzutage die größte Hoffung auf die Komponentenarchitektur gelegt. Zentrale Fragestellung, zu den einzelnen der unterschiedlichen Architekturen, ist dabei stets die Interoperabilität, also in welcher Weise unabhängig voneinander entwickelte Komponenten zur Laufzeit miteinander arbeiten. Ebenfalls im Interesse der Entwickler steht die Sprachenabhängigkeit. Zu guter Letzt ist die Plattformenabhängigkeit ein Ziel und die damit verbunden die Frage, wie Komponenten über Plattformen-, Netzwerke- und Systemgrenzen hinweg Informationen austauschen.
Zu den am weitesten entwickelten Komponentenarchitekturen zählen neben CORBA, die Gegenstand der weiteren Ausführung sein wird, noch die Enterprise Java Beans (EJB) von Sun sowie Microsofts COM+. An dieser Stelle soll kein Vergleich der unterschiedlichen Modelle erfolgen. Allerdings weisen die beiden anderen Architekturen jeweils Restriktionen auf, die die CORBA-Architektur nicht betreffen und daher hier kurz erwähnt werden sollen.
So ist der Einsatz von CORBA im Gegensatz zu EJB nicht an eine spezifische Programmiersprache und im Gegensatz zu COM+ nicht an eine spezifische Plattformumgebung gebunden und entspricht daher eher der Vorstellung eines universalen Komponentenmodells.

2. CORBA – Architektur

2.1. Was ist CORBA?

Die Common Object Request Broker Architecture (CORBA) ist eine von dem Standardisierungskonsortium Object Management Group (OMG) erarbeitete Spezifikation,
welche

> *„die Zusammenarbeit von Softwaresystemen über die Grenzen von Programmiersprachen, Betriebssystemen und Netzwerken hinweg regelt.“*[2]

CORBA stellt eine universelle Kommunikationsplattform dar, mit deren Hilfe Objekte zusammenarbeiten können, deren Implementierungen sich auf unterschiedlichen Plattformen befinden.

2.2. Funktionsweise von CORBA

Der Datenaustauschmechanismus erfolgt über einen Objekt-„Bus“, der Object Request Broker (ORB) genannt wird (siehe Abbildung 1 CORBA – Kommunikationsvorgang). Er sorgt dafür, dass Objekte bzw. Software-Komponenten durch Nachrichtenaustausch miteinander über Grenzen von Betriebssystemen und Hardwareplattformen hinweg kommunizieren können.

Neben dem ORB besteht die CORBA-Architektur aus vier weiteren Bereichen:

CORBAservices,
CORBAfacilities,
Domain Interfaces und
Application Objects.

Die ObjectServices stellen eine Erweiterung des ORBs auf Systemebene dar, indem Dienste (Transaction, Event, Naming, Security, Time, ...) bereitgestellt werden. Die Dienste sind hochgradig modular, elementar und redundanzfrei spezifiziert.

Eine Erweiterung der ObjectServices für endanwenderorientierte Fähigkeiten wird mit den Common Facilities zur Verfügung gestellt. Hier werden unter anderem printing facilities, database facilities, electronic mail facilities und ähnliche häufig benötigte Dienste bereitgestellt.

Standardisierte Schnittstellen für spezielle Anwendungsgebiete schließlich werden in den Domain Interfaces zusammengefasst. Diese Schnittstellen werden jeweils in Arbeitsgruppen (Task Forces) ermittelt und verabschiedet.

[2] [Wang_2000]

2.3. CORBA 3.0

In der CORBA-Spezifikation bis zur Version 2.3 werden verschiedene Problemfelder nicht adressiert, was zu Ad-hoc-Lösungen mit Objekten führte, die schwer zu entwickeln, wiederzuverwenden, zu verteilen und zu erweitern waren. Einige dieser Problematiken werden im folgenden angesprochen.

- **Entwicklungsprozess**
  Die CORBA-Spezifikation definiert keinen standardisierten Weg, um Server-Objekte zu implementieren. Im besonderen fehlen Richtlinien, wie Objekte zu verteilen und in ihrer jeweiligen Laufzeitumgebung zu installieren sind. Auch für die Aktivierung der Objekte durch das System müssen Ad-hoc-Strategien entwickelt werden.

- **Fehlende Unterstützung für die Erstellung von Server-Objekten**
  Die Spezifikation unterstützt viele Möglichkeiten, Servereigenschaften zu entwickeln. Es hat sich jedoch gezeigt, das oftmals nur ein kleiner Ausschnitt dieser Möglichkeiten effektiv genutzt wird, so dass eine automatische Erzeugung eben dieser wenigen Kombinationen wünschenswert wäre.

- **Verfügbarkeit der ObjectServices**
  Die Spezifikation sieht nicht vor, welche der ObjectServices zur Laufzeit verfügbar sein müssen. Resultierend daraus muss ein Entwickler Strategien entwickeln, wie die benötigten Dienste konfiguriert und aktiviert werden können.

- **Objektlebenszyklus**
  Da die Unterstützung des LifeCycleService keine Voraussetzung ist, müssen für die Bereitstellung der benötigten Funktionen für die Verwaltung eines Objektes über seinen gesamten Lebenszyklus hinweg oftmals individuelle Lösungen entwickelt werden.

Um dieser Problematik in einer angemessenen Weise gerecht zu werden und gleichzeitig wichtige Bereiche der modernen Softwareentwicklung zu adressieren, wurde die CORBA 3.0 – Spezifikation verabschiedet. Mit diesem neuen Standard adressiert die OMG vor allem 3 Gebiete, die im folgenden kurz angesprochen werden.

1. Varianten

Neben der normalen CORBA-Middleware gibt es künftig Varianten für bestimmte Einsatzzwecke. Minimum CORBA legt die minimale Funktionalität für ein System fest, das an der CORBA-Kommunikation teilhaben kann. Realtime CORBA legt die Voraussetzungen fest, die echtzeitfähige Implementierungen des CORBA-Standards erfüllen müssen. Fault Tolerant CORBA schließlich definiert die Voraussetzungen für eine ausfallsichere Nutzung der CORBA-Kommunikation.

2. Internet

Die verbreitete Nutzung des Internets beim Einsatz verteilter Anwendungen und der damit verbundenen Einsatz von Firewalls, der bei der Kommunikation zwischen Clients und CORBA-Objekten Berücksichtigung finden muss, resultierte in einer Spezifikation, die die CORBA-Kommunikation im Zusammenhang mit Firewall-Systemen regelt.

3. Komponenten

Ein Komponentenmodell legt die Grundlagen für die Zusammenarbeit von Softwarebausteinen fest. Dies ist Gegenstand der weiteren Ausführungen.

3. CORBA Components Model

Das Komponentenmodell von CORBA ist Gegenstand dieses Kapitels. Es soll zunächst ein Überblick über die unterschiedlichen Elemente des Modells gegeben werden, eher diese dann im weiteren Verlauf eingehender betrachtet werden.

Das Komponentenmodell ist eine Erweiterung des CORBA-Objektmodells und stellt als serverseitige Architektur eine Spracheneunabhängige Verallgemeinerung der Enterprise Java Beans dar. Die Hauptbestandteile des Modells sind Komponenten, Container, eine entsprechende Infrastruktur sowie ein standardisierter Entwicklungs- und Distributionsprozess.

Die Komponenten sind aus Entwickler- oder Anwendersicht nach wie vor die Elemente des Modells, mit denen sie hauptsächlich in Berührung kommen, werden dabei nun jedoch durch die anderen Elemente des Modells unterstützt. Der Entwickler kann sich ganz auf die Bereitstellung der Applikationslogik konzentrieren.

Eine Komponente ist eingebettet in einen Container, der eine Schale um sie bildet und Schnittstellen für eine transaktionsfähige, sichere und persistente Umgebung zur Verfügung stellt. Ein solcher Container wiederum nutzt die Dienste eines darrunterliegenden ORB bzw. eines Application Server. Ein einheitliches Distributionsformat sorgt dafür, dass Komponenten zwischen unterschiedlichen Implementierungen der Container ausgetauscht werden können.

Mit dem Komponentenmodell zielt man auf die Erreichung eines konsistenten und serverbasierten Frameworks für den Komponenteneinsatz in verteilten Umgebungen sowie die Unterstützung gebräuchlicher Techniken zur Herstellung von Server-Objekten.

3.1. Komponenten

> *„A component is a basic CORBA meta-type, i.e., it can be referenced by multiple object references of different types"*[3]

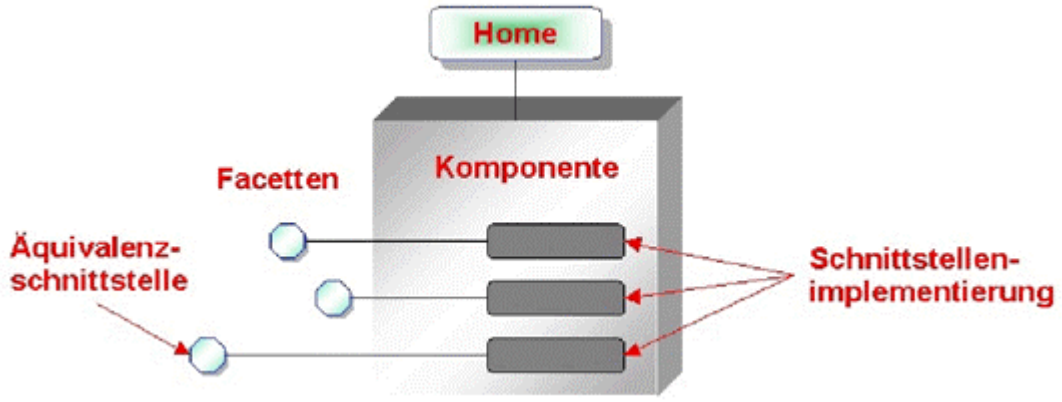
Durch das Komponentenmodell wird der Weg der Portabilität, den die Enterprise Java Beans-Spezifikation für die Java-basierten Applikations-Server ebnet, hin zur Interoperabilität über Systemgrenzen hinweg erweitert. Die schon angesprochene Anlehnung des Architekturmodells an die Enterprise Java Beans ermöglicht durch ein Standardmapping, CORBA-Komponenten als „beans" und umgekehrt „beans" als CORBA-Komponenten einzusetzen.

Zunächst werden die Elemente und Schnittstellen einer Komponente betrachtet, die aus Clientsicht, also aus der Sicht dessen, der die Dienste dieser Komponente nutzen will, relevant sind. Im Anschluss daran richtet sich der Fokus auf die Bereiche, die vor allem für den Entwickler der Komponente bedeutsam sind.

[3] [Wang_2000]

3.2. Clientsicht auf Komponenten

Die Referenz auf eine Komponente ist vergleichbar mit der auf ein normales CORBA-Objekt. Dadurch können Clients, die noch keine Komponenten kennen, diese wie eine bekannte Objektreferenz benutzen, wodurch die Kompatibilität zu den vorangegangenen Versionen gewährleistet wird.
Diese Referenz auf eine Instanz einer Komponente verweist immer auf eine ausgezeichnete Schnittstelle der Komponente, der sogenannten Äquivalenzschnittstelle. Diese Äquivalenzschnittstelle kann über Einfachvererbung Funktionen einer anderen erben, wobei diese Schnittstellen „supported interfaces“ genannt werden. Daneben wird es aber dem Komponentenentwickler auch ermöglicht, beliebig viele zusätzliche Schnittstellen zu implementieren, die Facetten oder „provided interfaces“ genannt werden. Diese zusätzlichen Schnittstellen werden mit Hilfe des Schlüsselworts supports kenntlich gemacht. Einem Client ist es möglich, über die Äquivalenzschnittstelle beliebige Facetten anzusteuern. Mit Hilfe von get_component(), einer neuen Methode eines Objektes, lässt sich jederzeit zur Äquivalenzschnittstelle der Komponente zurückkehren.



Durch die Möglichkeit der zusätzlichen Schnittstellen wird die Wiederverwendbarkeit und Erweiterbarkeit beträchtlich gesteigert. So ist es nun beispielsweise möglich, einer Komponente ergänzende Funktionalität in Form einer zusätzlichen Schnittstelle zu geben, ohne dass bestehende Clients der Komponente davon betroffen sind.
Der gesamte Lebenszyklus einer Komponente wird durch einen (eindeutigen) Home verwaltet, der die entsprechenden Schnittstellen zur Verfügung stellt. Clients können die angebotenen Schnittstellen nutzen, um auf eine standardisierte Weise den gesamten Lebenszyklus der Komponente zu kontrollieren. Die tatsächliche Implementierung dieses Lebenszyklus ist wieder abhängig von der zugrundeliegenden Architektur, da nur die Schnittstellen spezifiziert werden.
Jeder Komponententyp wird durch genau einen Home gesteuert. Ein Home kann optional Primärschlüssel benutzen, um so eine Zuordnung von Schlüsselwerten zu Komponenteninstanzen herzustellen. Bei einem Home, der keine Primärschlüssel benutzt, wird demnach jeweils nur eine Instanz der verwalteten Komponente erzeugt, ohne dieser Instanz einen eindeutigen Schlüssel zuzuordnen.
Referenzen auf alle verfügbaren Homes werden in einer zentralen Datenbank verwaltet, auf die ein Client über die Methode resolve_initial_references(„HomeFinder“) Zugriff erlangt.
Von dort ermöglichen es ihm sogenannte Finder-Methoden, nach vorhandenen Instanzen einer Komponente zu suchen.

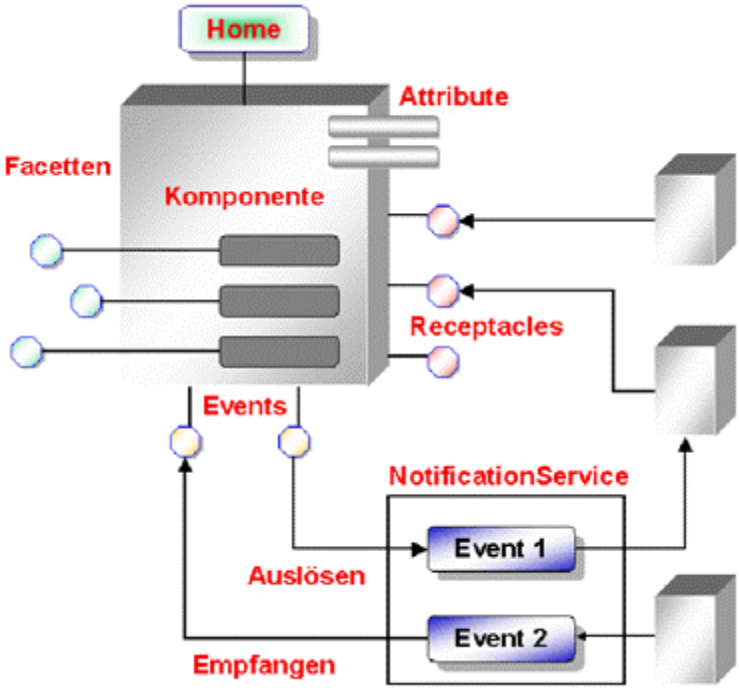
3.3. Entwicklersicht auf Komponenten

Eine der Nachteile der Spezifikationen bis einschließlich CORBA 2.3 war die fehlende Unterstützung bei der Erstellung häufig benötigter Techniken und Eigenschaften eines Objektes. Dies wird nun dadurch gelöst, dass Komponenten mit anderen Einheiten (ObjectServices, Komponenten, Clients, ...) über Schnittstellen, die Ports genannt werden, kommunizieren können. Es gibt vier unterschiedliche Ports (Facetten, Receptacles, Events und Attributes). Facetten wurden bereits im vorangegangenen Kapitel beschrieben, die restlichen werden im folgenden näher erläutert.

- **Receptacle**
  Facetten stellen eine Möglichkeit dar, Schnittstellen nach außen anzubieten. Daneben ist es allerdings auch sinnvoll, definierte Eingänge für eine Komponente zu spezifizieren, über die sie Funktionen ihrer Umgebung nutzen. Das Komponentenmodell bietet daher die Möglichkeit an, solche Schnittstellen, die die Komponente nicht selbst spezifizieren, sondern stattdessen importieren möchte, anzugeben. In der CORBA-Sprache werden solche Schnittstellen Receptacle genannt. Entwickler bringen mit der Angabe von Receptacles die Bereitschaft zum Ausdruck, ihre Komponente mit anderen Objekten / Komponenten zu verbinden. Ist diese (gewünschte) Verbindung etabliert, so spricht man von einer „object connection“.

- **Events**
  Als Ergänzung zur Interaktion von CORBA-Komponenten über Methodenaufrufe wird ein verteiltes Ereignismodell bereitgestellt, mit dem eine Komponente den Status einer anderen überwachen und reagieren kann, sobald eine bestimmte Statusänderung auftritt. Das Ereignismodell wird über den NotificationService vermittelt, der die Ereignisse über Kanäle weiterleitet, und orientiert sich an dem Observer Pattern. Entwickler spezifizieren das Interesse einer Komponente, Ereignisse zu senden oder zu empfangen, indem sie *event sources* bzw. *event sinks* der Komponentendefinition hinzufügen. Des weiteren kann noch angegeben werden, ob mehrere Empfänger ein Event empfangen sollen oder dies explizit nur einem Empfänger ermöglicht wird.

- **Attribute**
  Attribute schließlich ermöglichen eine Konfiguration der Komponente.



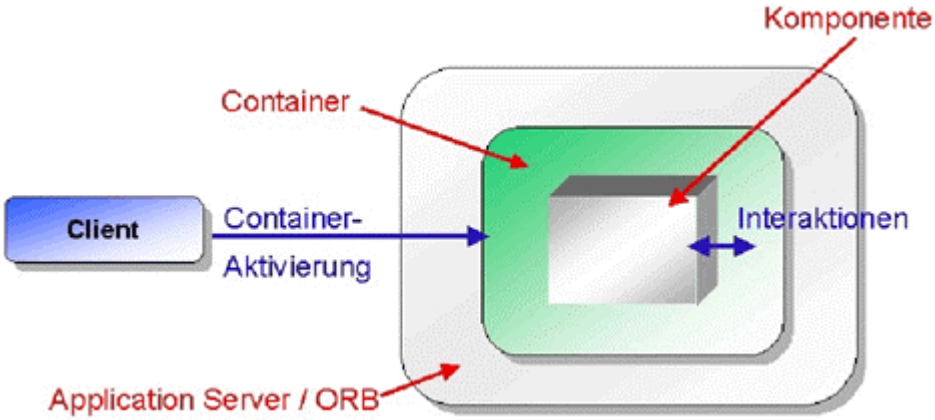
3.4. Container

Die den Containern zugrundeliegende Überlegung ist, dass eine Komponente kein Wissen über die Interaktion mit dem darunter liegenden System benötigt. Stattdessen wird durch einen Container eine Laufzeitumgebung für Komponenten bereitgestellt, die Container-APIs welche die Entwicklung und Konfiguration von Komponenten unterstützen.
Ein Container weist zwei unterschiedliche Arten von Schnittstellen auf: Die External APIs sowie die Container APIs.

- **External APIs**
  External APIs sind direkt durch andere Elemente des Modells (Clients, fremde Komponenten, ...) nutzbar. Hierzu zählen die Äquivalenzschnittstelle, die Facetten sowie der Home.

- **Container APIs**
  Bei den Container APIs unterscheidet man in Internal Interfaces und Callback Interfaces. Über Internal Interfaces ist es der Komponente möglich, die vom Container angebotenen Dienste zu nutzen. Beispielsweise stehen hier Methoden bereit, um die Transaktions-, Sicherheits- oder Ereignisschnittstellen abzurufen. Demgegenüber sind die Callback Interfaces von der Komponente zu implementieren, um dem Container Rückrufschnittstellen zur Verfügung zu stellen, über die beispielsweise eine Benachrichtigung über relevante Ereignisse erfolgt.

Art und Umfang der Nutzung dieser Schnittstellen sind natürlich maßgeblich von der Art der zu verwaltenden Komponente sowie der jeweiligen Aufgabe abhängig. Um Komponentenreferenzen mit oder ohne einem Persistenzstatus zu verwalten, existieren in dem Modell daher Container in zwei unterschiedliche Ausprägungen: Entity Container unterstützen persistente Komponenten, während Session Container für eine transiente Verwaltung der Komponenten zuständig sind.



Für die Interaktion von Containern mit den zugrundeliegenden CORBA-Diensten existieren die sogenannten „Container Implementation Types“. Bei der Verwendung eines transienten Containers ist nur eine zustandslose Interaktion möglich. Hierbei unterscheidet man zwei unterschiedliche Typen von Interaktionen in Abhängigkeit vom Verhältnis „Instanz zu Servant“. Wenn ein Servant sämtliche Instanzen eines Komponententyps repräsentiert, dann spricht man von einer Interaktion des Typs „stateless“, wohingegen beim Typ „conversational“ zu jeder Instanz ein eigener Servant geordnet wird. Bei persistenten Containern gibt es nur einen Containerimplementierungstyp, der „durable“ genannt wird. Auch hierbei wird jede Komponente mit einer eindeutigen Referenz auf einen Servant verbunden, allerdings kommt hierbei eine persistente Referenz zum Einsatz.

Daraus ergeben sich 4 mögliche Komponentenkategorien als Kombinationen von Containertyp, Container Implementation Type sowie der Verwendung / Nichtverwendung von Primärschlüsseln für die Komponenten (siehe auch Tabelle 1 CORBA-Komponenten).

1. **Service Components** sind zustandslos und damit nicht an einen Client gebunden.
2. **Session Components** besitzen einen transienten Zustand, verändern diesen aber in Abhängigkeit vom Clientaufruf, womit eine feste Assoziation mit dem Client für die Dauer einer Sitzung besteht.
3. **Process Components** besitzen einen persistenten Zustand, der Zugriff ist allerdings nur durch den Originalclient möglich.
4. **Entity Components** besitzen ebenfalls einen persistenten Zustand, durch die Verwendung eines Primärschlüssels ist der Zugriff jedoch von jedem Client aus möglich.

Komponentenkategorie	Containertyp	Containerimplementierungstyp	Primärschlüssel
Service	Transient	Stateless	Nein
Session	Transient	Conversational	Nein
Process	Transient	Durable	Nein
Entity	Persistent	Durable	Ja

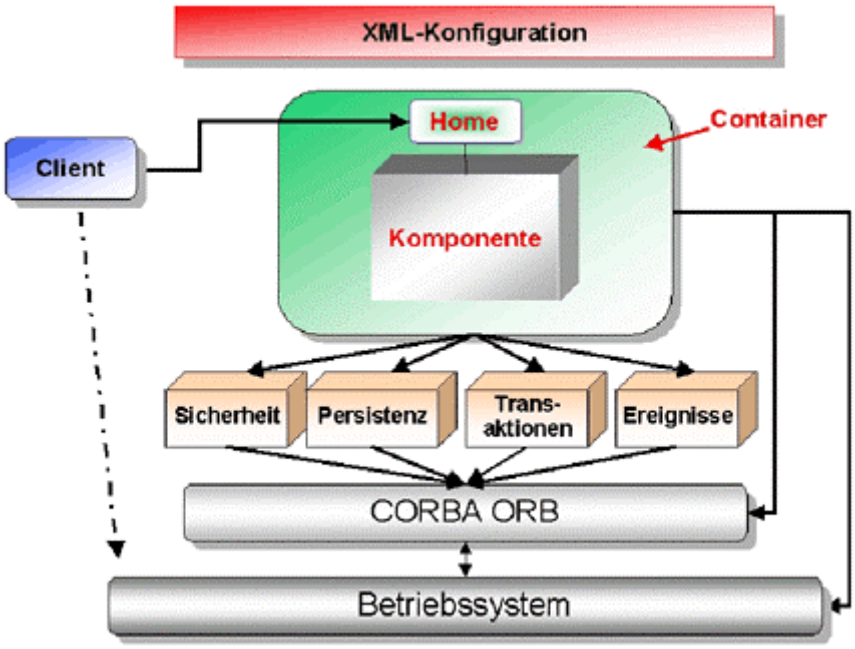
Container administrieren auch den Lebenszyklus des Komponenten-Servants. Dabei kontrollieren vier unterschiedliche Richtlinien die Aktivierung und Deaktivierung von Komponenten:

- **METHOD**: Die Aktivierung einer Komponente erfolgt vor, ihre Deaktivierung nach jedem Methodenaufruf.
- **TRANSACTION**: Hier erfolgt die Aktivierung der Komponente beim ersten Aufruf innerhalb der Transaktion, ihre Deaktivierung bei Abschluss der Transaktion.
- **COMPONENT**: Mit dem ersten Methodenaufruf erfolgt die Aktivierung der Komponente, eine Deaktivierung geschieht erst auf explizite Anfrage der Komponente.
- **CONTAINER**: Auch hier erfolgt die Aktivierung der Komponente mit dem ersten Methodenaufruf, der Container initiiert ihre Deaktivierung.

Die Möglichkeit METHOD ist nur bei Verwendung des Containerimplementierungstyps „*stateless*“ anwendbar, während beim Typ „*conversational*“ oder „*durable*“ alle 4 Möglichkeiten Verwendung finden können.

3.5. Schnittstellenbeschreibungen

Die Schnittstellen der CORBA-Objekte wurden bisher mit Hilfe der Interface Definition Language (IDL) beschrieben. Die IDL ist eine abstrakte Sprache, die keine Programmiersprache darstellt, da keine Implementierungsaspekte adressiert werden.
Mit der neuen Version der Spezifikation erfolgt die Beschreibung von Komponenten in einer Erweiterung der IDL, der Component IDL. Diese Erweiterung wurde notwendig, um die Abbildung von Aspekten des Komponentenmodells, die im CORBA-Objektmodell bis zur Version 2.3 noch nicht vorhanden waren, zu gewährleisten. Beispielhaft sei hier als ein Aspekt die Unterstützung der unterschiedlichen Ports genannt. Somit können Entwickler die (Component)-IDL zur Spezifikation der Operationen nutzen, die eine Komponente unterstützt soll. Auch hierbei ist natürlich die Kompatibilität zu vorangegangenen Versionen der IDL wünschenswert, was durch die Angabe eines äquivalenten Ausdrucks in der IDL für jeden Ausdruck der Component IDL ermöglicht wird. Durch diese Abbildung auf bisherige Sprachmerkmale wird eine Migration ermöglicht.
Allerdings reichten diese Erweiterungen der IDL für eine vollständige Schnittstellenbeschreibung von Komponenten nicht aus, es bedurfte noch weiterer Ergänzungen.
Hierzu wurde mit der **Component Implementation Definition Language** (CIDL) eine deklarative Sprache definiert, um die Implementierung und den Einsatz von Persistenzmechanismen einer Komponente zu beschreiben. Die CIDL stellt dem Entwickler abstrakte sprachliche Mittel zur Verfügung, um eine Komponente sowie deren Zusammenwirken mit Persistenzmechanismen zu beschreiben.
Die Entwicklung einer CORBA-Komponente beginnt mit der Angabe der Schnittstellen und der Erzeugung einer entsprechenden Beschreibung durch einen IDL-Compiler. Dieser Schnittstellenbeschreibung können nun weitergehende Aspekte bezüglich des Verhaltens in der CIDL hinzugefügt werden, woraus durch einen CIDL-Compiler sogenannte Implementierungsskeletons erzeugt werden. In einem letzten Schritt wird die Funktionalität der Komponente in der Zielprogrammiersprache hinzugefügt, und ein Compiler erzeugt aus den einzelnen Artefakten (CIDL-Skeleton, IDL-Skeleton und Source-Code) die Komponente.



Persistenzmechanismen werden mit Hilfe des Persistent State Service 2.0 angegeben, der in die CORBA 3.0-Spezifikation integriert wurde. Ähnlich zur IDL können auch hier abstrakte Schnittstellenbeschreibungen angegeben werden, die sich allerdings auf Datenobjekte beziehen. Für den Entwickler transparent werden mit Hilfe eines Compilers daraus programmiersprachliche Lese- und Schreiboperationen erzeugt, die den Zugriff auf zugrundeliegende Speichermedien managen.
Bezüglich der Speicherung von persistenten Zuständen sieht das Komponentenmodell zwei Möglichkeiten für die Zuständigkeiten vor. Ist der Container für die Durchführung der Persistenzmechanismen zuständig, spricht man folgerichtig von „*Container-managed Persistence*“. Der Container erzeugt dann automatisch die benötigten Speicher- und Ladeoperationen und führt diese aus. Wenn der Entwickler diese Aufgabe nicht dem Container übertragen, sondern stattdessen selbst für die entsprechenden Aktionen zuständig sein möchte, dann spricht man demgegenüber von „*Component-managed Persistence*“. Er kann sich dabei des Persistenzdienstes bedienen oder stattdessen proprietäre Mechanismen einsetzen.

4. Fazit und Ausblick

Der Markt für Web-Anwendungen leidet bislang darunter, dass sich die Programme nicht einfach von einem Applikations-Server auf einen anderen verlagern lassen. CORBA 3.0 hilft, diese Inkompatibilitäten bei Applikations-Servern einzudämmen.
Allerdings ist die Spezifikation CORBA 3.0 zum einen sehr umfangreich und komplex, zum anderen ist der Prozess der Komponenteninteraktion und –installation noch nicht vollständig geklärt. Es bleibt also abzuwarten, wann die ersten Implementierungen diese Spezifikation unterstützen, in welchem Umfang die Spezifikation umgesetzt wird und wie die vorgeschlagenen Werkzeuge zur Unterstützung des Entwicklungsprozesses die tatsächliche Softwareentwicklung unterstützen können.

Es gibt es eine Fülle von Anbietern, die CORBA – Implementierungen bis zur Version 2.3 der Spezifikation anbieten. Doch so sehr diese Vielzahl von Anbietern auch wünschenswert ist, so entstehen doch gerade dadurch auch gewisse Problematiken im Hinblick auf die weitere Entwicklung. So werden oftmals nur Teile der jeweiligen Spezifikation unterstützt, ein Vergleich der einzelnen Anbieter ist nur schwer möglich. Und gerade in letzter Zeit sind einige Anbieter dazu übergegangen, proprietäre Erweiterungen als Differenzierungsmerkmal ganz bewusst einzubauen, was Auswirkungen auf die Interoperabilität hat.
Als Ausblick bleibt die Gewissheit, dass der CORBA-Standard durch die Version 3.0 weiter vervollständigt wird. Die auf der Basis der neuen Version entworfenen Architekturen werden einfacher zu handhaben sein. Insbesondere durch die Standardisierung der Komponentenarchitektur wird CORBA an Bedeutung gewinnen, da es nun eine „echte“ Komponentenarchitektur für verteilte Systeme darstellt. Da die Grundkonzepte des bisherigen Standards nicht wesentlich angetastet werden, ist für bereits existierende Systeme nicht mit gravierenden Änderungen oder Erweiterungen zu rechnen und somit die Kompatibilität der Versionen gewährleistet.
Abschließend kann man festhalten, dass CORBA-Komponenten den Weg zu komplexen und umfangreichen industriellen Softwarelösungen ebnen. Die Herstellerunabhängigkeit, Portabilität, die Vielzahl der angebotenen Produkte sowie die weitere Annäherung an die Enterprise Java Beans sind Gründe, die einen Erfolg der CORBA-Architektur auf Basis der Version 3.0 wahrscheinlich machen.

5. Quellen:

[OMG_1999] OMG: CORBA **Components – Joint Revised Submissio**n; OMG TC Document, URL: http://www.omg.org/ (Stand: 20.07.2000), (1999)

[OMG_ORB_1999] OMG: **Common Object Request Broker Architecture, Revision 2.**3; OMG TC Document, , URL: http://www.omg.org/ (Stand: 20.07.2000), (1999)

[Wagner_2000] Wagner, Michael: Architekturen stützen Software-Bausteine. In Information Week, Ausgabe 2/2000 (2000)

[Wang_2000] Wang, Nanbor, Schmidt, Douglas C., O´Ryan, Carlos: **An Overview of the CORBA Component Mode**l. Eingereicht als ein Kapitel in: **Component-Based Software Engineering: Putting the Pieces Togethe**r, George Heineman und Bill Councill, Addison-Wesley (2000)

[Corba_2002] CORBA® BASICS, OMG TC Document, URL: http://www.omg.org/ (Stand: 16.05.2002), (2002)