Zeus Framework

Cell Computing Model

Autor	Benjamin Hadorn
E-Mail	b_hadorn@bluewin.ch
Ablage/Website	http://www.xatlantis.ch
Datum	13.09.08
Version	0.6.3

Inhaltsverzeichnis

1 Einleitung

Was ist Zeus-Framework und weshalb braucht es schon wieder so ein Framework? Diese Fragen versuchen wir in diesem Dokument zu beantworten und zu zeigen, dass trotz massiver Softwareentwicklung dieses doch sehr junge Ingenieurwesen noch sehr in den Kinderschuhen steckt.

Haben Sie sich auch schon gefragt, warum bei jedem neuen Projekt man das Rad wieder von neuem erfinden soll?

Diese Abhandlung wird Ihnen wohl kaum eine Weltformel vermitteln, aber es soll zeigen wie mit guter Software Architektur viele Probleme bereits zum Beginn eines Projekts eliminiert werden können. Vieles werden Sie wohl an Hochschulen bereits gehört haben, wir versuchen dieses Wissen zusammen mit dem Paradigma von Zeus-Framework zu vertiefen und zeigen Ihnen einen möglichen (aber nicht den einzig wahren) Weg, Software in hoher Qualität, Wartbarkeit und vernünftigem Aufwand zu entwickeln.

2 Software

Betrachten wir eine objektorientierte Software, so sprechen wir von Objekten und Klassen. Objekte sind Instanzen einer Klasse.

Die Software besteht also aus vielen Objekten die sich gegenseitig kennen. Die Beziehungen zu anderen Objekten nennen wir **Beziehungsnetz**. Das Beziehungsnetz kann als mathematischer Graph angesehen werden. Dabei ist prinzipiell alles möglich, wie uni- und bidirektionale Beziehungen und zyklische Beziehungen über mehrere Objekte.

Die gesamte Software besitzt normalerweise einen oder mehrere Wurzel-Objekte (Basisobjekte). Von hier aus wird das Beziehungsnetz aufgebaut.

2.1 Probleme

Bereits eine kleine Software mit wenigen Objekten kann ein relativ kompliziertes Beziehungsnetz besitzen (Siehe Abbildung). Diese Vernetzung bringt folgende Probleme mit sich:

- Wer hat die Vernetzung während seiner Entwicklungsarbeit und Wartung der Software stets vollständig im Kopf?

- Bei starker Vernetzung können sich Fehler massiv auf andere Objekte auswirken
- Änderungen bewirken unverhoffte negative Auswirkungen bei anderen Objekten
- usw.

Die Probleme werden wir in folgende Kategorien unterteilen und danach separat behandeln. Die Kategorien heissen

- Qualität
- Wartbarkeit
- Erweiterbarkeit
- Verständlichkeit

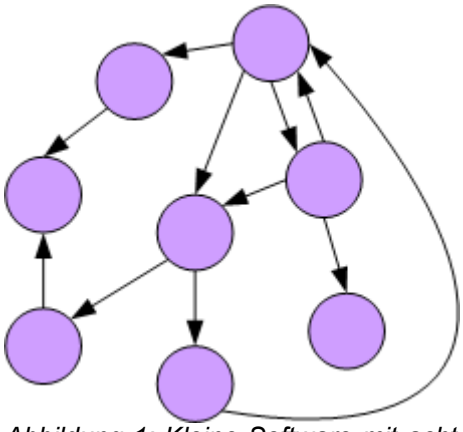


Abbildung 1: Kleine Software mit acht Objekten. Die Vernetzung der Objekte kann auch bei geringer Anzahl Objekten bereits sehr kompliziert werden.

2.1.1 Qualität

Unter Softwarequalität verstehen wir folgende Anforderungen:

- führt die Software die Aufgabe in gewünschter Zeit mit gewünschter Genauigkeit aus.
- wie geht die Software mit Ressourcen um.
    - Gibt sie alle Ressourcen wieder frei.

- wie ist das Laufzeitverhalten.

- Wie wird sichergestellt, dass die Objekte während der Verwendung gültig beiben (nicht plötzlich freigegeben werden)

Als Qualitätsmerkmal kann auch Erweiterbarkeit, Wartbarkeit und Verständlichkeit betrachtet werden. Diese Punkte betrachten wir aber einzeln.

2.2 Wartbarkeit

Bei der Wartbarkeit betrachten wir, wie einfach es ist Fehler zu lokalisieren und zu beheben. Dabei betrachten wir 2 Situationen, mit denen wir klar kommen sollten:

- Fehlerkorrektur in der Testphase (einfach)

- Fehlerkorrektur beim Produkt welches bereits Jahre im Einsatz ist (schwierig)

2.2.1 Erweiterbarkeit

Bei kleiner Software kann ein Beziehungsnetz durch Weitergabe von Objektreferenzen erzielt werden. Bei grosser Software mit tausenden von Objekten ist dies nicht einfach realisierbar. Betrachten wir folgende Abbildung. Durch eine Softwareerweiterung muss das orange Objekt eine neuen Beziehung (rot) zu einem bereits existierenden Objekt im Beziehungsnetz haben. Wie kriege ich denn die richtige Referenz?

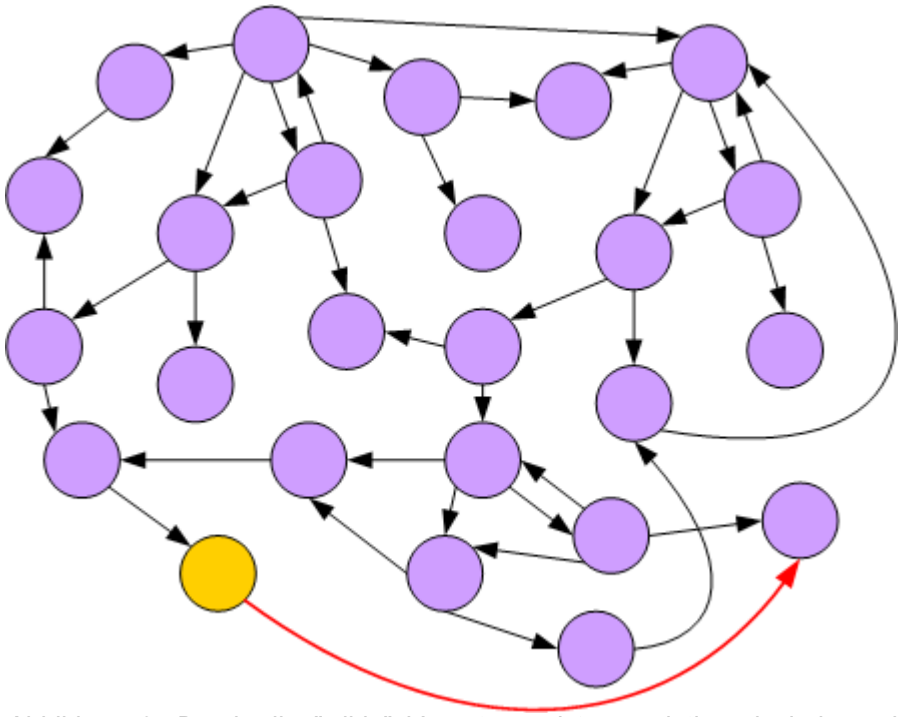


Abbildung 2: Durch die "wilde" Vernetzung ist es relativ schwierig und aufwändig, eine neue Beziehung einzufügen.

2.3 Die Lösung: Zeus-Framework

Um zu zeigen wie das Konzept des Zeus-Frameworks funktioniert, versuchen wir zunächst die oben genannten Probleme zu analysieren und zu lösen.

Scheinbar ist die Vernetzung ein zentrales Kriterium, ob Software flexibel und übersichtlich ist. Zu komplizierte Vernetzung gefährdet die Qualität, zu einfache Vernetzung ist nicht flexibel und endet meist in grosser Redundanz, welche ich dann bei der Wartbarkeit negativ auswirkt.

2.3.1 Beziehungsnetz als Baum Struktur

Als erste Massnahme schlagen wir vor, das Beziehungsnetz als allgemeinen Graphen zu einem Baum zu vereinfachen:

- Ein Objekt hat immer ein Vater (ausser das Wurzelobjekt)
- Ein Objekt kann mehrere Kinderobjekte haben
- Direkte Beziehungen zu Vorgängerobjekten (Vater, Grossvater etc.) sind nicht

erlaubt.

Diesen Baum nennen wir Objekthierarchie. Es ist offensichtlich, dass die Objekthierarchie nicht alle Beziehungen beinhaltet, welche für eine Software von Nöten ist. Wir werden später das Konzept der dynamischen Beziehungen vorstellen. Betrachten wir doch zuerst einmal die Objekthierarchie. Sie bringt in mancher Hinsicht eine klare Verbesserung:

- alle Kinderobjekte werden durch den Vater erzeugt und verwaltet.
    - dadurch ist die Zugehörigkeit klar definiert.
- durch das Navigieren im Objektbaum kann ein Objekt schnell gefunden werden.
    - Suchen im Graphen ist massiv aufwendiger
- Baumstrukturen repräsentieren in vieler Hinsicht die Bedürfnisse der Anwender und Softwareentwickler.
    - Klarer und verständlicher Aufbau von Software

Es gibt auch Einschränkungen; beliebige Beziehungen sind nicht möglich. Deshalb wird die Objekthierarchie vor allem zum Erstellen der Objekte verwendet.

2.3.2 Dynamische Beziehungen

Betrachten wir einen Objektbaum, so möchten wir zum Beispiel eine Objektbeziehung zwischen 2 Geschwister-Objekten haben, wenn Geschwister A die Dienstleistung von Geschwister B in Anspruch nehmen möchte.

Da beim Erzeugen des Objektbaums das Geschwisterpaar noch nicht vollständig vorhanden sein kann, muss die Beziehung zu einem späteren Zeitpunkt erfolgen. Alle Beziehungen, die nicht während der Erzeugung der Objekte hergestellt werden können, nennen wir **dynamische Beziehungen**.

Dynamische Beziehungen beinhalten alle Beziehungen, ausser Vater-Kind und Kind-Vater Beziehungen, also alle Beziehungen des Objektbaums.

Um dynamische Beziehungen herstellen zu können, muss eine Navigation im Objektbaum vorhanden sein, damit die benötigten Objekte gefunden werden können. Vergeben wir jedem Objekt einen Namen, so können wir, ähnlich wie in einem Dateisystem, durch Angabe eines Pfads die Objekte wieder finden.

2.3.3 Initialisieren der Software

Wie wir gesehen haben, brauchen wir eine strukturierte Initialisierung der Software. Dies geschieht in 2 Zyklen:

- Erster Zyklus dient zum Erstellen des Objektbaums
- Zweiter Zyklus dient zum Ermitteln der dynamischen Beziehungen.

Während der erste Zyklus durch den Konstruktor implementiert werden kann, verwenden wir für den zweiten Zyklus die Methoden `unfreeze()`.

2.3.4 Beenden der Software

In genau umgekehrter Reihenfolge zu der Initialisierung können wir die Software herunterfahren und beenden:

- Alle dynamischen Beziehungen freigeben
- Objektbaum freigeben

2.3.5 Laufzeit Dynamik

Die vorgestellten Konzepte zeigen zwar, wie sich die Software initialisiert und sich auch wieder beendet. Wie sieht es denn aus, wenn Softwareteile viel später erstellt werden oder einfach gelöscht werden, weil sie nicht mehr benötigt werden?

Dieses Verhalten nennen wir Laufzeitdynamik der Objekthierarchie. Folgende Aktionen sind darin enthalten:

- Zu einer beliebigen Zeit neue Objekte erzeugen und der Software zur Verfügung stellen
- Zu einer beliebigen Zeit bestehende Objekte freigeben.

Dieses Verhalten ist dann gewünscht, wenn dynamisch neue Aufgaben erledigt werden müssen oder nicht mehr benötigte Ressourcen freigegeben werden sollten.

Die Laufzeitdynamik bringt folgendes Problem mit sich; Wie können wir sicherstellen, dass der Zugriff auf ein Objekt noch gültig ist?

Das einfachste Konzept ist die Referenzzählung. Solange ein Objekt eine Referenz auf ein anderes Objekt besitzt, darf das Objekt nicht gelöscht werden.

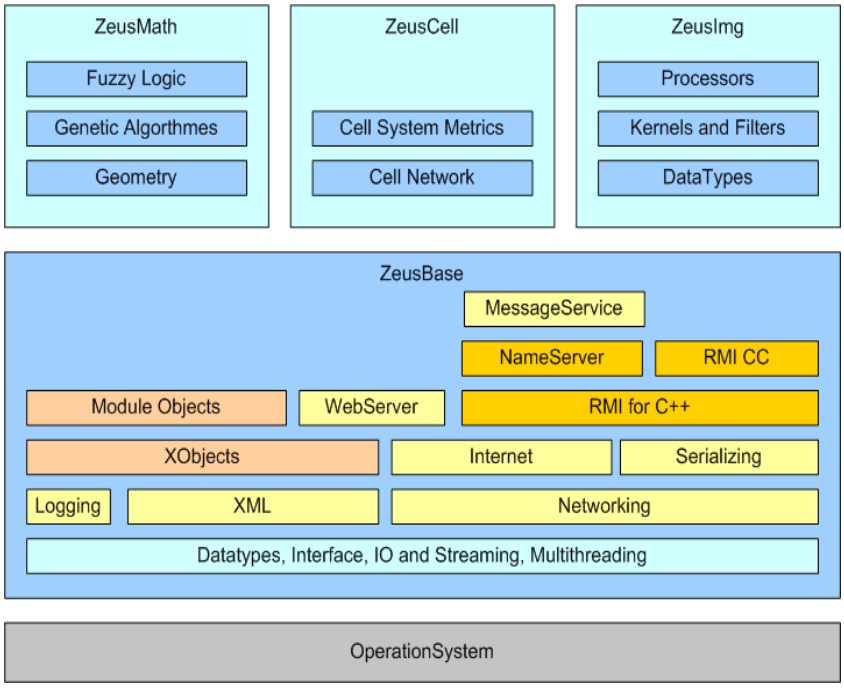
3 Zeus-Framework

Das Zeus-Framework ist eine Implementation des Cell Computing Model (CCM). Das CCM ist aber nur eines von vielen Konzepten, die in diesem Framework umgesetzt wurden. Dieses Dokument beschreibt, was mit dem Zeus-Framework Applikationen realisiert werden kann, und wie die einzelnen Konzepte umgesetzt wurden.

Das Zeus-Framework ist in mehrere Pakete unterteilt, welche in verschiedenen Dokumenten beschrieben werden.

- ZeusBase: Basis Paket des Zeus-Frameworks
- ZeusMath: Mathematisches Framework
- ZeusCell: Cell Computing Network (Grid- und Cluster-Computing)
- ZeusImg: Framework für Signal- und Bildverarbeitung

Abbildung 3: Struktur des Zeus-Framework



3.1 Das System

Das Zeus-Framework besitzt zwei Haupteigenschaften:

- modulare Struktur mit Softwarekomponenten in einem abgeschlossenen System (statische Eigenschaft)
- dynamische Kommunikation mit anderen gleichartigen und fremden Systemen. Zudem autonomes Anpassen an neue Situationen. (dynamische Eigenschaft)

Diese Eigenschaften ermöglichen ein breites und umfassendes Einsatzgebiet des Frameworks. Weitere Eigenschaften, wie

- portabel auf verschiedene Systeme (Linux und Windows bereits getestet)
- schnell und effizient (In C++ geschrieben)
- keine Lizenzgebühren (Open Source)

zeichnen das Framework ebenfalls aus.

Es setzt moderne Konzepte wie eXtensible Markup Language (XML) [XML01], Remote Method Invocation (RMI) und Objektserialisierung um und bietet eine einfache objektorientierte API (Application Programming Interface).

3.2 Einsatzgebiet

Das Zeus-Framework kann auf sehr verschiedene Arten eingesetzt werden:

- Durch seine modulare Struktur eignet es sich gut für grosse Projekte. Mehrere Entwickler können gleichzeitig an einem Produkt arbeiten. Integration von Softwarekomponenten ist sehr einfach möglich.
- Das Framework kann Module aus verschiedenen Entwicklungsumgebungen laden und verwalten, wie zum Beispiel .Net oder Delphi
- Einsatz für hochdynamische, verteilte Anwendungen, bei denen künstliche Intelligenz zum Einsatz kommt.
- Es eignet sich für parallele Verarbeitung von Daten (Clustering).
- Es kann zur Entwicklung von Anwendungen dienen, bei denen Java oder C# aus Grund der schlechteren Effizienz nicht eingesetzt werden kann (Bsp. bei

hardwarenaher Software)

- Bedien- und Kommunikationssoftware für Maschinensteuerungen

3.2.1 Clustering

Durch Clustering können mehrere PCs zu einem Grosssystem zusammengeschaltet werden. Diese PCs verarbeiten Daten parallel und lösen ein Problem gemeinsam. Die Arbeitslast ist je nach Problem verschieden und die PCs reagieren selbständig auf die Änderung der Auslastung. Dies ist ein typischer Einsatz des Cell Computing Model.

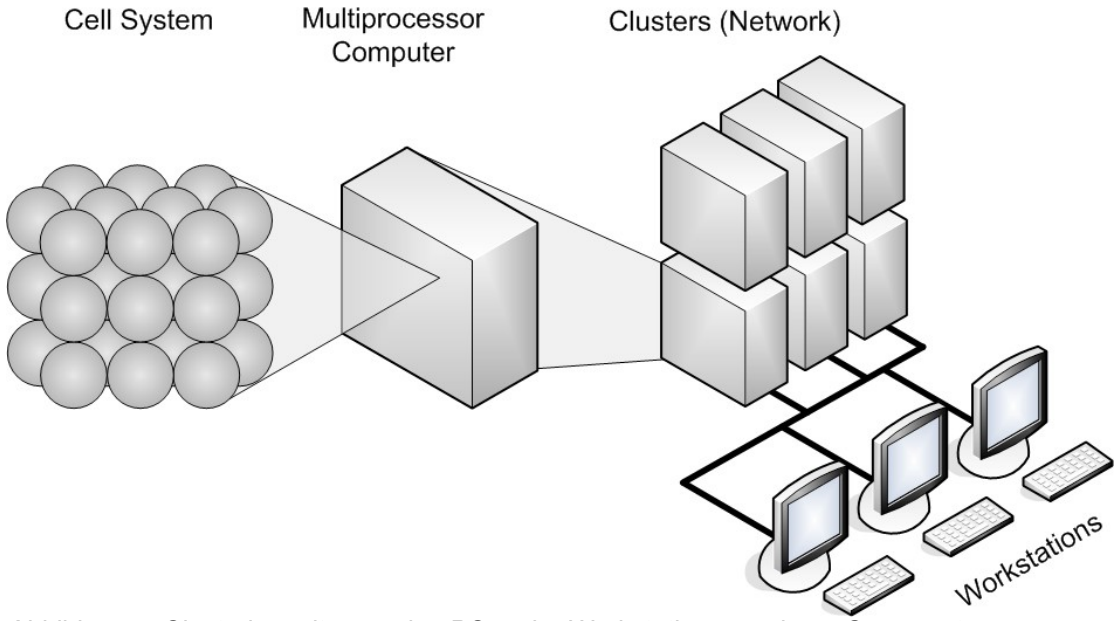


Abbildung 4: Clustering mit normalen PCs oder Workstations zu einem Grosssystem

Die Zellensysteme werden auf den PCs installiert, welche untereinander mit einem Netzwerk verbunden sind. Durch eine Registrierungsstelle (Namensdienst), welche auf einem der PCs läuft, können sich die Zellensysteme gegenseitig finden und somit miteinander kommunizieren. Die PCs, welche gemeinsam eine Registrierungsstelle benötigen, bilden einen so genannten Bearbeitungscluster.

Mehrere Bearbeitungscluster können nun über die Registrierungsstellen miteinander gekoppelt werden. Die Registrierungsstellen tauschen Informationen untereinander aus, welche die einzelnen Zellensysteme betreffen. Zellensysteme von verschiedenen Bearbeitungscluster können so miteinander kommunizieren.

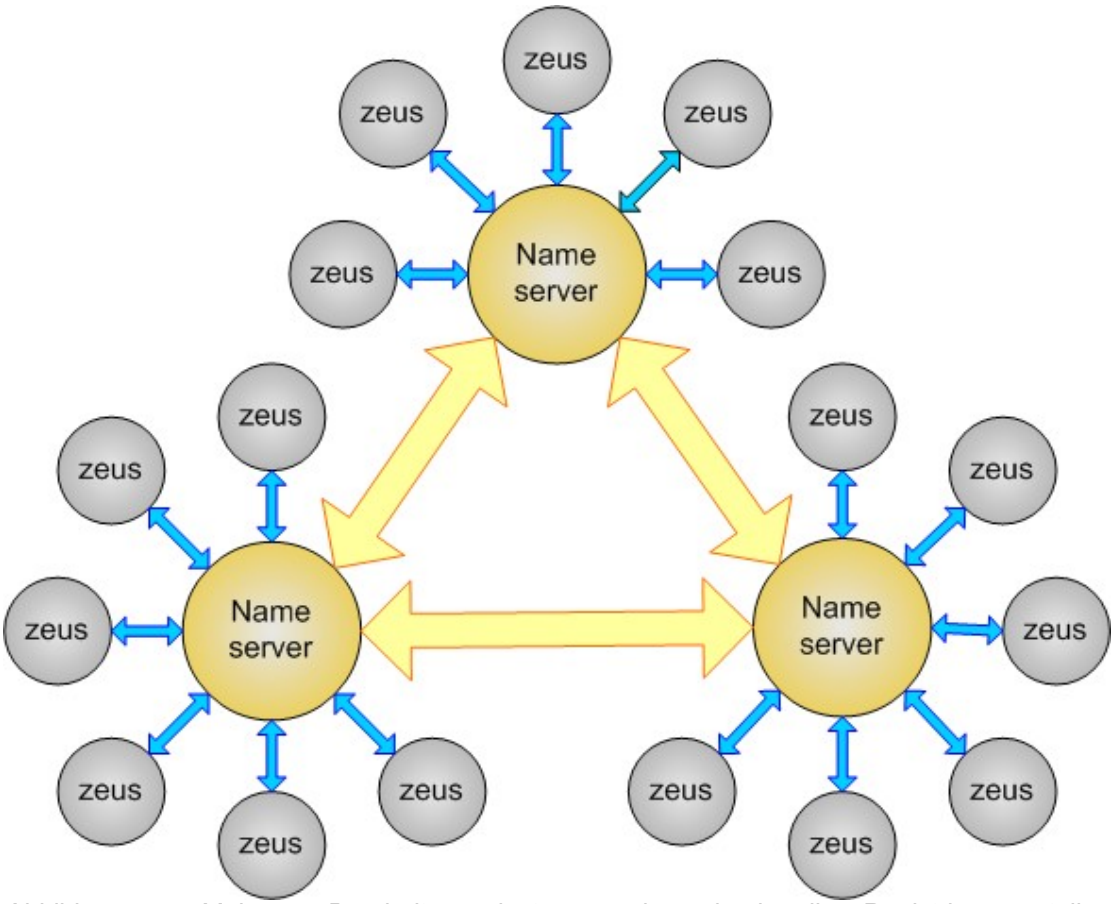


Abbildung 5: Mehrere Bearbeitungscluster werden durch die Registrierungsstellen miteinander gekoppelt. Dadurch können Systeme verschiedenster Grösse realisiert werden.

3.2.2 Bediensoftware für Maschinen

Bei der Bediensoftware für Maschinen spricht man etwa auch von Human Machine Interface (HMI). Das HMI hat die Aufgabe Maschinendaten möglichst schnell zu visualisieren, Benutzereingaben an die Maschine weiterzuleiten und Aktionen auf der Maschine auszuführen.

Das Hauptproblem ist fast immer die Zeit. Je nach Datenmenge und Art der Maschinensteuerung ist die Kommunikationsschnittstelle ein Engpass. Aber auch die Aufbereitung und das Rechnen der Daten kann sehr viel Zeit in Anspruch nehmen.

Deshalb muss die Übertragung der Daten von der Maschine zur Bedienoberfläche und zurück möglichst schnell und optimal laufen. Das Zeus-Framework wird hier einerseits bei der Darstellung und Verarbeitung der Daten und anderseits bei der Kommunikation verwendet.

Es ist vorstellbar, dass in Zukunft nicht mehr eine zentrale Maschinensteuerung mit PLC (Programmable Logic Controller) und CNC (Computer Numeric Control) existiert, sondern dass sich mehrere Prozessoren in einem Maschinenraum befinden und die Daten möglichst nahe bei der Messstelle auswerten. Hier könnte das Zeus-Framework ebenfalls zum Einsatz kommen. Durch sein dynamisches Verhalten kann sich die Software der Auslastung anpassen und auch in kritischen Momenten noch genügend reagieren. Diesbezüglich mussten aber noch allgemeine Konzepte erarbeitet werden.

4 Module Object Model Spezifikation 2.0

Die Module Object Model Spezifikation dient der allgemeinen Vereinbarung und der Definition von Objekten in der Framework-Umgebung. Das Zeus-Framework ist eine Referenzimplementation dieser Spezifikation.

Das Module Object Model (MOM)[1]definiert den Aufbau eines dynamisch und modularen Frameworks. Das MOM besteht primär aus einem Baum von Softwarekomponenten (Objekten), die dynamisch geladen werden können.

4.1 Eigenschaften des MOM

Das MOM hat durch seine einheitliche Entkopplung von Komponenten folgende Eigenschaften:

- Hohe Wiederverwendbarkeit der Komponenten. Das ermöglicht hervorragendes Software-Recycling.
- Dezentrale Entwicklung von Softwarekomponenten. Durch klare Schnittstellen können komplexe Systeme dezentral entwickelt werden.
- Stufenweises Entwickeln. Das System muss nicht schon beim Projektstart ins Detail geplant werden. Komponenten können auch später spezifiziert, entwickelt und integriert werden.
- Erweiterbarkeit der Software. Durch sich ändernde Kundenwünsche kann die Software auch später noch einfach erweitert werden.
- Durch die flexible Konfiguration kann die Software nach Kundenwunsch angepasst werden. Es können beliebige Komponenten in den Objektbaum eingehängt oder entfernt werden.
- Gute Wartbarkeit von komplexen Systemen. Einzelne Komponenten können ausgewechselt oder neu geschrieben werden, ohne dass das gesamte System neu bearbeitet werden muss.

Das Modell hat auch seine Schwachpunkte, die aber durch klare Richtlinien bei der Entwicklung wieder wett gemacht werden können:

- Die Vernetzung der Komponenten ist nicht begrenzt. Legt ein Projekt-Team die

1 MOM 1 wurde bei Studer.com definiert. [STWIN]

Vernetzung nicht fest, kann ein System entstehen, welches sehr viele und unüberblickbare Abhängigkeiten besitzt.

4.2 Aufbau und Definition

Die MOM Spezifikation wird in 2 Anforderungskategorien unterteilt:

1. Die Basis-Anforderungen definieren die MUSS-Kriterien

2. Die erweiterten Anforderungen definieren Zusätze, welche implementiert werden können.

Jedes Objekt ist durch eine einheitliche Schnittstelle von den anderen Objekten entkoppelt. Es gibt 2 Arten von Objekt-Typen:

- *X-Objekt*: Das X-Objekt ist ein einfaches Objekt, welches eine Softwarekomponente repräsentiert. Das X-Objekt implementiert die Basis-Anforderungen der MOM Spezifikation.

- *Modul*: Das Modul ist eine Komponente welche eine ganze Software-Bibliothek repräsentiert. Das Modul implementiert die Erweiterungen der MOM Spezifikation. Zudem kann das Modul eine Session von einer Bibliothek erzeugen (Kontext).

4.2.1 Basis Anforderungen

Das MOM beschreibt sich folgendermassen:

- X-Objekte können Endknoten sein, welche im System eine Dienstleistung oder eine bestimmte Aufgabe erfüllen.

- X-Objekte können Knoten sein, welche andere X-Objekte zu logischen Einheiten gruppieren.

- X-Objekte können eingefroren oder freigeschaltet werden.

    - Beim Einfrieren der X-Objekte werden alle Referenzen freigegeben, die von anderen X-Objekten stammen. Eingefrorene X-Objekte reagieren nicht mehr.

    - Beim Freischalten können die benötigten Referenzen von anderen X-Objekten angefordert und gebraucht werden.

- Das MOM definiert den Datenaustausch zwischen X-Objekten durch vordefinierte Schnittstellen.

- Ein X-Objekt kann seine Kinderobjekte verzögert erstellen. Die Kinderobjekte werden erst erstellt, wenn jemand aus dem System ein solches Kinderobjekt referenzieren möchte.

4.2.2 Erweiterungen

Die Erweiterungen definieren spezielle X-Objekte, die *Module*:

- Module können aktiviert oder deaktiviert werden. In einem System ist immer nur ein Modul aktiv (Siehe Abbildung 6). Die Aktivierung wird vor allem bei graphischen Anwendungen verwendet. Das aktive Modul erhält hier den Eingabefokus.

- Module laden ihre Ressourcen (Shared Libraries, Dokumente oder stellen Verbindungen zu anderen Systemen her). Hat ein Modul, welches aktiviert oder von dem Objekte angefordert wurde, seine Ressourcen noch nicht geladen, erfolgt nun der Ladeprozess. Beim Deaktivieren bleiben die geladenen Ressourcen meistens erhalten.

- Manager sind Module die andere Module verwalten und zu logischen Gruppen zusammenfassen.

- Werden in einem Objektbaum Module und Manager eingesetzt, ist es wichtig, dass jedes Modul und jeder Manager ein Kindobjekt von einem Manager ist. Der Wurzelknoten muss also ein Manager sein. Nur so ist der Aktivierungsprozess nach Definition gewährleistet.

Jeder Manager und jedes Modul kann beliebig viele X-Objekte als Kinderobjekte besitzen.

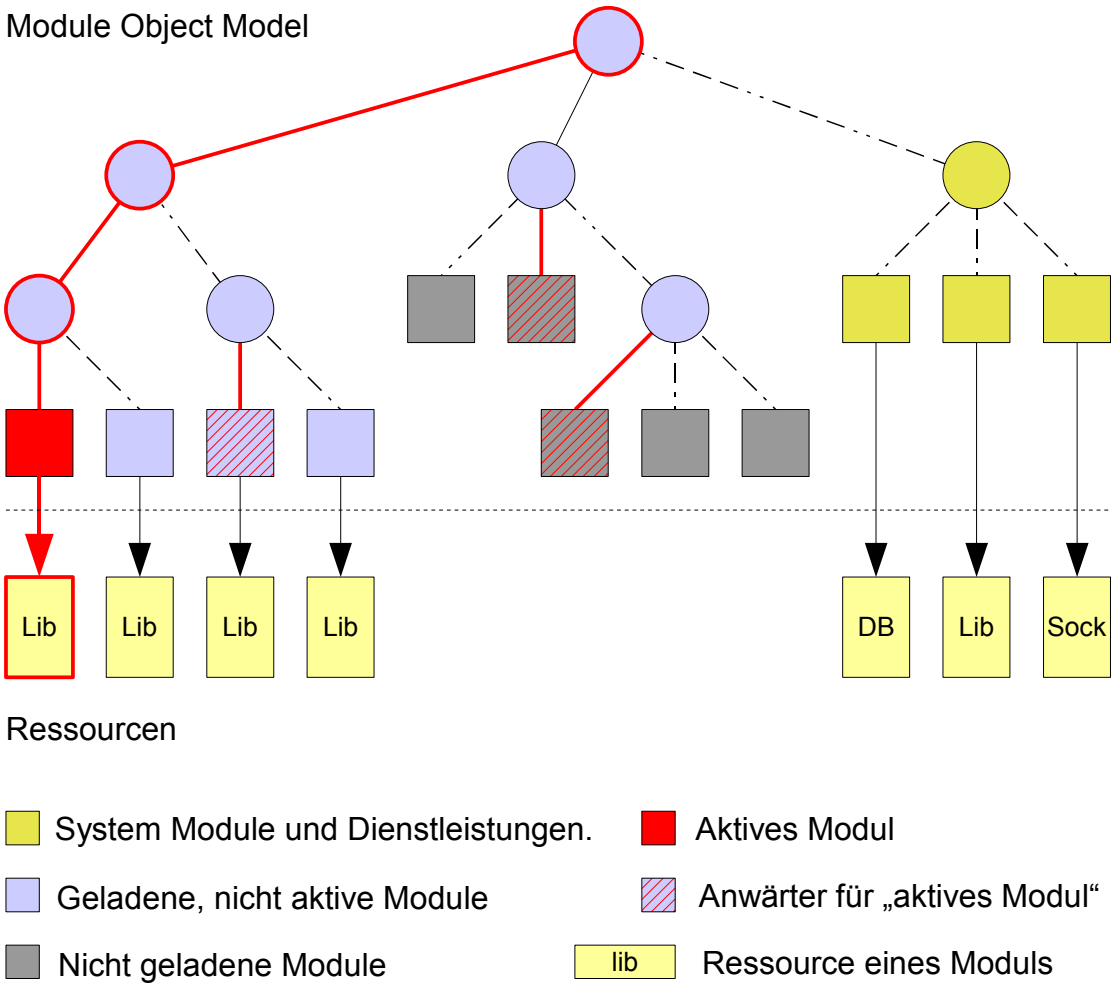


Abbildung 6: Diese Darstellung verdeutlicht den Aufbau und die Funktionsweise des MOM mit Modulen. Kreise symbolisieren die Manager, die Quadrate die Module. Das aktive Modul (rot) ist durch einen Aktivierungspfad gekennzeichnet. Die Systemmodule werden immer geladen. Sie stehen dem System generell zur Verfügung.

A Versionsgeschichte

Änderungen am Dokument sind hier festgehalten

Version	*Beschreibung*	*Datum*
1.0	Konzept von Zeus-Framework	
1.1	Anpassen an neue Zeus-Framework Struktur	13.09.2008

B Literaturverzeichnis

[XML01]: W3C, Extensible Markup Language (XML),

[STWIN]: Benjamin Hadorn, Programmieranleitung zum StuderWINFrame, 2003

C Index