

Wissensvermittlung in der Praxis: Modellierung des Apache HTTP Servers

Bernhard Gröne, Andreas Knöpfel, Rudolf Kugel

Zusammenfassung Analyse und Verständnis bereits vorhandener Software gehört zum Tagesgeschäft in der Software-Branche. Die Komplexität von Softwaresystemen und der daraus entstehende Bedarf für Arbeitsteilung lassen die effiziente Vermittlung dieses Wissens zu einem wichtigen Faktor werden. Eine gute Modellierung eines Softwareprodukts und deren didaktische Aufbereitung tragen erheblich zur Vermittlung des Wissens bei.

Als Beispiel wird ein Ausschnitt aus der Modellierung des Apache HTTP Servers vorgestellt, die im Rahmen eines Seminars am Hasso-Plattner-Institut (HPI) entstanden ist und bei Industrievorträgen bereits mehrfach erfolgreich eingesetzt wurde. Unter Verwendung der Modellierungstechnik SPIKES führen Pläne von der Übersicht bis hin zum Quelltext. Dabei wird besonderer Wert auf Aufbaustrukturen gelegt.

Die Erfahrungen zeigen, dass sich die Mühe der Modellierung und deren didaktischen Aufbereitung lohnt, wenn das Softwareprodukt eine gewisse Komplexität hat und das Wissen vom Modellierungsteam an viele andere weitergegeben werden soll.

1 Einleitung

Es gehört zum Tagesgeschäft eines Software-Entwicklers, sich mit bereits vorhandener Software auseinanderzusetzen. Um sie verstehen und sinnvoll einsetzen oder modifizieren zu können, muss er sich ein Modell des betrachteten Softwaresystems schaffen, also der ausgeführten Software in ihrer Einsatzumgebung. Arbeitet er in einem Team, ist es sinnvoll, dass alle Mitglieder das gleiche Modell des Softwaresystems im Kopf haben, um nicht aneinander vorbei zu reden. Aus Gründen der Arbeitsteilung ist es erwünscht, dass ein Mitglied ein Softwareprodukt oder Teile davon modelliert und sein Wissen an die anderen weitergibt.

Die Situation der arbeitsteiligen Modellierung wurde im Seminar “Anwendungsanalyse” am HPI erzeugt, die 60 Studenten aus dem 4. Fachsemester des Studiengangs Softwaresystemtechnik durchführten. Als Modellierungsobjekt wurde der Apache HTTP Server gewählt, weil es sich dabei um ein Softwareprodukt handelt, das sich weltweit im Einsatz befindet, eine gewisse Komplexität erreicht, dessen Quelltexte offen liegen und das ständig weiterentwickelt wird¹. Der Schwerpunkt des Seminars lag in der Vermittlung des

¹Zur Geschichte des Apache und der Organisation der Entwicklung siehe auch [En00, S.5–15].

erarbeiteten Wissens. Die Teilnehmer konnten am Ende die Bedeutung beliebiger Stücke des Sourcecodes für das Gesamtsystem erklären.

Dem Seminar ging eine Modellierung des Apache HTTP Servers durch die Autoren voraus. Das einzige Tool, das diese Arbeit wesentlich unterstützte, war ein Syntax-Highlighter und Hyperlinker, um die C-Quelltext-Dateien nach HTML zu konvertieren. Die Studenten erarbeiteten sich im ersten Teil des Seminars das Systemmodell, wie es in den Abschnitten 3.1 und 3.2 vorgestellt wird, inklusive der Konzepte, Protokolle und Datenstrukturen, die beim Einsatz eines HTTP Servers eine Rolle spielen. Die Modellierung der Autoren wurde den Studenten erst danach in einer Vorlesung vorgestellt und diente ihnen dann als Grundlage für die Analyse der Quelltexte im zweiten Teil des Seminars, bei der auch Petrinetze wie in Abbildung 4 entstanden sind.

Nach dem Seminar haben die Autoren die Modellierungsergebnisse zu einem einstündigen Vortrag verdichtet ([Gr01a]), der bereits mehrfach vor Entwicklern und Managern der Firmen SAP und SAP Portals gehalten und sehr positiv aufgenommen wurde.

Im folgenden Abschnitt wird die verwendete Modellierungstechnik und grafische Notation kurz vorgestellt. In Abschnitt 3 wird ein Ausschnitt aus der Modellierung des Apache HTTP Servers vorgestellt. Beginnend mit dem Systemmodell wird über die innere Struktur des Apache auf Prozessebene bis zum codenahen Ablaufplan eine Modellierung vorgestellt, die sowohl für den Überblick und Konzeptverständnis als auch für den Zugang zu den Quelltexten geeignet ist.

2 Die Modellierungstechnik

SPIKES² ist eine Modellierungstechnik, die von Professor Dr.-Ing. Siegfried Wendt und seinen Mitarbeitern entwickelt und in langjährigen Industrieprojekten mit Siemens und SAP verfeinert wurde. Ihre wesentlichen Merkmale sind:

- Eine für den Menschen gut erfassbare Darstellung, die nach didaktischen Aspekten aufgebaut ist [KA01].
- Eine klare Begriffswelt zur Beschreibung informationeller Systeme [We01].
- Die Trennung der Aspekte Ablauf, Aufbau (Systemstruktur) und Wertebereichsstruktur (Datenstrukturen) in der Darstellung
- Die Eignung der Beschreibungs- und Darstellungsmittel für verschiedene Abstraktionsstufen [Bu98], die damit auch die Modellierung software-intensiver Systeme wie beispielsweise die SAP R/3-Basis ermöglichte.

²Structure Plans for Improving Knowledge Transfer and Engineering of Systems

Aufbaupläne Abbildung 1 zeigt links einen Aufbauplan³ zum Systemmodell. Die Notation ist folgende: Rechtecke stellen Akteure dar, also aktive Systemkomponenten, die Verhalten zeigen. Kreise beziehungsweise Rechtecke mit stark abgerundeten Ecken stellen Speicher dar, also passive Systemkomponenten, in denen Werte und Wertänderungen zu beobachten sind. Kleine Kreise symbolisieren Kanäle, die Verbindungselemente zwischen Akteuren.

Ablaufpläne Die Darstellung des Verhaltens eines Akteurs oder des Zusammenspiels mehrerer Akteure (Protokoll) werden in SPIKES Petrinetze verwendet. Ein Beispiel dafür ist die Abbildungen 1 rechts. Dabei bezeichnet die Beschriftung einer Transition (dargestellt als Rechteck) eine Operation eines Akteurs. Führen mehrere Kanten von einer Stelle (dargestellt als Kreis) weg, werden die Kanten mit den Bedingungen für den Markenfluss beschriftet. Im Beispiel gibt es unterhalb der Transition “read request” eine Stelle, die mit “HTTP Method” beschriftet ist, sowie drei Kanten, die von ihr weg führen. Diese Notation bedeutet, dass im Falle, dass “HTTP Method = GET” wahr ist, die mittlere Transition schalten kann; falls die HTTP Methode nicht unterstützt wird, schaltet die rechte, ansonsten die linke, die hier nicht weiter dargestellt ist. Diese Unvollständigkeit dient der Vereinfachung.

3 Apache

3.1 Systemmodell zum einfachen HTTP Server

Apache ist ein HTTP Server, also eine Software, mit der Requests von Clients gemäß des **HyperText Transfer Protocol** beantworten werden können. Der Client, normalerweise ein Web-Browser, fordert in seinem HTTP Request eine Ressource an, beispielsweise eine HTML-Seite oder ein Bild. Der Server schickt mit seiner Response die passenden Daten zurück. Die Darstellung oder Verarbeitung der Daten ist Aufgabe des Clients. Mit jedem Request wird immer genau eine Ressource angefordert.

Abbildung 1 zeigt auf der linken Seite einen Aufbauplan zum allgemeinen Systemmodell. Es ist ein sehr einfaches Modell: Oben ist der Benutzer, der mit seinem Web-Browser über Bildschirm, Maus und Tastatur kommuniziert (vereinfacht als Kanal dargestellt). Der Browser stellt über Netzwerk eine Verbindung mit dem HTTP Server her und benutzt HTTP zur Kommunikation. Der HTTP Server greift rein lesend auf Dateien im Dateisystem zu. Nach einer Abbildung der angeforderten Ressource auf einen Dateinamen schickt er den Inhalt der Datei als Response dem Browser zurück. Es ist die Aufgabe des Browsers, die Darstellung der Daten vorzunehmen und gegebenenfalls weitere Ressourcen anzufordern, beispielsweise in eine HTML-Seite eingebettete Bilder.

Auf der rechten Seite von Abbildung 1 ist das Verhalten eines HTTP Servers dargestellt. Eine Implementierung eines simplen HTTP Servers, der zwar nur die GET-Methode be-

³Zu Aufbauplänen siehe auch [We91, S.217ff], Thema Instanzenetze

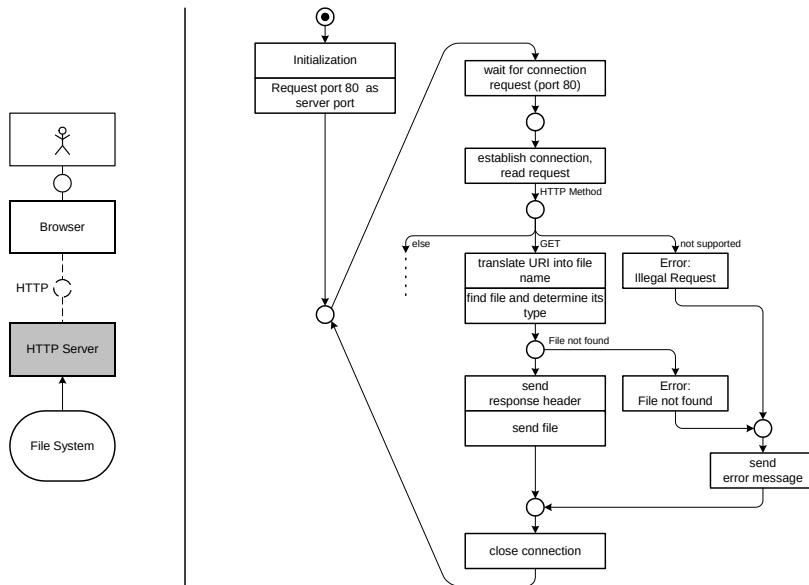


Abbildung 1: Aufbauplan zum Systemmodell von HTTP Server und Client und Ablaufplan zum Verhalten eines HTTP Servers

herrscht, mit dem aber der größte Teil der üblichen HTTP Requests bedient werden kann, ist mit 85 Zeilen Python-Code beziehungsweise 130 Zeilen C-Code möglich [Gr01b].

Übliche HTTP Server Produkte sind wesentlich komplexer. Die Source-Distribution der betrachteten Apache-Version 1.3.17 ist 780 Dateien groß, wovon 215 Dateien ca. 100 000 Zeilen C-Quelltext enthalten. Es muss also zuerst geklärt werden, was die Komplexität des Apache HTTP Servers gegenüber der simplen Implementierung ausmacht.

3.2 Systemmodell Apache

Abbildung 2 zeigt in einem Aufbauplan das Systemmodell des Apache HTTP Servers in seiner Umgebung. Gegenüber dem einfachen Systemmodell in Abbildung 1 links gibt es folgende Änderungen:

- (Oben) Der HTTP Server ist in der Lage, HTTP Requests von mehreren Clients nebenläufig abzuarbeiten.
- (Linke Seite Mitte) Ein Administrator kann während des Betriebs mit dem HTTP Server kommunizieren und ihn steuern. Die Konfiguration ist über Textdateien möglich (Speicher: Configuration data).
- (Rechte Seite Mitte) Apache kann als HTTP Client auftreten, um in der Proxy-Funktionalität HTTP Requests an andere HTTP Server weiterzuleiten und deren Response an den anfordernden Client zurückzuschicken.

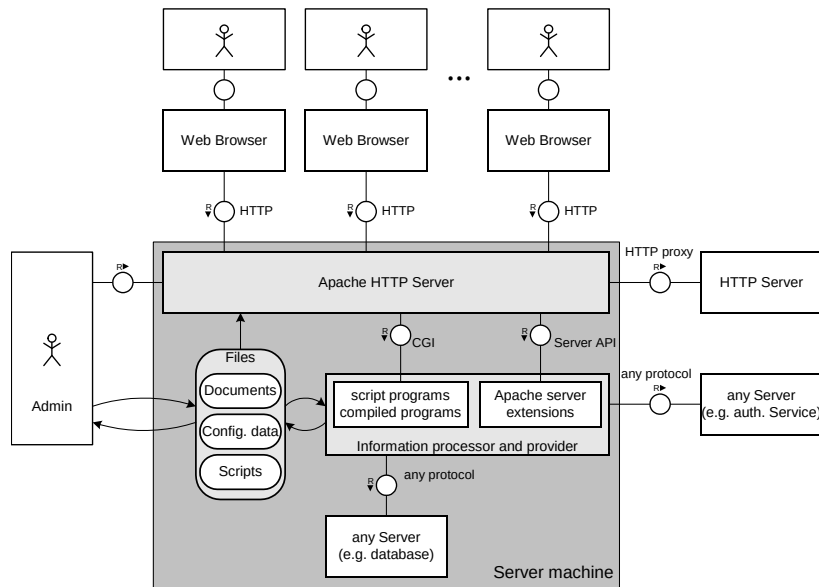


Abbildung 2: Systemmodell: Apache HTTP Server und seine Umgebung

- (Mitte) Über die CGI Schnittstelle kann Apache beliebige Programme ausführen, die auf Ressourcen im Dateisystem oder auf andere Server auf der gleichen oder einer über Netzwerk erreichbaren Maschine zugreifen und über Apache die Response an den Client zurückschicken können.
Über eine spezielle Server-API (Modul-API, gut beschrieben in [SM99]) kann Apache um neue Funktionen erweitert werden.

Weitere Eigenschaften, die die Komplexität erhöhen, sind die vollständige Implementierung aller HTTP-Versionen bis HTTP/1.1 (siehe RFC 2616 [Fi99]), die Übersetzbarkeit der gleichen Quelltext-Distribution auf vielen verschiedenen Plattformen sowie die Infrastruktur zur einfachen Anpassung an die jeweilige Systemkonfiguration, die Robustheit (24/7-Betrieb) und Skalierbarkeit des Servers sowie die Möglichkeit der Konfigurationsänderungen im laufenden Betrieb.

3.3 Verständnis der Quelltexte

Eine besondere Stärke des Apache HTTP Servers ist sein Modulkonzept. Module sind dynamisch ladbare Codestücke, die Informationen und Handler-Routinen zur Verfügung stellen, die zu bestimmten Zeitpunkten aufgerufen werden. Es handelt sich dabei um die Zeitpunkte Initialisierung, Einlesen von Konfigurationsdaten sowie die Schritte der Request-Bearbeitung. Eine ausführliche Beschreibung bieten [Gr01b] und [SM99].

Das Modul-Konzept hat für das Verständnis der Apache Quelltexte eine große Bedeutung. Da nämlich die Hauptfunktionalität eines HTTP-Servers, die Beantwortung von Requests

gemäß HTTP, in Apache-Module “ausgelagert” ist, genügt es, die Modul-API zu betrachten, um die Mechanismen zu verstehen. Damit reduziert sich die Zahl der Codestücke, die zum Verständnis des Apache HTTP Servers erforderlich ist, auf folgende:

- Die Aktivierung und Deaktivierung⁴ des Servers sowie der Restart-Loop (siehe dazu Abbildung 4).
- Der Request-Response-Loop eines Child Servers. Hier kann nachvollzogen werden, zu welchen Zeitpunkten Handler-Routinen der Module aufgerufen werden.
- Der Master-Server-Loop, der die Child Server Prozesse überwacht und deren Anzahl regelt.
- Kleinere Codestücke, wie etwa die Speicherverwaltung (Pools), die Realisierung eines Mutex auf der jeweiligen Plattform oder das Einlesen der Konfigurationsdaten.

3.4 Modell des Apache auf Prozess-Ebene

Zur Realisierung der nebenläufigen Bearbeitung eintreffender Requests werden bei Apache Prozesse bzw. Threads (Windows-Version) verwendet. Im folgenden wird ausschließlich die Implementierung mit (Unix-) Prozessen betrachtet.

Abbildung 3 zeigt die Struktur eines Apache HTTP Servers zur Laufzeit. Dessen Umgebung ist nur noch skizziert mit den Clients oben, dem Dateisystem auf der rechten Seite sowie Administrator und den Konfigurationsdaten auf der linken Seite. Der Apache HTTP Server ist hellgrau unterlegt und zeigt eine innere Struktur:

- Auf der linken Seite ist der sogenannte Master Server. Er nimmt als Stellvertreter des gesamten Apache HTTP Servers die Steuerbefehle entgegen, die ihm der Administrator in Form von Signalen mit dem Unix-Befehl `kill` schickt. Er ist auch derjenige, der die globalen Konfigurationsdaten einliest. Der Master Server beschränkt sich darauf, die child Server zu erzeugen oder zu beenden⁵. Für die Erzeugung benutzt er den `fork()`-Aufruf, der bewirkt, dass ein Child Server eine exakte Kopie der Speicherbelegung und der reservierten Systemressourcen des Master Servers bekommt. Auf diese Weise kennt ein Child Server die eingelesenen Konfigurationsdaten.
- In der Mitte sind die Child Server dargestellt. Sie übernehmen die Bearbeitung der HTTP Requests. Jeder einzelne Child Server kann genau einen Request zur gleichen Zeit bearbeiten. Um also 25 Requests nebenläufig bearbeiten zu können, sind 25 Child Server Prozesse nötig.

⁴Die Begriffe Aktivierung und Deaktivierung sind Teil des Schaffungsprozessmodells, das in [Ga98] beschrieben wird.

⁵In Abbildung 3 gibt es einen gestrichelt umrandeten Bereich, der die darin vorkommende Strukturvarianz darstellt. Der Master Server hat schreibenden Zugriff auf diesen Bereich, ist also für die Strukturvarianz in diesem Bereich verantwortlich.

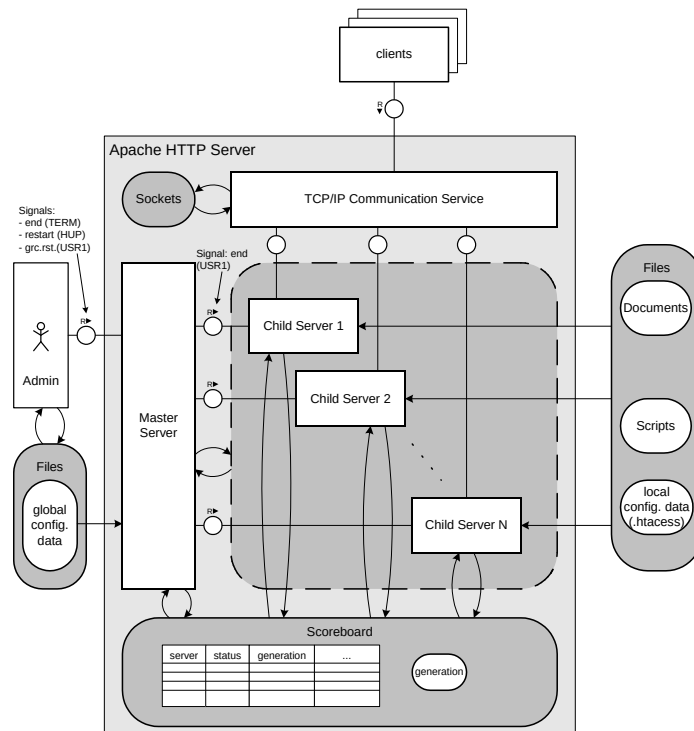


Abbildung 3: Aufbauplan zum Preforking–Prozessmodell

- Oben ist der TCP/IP Communication Service dargestellt, der eigentlich Teil des Betriebssystems ist. Die Funktion des TCP/IP Communication Service ist vergleichbar mit einer Telefonzentrale. Er nimmt die Anfragen der Clients zur Herstellung einer TCP–Verbindung entgegen und stellt die Verbindung bei Annahme durch den nächsten wartenden Child Server zu diesem durch. Die Verbindungen verwaltet er mit Hilfe der Socket–Datenstrukturen.
- Der Master Server sowie alle Child Server haben Zugriff auf einen gemeinsamen Speicherbereich, der unten dargestellt ist. Darin befindet sich das Scoreboard, in dem der Master Server einen Eintrag für jeden Child Server anlegt, in dem die Child Server ihren aktuellen Status eintragen. Weitere Informationen sind unter anderem die Prozessnummer eines Child Servers sowie dessen Generation. Zu den Generationen siehe weiter unten unter “graceful restart”.

Preforking Bei Apache wird das sogenannte Preforking–Server–Modell eingesetzt, das heißt, dass der Master Server zu Anfang eine bestimmte Anzahl an Child Server Prozessen auf Vorrat erzeugt, so dass bei einer einkommenden Verbindungsanfrage auch garantiert ein Child Server zur Annahme bereit ist. Unterschreitet im Betrieb die Anzahl der freien

Child Server eine in der Konfiguration eingestellte Grenze, erzeugt der Master Server weitere Prozesse. Beim Überschreiten einer Grenze beendet er Child Server Prozesse. Beendet sich ein Child Server Prozess wegen eines Programmfehlers oder durch andere Einwirkungen, ersetzt der Master Server diesen durch einen neuen Prozess.

Graceful Restart Um Änderungen an den Konfigurationsdateien beim Apache HTTP Server wirksam werden zu lassen, hat der Administrator zwei Möglichkeiten. Bei der einen weist er den Master Server an, einen Neustart (restart) durchzuführen. Dabei beendet der Master Server alle Child Server Prozesse, liest die Konfiguration neu ein und startet einen neuen Satz von Child Servern. Der Nachteil an dieser Prozedur besteht darin, dass bestehende TCP-Verbindungen abgebrochen werden, bevor die dazugehörigen Requests beantwortet werden konnten.

Die andere Möglichkeit erlaubt es, im laufenden Betrieb die Konfiguration zu ändern, ohne laufende Request-Bearbeitungen zu unterbrechen. Sie heißt “graceful restart” und macht Gebrauch von den Generationsinformationen im Scoreboard (siehe Abbildung 3 unten). Der Master Server zählt bei jedem restart eine globale Generationsvariable im Scoreboard hoch und trägt diesen Wert bei jedem neuen Child Server Prozess ein. Bei einem graceful restart beendet er nur die Child Server, die untätig (Status *ready*) sind und ersetzt diese durch Child Server der neuen Generation. Schließt ein Child Server Prozess die Beantwortung eines Requests ab, prüft er, ob seine Generation im Scoreboard noch mit der globalen Generation übereinstimmt. Ist das nicht der Fall, beendet sich der Prozess selbst. Der Master Server bekommt die Mitteilung über das Ende dieses Prozesses und ersetzt ihn durch einen Child Server der neuen Generation.

Die beschriebenen Abläufe sind im Ablaufplan in Abbildung 4 dargestellt. Es handelt sich dabei bereits um eine sehr codenahe Darstellung. Konkret betrifft das die C-Funktionen `REALMAIN()`, `standalone_main()` und `child_main()` in der Quelltextdatei `http_main.c`. Zu den grau unterlegten Unternetzen “child_main” und “idle_server_maintenance” existieren Ablaufpläne, die weiter ins Detail gehen. Das betrifft insbesondere die Request-Bearbeitung in `child_main`.

4 Diskussion der Modellierung

Aufbaupläne wie in den Abbildungen 2 und 3 dienen den Studenten als Grundlage, um die Quelltexte verstehen und Ablaufpläne wie in Abbildung 4 erstellen zu können. Gerade nebenläufige Systeme weisen dynamische Strukturen auf, die sich im Quelltext schwer nachvollziehen lassen. In den Aufbauplänen kann man typische Probleme nebenläufiger Systeme⁶ besser erkennen, weil nebenläufig agierende Systemkomponenten als verschiedene Akteure dargestellt und benannt werden können. Beispielsweise existiert in Abbildung 3 ein Zugriffskonflikt beim Scoreboard, das als rein passive Systemkomponente modelliert ist.

⁶Eine detaillierte Betrachtung bietet [Ta01].

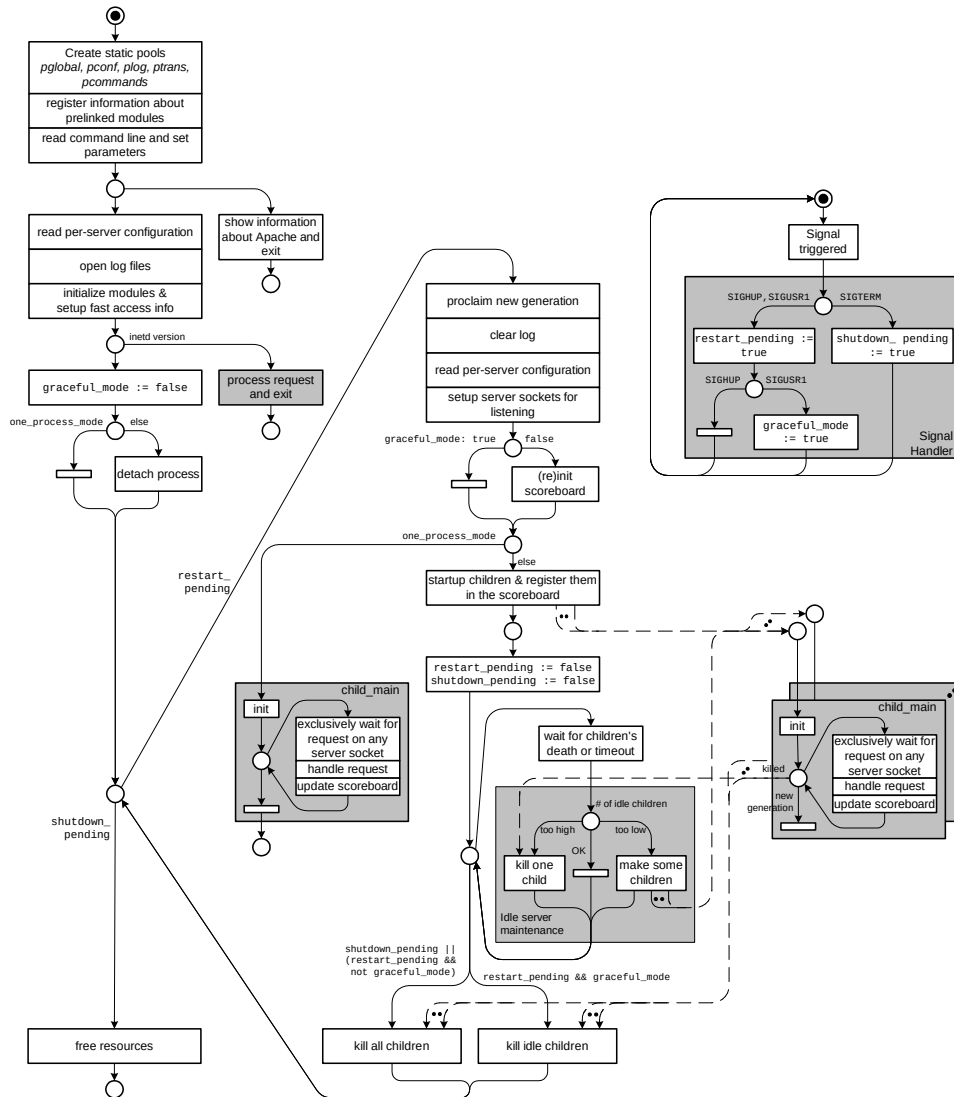


Abbildung 4: Hauptablauf des Apache HTTP Servers

Die Bedeutung mancher Quelltext-Passagen lässt sich erst dann zeigen, wenn die Einsatzumgebung in die Modellierung einbezogen wird, also das Gesamtsystem unter diesem Aspekt betrachtet wird. Um den Leser bei der Darstellung komplexer Sachverhalte nicht in Details ertrinken zu lassen, ist es erforderlich, für verschiedene Aspekte auch verschiedene Pläne zu erstellen, in denen eine Beschränkung der Darstellung auf die für den Aspekt wesentlichen Sachverhalte erfolgt.

Um die gute grafische Auffassungsgabe des Menschen zu unterstützen, spielt auch das

Layout eine wichtige Rolle. Dazu gehört neben der Anordnung der Planelemente (Wiedererkennung von Mustern) auch deren Höhe und Breite, Strichstärken von Kanten und Umrandungen, Schattierungen und abgerundete Ecken bei Kanten (damit sie besser mit dem Auge verfolgt werden können).

5 Resümee

Der Apache HTTP Server bietet eine Vielzahl von Details, die eine Betrachtung wert sind. Die oben angedeutete Modellierung bietet eine Grundlage, die den Zugang und die Einordnung von Konzepten und Quelltext-Passagen des Apache HTTP Servers wesentlich erleichtert.

Der hohe Aufwand einer guten Modellierung und vor allem auch ihrer didaktischen Aufbereitung macht sich bezahlt, wenn Wissen über Softwaresysteme effizient vermittelt werden soll.

Literatur

- [Bu98] Bungert, Andreas: Beschreibung programmierter Systeme mittels Hierarchien intuitiv verständlicher Modelle, Shaker Verlag, Aachen, 1998.
- [En00] Engelschall, Ralf S.: Apache Desktop Reference. Addison–Wesley, 2000.
- [Fi99] Fielding, et al: Hypertext Transfer Protocol — HTTP/1.1, RFC 2616, 1999.
- [Ga98] Gales, Frank: Ein Beitrag zur Begriffswelt der Schaffung programmierter Systeme. Shaker Verlag, Aachen, 1998.
- [Gr01a] Gröne, Bernhard: Anwendungsanalyse am Beispiel des Apache HTTP Servers. Vortragsunterlagen, HPI, 2001
- [Gr01b] Gröne, B.; Knöpfel, A.; Kugel, R.; Schmidt, Oliver: Modellierung des Apache HTTP Servers, <http://www.hpi.uni-potsdam.de/apache>
- [KA01] Keller, Frank; Apfelbacher, Remy: SPIKES Handbook, HPI, 2001
- [SM99] Stein, Lincoln; MacEachern, Doug: Writing Apache Modules with Perl and C. O'Reilly, 1999.
- [Ta01] Tabeling, Peter: Ein Metamodell zur architekturorientierten Beschreibung komplexer Systeme, eingereichter Beitrag zur GI-Konferenz "Modellierung 2002", Hasso-Plattner-Institut 2001.
- [We91] Wendt, Siegfried: Nichtphysikalische Grundlagen der Informationstechnik, Springer Verlag Heidelberg, 1991.
- [We01] Wendt, Siegfried: Ein grundlegender Begriffsrahmen für das Wissensmanagement im Software-Engineering, Tagungsband KnowTech 2001 Dresden.