FACHBEREICH INFORMATIK
Abteilung Software Engineering

Proposal einer Diplomarbeit

Empirische Bewertung von Performance-Analyseverfahren für Software-Architekturen

29. April 2004

Bearbeitet von:
Heiko Koziolek
Schützenweg 42
26129 Oldenburg
heiko.koziolek@informatik.uni-oldenburg.de

Betreut von:
Erstgutachter: Jun.-Prof. Dr. Ralf Reussner
Zweitgutachter: Prof. Dr. Wilhelm Hasselbring
Betreuer: Dipl. Wirtsch.-Inform. Steffen Becker

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Performance in modernen Softwaresystemen

Obwohl immer schnellere Rechnersysteme konstruiert werden, ist die Performance von Softwaresystemen noch immer ein schwerwiegendes Problem in der Softwaretechnik. Unter Performance versteht man dabei den Grad, zu dem eine Software Anforderungen an Antwortzeiten und Durchsatz einhalten kann [SW02].

Die Gründe für die anhaltenden Performance-Probleme trotz schneller Hardware sind vielfältig. Anwendungssysteme werden ständig komplexer und bestehen nun häufig aus über das Internet verteilten, zusammenarbeitenden Komponenten. Zudem ergeben sich ganz neue Anforderungen, die selbst moderne Systeme an ihre Leistungsgrenze führen. Eine Web-Applikation wird beispielsweise zu Stoßzeiten von Hunderttausenden von Benutzern gleichzeitig verwendet. Liefert der Anbieter nicht die vom Kunden gewünschte Performance, sind Konkurrenzangebote im Internet nur wenige Mausklicks entfernt. Ein weiteres Beispiel ist eine Software zur Steuerung von Airbags, die innerhalb von Millisekunden komplexe Sensordaten auswerten muss. Können die äußerst geringen Antwortzeiten nicht eingehalten werden, sind Menschenleben gefährdet.

1.2 Software-Entwicklung und Performance-Analysen

In der Praxis werden Performance-Aspekte während der Entwicklung von Softwaresystemen in den frühen Entwurfsphasen oftmals vernachlässigt oder ganz ausgeklammert. Viele Entwickler sind der Ansicht, dass Performance-Probleme erst dann behoben werden müssen, wenn sie wirklich auftreten, also wenn das System bereits annähernd fertig implementiert ist und unter realen Bedingungen getestet werden kann. Diese so genannte 'fix-it-later'-Einstellung [SW02] lässt außer acht, dass die meisten Performance-Probleme in der Architektur eines Softwaresystems begründet liegen und die zugehörigen Fehler bereits im Entwurf gemacht wurden. Architekturmängel können aber nach erfolgter Implementierung meist gar nicht oder nur mit großem Aufwand behoben werden, da dann große Teile des Programmcodes neu geschrieben werden müssen, was hohe Kosten verursacht.

Aus diesem Grund wird versucht, Performance-Analysen bereits in den Entwurfsprozess der Software einzubeziehen. Schon während der Modellierung einer Applikation sollen Schätzungen über die Ausführungszeiten von Programmsegmenten und die Belastung von Hardware-Elementen in die Planungen einbezogen werden. So können im Idealfall solche Performance-Probleme, die aus der Software-Architektur resultieren, von vornherein vermieden werden. Die Mehrkosten für frühe Performance-Analysen liegen für typische große Softwaresysteme in der Größenordnung von 1-3 Prozent der gesamten Projektkosten [SW02] und sind damit weit unterhalb der Kosten für ein nachträgliches Redesign der fertigen Implementierung.

1.3 Software Performance Engineering

Bereits 1981 prägte Connie U. Smith den Begriff *'Software Performance Engineering'* [Smi81]. Darunter versteht man einen 'systematischen, quantitativen Ansatz zur Konstruktion von Software-Systemen, die Performance-Vorgaben einhalten'.

In der Folgezeit entstanden eine Reihe von unterschiedlichen Performance-Analyseverfahren für Software-Architekturen, über die in Kapitel 3 ein erster Überblick gegeben wird. Ein Standardverfahren gibt es bis heute jedoch nicht, auch ist die Integration von Performance-Analysen in führende CASE-Werkzeuge noch nicht erfolgt, so dass man nur von einer bedingten Praxistauglichkeit der meisten Verfahren sprechen kann [DS00].

Fallstudien zur Validation der unterschiedlicher Verfahren liegen bisher kaum vor. In [BMDI04] wird die Software-Architektur eines Marine-Kommunikationssystems mit einem analytischen und einem simulationsbasierten Performance-Analyseverfahren in einer Feldstudie untersucht. Die Autoren vergleichen beide Methoden anhand verschiedener Kriterien, wie z.B. Schwierigkeit der Ableitung von Performance-Modellen, Allgemeingültigkeit, Skalierbarkeit und Möglichkeit zur Messung anderer nicht-funktionaler Eigenschaften.

2 Konzeption

2.1 Diskussion der Methodik

In der anzufertigen Diplomarbeit sollen verschiedene Verfahren zur Performance-Analyse von Software-Architekturen validiert werden. Im Idealfall sollte dazu ein kontrolliertes Experiment durchgeführt werden, das unter den empirischen Methoden den höchsten Grad an Vertrauen in die Beobachtungen genießt [Pre01]. Ein solches Experiment vergleicht mehrere Fälle, die sich nur in einem Merkmal (in diesem Fall wäre das das Verfahren) unterscheiden. Dabei können die beobachteten Effekte direkt auf die Änderungen dieses Merkmals zurückgeführt werden, da alle anderen Faktoren konstant gehalten werden. Allerdings wird für die Durchführung eines kontrollierten Experiments eine größere Anzahl von Versuchspersonen benötigt, um deren fachliche Qualifikation und Motivation als einen das Ergebnis verfälschenden Faktor durch statistische Mittelung auszuschließen. Auf eine ausreichende Anzahl von Testpersonen kann jedoch im studentischen Umfeld nicht zurückgegriffen werden. Auch eignen sich die verschiedenen Performance-Analyseverfahren nur bedingt für ein solches Experiment, da alle Verfahren unterschiedliche Eingaben für ihre Analysen erwarten.

Aus diesem Grunde soll die Untersuchung in Form einer Fallstudie durchgeführt werden, bei der eine kleine Zahl konkreter Einzelfälle mit den zur Verfügung stehenden Testpersonen untersucht wird. Bei der Analyse der Ergebnisse muss daher genau untersucht werden, ob die beobachteten Effekte tatsächlich auf die Eigenheiten der angewandten Verfahren zurückzuführen sind und nicht in den subjektiven Unterschieden der Teilnehmer begründet liegen.

2.2 Experimententwurf

Als konkrete Aufgabenstellung für die Testpersonen sollen Entwurfsentscheidungen bei der Entwicklung einer Web-Applikation bewertet werden. Dazu kann ein in der Palladio-Gruppe entwickelter, prototypischer Web-Server verwendet werden. Die verschiedenen Entwurfsalternativen werden von Yvette Teiken, die in ihrem individuellen Projekt den Web-Server um dynamische Seitengenerierung erweitert, in Form von UML-Diagrammen geliefert. Nach evtl. Überarbeitung der Diagramme sollen diese den Testpersonen vorgelegt werden, die zuvor über verschiedene Performance-Analyseverfahren instruiert wurden. Deren Aufgabe besteht dann darin, die in den Verfahren vorgesehenen Performance-Modelle zu erstellen und Performance-Attribute wie Antwortzeiten und Ressourcenauslastung zu berechnen. Es werden mehrere Architekturalternativen vorgelegt, von denen dann die Testpersonen nach Möglichkeit jeweils eine favorisieren sollen.

In der Auswertung der Ergebnisse kann dann untersucht werden, welche Methode die realistischsten Performance-Werte liefert oder ob ein Zusammenhang zwischen Methodik und Ergebnis gar nicht besteht, also die Performance-Werte zu sehr von den darin enthaltenen Schätzungen abhängen. Darüberhinaus sollen eine Reihe von Eigenschaften der Verfahren wie z.B. Schwierigkeit der Ableitung des Performance-Modells, Übertragbarkeit auf verschiedene Architekturtypen, Grenzen bzgl. der Komplexität der zu untersuchenden Architektur usw. untersucht werden. Konkrete Untersuchungsaspekte für die vorläufig ausgewählten Verfahren finden sich in Kapitel 3 jeweils am Ende des Abschnitts über das Verfahren. Diese Aspekte sollen bis zum Beginn des Experiments vervollständigt und verfeinert werden. Darüberhinaus sollen die Testpersonen über ihre subjektiven Eindrücke über die Verfahren befragt und um Verbesserungsvorschläge gebeten werden.

2.3 Artefakte

Konkret sind im Rahmen der Diplomarbeit zu erstellen:

- Literaturüberblick über die existierenden Verfahren
- Einleitender Vortrag für die Testpersonen (über Performance Engineering im Allgemeinen)
- Einführung in die für die Fallstudie ausgewählten Verfahren jeweils mit Anwendungsbeispiel
- Überblick über die einzelnen Verfahren als Arbeitsmaterial für die Testpersonen
- Aufgabenstellung für einen Vortest
- Aufgabenstellung für den Haupttest auf Basis einer in UML vorgegebenen Architektur
- evtl. Implementierungsarbeit im Zuge der Aufgabenerstellung

- Auswertung der Ergebnisse mit Herausarbeitung der Fehler bzw. Schwachstellen einzelner Verfahren

2.4 Risikomanagement

Eine Reihe von Risiken können den Erfolg der Studie gefährden:

- **Geringe Teilnehmerzahl:** Da die Versuchspersonen im Rahmen einer Lehrveranstaltung angeworben werden, ist die Studie auf eine Reihe von freiwilligen Teilnehmern angewiesen. Ein zu geringe Teilnehmerzahl (unter 10 Personen) würde eine Auswertung der Ergebnisse erschweren, da so kaum noch individuelle Unterschiede zwischen den Personen ausgeglichen werden können.

- **Fristen:** Es besteht die Gefahr, dass die anzufertigen Materialen für Einführungen und Aufgabenstellung nicht rechtzeitig fertig werden, und die Versuchspersonen nicht mehr aus der dann bereits abgelaufenen Lehrveranstaltung genommen werden können. In diesem Fall könnten notfalls Mitarbeiter der Abteilung Software Engineering als Testpersonen angeworben werden.

- **Ergebnisverfälschende Faktoren:** Möglicherweise beeinflusst bereits die Vorstellung der einzelnen Methoden selbst die Effekte der Studie. Falls bei der Einführung bestimmte Punkte nicht richtig oder unklar dargestellt werden, könnten die Testpersonen Fehler bei der Anwendung machen. In diesem Fall wäre es schwierig konkrete Ergebnisse aus der Studie zu ziehen. Auch die Komplexität der Aufgabenstellung könnte die Testpersonen möglicherweise über- oder unterfordern, so dass hier ein angemessener Detaillierungsgrad gefunden werden muss. Diese Risiken sollen durch die Durchführung eines internen Vortests der Aufgabenstellung in der Abteilung Software Engineering vermindert werden.

3 Grundlagen

Ein kurzer Überblick soll nun darstellen, wie sich Performance-Analyseverfahren in den Softwareentwicklungsprozess integrieren lassen und welche Notationen dabei in der Vergangenheit verwendet wurden. Auf die verläufig als relevant für die Studie eingestuften Verfahren wird danach näher eingegangen.

3.1 Performance-Analyse von Software-Architekturen

Ausgangspunkt der Performance-Analyse ist ein Modell der Software, welches während der frühen Entwicklungsphasen entsteht. Seit ihrer Standardisierung 1997 durch die Object Management Group (OMG) wird dazu häufig die Unified Modelling Language (UML)

[OMG03b] verwendet. Die UML-Notation ist aufgrund ihrer Flexibilität bei der Spezifikation von statischen und dynamischen Elementen von Software-Systemen und ihrer einfachen Handhabung bei Entwicklern beliebt. Allerdings ist sie nur eine semi-formale Notation, bei der keine exakte Semantik standardisiert ist.

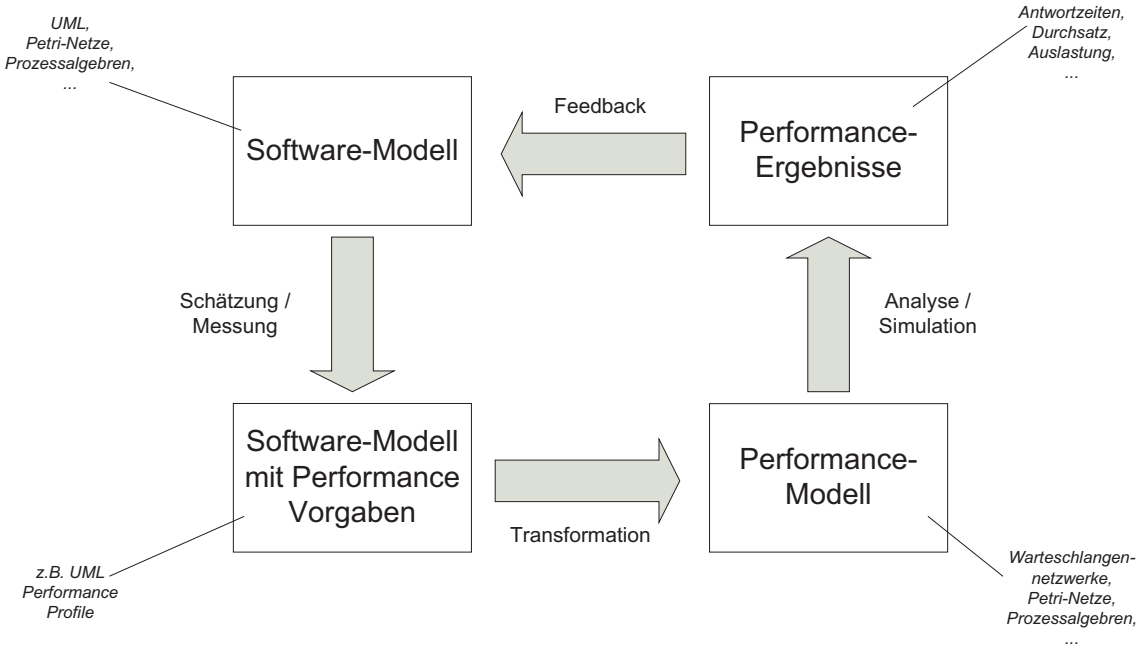


Abbildung 1: Generischer Prozess der Performance-Analyse

Balsamo et. al. [BMIS02] nennen außer der UML als weitere Notationen zur Modellierung von dynamischen Software-Spezifikationen für die Performance-Analyse endliche Automaten, Prozessalgebren, Petri-Netze, Message Sequence Charts und Use Case Maps. Prozessalgebren und Petri-Netze sind nicht nur aufgrund ihrer formalen Spezifikation mit exakter Semantik interessant. Sie eignen sich auch direkt als Performance-Modelle, so dass die funktionalen und nicht-funktionalen Eigenschaften (wie z.B. die Performance) innerhalb eines Modells dargestellt werden können. Jedoch sind diese Notationen mit einer höheren Lernkurve verbunden und in der Praxis kaum verbreitet. Prozessalgebren werden z.B. im Verfahren [BCD00] verwendet, während [KP00] und [BDM02] ihre Analysen mit Stochastischen Petri-Netzen durchführen.

Auf Basis des Software-Modells werden anschließend Performance-Modelle erstellt. Man unterscheidet zwischen drei unterschiedlichen Modellarten [BGM03]:

- **Analysemodelle** sind am weitesten verbreitet und basieren meistens auf Warteschlangennetzwerken ([SW02], [MA02], [PS02], [CM02], [GM01] und [Käh01]), Petri-Netzen oder Prozess-Algebren. Zur ihrer Auswertung werden die darin enthaltenen Markov-Ketten berechnet. Allerdings sind diese Modelle in ihrer Ausdruckskraft beschränkt und können beispielsweise Nebenläufigkeit oder simultane Ressourcennutzung nicht effizient abbilden.

- **Simulationen** ([BMIS02], [AS99], [MLH+00]) hingegen besitzen in ihrer Ausdruckskraft nicht die Einschränkung von Analysemodellen und können bezüglich ihres Abstraktionsniveaus beliebig detailliert sein und. Sie messen Performance-Werte auf einem Testsystem und erfordern bei komplexeren Systemen für die Ausführung oft einen hohen Aufwand an Rechenzeit. Die Ergebnisse der Ausführung müssen dann mit statistischen Methoden ausgewertet werden.

- **Hybride Modelle** beinhalten sowohl analytische Elemente als auch Simulationstechniken, allerdings gibt es hier noch keine brauchbare Ansätze.

Zur Erstellung der Performance-Modelle kann bei Verwendung der UML unterstützend das 'UML Profile for Schedulability, Performance and Time' verwendet werden, das 2003 von der OMG offiziell spezifiziert wurde. [OMG03a] Es erlaubt die standardisierte Beschriftung von UML-Diagrammen mit Performance-relevanten Informationen, die dann tool-gestützt zur Erstellung von Performance-Modellen genutzt werden können.

Die Ergebnisse der Auswertung von Performance-Modellen (z.B. überlastete Ressourcen in der Architektur) sollen idealerweise direkt in das ursprüngliche Software-Modell übertragen werden. [BMIS02] Auf diese Weise kann der Entwickler sofort die Schwachpunkte seiner Architektur erkennen, Anpassungen vornehmen und verschiedene Alternativen ausprobieren. Dieser Ansatz der Rückmeldung von Ergebnissen ist noch kaum umgesetzt worden, da die benutzten Performance-Modelle wie z.B. Warteschlangennetzwerke oder Petri-Netze gegenüber UML-Modellen meist weniger detailliert sind und eine direkte Zuordnung der Performance-Probleme in die Original-Notation erschweren. [BGM03]

Im folgenden werden die Performance-Analyseverfahren von Smith/Williams, Menasce/Almeida sowie Balsamo/Marzolla überblicksartig näher erläutert. Die ersten beiden Verfahren sind nach initialer Literaturanalyse von 16 verschiedenen Ansätzen die momentan am weitesten ausgearbeiteten und beruhen jeweils auf umfangreichen Monografien. Die Methode von Balsamo/Marzolla wurde ausgewählt, weil sie zur Klasse der simulationsbasierten Verfahren gehört und bereits das UML Performance Profile verwendet, das in der Studie ebenfalls untersucht werden soll.

3.2 Verfahren von Smith/Williams (SPE)

Smith/Williams geben zu ihrer SPE-Methode [Smi90] [SW02] ein ausführliches, schrittweises Vorgehensmodell zur Integration in den Software-Entwicklungsprozess an.

Nach allgemeiner Abgrenzung von Performance-Risiken der zu entwerfenden Applikationen (1) werden zunächst Performance-kritische Use-Cases gesucht (2), die dann mit einer erweiterten Form von UML-Sequenzdiagrammen ausführlich modelliert werden. Dabei werden solche Szenarien (also Abfolgen von Benutzerinteraktionen) ausgewählt, die häufig vorkommen und besonders Performance-kritische Elemente enthalten (3). Für diese Szenarien werden konkrete Performance-Vorgaben festgelegt (4). Beispielsweise darf die

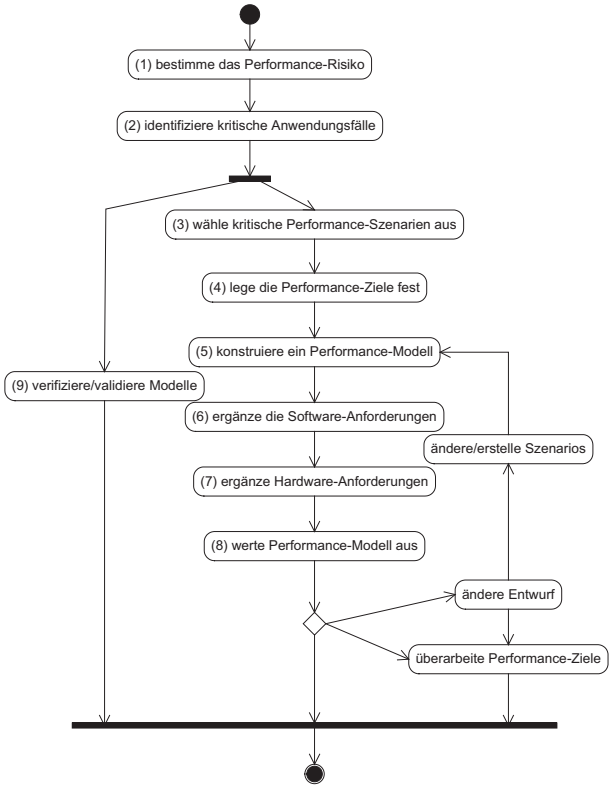


Abbildung 2: SPE-Prozess

maximale Antwortzeit für eine Benutzeranfrage höchstens fünf Sekunden betragen oder es müssen mindestens zehn Anfragen pro Minute verarbeitet werden können.

Aus den Sequenzdiagrammen für die wichtigsten Szenarien wird dann das so genannte 'Software Execution Model' erstellt, das in Form von Ausführungsgraphen (ähneln UML-Statecharts) repräsentiert wird (5). In die Ausführungsgraphen fließen Informationen über die Ausführungshäufigkeit und -wahrscheinlichkeit einzelner Aktionen ein (6). So bestellt beispielsweise ein Kunde bei einem Web-Shop durchschnittlich drei Artikel oder tätigt bei 40 Prozent seiner Online-Banking-Sitzungen eine Überweisung. Diese Werte werden entweder statistisch bestimmt, in Performance-Walkthroughs ermittelt oder aufgrund von Erfahrung abgeschätzt. Jedem Knoten eines Ausführungsgraphen wird eine Software-Anforderung (in Form von z.B. Prozessorbelastung oder Netzwerkzugriffen) zugeordnet, wobei 'best-case' und 'worst-case'-Anforderungen festgelegt werden. Beispielsweise braucht eine Aktion im best-case eine Netzwerknachricht und zwei Datenbankzugriffe.

Die Software-Anforderungen werden dann in einer Tabelle auf die Computer-Anforderungen abgebildet (7). So braucht z.B. ein Datenbankzugriff auf dem Zielsystems zur Ausführung 500.000 CPU-Instruktionen und zwei Festplattenzugriffe. Da man die Zeit für die Ausführung dieser Aktionen kennt (Prozessor- bzw. Festplattenspezifikation), kann man nun für den gesamten Ausführungsgraphen die Antwortzeit ausrechnen. Bei der Be-

rechnung werden die vorher angegebenen Wahrscheinlichkeiten für bestimmte Aktionen berücksichtigt. Im Normalfall werden sowohl best-case als auch worst-case Zeiten berechnet. Automatisiert man die Berechnung mit Tools (z.B. SPE-ED von Smith/Williams), können schnell verschiedene Entwurfsalternativen getestet werden (8).

Dieses 'Software-Execution-Model' auf Basis von Ausführungsgraphen ist ein einfaches Modell und berücksichtigt noch nicht, dass meist mehrere Benutzer oder Prozesse gleichzeitig um begrenzte Ressourcen (wie CPU, Datenbanken, Netzwerkverbindungen) konkurrieren. Falls mit dem 'Software-Execution-Model' keine Performance-Probleme mehr gefunden werden, wird deshalb danach ein weiterführendes 'System-Execution-Model' erstellt. Dieses wiederum benutzt zur Modellierung von konkurrierenden Ressourcen Warteschlangennetzwerke. Damit kann dann z.B. die durchschnittliche Zeit berechnet werden, die ein Prozesse in einer Warteschlange auf seine Bearbeitung wartet oder die Auslastung einer Ressource. Auf diese Weise können Flaschenhälse in der Hardware-Architektur identifiziert werden.

Zu untersuchende Aspekte: Ein kritischer Punkt dieses Verfahren ist die feingranulare Abschätzung der Performance-Attribute einzelner Schritte. Es bleibt zu zeigen, ob diese Werte bereits im Entwurfsstadium realistisch vorhergesagt werden können. Das Verfahren verwendet proprietäre Notationen wie die erweiterten Sequenzdiagramme und Ausführungsgraphen. Hier stellt sich die Frage, ob die Testpersonen diese Notationen annehmen und als sinnvoll erachten. Weiterhin soll das Tool SPE-ED kritisch untersucht werden.

3.3 Verfahren von Menasce/Almeida

Das Verfahren von Menasce und Almeida [MA02] [MAD04] ist speziell auf die Performance-Analyse von Web-Applikationen ausgerichtet. Es unterscheidet sich insofern von dem Verfahren von Smith und Williams, als dass eine Software-Architektur nicht explizit modelliert wird, sondern nur die im Web-Umfeld üblichen Client/Server-Systemen betrachtet werden. Es geht von fertigen Anwendungen aus, für die das umgebene System bestimmte Performance-Vorgaben einhalten muss. Im Kern basiert das Verfahren allerdings auch auf analytischen Performance-Modellen, die auf Warteschlangennetzwerken beruhen. Zusätzlich gibt es ein 'Workload-Model', mit dem die speziellen Anforderungen von Web-Applikationen bezüglich der Systemauslastung modelliert werden. Um auch eine betriebswirtschaftliche Perspektive einzubeziehen gibt es darüber hinaus noch ein 'Cost Model' zur Abschätzung der Kosten.

Ziel dieser Methode ist es, eine adäquate Kapazität für eine Web-Applikation zu bestimmen. Dazu gehört die Einhaltung von Performance-Vorgaben, so genannter Service Level Agreements (SLA), auf einer meist schon bestehenden Hardware/Software-Infrastruktur. Nebenbei sollen Kostenvorgaben durch das Management eingehalten werden.

In Abbildung 3 sind die einzelnen Schritte des Verfahrens dargestellt. Als erstes wird

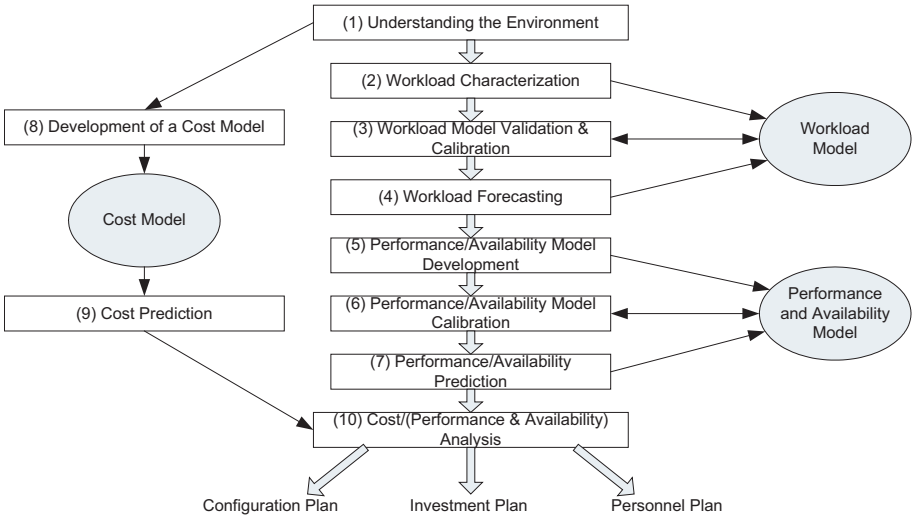


Abbildung 3: Menasce-Prozess

die bestehenden System-Infrastruktur untersucht (1), was z.B. die vorhandenen Rechner, Netzwerke und Applikationen beinhaltet. Dazu werden die möglichen Spitzenauslastungen des Systems ermittelt und die zugehörigen SLAs definiert.

In den nächsten drei Phasen (2-4) wird das 'Workload-Model' konstruiert und ggf. angepasst. Die globale Auslastung der Applikation wird dabei in lokale Lasten einzelner Komponenten zerlegt und durch Messungen oder Schätzungen mit konkreten Werten belegt. Vorhersagen für die zu erwartende Systembeanspruchung werden mit statistischen Verfahren ermittelt.

Nach Konstruktion des 'Workload-Models' folgt in den nächsten drei Phasen (5-7) die Konstruktion und Analyse des 'Performance-Models'. Die Parameter dieses Modells unterteilen sich in System-Parameter (z.B. die maximale Anzahl von Verbindungen, die ein Web-Server annehmen kann oder die Regeln für die Lastverteilung auf mehrere Server), Ressourcen-Parameter (z.B. Festplattenzugriffszeiten, CPU-Geschwindigkeit) und Workload-Parameter, die aus dem 'Workload-Model' gewonnen werden. Aus diesen Parametern werden mit Hilfe des 'Performance-Models' die gesuchten Performance-Daten wie Antwortzeiten, Durchsatz, Ressourcenauslastung und die Länge von Warteschlangen berechnet.

Die Modellierung des 'Performance-Models' selbst erfolgt auf zwei unterschiedlichen Ebenen. Auf der System-Ebene wird das gesamte System als eine Black-Box betrachtet, also der innere Aufbau des System vernachlässigt und nur die eingehenden Anfragen und abgegebenen Antworten untersucht. Auf der Komponenten-Ebene modelliert man die einzelnen Ressourcen, aus denen das System besteht (z.B. Web-Server, Netzwerkverbindungen, Datenbanken) durch ein Warteschlangennetzwerk. Dabei werden offene (mit einer unbegrenzten Anzahl von Anfragen) und geschlossene (feste Anzahl von Anfragen) Warteschlangennetzwerke unterschieden. Die möglichen Anfragenmengen unterteilen sich in einklassige (nur gleichartige Anfragen) und mehrklassige (unterschiedliche Arten von

Anfragen) Mengen.

Während der Konstruktion von 'Workload-Model' und 'Performance-Model' wird gleichzeitig das 'Cost-Model' entwickelt (8-9). Die Kosten werden in Startup-Kosten und laufende Kosten unterteilt. Der Detaillierungsgrad der Modellierung ist dabei beliebig. Es werden Kosten für Hardware, Software, Telekommunikation, Dienste von Drittanbietern und Personal abgeschätzt.

Nach Fertigstellung aller Modelle können dann die zu erwartenden Kosten mit der gewünschten Performance verglichen und verschiedene Szenarios abgewogen werden (10). Für jedes Szenarios kann die Performance des gesamten Systems als auch jeder Basis-Komponente vorhergesagt werden und eine Abschätzung der entstehenden Kosten vorgenommen werden. Aus diesen Analysen entstehen dann neue Pläne für die Konfiguration der System-Architektur, die neuen Investionen und das Personal.

Das Vorgehensmodell ist bei diesem Verfahren nicht so straff vorgegeben wie bei Smith/Williams. Vielmehr werden im entsprechenden Buch eine Fülle von Formeln und kleineren Methoden in loser Folge vorgestellt. Werkzeuge existieren nur in Form von Excel-Tabellen, eine Modellierung mit UML-Diagrammen ist nicht vorgesehen.

Zu untersuchende Aspekte: Da sich dieses Verfahren stark auf die Verwendung von Warteschlangennetzwerken stützt, sollen die Vor- und Nachteil dieser Modellierungsmethode näher untersucht werden. Insbesondere die Grenzen von Warteschlangennetzwerken bzgl. nebenläufiger Ressourcennutzung und Komplexität der Architektur sind in der Fallstudie zu beleuchten. Fraglich bleibt auch, wie sich die mangelnde Toolunterstützung auf die Praktikabilität des Verfahrens auswirkt.

3.4 Verfahren von Balsamo/Marzolla (umlPSI)

Das noch recht neue Verfahren von Balsamo/Marzolla [BGM03] [BM03] ist nicht wie die beiden vorangegangenen Verfahren analytisch, sondern beruht auf Simulationen. Es lässt sich aufgrund der Nutzung von UML-Diagrammen und der Verfügbarkeit eines Tools zur automatischen Erstellung der Simulationen leicht in den Software-Entwicklungsprozess integrieren. Ein manuelles Erstellen der Performance-Modelle wie in den anderen Methoden ist nicht notwendig.

Ausgangspunkt des Verfahrens sind Use-Case-, Deployment- und Aktivitätsdiagramme, mit denen die zu untersuchende Architektur unter Benutzung eines CASE-Tools (ArgoUML) modelliert wird. Diese Diagramme werden mit Hilfe des UML Performance Profiles [OMG03b] mit Performance-Angaben versehen. Die anschließend im Datenformat XMI [OMG02] vorliegenden Diagramme dienen dem Simulationstool UML-Performance-Simulator (umlPSI) als Eingabe. Dieses erzeugt automatisch eine prozessorientierte Simulation, für die noch die Anzahl der Jobs im System sowie Konfidenzintervalle angegeben werden müssen.

Die Ausführung der Simulation liefert die mittlere Antwortzeit eines Szenarios der Architektur zurück. Aufgrund der Verwendung des UML Performance Profiles können die

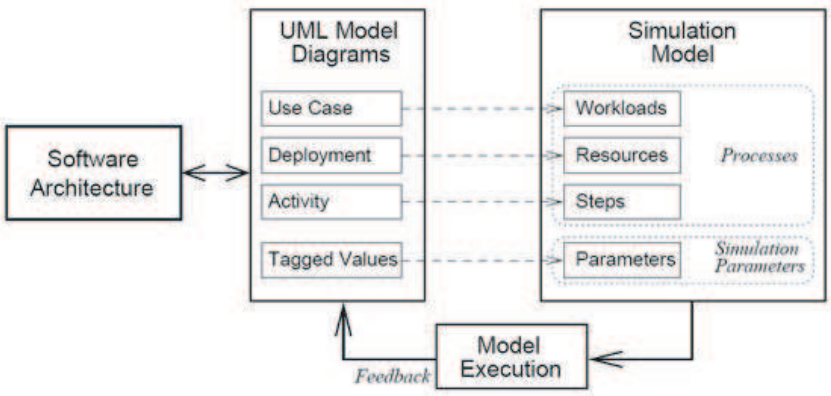


Abbildung 4: Abbildung von UML-Diagrammen auf ein Simulationsmodell

gemessenen Werte direkt in die XMI-Dateien zurückgeschrieben werden, da das Profile für die Ergebnisse eigene 'tagged-values' vorsieht. Auf diese Weise kann der Entwickler im CASE-Tool sofort die in seiner Architektur vorhandenen Performance-Probleme zuordnen und schnell verschiedene Alternativen testen.

Für dieses Verfahren liegen zwei Fallstudien der Autoren vor. [BM03] [BMDI04].

Zu untersuchende Aspekte: Bei diesem Verfahren steht die Untersuchung der Brauchbarkeit des UML Performance Profiles und des umlPSI-Tools im Vordergrund. Da das UML Performance Profile noch recht neu ist, liegen entsprechende Arbeiten zur Validierung noch nicht vor. Das umlPSI-Tool liegt nur in einer Alpha-Version vor und funktionierte in Vortests noch nicht stabil. Trotzdem sollen Möglichkeiten zur Erweiterung des Tools bzgl. andere Quality-of-Service Attribute und der Integration in andere CASE-Tools evaluiert werden. Auch das zugrundeliegende Simulationsmodell ist in Hinsicht auf die Vor- und Nachteile gegenüber analytischen Verfahren zu bewerten.

4 Organisatorisches

4.1 Betreuer

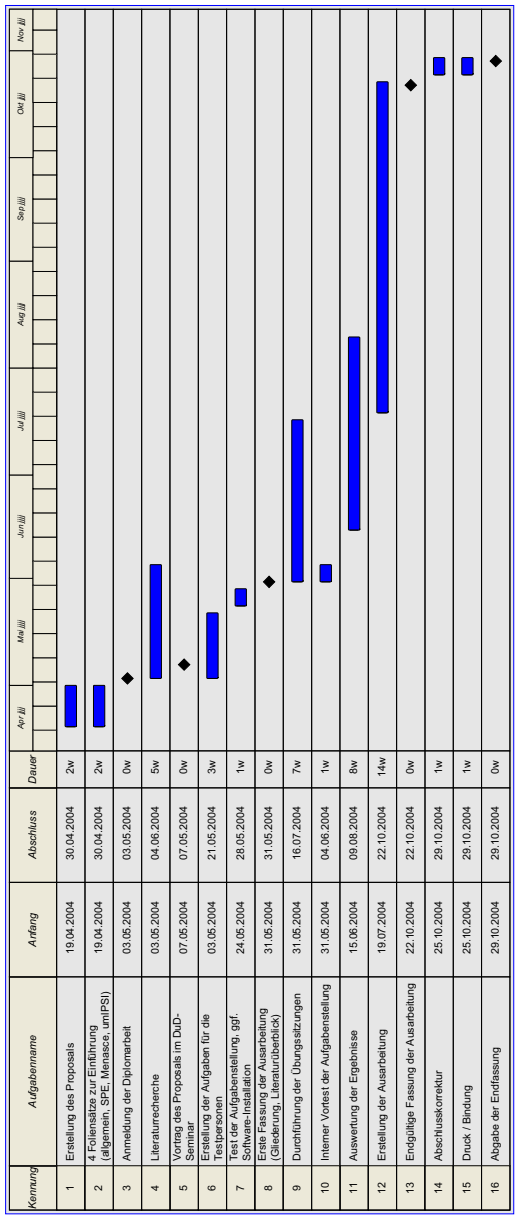
- **Erstgutachter:** Jun.-Prof. Dr. Ralf Reussner
- **Zweitgutachter:** Prof.-Dr. Wilhelm Hasselbring
- **Betreuer:** Dipl.-Wirtsch.-Inf. Steffen Becker

4.2 Entwicklungsumgebung

- Tools wie SPE-ED und umlPSI müssen installiert werden
- Für die Erstellung von UML-Diagrammen wird ArgoUML bzw. Poseidon verwendet.

5 Zeitplan

Nach der DPO3 beträgt die Dauer der Diplomarbeit sechs Monate. Eine genauere Aufstellung des Zeitplans ist in der folgenden Abbildung zu finden. Während der Diplomarbeit finden in der Regel wöchentliche Besprechungen mit den Betreuern statt.



Kennung	Aufgabenname	Anfang	Abschluss	Dauer
1	Erstellung des Proposals	19.04.2004	30.04.2004	2w
2	4 Foliensätze zur Einführung (allgemein, SFE, Menasce, IumPSI)	19.04.2004	30.04.2004	2w
3	Anmeldung der Diplomarbeit	03.05.2004	03.05.2004	0w
4	Literaturrecherche	03.05.2004	04.06.2004	5w
5	Vortrag des Proposals im DuD-Seminar	07.05.2004	07.05.2004	0w
6	Erstellung der Aufgaben für die Testpersonen	03.05.2004	21.05.2004	3w
7	Test der Aufgabenstellung, ggf. Software-Installation	24.05.2004	28.05.2004	1w
8	Erste Fassung der Ausarbeitung (Gliederung, Literaturüberblick)	31.05.2004	31.05.2004	0w
9	Durchführung der Übungssitzungen	31.05.2004	16.07.2004	7w
10	Interner Vortest der Aufgabenstellung	31.05.2004	04.06.2004	1w
11	Auswertung der Ergebnisse	15.06.2004	09.08.2004	8w
12	Erstellung der Ausarbeitung	19.07.2004	22.10.2004	14w
13	Endgültige Fassung der Ausarbeitung	22.10.2004	22.10.2004	0w
14	Abschlusskorrektur	25.10.2004	29.10.2004	1w
15	Druck / Bindung	25.10.2004	29.10.2004	1w
16	Abgabe der Endfassung	29.10.2004	29.10.2004	0w

Abbildung 5: Zeitplan der Diplomarbeit

Literatur

[AS99] L. B. Arief and N. A. Speirs. Automatic generation of distributed system simulations from uml. In *Proceedings of ESM '99, 13th European Simulation Multiconference*, 1999.

[BCD00] M. Bernardo, P. Ciancarini, and L. Donatiello. Aempa: A process algebraic description language for the performance analysis of software architectures. In *Proceedings of WOSP2000*, 2000.

[BDM02] S. Bernardi, S. Donatelli, and J. Merseguer. From uml sequence diagrams and state-charts to analysable petri net models. In *Proceedings of WOSP2002*, 2002.

[BGM03] S. Balsamo, M. Grosso, and M. Marzolla. Towards simulation-based performance modeling of uml specifications. Technical report, Dipartimento di Informatica, Universitá Ca' Foscari di Venezia, 2003.

[BM03] S. Balsamo and M. Marzolla. A simulation-based approach to software performance modeling. Technical report, Universitá Ca' Foscari di Venezia, 2003.

[BMDI04] S. Balsamo, M. Marzolla, A. DiMarco, and P. Inverardi. Experimenting different software architectures performance techniques: A case study. In *Proceedings of the fourth international workshop on Software and performance*, pages 115–119. ACM Press, 2004.

[BMIS02] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni. Software performance: state of the art and perspectives. Technical report, Dipartimento di Informatica, Universita Dell'Aquila, 2002.

[CM02] V. Cortellessa and R. Mirandola. Prima-uml: a performance validation incremental methodology on early uml diagrams. In *Proceedings of WOSP*, 2002.

[DS00] Evgeni Dimitrov and Andreas Schmietendorf. Uml-basierte performance engineering. In *Proceedings des 1. Workshop Performance Engineering in der Softwareentwicklung*, 2000.

[GM01] H. Gomaa and D. A. Menasce. Performance engineering of component-based distributed software systems. *Lecture Notes in Computer Sciece 2047*, 2001.

[Käh01] P. Kähkipuro. Uml-based performance modeling framework for component-based distributed systems. *Lecture Notes in Computer Science 2047*, 2001.

[KP00] P.J.B. King and R.J. Pooley. Derivation of petri net performance models from uml specifications of communications software. *Lecture Notes in Computer Sciece 2047*, pages 262–276, 2000.

[MA02] Daniel A. Menasce and Virgilio A.F. Almeida. *Capacity Planning for Web Services*. Prentice-Hall, 2002.

[MAD04] Daniel A. Menasce, Virgilio A.F. Almeida, and Lawrence W. Dowdy. *Performance by Design*. Prentice Hall, 2004.

[MLH+00] M. De Miguel, T. Lamobolais, M. Hannouz, S. Betge-Brezetz, and S. Piekarec. Uml extensions for the specifications and evaluation of latency constraints in architectural models. In *Proceedings of WOSP*, 2000.

[OMG02] Object Management Group OMG. Xml metadata interchange (xmi) specification, Jan. 2002.

[OMG03a] Object Management Group OMG. Uml profile for schedulability, performance and time. http://www.omg.org/cgi-bin/doc?formal/2003-09-01, 2003.

[OMG03b] Object Management Group OMG. Unified modelling language (uml), version 1.5. http://www.omg.org/cgi-bin/doc?formal/03-03-01, 2003.

[Pre01] Lutz Prechelt. *Kontrollierte Experimente in der Softwaretechnik*. Springer Verlag, 2001.

[PS02] D.C. Petriu and H. Shen. Applying the uml performance profile: graph grammar-based derivation of lqn models from uml specifications. *Lecture Notes in Computer Science 2324*, 2002.

[Smi81] C.U. Smith. Increasing information systems productivity by software performance engineering. In Donald R. Deese, Robert J. Bishop, Jeffrey M. Mohr, and H. Pat Artis, editors, *Int. CMG Conference*, pages 5–14. Computer Measurement Group, 1981.

[Smi90] C.U. Smith. *Performance Engineering of Software Systems*. Addision-Wesley, 1990.

[SW02] C.U. Smith and L. Williams. *Performance Solutions: A Practical Guide To Creating Responsive, Scalable Software*. Addision-Wesley, 2002.