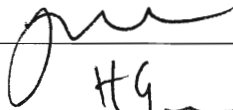


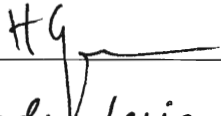
A METHODOLOGY FOR MAKING EARLY COMPARATIVE  
ARCHITECTURE PERFORMANCE EVALUATIONS

by


Gerald S. Doyle  
A Dissertation  
Submitted to the  
Graduate Faculty  
of  
George Mason University  
In Partial Fulfillment of  
The Requirements for the Degree  
of  
Doctor of Philosophy  
Computer Science

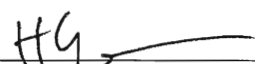
Committee:

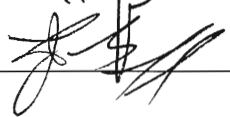
  
\_\_\_\_\_ Dr. Elizabeth White, Dissertation Director

  
\_\_\_\_\_ Dr. Hassan Gomaa, Committee Member

  
\_\_\_\_\_ Dr. Alex Levis, Committee Member

  
\_\_\_\_\_ Dr. Robert Simon, Committee Member

  
\_\_\_\_\_ Dr. Hassan Gomaa, Department Chair

  
\_\_\_\_\_ Dr. Lloyd J. Griffiths, Dean, The Volgenau  
School of Information Technology and  
Engineering

Date: 3 Dec 2010  
Fall Semester 2010  
George Mason University  
Fairfax, VA

A Methodology for Making Early Comparative  
Architecture Performance Evaluations

A dissertation submitted in partial fulfillment of the requirements for the degree of  
Doctor of Philosophy at George Mason University

By

Gerald S. Doyle  
Masters of Science  
George Mason University, 2000  
Masters of Science  
Naval Postgraduate School, 1979  
Bachelor of Science  
United States Military Academy, 1973

Director: Elizabeth White, Professor  
Department of Computer Science

Fall Semester 2010  
George Mason University  
Fairfax, VA

Copyright 2010 Gerald S. Doyle  
All Rights Reserved

## **DEDICATION**

I dedicate this dissertation to my family, friends, and co-workers who donated the gifts of time, patience and encouragement to advance this work to completion.

## **ACKNOWLEDGEMENTS**

First and foremost I would like to express my gratitude to my family, friends and co-workers who provided me the time, flexibility and encouragement needed to complete this effort. The patience of Charlotte, Laura, Kim, Bruce and Bruce can not be overstated. I would like to thank Dr. Elizabeth White for her guidance in my study of architectures over the years as well as for her many suggested improvements to the manuscript. I express my appreciation to the committee for their diligent review of this work, and would further like to acknowledge the support of the broader GMU faculty in preparing me for this effort. In particular I would like to thank Dr. Sood for his insight into how to approach the dissertation, and Dr. Alexander Levis for broadening my engineering perspective on many difficult issues.

## TABLE OF CONTENTS

	Page
LIST OF TABLES .....	ix
LIST OF FIGURES .....	x
ABSTRACT .....	xiii
1 Introduction .....	1
1.1 Introduction .....	1
1.2 Systems Development Process .....	3
1.3 Uncertainties Encountered in the Systems Development Process.....	7
1.4 Performance Analysis to Date.....	9
1.5 Architecture Performance Evaluation Options.....	11
1.6 Performance Analysis Improvement Strategy .....	12
1.7 Work Focus.....	13
1.8 Research Strategy.....	16
1.9 Definitions .....	19
1.10 Research Approach.....	21
1.11 Assumptions.....	24
1.12 Organization.....	25
2 Related Work .....	26
2.1 Software Architecture Foundational Work.....	26
2.2 Architecture Description Languages.....	27
2.3 System Performance Evaluation .....	29
2.3.1 Queuing Networks.....	30
2.3.2 Petri Nets .....	32
2.3.3 Stochastic Process Algebras .....	33
2.3.4 Simulation Models.....	33
2.3.5 Combined Models .....	34
2.3.6 Component-based Architectures.....	34
2.3.7 Architecture Design Domain Specific Considerations .....	37
2.4 Other Domain Specific Concerns .....	39
2.5 Identifying Good Attributes for a Performance Analysis Approach.....	40
2.6 Summary.....	43
3 Architecture Performance Viewed as a Specific Experiment.....	44
3.1 Introduction .....	44
3.2 Early Performance Estimation Methodology .....	45
3.3 Defining an Experiment.....	47

3.4	Performance Estimation Fundamentals .....	50
3.5	Identifying Uncertainty Sources.....	51
3.5.1	Value or Data Based Performance Uncertainties.....	51
3.5.2	Algorithm-Based Performance Uncertainties .....	52
3.5.3	Topology-Based Performance Uncertainties.....	55
3.5.4	Synchronization-Based Performance Uncertainties .....	56
3.5.5	Load-Based Performance Uncertainties .....	57
3.5.6	Sizing-Based Performance Uncertainties.....	57
3.6	Summary.....	58
4	Comparing Architecture Performance Potential .....	60
4.1	Introduction .....	60
4.2	Deriving a Performance Probability Integral .....	61
4.3	Verifying the Probabilistic Analysis.....	68
4.4	Example 1 – Comparing Two Normal Performance Descriptions .....	68
4.5	Example 2 –Possibly Overlapping Performance Descriptions.....	73
4.6	Example 3 - Comparing Discrete Performance Descriptions .....	77
4.7	Example 4 – Asymmetric Performance Descriptions.....	79
4.8	Summary.....	81
5	Constructing Architecture Performance Descriptions .....	82
5.1	Introduction .....	82
5.2	Summation .....	84
5.3	Quotient.....	86
5.4	MIN.....	89
5.5	MAX .....	91
5.6	Composite .....	95
5.7	Summary.....	99
6	An Illustrative Practical Architecture Example .....	101
6.1	Introduction .....	101
6.1.1	Selecting Specific Architectures for Demonstration.....	105
6.2	Assigning Delay Descriptions to Architecture Elements.....	107
6.2.1	Characterizing the Architectural Elements .....	110
6.2.2	Architecture A.....	111
6.2.3	Architecture B.....	115
6.2.4	Architecture C .....	117
6.2.5	Architecture D .....	119
6.3	Comparing Results Across Architectures .....	121
6.3.1	Scaling Issues.....	123
6.4	Summary.....	125
7	Two Larger Examples > .....	126
7.1	Example One: Data Exfiltration .....	127
7.1.1	Problem Definition.....	127
7.1.2	Large-Grain Delay Descriptions .....	128
7.1.3	Alternative Problem Architectures .....	130
7.1.4	Delay Characterization.....	132

7.1.5	Data Characterization .....	132
7.1.6	Characterizing Delay $D_{L1}$ .....	133
7.1.7	Characterizing Delay $D_{L2}$ .....	137
7.1.8	Characterizing Delay $D_{L3}$ .....	139
7.1.9	Characterizing Delay $D_{L4}$ .....	140
7.1.10	Combining Component Performance Descriptions .....	141
7.1.11	Comparing the VIDEO Performance of the Alternatives .....	142
7.1.12	Example One Summary .....	144
7.2	Example Two: A Service Oriented Architecture Based Service .....	144
7.2.1	Problem Description.....	145
7.2.2	Large Grain Delay Descriptions .....	148
7.2.3	Delay Characterization.....	148
7.2.4	Data Characterization .....	149
7.2.5	Delay Characterization to the Photo Server, P.....	149
7.2.6	Delay Characterization to the Search Engine, S .....	150
7.2.7	Delay Characterization to the Directions Server, D .....	151
7.2.8	Combining Server Delays .....	153
7.2.9	Example Two Summary .....	158
7.3	Summary.....	159
8	The CAPE Tool .....	160
8.1	Introduction .....	160
8.2	The CAPE Tool Design .....	161
8.3	Code Snippet Functionality .....	165
8.3.1	Function Definitions.....	166
8.3.2	Modeling Topological Aspects of the Architecture.....	167
8.3.3	CAPE Evaluation of the PPI.....	169
8.4	Library Support.....	170
8.4.1	Results .....	171
8.4.2	A Simple Complete Example.....	172
8.4.3	Summary.....	174
9	Methodology Validation .....	176
9.1	Introduction .....	176
9.2	Modeling Technique .....	178
9.3	Result Comparison Techniques .....	179
9.3.1	Chi Squared Goodness of Fit Testing .....	179
9.3.2	Norm Based Difference Measurement - MESA vs. CAPE .....	182
9.4	Validation Examples.....	183
9.4.1	Example One – The Army Tactical Environment.....	183
9.4.2	Example Two – Real Estate Service .....	188
9.5	Comparing MESA and CAPE Results Quantitatively .....	193
9.5.1	Hypothesis Testing - MESA vs. CAPE .....	193
9.6	Conclusion .....	199
9.7	Summary.....	204
10	Contributions and Future Research .....	205



10.1	Introduction .....	205
10.2	Research Contributions.....	207
10.3	Future Research .....	209
10.3.1	Expand Offered Workload Analysis.....	209
10.3.2	Improve Model Implementation Efficiency.....	210
10.3.3	Applying CAPE to the Design of Software Architectures.....	210
10.3.4	Simplify Performance Specifications Graphical User Interface ...	211
10.3.5	Generate Parameterized Pre-built Architectural Entity Models ...	211
10.3.6	Quantify Improvement Factor.....	212
10.3.7	Specify an Appropriate Multi-Attribute Utility Function .....	212
10.3.8	Specify a Compatible Cost Model for a Bayes Decision .....	213
10.3.9	Summary.....	214
A	Function Model and Elementary Function Operations.....	215
A.1	Introduction .....	215
A.2	Representing Functions of One Variable.....	215
A.3	Operations on Functions .....	216
A.4	Scalar Multiplication .....	216
A.5	Shift Operation .....	216
A.6	Integration Operation .....	216
A.7	Differentiation Operation .....	217
A.8	Convolution Operation .....	217
A.9	makeCanonical() Operation .....	217
A.10	Representing Functions of Two Variable.....	217
A.11	Quasi-Arbitrary Random Variable Generation Functions .....	218
B	Detailed Examples of Uncertainty.....	221
C	Computational Methods and Verification .....	224
C.1	Introduction .....	224
C.2	Summation .....	224
C.3	Quotient.....	228
C.4	MIN.....	232
C.5	MAX .....	234
D	PLOT Library Functionality .....	237
	REFERENCES .....	239

## LIST OF TABLES

Tables	Page
Table 2.1 Performance and Domain Specific Architectures .....	39
Table 2.2 Desirable approach attributes .....	41
Table 6.1 Standard data table for comparing transition delays. ....	108
Table 6.2 Graph abbreviations used in analysis tables that follow .....	109
Table 7.1 Summary of data exfiltration system performance estimates .....	130
Table 7.2 Table of architectures considered .....	131
Table 7.3 PPI Results for Data Exfiltration .....	143

## LIST OF FIGURES

Figures		Page
Figure 1.1	V-type systems development process	5
Figure 1.2	Implementations possible from the systems development process	6
Figure 3.1	Visual description of the experiment	48
Figure 3.2	Curve characterizing processing and transmission time tradeoff	53
Figure 3.3	Computation and communications normalization	54
Figure 4.1	Delay density function and cumulative distribution function	62
Figure 4.2	Calculating the probability System A outperforms System B	64
Figure 4.3	Evaluation of the $P(A < t')$ for increasing values of $t'$	65
Figure 4.4	Simulation results for comparing systems from Figure 4.2	71
Figure 4.5	PPI values vs. difference in mean ( $B - A$ )	72
Figure 4.6	PPI calculation: non-overlapping performance descriptions	73
Figure 4.7	PPI calculation: overlapping performance descriptions	74
Figure 4.8	PPI simulation: overlapping performance descriptions	75
Figure 4.9	PPI calculation: complete overlap of performance descriptions	76
Figure 4.10	PPI simulation: narrow overlapping architecture	76
Figure 4.11	PPI calculation: discrete example	78
Figure 4.12	PPI simulation: discrete example	79
Figure 4.13	Non-symmetric performance description comparison	80
Figure 4.14	PPI simulation: Non-symmetric performance description	80
Figure 5.1	Summing process for activity performance descriptions	85
Figure 5.2	Source-sink performance descriptions for quotient calculation	88
Figure 5.3	Example case where first result is sufficient	90
Figure 5.4	Minimum function computation result	91
Figure 5.5	Timing for a subtasks processed in parallel	93
Figure 5.6	Maximum function computation result	94
Figure 5.7	PDF result of combing two data classes in a common channel	96
Figure 5.8	A single class of data is transmitted over any of multiple paths	97
Figure 5.9	Simulation confirmation of weighted composite calculations	98
Figure 5.10	Composite delay result for example two path calculation	99
Figure 6.1	Classical three tier architecture	103
Figure 6.2	XML Tree Transform Process Example	104
Figure 6.3	Alternative example architectures	105
Figure 6.4	Architecture A and associated component delays	113
Figure 6.5	Performance description for architecture A	114
Figure 6.6	Architecture B and associated delay tables	116
Figure 6.7	Performance description for architecture B	117

Figure 6.8	Architecture C and associated delay tables	118
Figure 6.9	Performance description for architecture C	119
Figure 6.10	Architecture D and associated delay tables	120
Figure 6.11	Performance description for architecture D	121
Figure 6.12	Plot of all four architecture delay descriptions	122
Figure 6.13	Summarizing all architectural performance results	123
Figure 6.14	Establishing related propagation delays and data rates	124
Figure 6.15	Result of scaling the satellite delay to the terrestrial delay	125
Figure 7.1	Data exfiltration architecture and labeling convention	131
Figure 7.2	LEO satellite accessibility times	134
Figure 7.3	Critical points for LEO satellite transmit delay	135
Figure 7.4	Architecture $A_{DE}$ , characterization of delay $D_{L1}$	136
Figure 7.5	Combined delay contributions for $D_{L1}$	137
Figure 7.6	Architecture $A_{DE}$ , characterization of delay $D_{L2}$	138
Figure 7.7	Architecture $A_{DE}$ , characterization of delay $D_{L3}$	139
Figure 7.8	Architecture $A_{DE}$ , characterization of delay $D_{L4}$	140
Figure 7.9	Video architecture performance descriptions	141
Figure 7.10	Initial activity diagram for SOA real estate service	146
Figure 7.11	Symbolic SOA real estate service architecture graph	147
Figure 7.12	Architecture SOA, Photo Server delay	150
Figure 7.13	Architecture SOA, Aerial Search Engine delay	151
Figure 7.14	Architecture SOA, Directions Server delay	152
Figure 7.15	Cumulative results for describing the delays of each server	154
Figure 7.16	Projected performance of original example SOA	155
Figure 7.17	Server contributions with parallel Photo Server	156
Figure 7.18	Performance of parallel Photo Server architecture	157
Figure 7.19	Projected performance of modified example SOA	158
Figure 8.1	NetBeans IDE with partial code templates	162
Figure 8.2	CAPE tool structure - PDF analysis	164
Figure 8.3	CAPE tool structure - PPI analysis	165
Figure 8.4	Building a Function point-wise within the code	166
Figure 8.5	CAPE pdf functions validating the chapter nine Army example	167
Figure 8.6	CAPE used to compute PPI	169
Figure 8.7	CAPE text output from example	172
Figure 8.8	CAPE input for example analysis	173
Figure 8.9	Raw macro result once executed within PPT	174
Figure 9.1	Top level MESA model for Army SOA example	185
Figure 9.2	Detailed CAPE delays with bounding values	186
Figure 9.3	PDF comparison of CAPE and MESA results Army example	187
Figure 9.4	MESA model for the real estate service (single photo server)	189
Figure 9.5	Single photo server delay contributions	190
Figure 9.6	Total delay for single photo server	191
Figure 9.7	Transmission delay contribution for the two photo server case	192
Figure 9.8	Cumulative delays for the two photo server case	192

Figure 9.9	Initial $\chi^2$ values vs. count needed to meet rules of thumb	194
Figure 9.10	Plot of $\chi^2$ values over time	195
Figure 9.11	Absolute MESA-CAPE estimate difference— Army example	197
Figure 9.12	Uncertainty contributions before design decisions are made	201
Figure 9.13	Changing performance bounds as decisions are made	202
Figure 9.14	Boehm software development cost uncertainty description	203
Figure C.0.1	Fifteen independent stage uniformly distributed delay line	224
Figure C.0.2	Iterative summation of uniform densities	225
Figure C.0.3	An irregular pdf describing system performance	227
Figure C.0.4	Another pdf describing system performance	227
Figure C.0.5	Convolution of pdfs from Figure C.3 and Figure C.4	228
Figure C.0.6	Relationship of transmit time to data size and data rate	229
Figure C.0.7	Quotient cumulative distribution function for the example	230
Figure C.0.8	Example probability density function	231
Figure C.0.9	Simulation of one million example quotients	231
Figure C.0.10	Example two input minimum geometry calculation	232
Figure C.0.11	Calculated cdf MIN for two input example	233
Figure C.0.12	PDF for minimum of the uniform joint pdf	234
Figure C.0.13	Example two input maximum pdf geometry calculation	235
Figure C.0.14	Example cumulative distribution – two input example	235
Figure C.0.15	Example two input maximum pdf calculation	236

## **ABSTRACT**

### **A METHODOLOGY FOR MAKING EARLY COMPARATIVE ARCHITECTURE PERFORMANCE EVALUATIONS**

Gerald S. Doyle, PhD

George Mason University, 2010

Dissertation Director: Dr. Elizabeth L. White

Complex and expensive systems' development suffers from a lack of method for making good system-architecture-selection decisions early in the development process. Failure to make a good system-architecture-selection decision increases the risk that a development effort will not meet cost, performance and schedule goals. This research provides a method to mitigate that risk based on the idea that a development can be characterized as the management of uncertainties in a probabilistic experiment. The method developed shows how to estimate the probability that an arbitrary implementation of one system-architecture will perform better than an arbitrary implementation of an alternate system architecture.

The analysis technique presented acknowledges that many implementation uncertainties exist at system-architecture-selection time and identifies steps that

can be used to characterize these uncertainties. The process by which uncertainty descriptions are combined into architectural performance descriptions is presented. Once all alternative system architecture performance descriptions are developed relative system architecture performance comparisons can be made.

After the analysis technique is described, three examples are considered. The first example is a simple three tier web-enabled database application. This small web application is used to illustrate the analysis method and demonstrate some methods for characterizing uncertainties. The next two examples are more complex. These examples expose a broader set of uncertainties and show how to handle cases where large numbers of uncertainties exist. Sections on validation of results follow. The dissertation concludes with a list of future research opportunities in this area.

## **Chapter 1**

### **Introduction**

#### **1.1 Introduction**

Developing a large system is a complex task that requires an understanding of the functional and non-functional requirements as documented in the systems requirements specification. Functional requirements relate to what the system does, and non-functional requirements relate to system quality. Non-functional requirements (often called quality attributes) include reliability, modifiability, portability, security, and performance, among others. Requirements relating to quality influence early system development much like functional requirements. Unlike functional requirements, performance requirements are often addressed later in the development process, and often addressed in an unstructured manner.

System architectures can be used to help manage the complexity inherent in systems development. A system architecture is a high level description of the proposed solution. It is valued for its ability to describe the large grain structure



of the objective system, suppressing unnecessary detail. Since a system's architecture affects all types of requirements, it is critical that a suitable system architecture be selected early in the development process.

A system architecture can be characterized as an enumeration of the system's large-grain components and the connections between those components. There are several techniques for describing system architectures. The earliest work in the software field used boxes (components) of various shapes, combined with lines (connectors) to characterize the interactions between components. This description based on combining boxes and lines creates a high-level pictorial representation of the proposed system structure. Many current description techniques are similar.

The most concise insight on the impact that the system architecture has on system performance comes from work done by Clements and Lakos. "Whether or not a system will be able to exhibit its desired (or required) quality attributes is largely determined by the time the architecture is chosen." [CINo96] Similarly Lakos [Lako96] writes, "If we fail to address our performance goals in the beginning, we may adopt architectures or coding practices that will preclude our ever achieving these goals, short of rewriting the entire system."

While there are a number of important quality attributes associated with systems, performance is the one that is the focus of this research.

## **1.2 Systems Development Process**

A systems development process is established in order to make the development process repeatable and to ensure that the product will have known quality. It usually enumerates a sequence of steps that the developer should follow. System development success hinges on meeting established goals for three criteria: cost, schedule, and performance [OMBC09]. Failing to achieve any of these goals compromises the project.

One of the primary sources of failing to meet goals is related to change. Changes resulting from either re-architecting or redesigning increase cost and extend schedules. The costs associated with making changes are accrued in activities like revising functional requirements, changing quality attribute specifications, modifying the system architecture, and changing the system design to fix unmet quality goals and functional requirements. System development costs are not uniformly distributed over the system lifecycle and neither are the costs of change. There is general agreement that mistakes made early in the system development process are more costly to correct than those made later in the lifecycle [WiKe00]. There are disproportionately large expenses associated with architectural modification due to the deep understanding that is required to successfully make significant architectural changes and due to the magnitude and number of the changes that are routinely

needed [CIDB98]. Reducing the risk of making an inappropriate architecture-selection-decision will decrease the expected cost of the development.

There are many approaches to systems development. A typical system is built using a development model like the classical waterfall model, waterfall models with feedback, V-models [Somm04], or alternatives. The development model establishes a sequence of activities or steps that guide developers as they plan the work to be done. This series of steps usually begins with requirements specification and runs through end-of-life disposition. While different development approaches address different development concerns, each approach includes some type of architecture design phase followed by a detailed design phase. Most include feedback mechanisms between the phases. The waterfall model with feedback, for example, helps accommodate the reality that the dividing line between the architecture design and detailed design phases is imprecise. The model's feedback paths allow information learned while performing the detailed design to force changes in the architecture design. Architectural changes as well, often directly influence detailed design decisions. An architecture design phase is usually identified early in the step sequence. Once a particular problem is identified and system requirements are specified, an architecture-design effort identifies a set of potential or candidate architectures to be considered.



architectures. Below them are the possible detailed designs, followed by the implementations.

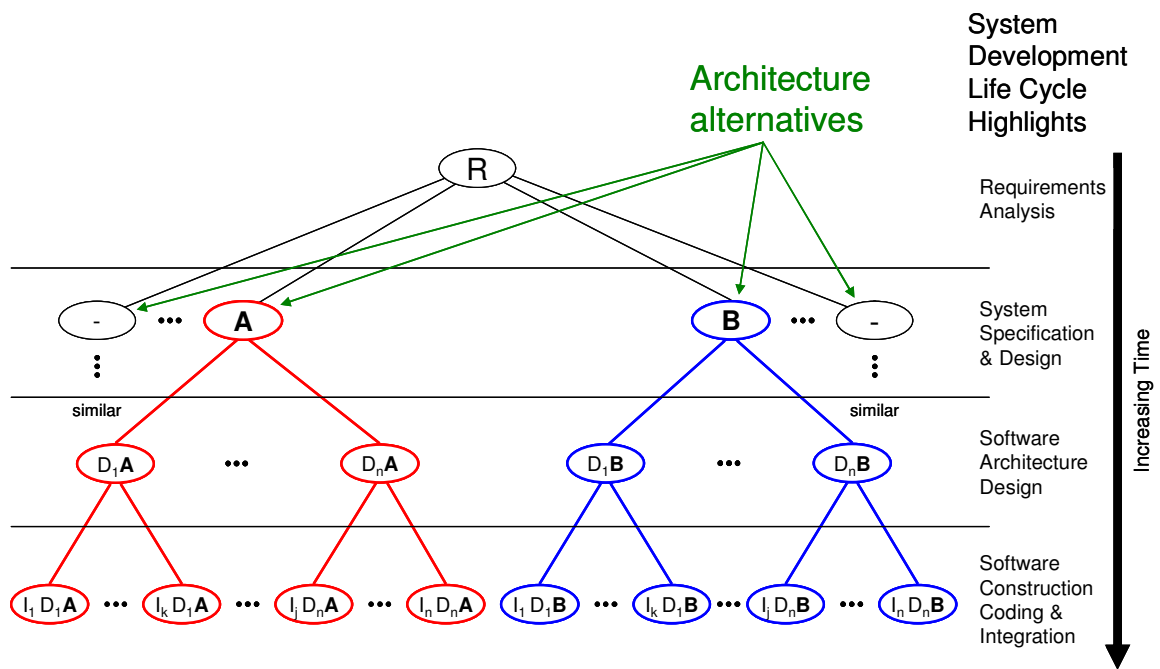


Figure 1.2 Implementations possible from the systems development process

From that set of initial architectures, either a specific architecture may be selected or the architecture-selection decision may be delayed until further analysis is conducted. When the selection decision is delayed, another set of

steps is executed to refine each of the architecture options. The developer refines broader descriptions to more specific ones. For each of the choices considered, the product's required functions are assigned to architecture specific sub-systems or components. When the functions or the components to which they are assigned are complex, either may be decomposed again (possibly a number of times). This iterative process defines the structure of the detailed design. Eventually, one specific detailed design is selected from a number of possible alternatives.

In a similar way, for each design there is a set of reasonable implementations. As the implementation options are examined, the design process eventually locks-down the types of data structures to be used, the control flow of the computations, the amount of concurrency that will be provided, as well as a number of other factors. Each of these low level design decisions can be viewed as the selection of one specific choice from a number of options. Before the decision is made however, each decision carries with it an amount of uncertainty in the system performance.

### **1.3 Uncertainties Encountered in the Systems Development Process**

That there are undefined parameters early in the system design, and that there are many design decisions yet to be made in the development processes leads to two conclusions: 1) not knowing the performance impact of the specific

implementation decisions (those yet to be made) means that uncertainties exist, and 2) both the sources and amounts of uncertainty will likely change over time. The uncertainties present in the earlier development stages are reduced as the detailed design decisions are made.

At architecture-selection time, there are a large number of implementation details that have not been decided. The uncertainty generated by not knowing these details prevents the designer from comprehensively and hence precisely assessing the expected performance of the final product. Any performance analysis method to be used early in development must be able to handle the fact that design details will be missing when architectural evaluations are conducted. Over the early to middle part of the development, the percentage of architectural design usually decreases as the product design matures, while the amount of detailed design increases as the design is refined and gets closer to the actual implementation. It is the specification of these details during the subsequent course of the system design that collectively determines the final performance of the instantiated system.

There are also non-technical issues that can affect the quality attributes resulting from executing the systems development process. Issues not directly related to the system actually built, but to the processes and people that are used to generate that implementation. A designer's capabilities, i.e., their knowledge

strengths and weaknesses, can vary widely. Different implementers will be either more or less efficient and do either a better or worse job in creating the implementation. Each of these non-technical uncertainties may lead to a different implementation and hence different performance expectations.

Until the implementation exists, an exact performance characterization of an “as-produced” system can not be established. Estimated values (with uncertainties) must be used for analysis. This research presents a structured method for managing these types of uncertainties.

#### **1.4 Performance Analysis to Date**

Performance analysis is routinely done near the end of a development cycle and is informed by a number of design and code artifacts. Performance analysis near the beginning of the development cycle does not have access to this information. While most development processes already include an uncertainty reduction characteristic realized through iterative feedback, most performance evaluation methods do not. There are three methods often used to assess and improve system performance: code profiling, queuing analysis and modeling and simulation.

Over the past decade, engineers have begun to address performance issues but later in the development effort. They often use a cyclic performance-



improvement approach. There are two general cases. In the first case, available code elements are profiled so that code sections consuming larger proportions of time can be identified. Recoding "slow" sections then changes the performance. The profiling process is then repeated to quantify the performance of the modified system. The newly measured performance is compared with the desired performance goal. If the goal is not met, the profile-rewrite cycle continues until no further improvement seems possible [WiKe00]. In those cases, if performance is still considered to be unacceptable, system redesign or architectural changes provide the next target for making improvement [Bulk00].

In the second general case, the anticipated detailed design is analyzed using either queuing theory or simulation. These closely related methods are usually applied after the architecture-selection decision has been made. For both queuing models and simulation, detailed system models are usually created. These models typically require detailed design information, e.g., component topologies, data flow rates, queuing model strategies, execution time constants, etc. to characterize the proposed system effectively. The queuing or simulation approaches target performance estimation earlier than those that actually make measurements on code, but the required information is still not available until late in the development's detailed design phase. At this point, the developer can choose to wait for detailed design values to become available, or can provide estimates of structure and performance without knowing these details. In the first

case, waiting until later in the design process, the evaluation is delayed. In the second case, uncertainties are injected into the evaluation process. While the estimates such queuing and simulation models produce could give insight into performance uncertainty, there is seldom an attempt to characterize it.

### **1.5 Architecture Performance Evaluation Options**

In selecting an approach to evaluate architecture performance, there are at least three ways to proceed. From a conceptual point of view, if given sufficient resources, all implementations could be instantiated and tested across the anticipated input datasets to provide data for making performance comparisons (using a suitable performance metric). This technique would identify the architecture which produced the best performing implementation. This approach is impractical. Even for a small system, the set of implementations can be very large. At the opposite extreme, a single architecture could be selected (perhaps at random) to be developed. This is routinely what is done as state-of-the-art today. A third and more beneficial approach would be to perform an analysis on each of the alternative architectures to approximate the likely performance attributes of each, then base architecture selection on these estimated values.

By necessity such an analysis would have to speculate or assume the assignment of functions to components, the internals of component design, specify details of implementation, etc. Since the true design decisions have not

yet been made, such estimates could provide a viable way to proceed. In cases where there is additional information, informed decisions can be made to eliminate early implementations that are projected to perform less well, thus pruning the previously described tree, Figure 1.2. While this approach can reduce the size of the potential outcome space it does not routinely identify a single architecture as being the best to select.

## **1.6 Performance Analysis Improvement Strategy**

This work focuses on those systems whose development success is closely tied to meeting performance expectations. The ability to make a good architecture-selection decision is critical to achieving that outcome. Moving performance analysis earlier in the development process requires these performance uncertainties to be managed in a structured manner. This effort proposes a methodology for making architecture-selection decisions in a way that should reduce overall system development costs by reducing architecture-selection-risk. It is the early characterization of the system's performance that is the focus of this research effort. Since there are many uncertainties associated with the development, having a structured process to manage these uncertainties would be beneficial.

At architecture-design time there is no way to determine what the actual implementation will be. There is also no way to tell how well the actual

implementation will perform. Hence, a probabilistic approach is appropriate. The assertion made here is that the performance of the "to be" instantiated system can be viewed as a random variable which is characterized by a performance probability density function (pdf). This random variable can be thought of as the rule for mapping an event (architecture selected) to a number (the actual performance of system built constrained by the architecture selected). This probabilistic approach allows development planners to include not only design performance uncertainties, but uncertainties associated with the teams performing activities and the processes used to create and implement the designs. Taking this approach, the performance-concerned developer can reduce the risk of selecting an architecture which can not meet the desired performance specifications by assessing the probability that a typical implementation of one particular architecture will perform better than a typical implementation of another.

## **1.7 Work Focus**

This work concentrates on performance-centric development process improvements and focuses on how to improve the architecture-selection process as this will reduce costs associated with fixing performance problems that pop up later in the development that result from poor architecture selection. Any method proposed to replace existing performance evaluation techniques must have attributes that make it better than current practice. Based on a review of related

work as discussed in chapter two, the most important attributes can be enumerated in four categories.

#### 1) Earliness in the design process

Executing performance analysis earlier in the development process is considered beneficial as it reduces performance-failure risk, and can thereby reduce costs. Substantial work has already been done on evaluating designs (and hence indirectly architectures) late in the development process when code has already been delivered. Much less work has been done for the design phase and still less addresses analysis at architecture design time.

The method proposed here addresses performance-goal-achievement by managing uncertainty directly. The approach can be applied early in the development when there is less detailed information available. As the final design matures some of this uncertainty will be eliminated, while other uncertainties like system load may remain (usage of the system may change over time). As the design evolves from architectural design through detailed design, the characteristics of components and connections will be resolved and the system performance uncertainty will be reduced.

## 2) Accommodation of all available data

The methodology developed should be able to accommodate both actual performance values and performance estimates for cases where true values are unknown. Estimates are most useful before decisions have been made about the precise performance of platforms, communications paths, algorithms, etc. The process should manage both the estimated values and the value uncertainties to ensure that final performance comparisons depict meaningful performance differences. It should be sufficient to use relative performance estimates for alternatives' performance descriptions, since there can be a large number of potentially good solutions to be evaluated.

## 3) Repeatability

The process must be repeatable. The results obtained should not depend on the skills of the analyst. The results should be consistent for different executions using a particular dataset. It must produce predictable and understandable results.

## 4) Practicality

In complex systems, there are a large number of decisions that need to be made. As each decision is made, the uncertainty in system performance is reduced.

The decision making process employed must avoid introducing exhaustive state-space searches which are often not feasible [AABI00] when large numbers of options must be considered. As well, the method adopted should be practical and useful to a broad set of analysts.

## **1.8 Research Strategy**

State-of-the-art methods used for performance management might be referred to as "continuously optimistic." One follows good design guidelines in generating the solution and then checks to see how well that design performs once implemented. For software intensive systems, performance shortcomings are handled through redesign or recode and re-measurement efforts. For the broader system design problem, reconfiguration of elements and modification to the data-flow topology can provide improvements. The method developed in this research takes an "estimate before build" approach. It is patterned on the INCOSE [ICSE07] and Department of Defense Systems Engineering processes [DODI08] but modifies certain activities to account for the estimation and management of uncertainties throughout the development. While performance budgeting can be done either top-down or bottom-up, the method developed as part of this research uses the bottom-up approach since those mathematical tools are more tractable.

Conceptually the process is simple. In all of these approaches, top level system requirements are defined and then the budgets for critical quantities are allocated to subsystems. In a weapons system development, the quantities budgeted might include system weight, power, heat dissipation, etc. In the method described in this research, it is delay that is allocated across the architectural components and connections of each of the alternative solutions. This allocation is done in a hierarchical manner. At any particular level these components may in turn be composed of sub-elements. When that is the case, performance budgets are again sub-allocated to these smaller elements. This process continues until all elements have been assigned a performance allocation. After each allocation, the provided techniques are used to combine assigned budgets along with their uncertainties to ensure end-to-end or total system performance meets the goal. At the lowest level, individual components are built or selected to meet these derived performance budgets. There are two difficulties: 1) there is more than one way to split up a performance budget across a set of elements, and 2) in many developments, some components may already exist, and the performance of the top-down approach needs to end up with a budgeted value that matches the performance values of those existing components when the analysis is complete.

An alternative approach using the same tools is possible. The performance of the lowest elements in the architecture design can be characterized and the



performance of the overall system built up incrementally from such descriptions. When performance is estimated this way, one does not know in advance whether that the requested system requirements will be met.

Both of these sets of problems can be solved by redefining the problem to be one based on anticipating performance differences between architecture options. My research extends current performance evaluation methods and concepts by considering the uncertainties in performance estimates of components and connections and provides methods of combining these uncertainty values into an overall, end-to-end estimate of system performance in a relative way so that comparisons can be made between alternatives.

The management of uncertainties when done as proposed can further assist the developer in analyzing implementation costs (work that is outside the scope of this effort). The Bayes Criteria [MeCo78] describes how to combine alternative cost with the probabilities of selecting those alternatives. A Bayes approach could leverage the probabilistic descriptions developed here and provide a basis for future work directed at further reducing the cost of system developments by balancing performance risk against cost.

## 1.9 Definitions

There are multiple definitions of architecture and performance in common use as related to analysis in this field. These terms as used for this research are defined next.

*Architecture.* A system architecture is classically defined as the enumerated set of entities (that perform system functions), how they are connected, and how they evolve with time [DODA97]. It routinely contains at a minimum, identification of the components or large-grain pieces of the system, as well as descriptions of the connections between those pieces. The quality of an architecture is directly related to its ability to constrain eventual implementation options to ones that are likely to meet both the desired quality goals and the functional requirements. Architectures are intended to maximize the flexibility of developers to make lower level design decisions while ensuring that these developers can meet system quality and functional requirements. This view is consistent with many others like [FrBo98] [UcYa00] which identify components, connectors, and ports as key features.

*Performance.* Performance is defined to be the number of events of interest that occur per unit of time. It is either the rate at which a set of activities is completed, or the time required to complete a specific activity. This intentionally broad definition allows us to qualify the performance of a system differently depending

on the types of information that may be available to characterize the system elements and the related uncertainties associated with activity performance descriptions.

Since much of this work focuses on probabilistic results and the manipulation of functions of random variables, the following definitions are provided. While these terms have other meanings in the computer science field, they are used in the following context with these definitions.

*Experiment.* An experiment is an activity that produces an identifiable outcome. Common experiments include the roll of a die, the toss of a dart, etc. In this context, the experiment is routinely the selection of an architecture. The associated result or outcome is the performance value that this selected architecture will have based on an input selected from the feasible system inputs. For this work, the experiment or the space of all outcomes is the closed interval of the real line  $[0, X]$ , where  $X$  is some large rational number.

*Event.* An event is a collection of outcomes to which we can assign probabilities. The set of events constitutes a mathematical field. In the context of this work, all "sensible" outcomes will qualify as events [Papo65]. There exist several technical restrictions on what can be an event. These restrictions are not listed here since for architecture selection purposes they are not important.

## **1.10 Research Approach**

This dissertation provides a method for improving the system development process by identifying a technique for making quantitative comparative performance assessments of alternative system architectures early in the development life cycle.

The problem statement for this effort is:

There are many methods that can be used to evaluate system performance. Most are applied to systems that are either at least partially implemented or where detailed component and connector performance information has been clearly defined. Approaches based on queuing simulations produce precise results where the detailed design factors are known. When design parameter values are not clearly established however, a queuing approach usually has to resort to checking combinations of sets of possible values. This assures that existing uncertainty can be reflected in the end-to-end performance estimates which will vary depending on the outcomes future design decisions. Examining large numbers of cases takes time. There aren't generally accepted techniques that can directly manage the performance uncertainty that exists at architecture selection time. Statistical techniques exist that could be used to reduce the

number of solution options to be analyzed and as such should reduce the time required for analysis before the architecture selection is made.

The thesis statement for this research is:

Consider a set of proposed system architecture options (architectural designs) and a probabilistic description of the performance of the architectural elements that make up those designs. For each ordered pair of architectural options in that set, one can calculate an estimate of the probability that the first option will perform better than the second option should both options be implemented.

In executing this research, the following activities were conducted:

- Five fundamental functions were identified for use in architecture performance modeling.
- Algorithms were implemented to evaluate each of these five functions for quasi-arbitrary descriptions of performance uncertainties.

- A method was developed to generate a probabilistic description of the performance of an architecture based on combining the descriptions of the constituent elements using the five functions.
- A method was derived to compute the probability that one particular architectural description will produce an implementation which will perform better than a similar implementation of an alternative architecture.
- The method was applied to a small but realistic three-tier information management example problem.
- The method was applied to two more complex architecture examples to demonstrate the analysis required to identify and characterize architecture component and connection uncertainties.
- Discrete element simulations were run on connection and processing elements. These simulations established that one can estimate an element's mean performance consistent with theory. It further established that one can estimate the element performance uncertainty around the mean. This second descriptive characterization is not routinely available through analytic-closed-form efforts.

### 1.11 Assumptions

The analysis process is based on a number of assumptions:

- 1) Components – component performance does not vary over time, and the conversion of data from input to output can be characterized by a single probability density function.
- 2) Connectors – the delay of connectors is approximated by the quotient of the probability density function for the size of transmitted elements divided by the probabilistic description of the link data rate.
- 3) Contention for resources can be simulated by subtracting resources used by other tasks (both computational and transmission) from those available for the task under analysis.
- 4) Congestion/bottlenecks - . When network congestion or bottlenecks can be anticipated, they are included in the low level performance descriptions of the connectors. When they are not or cannot be anticipated, they will not be reflected in the performance characterizations.

## **1.12 Organization**

Chapter one presented the context of the problem and reason for addressing this research goal as well as the research approach. Chapter two surveys related work in this area. Chapter three considers the architecture selection problem as a type of probabilistic experiment. Uncertainties to be expected are identified and the underlying performance design problem is characterized. Chapter four assumes that the uncertainties of the architecture selection process can be characterized as a probability density function. It then explains how to compare architecture alternatives in terms of their likelihood of performing better than other alternatives. Chapter five explains how to combine the probability density functions (pdfs) describing the performance of subsystems into a probability density function which describes the performance of the entire architecture alternative. Chapter six uses a classic three tier database problem as an example of the techniques described in chapters four and five. Chapter seven takes two more realistic problems and applies the described techniques to identify which of several alternative architectures perform better. Chapter eight describes the CAPE tool built as part of this research. Chapter 9 describes the validation methods used and a preliminary error bound for the architecture comparison metric. Chapter ten identifies the research contributions and future research directions.



## **Chapter 2**

### **Related Work**

This system performance research effort is largely built upon established results in four areas: foundational work in software architecture, architecture description languages, performance evaluation requirements and performance evaluation analysis. Much of the work discussed here provided the rationale for the approach taken in this research effort. Other portions of the work discussed are indicative of the amount of active research ongoing with regard to performance evaluation (much of it architecture related) as well as suggesting that the approach documented here has not already been attempted.

#### **2.1 Software Architecture Foundational Work**

A 1992 paper written by Perry and Wolf [PeWo92] began the study of software architectures as a significant factor in creating quality software. This paper addressed the elements, form and rationale for using a particular structure to develop a software capability. This is the start of the association of quality attributes (performance) with architectural elements. In the mid-1990s, papers on the topic were published in numbers. The theme of these papers was that software projects have architectures, either generated intentionally or evolved

from the detailed system design, and that architectures can be categorized into a relatively small number of styles [BaCK98]. Some researchers formalized the description of those architectural styles [AlGa94] [GaPZ94]. These styles provide a categorization scheme for identifying useful elements when considering performance contributions. Shortly thereafter Garlan and Shaw published a seminal work [GaSh96] which began to enumerate software architectural styles and the quality attributes of those styles. The identification or classification of architecture styles is important because it provides a fundamental way to look at the elements of a software system and hence points to the fundamental entities that must be considered when performance is an important consideration. Substantial analysis continues as investigators study the relationship between architectural styles and software quality. Much of the work that followed focused on Architecture Description Languages in an attempt to make the definition and manipulation of architectures more precise.

## **2.2 Architecture Description Languages**

Precise descriptions of architectures, or perhaps more accurately the fact that architectural descriptions are currently not precise with respect to performance concerns, is critical to establishing an appropriate method for analysis. Efforts were made to increase the precision of the architectural descriptions. The concept of an Architectural Description Language (ADL) was developed to provide a basis for capturing the essential elements and relationships between

those elements in formal languages to express software architectural concepts and structures. The role of the ADL is to express the software structure in a notation appropriate for manipulation by supporting tools. These languages focus on the large-scale or high-level design and provide developers and designers with an analysis-appropriate description of the system under study. Their value rests in their ability to clearly define relationships between entities and to provide a common understanding among readers of the underlying "structure" of the concept to be presented. The languages make it possible to explore the functional properties of architectures and provide a means to concisely define the characteristics of system elements in a precise and expressive manner.

There are a number of ADLs, each focusing on a slightly different aspect of overarching architectural concerns. A survey of architectural description languages is covered in [Clem96]. Some of the more well-known languages include: ACME [GaMW97], Aesop [GaAO94] [Gar95], C2 [MORT96], Gestalt, LILEANNA [Trac93], MetaH [Vest96], SADL [MoRi97], UniCon [SDK+95], Weaves [GoRa91], Regis [MaDK94], Rapide [LKA+95] [LuVe95], Darwin [Darw94], and Wright [GaSh96] [Alle97]. Some, like Rapide, are executable, while most are not. Nearly all include analysis capabilities addressing specific architectural concerns. None of these ADLs specifically addresses performance issues, although each reference gives insight into places where performance

related information could be inserted into a specification so that they would be more useful in studying performance issues.

### **2.3 System Performance Evaluation**

The earliest work on performance associated with software architectures focused on the large grain issues relating to interoperability, the matching of interfaces and the extent to which systems composed of parts could function properly. A paper by Garlan, Allen and Ockerbloom [GaAO95] is characteristic and addresses the impact of architectural considerations on system performance in a fundamental way. Their analysis focuses on the types of mismatches that can occur between components and connectors and how these mismatches can remain undiscovered until well into the development life cycle. Their technique concentrates on identifying problems, (i.e., assumptions about component characteristics, connectors, and the global architectural structure) before they become part of the design. The authors discuss methods to avoid making improper assumptions about these architectural elements. They provide insight into early interface indicators that may warn of failure when the elements are assembled. Unfortunately, the paper does not address a specific methodology for correcting these types of deficiencies. There are many other papers of this type. They generally avoid addressing performance in terms of metrics or in a numerically quantifiable way. Some efforts continue to analyze potential

interface mismatches and other model checking techniques in work done by Campos and Merseguer [CaMe06].

In 2004, Balsamo et. al. summarized the state of performance work in published approaches focusing on three evaluation factors: earliness of integration, model used for performance estimation and the amount of automation available. Performance work had progressed from identifying failures to the actual prediction of performance indices such as response time, throughput and utilization. For my research, the first two are of most interest. The paper [BaMI04] exposes the principal research techniques, and suggests that the benefits of probabilistic analysis show promise. The survey divides the work into models and evaluation methods. The models fell largely into four categories: queuing networks, stochastic Petri nets, stochastic process algebra, and simulation models. Evaluation methods were divided into analytical techniques and simulation.

### **2.3.1 Queuing Networks**

Queuing networks characterize the proposed solution of a number of researchers in this area. In early work, CHAM (CHemical Abstract Machine) [BaIM98] [CoIW99] employs an N-layer Queuing Network Model and was used to estimate performance based on a network automatically derived from an architecture description language. The ATLAS Transformation Language can be used to

automatically map UML models into queuing networks [CoGM08]. With the growing acceptance of UML, recent efforts started to focus on the mapping of Message Sequence Chart descriptions of architectures to queuing networks [AABI00]. This work is one of the reasons for suggesting that UML sequence diagrams could be effective in a probabilistic analysis. Similarly, UML diagrams were automatically and consistently translated into queuing-network-based performance models in [CoMi00] and [Ambr05]. [BaMa05] started with annotated UML use cases, activity and deployment diagrams to create performance models based on multi-chain and multi-class Queuing Networks (QN) annotated according to the UML Profile for Schedulability, Performance and Time Specification. Other attempts have been made to use this specification to construct models to make quantitative predictions regarding performance characteristics. These efforts demonstrated generating Layered Queuing Network (LQN) performance models based on a graph-grammar and a UML model annotated with performance information [PeSh02]. An XSLT approach to transform UML artifacts into LQNs is described in [GuPe02]. Some have attempted to move the evaluation of performance even earlier than architecture-selection time back to scenario definition time. This is the case with the Scenario to Performance (S2P) algorithm described in [PeWo05] which transforms scenario models automatically into performance models and uses the LQNGenerator tool to build layered queuing network performance models.

### 2.3.2 Petri Nets

Petri nets come in a number of forms and can be used for modeling and evaluating software architecture performance. Colored Petri Nets are used in [XuKu98] to model a mobile-phone-family execution architecture and characterize both the time and space performance of a large software intensive system. Interval Time Petri Nets (Petri Nets that have random firing time intervals) are used for a similar purpose in [NaBG09]. PEPA nets, a type of colored stochastic Petri nets, are used with stochastic process algebras to derive steady state performance measures in [GHKR04] and in [CGHT99]. As part of a larger security analysis effort, Generalized Stochastic Petri Nets (GSPNs) are used in [CoTr08] to inform on the performance impacts of security related architectural decisions. Additionally, [Cort05] discusses the need for performance anthologies based on evidence from operational profiles (annotating performance parameters onto software models) and their translation into Stochastic Petri Nets to quantify performance. Several UML diagram types provide information for the Software Performance Engineering process which is capable of generating GSPNs as described in [LoMC04]. Similarly state charts and sequence diagrams provide the input to automatically generated GSPNs in [BeDM02]. Analysis using GSPNs to generate worst and best case system performance is discussed in [NaBG09].

### **2.3.3 Stochastic Process Algebras**

Stochastic process algebras provide another strategy for addressing the performance estimation task. In [LTK+02], UML diagrams provide information to characterize a generalized semi-Markov process used to do performance analysis. They are also used in TIPPTool [HHK+00] to address compositional performance modeling. The architecture description language AEMPA characterizes the syntax and semantics of architecture descriptions in [BeCD00] and is used for performance analysis.

### **2.3.4 Simulation Models**

Simulation models are the basis of a number of different approaches that do not fit neatly into the above categories. Initially mean-value-analysis was used in performance modeling of combined computer-communication systems. Later, simulations were used to generate performance intervals which better characterized systems than these single mean value numbers [LuHa98]. A simulation environment called Arena has been used with UML data to simulate performance in a meta-modeling environment [TKMG08]. In [SaJP05] the simulation output is used to inform where improvements can be made in the architecture. [BeKR07] addresses a method to parameterize architectural features using the Palladio Component Model, a domain specific modeling language, to gain insight into performance. Activity diagrams are simulated to characterize performance with both exponentially distributed and deterministic



delays in [LTK+02]. Indicative of efforts considering hardware parameters before design and relating them to performance expectations is [Alsa04a] where disk and CPU delays drive architectural analysis. New UML stereotypes are introduced in [AmCI01] where use case, activity and deployment diagrams representing performance, complete the architecture performance description. An algorithm for deriving a simulation model from an annotated UML software architecture is explained in [BaMa03]. Sequence diagrams and state charts are used to represent execution paths in [BeDM02] where performance estimates are derived.

### **2.3.5 Combined Models**

Combining models is common as well. [BaBS02] uses Stochastic Process Algebras and Queuing Networks to leverage formal techniques and verify functional properties. A component-based predictive analysis built on stochastic process algebras and simulation forms a system called NICE that addresses communications performance concerns [BMMI04].

### **2.3.6 Component-based Architectures**

Component-based architectures are often handled a bit differently. With components, the estimation problem may be simpler particularly if there is some experimental information available. A specification might already exist for estimating performance even before components have been built. This need to

determine or estimate the performance of elements is consistent with the information needs of this research's proposed approach. Establishment of a set of scenarios based on Use Cases to enumerate the sequence of tasks or component responsibilities to be performed by the system is covered in [PIFa02]. Once element characteristics are established, the challenge shifts to one of combining results to describe a complete system estimate [GrMi04]. Performance estimation with components can be difficult since systems already built on component technology [CLGL05] have existing interfaces. When these interfaces do not match parts of the business process, performance degradations often result. Components routinely have interfaces that reflect their underlying function; changes in processes will cause changes in component coordination and performance. How to manage the expected estimate changes that crop up from a substitution of elements is described in [AaHT02]. The importance of workload characteristics and service demand is discussed using an XML Component-Based Modeling Language in the analysis presented in [WuXi04]. Executable models are described in [WHKT08] where the authors identify a set of architectures with a component-based modeling language called COLA. Component based design benchmarking and profiling are explained in [CLGL05] which includes case studies in CORBA and J2EE. Combining RT-UML and CB-SPE (an automated approach for composing elements that addresses performance engineering) [BeMi04], provides insight into methods for parameterizing performance as well as combining elements. COTS-Aware

Requirements Engineering and Software Architecting, (CARE/SA) [ChCo05] is introduced as a method to iteratively match, rank, and select COTS components and subsequently aggregate their functional and non-functional performance into an architecture. Other informative but more generic approaches are included in [BFG+04] and [CoTr08] which address using patterns in a building block approach to give insight into architecture performance.

Once a model has been established, there remains the need to populate that model with information tailoring it to the specific architecture being evaluated. A number of contributions can assist in this process. Model information exchange formats have been designed to facilitate both the flexibility of bringing different models together, as well as helping to define the minimal set of characterizing parameters that are needed. A useful exchange schema is discussed in [MoSW08] and descriptions of XML interchange formats are discussed in [SLC+05]. These approaches suggest that, in the future, performance evaluation tools may be combined in a plug-and-play manner. Others have extended architecture description languages to include the ability to maintain performance data. A review of different notations or languages useful for representing performance characteristics exists in [CoMI03]. A number of UML extensions have been proposed: Performance Aware Software Development (PASD) includes information on resource demand budgets. Others use UML 2.0 sequence diagrams and the newer structured control constructs, like

CombinedFragments (an expression for sets of interaction fragments), express concurrency [ShVN08] or use the composite structure diagram [PSG+09]. State charts are refined in [LoMC04] and UML 2.0 versions of activity diagrams are demonstrated in [CGH+04]. Service composition processes are described using Business Process Execution Language for Web Services in [AmBo07].

### **2.3.7 Architecture Design Domain Specific Considerations**

The Earth Observing System (EOS) Data and Information System (EOSDIS) is a geographically-distributed, large-scale, data-intensive system in [GoMK95]. EOSDIS has been used as an example to describe the challenges of satisfying functional and performance requirements. A queuing network model approach was taken as the basis for doing architecture performance design of large-scale distributed data intensive information systems. Focusing on meeting performance requirements in the architecture design of distributed client/server applications, [MeGo00] advocates integrating design and performance modeling activities using software performance engineering language. Component service demand parameters are derived by the compiler for this language from the system specification. The work shows results from using this analysis technique and shows how performance concerns can be informed by this type of design analysis. The design of reusable component interconnection in client/server systems is investigated in [GoMS01]. Component interconnection patterns describe synchronous, asynchronous, and brokered communication between

client and server using Unified Modeling Language (UML). The approach demonstrates the benefits of understanding component interaction patterns early in the design. Alternative client server communications strategies are described using Unified Modeling Language (UML) for reusable component interconnection in client/server systems. Use of these interconnection patterns in [GoMe00] shows the design benefit when implementing new distributed applications. [PeGo04] applies colored Petri net (CPN) templates and UML artifacts to concurrent software architectures behavioral modeling and other concurrent objects in [PeGo06]. They consider dynamic object-oriented architectures and describe the functional flow used to analyze the architecture's concurrent behavior. They show how design quality can be improved prior to implementation based on the behavioral properties. Real-time, reactive concurrent architecture designs are considered in [PeGo06]. The approach presented addresses concurrent object-oriented software designs mapped into stereotyped UML diagrams and transformed into reusable colored Petri net (CPN) templates. These CPNs are used to model then analyze behavioral properties of the software architecture. A Self-Architecting Software System called SASSY is a framework that helps domain experts take a system requirements specification and generates a corresponding base architecture. From this base architecture, SASSY derives another architecture that optimizes a multidimensional system utility function based on alternative QoS metrics

[MEG+10]. Problems and solutions for estimating missing service demand and other parameters needed for queuing models is covered in [Mena08].

## 2.4 Other Domain Specific Concerns

Finally, there is a significant body of work related to performance in domain-specific architectures. This work on general analysis techniques, benefits from extending some domain specific methods currently in use. Some of the domain specific ideas leading to these extensions are listed in Table 2.1.

Table 2.1 Performance and Domain Specific Architectures

Topic	Reference
distributed server systems	[TWG+00]
defining a performance engineering maturity model	[ScSR00]
software product lines	[TaPe08]
real time and embedded systems	[TrGi08]
reliability estimation	[ShTr05]
performance requirements	[HoWi07]
static concurrency analysis	[NACO97]
megaservices - large-scale applications	[LiLW02]
real-time telecommunications applications	[LuJR98]

Table 2.1 Performance and Domain Specific Architectures continued

Topic	Reference
RT-UML	[GrMi04]
managing quality attributes	[GrBo98]
real-time systems and work-flow	[EsWi01]
distributed and parallel computing	[FaSS00]
hosting centers or "data centers"	[DeAM08]
Java 2 Enterprise Edition (J2EE) or the Common Object Request Broker Architecture (CORBA)	[DePE04]
recording of performance data system features and application details	[Verl06]
resource-sharing systems as in large-scale computers with client-server architectures	[HeHK02]
testing large-scale multi-tiered collaboration products	[GuSh08]

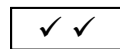
## 2.5 Identifying Good Attributes for a Performance Analysis Approach

Table 2.2 clarifies the performance attribute categories previously identified and list many of the specific desirable traits that would characterize a good performance analysis process. The topics listed in Table 2.2 were generated during the review of the literature done for this work. The identified references are the sources that came to light during this research, and there is no attempt

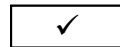
made here to identify them as either the first or only source of each idea. The research developed in this effort attempts to incorporate at least the spirit if not the actual content of each of these desirable attributes.

In Table 2.2, the “Achieved” column represents methodology attribute attainment:

#### Attribute attainment in my work:



well addressed here



more could be done in the future



only partially addressed here



not addressed

Table 2.2 Desirable approach attributes

Achieved	Desirable Attribute	Reference
✓ ✓	address the architectural level system description	[ShJT05]
✓	reduce complexity to realistic levels	[AABI00]
✓ ✓	identify and characterize workload data	[AKLW02]
✓	avoid building models "by hand"	[GuPe05]
✓	provide the benefits of executable UML designs with supplemental data for further analysis	[TKMG08]
✓ ✓	expose the benefits of difficult mathematical constructs without demanding study in these areas	[PSG+09]



Table 2.2 Desirable approach attributes continued

Achieved	Desirable Attribute	Reference
✓	bridge the gap between current UML-based tools and analysis techniques already established	[GuPe05]
✓	include information additional to that which is routinely expressed in UML architecture artifacts	[Hoeb00]
✓	use scenarios to characterize the environment of a system	[PeWo02]
✓	augment data processing characterizations to model both communications and processing delays	[Verd07]
✓	expose performance properties in a formal framework	[BeCD00]
	use automated transformations of existing description techniques	[CoGM08]
✓	address granular quantities like response times, throughput, scalability , capacity, etc	[DePE04]
✓ ✓	parameterize loading factors which influence performance like workload variability	[lyRo02]
	addressed quality of service early in the stages of development	[BaMa03]
	be able to identify sub-system specification conflicts in component interactions	[CoIW99]
✓	be executable by non-experts in performance analysis	[PSG+09]
✓	architectural decisions should inform proposed solutions to meet user performance requirements	[BFG+04]
	be compatible or interoperable with existing software development standards	[AABI00]

Table 2.2 Desirable approach attributes continued

Achieved	Desirable Attribute	Reference
	leverage available development artifacts routinely generated in a structured development process	[CoMi00] [PaSH01]
	expose architectural mismatches	[BaBS02]

## 2.6 Summary

This section reviewed the Architecture Description Language foundations of architecture analysis. These formal methods for describing architectures bring increased clarity to the design process, however there is no ADL available today which provides for the evaluation of performance. Many tools do exist for the formulation of performance estimates later in the design process. These include but are not limited to: queuing networks, petri nets, stochastic process algebras, simulation models, combined models, as well as component-based architecture considerations. The chapter concludes with descriptions of references addressing domain specific architecture performance issues, and a listing of the desirable attributes that a method or process should support to effectively evaluate performance at architecture design time.

## **Chapter 3**

### **Architecture Performance Viewed as a Specific Experiment**

#### **3.1 Introduction**

System developments are generally executed to solve a specific problem or class of problems. Architectures generated early in the system development process are established to constrain the proposed solution space of the problem being solved. One of the benefits of selecting an architecture is that it can systematically rule-out undesirable solutions while at the same time, it can allow developers maximal flexibility in defining the implementation. The goal of this research is to develop a method for making performance-based architecture selections. Given a set of architecture alternatives, the objective is to select the architecture which will most likely yield the best performing implementation.

At architecture-selection time there are significant uncertainties. These uncertainties are caused by unknown factors and influences. They characterize the development decisions yet to be made for the design and implementation of the proposed solution. The method developed in this research identifies, estimates and manages these uncertainties to inform the architecture-selection decision. In the sections that follow, the term “performance description” will be

used to describe the manner in which performance uncertainties are described. A performance description will always have the form of a probability density function that states the delay characteristics that a system, architecture or implementation will be expected to exhibit. It quantifies the performance uncertainties of the entity under discussion.

The process developed in this research is accomplished in four steps: 1) identify the uncertainties associated with design and implementation, 2) quantify these uncertainties, 3) show how to combine these uncertainties into a total or end-to-end performance description, and 4) use the end-to-end performance description to compare the performance of architecture alternatives. This chapter identifies performance uncertainty sources and identifies some factors useful in characterizing them. Chapter four shows how to compare architecture-performance descriptions once they have been built. Chapter five shows how to combine the performance descriptions of components and connectors into higher level performance descriptions.

### **3.2 Early Performance Estimation Methodology**

As listed above, uncertainty management embodies four steps. In step one, the significant architectural elements (computing nodes and communications links) of each of the alternative system architectures are identified. This may be done by describing each architectural option as a UML diagram. UML collaboration

diagrams and sequence diagrams are effective in identifying the processing steps of a solution, but their use is not a requirement for applying this methodology. What is important is to generate a list all intermediate products and their locations. With this list, map the product-locations pairs into graph nodes, and transitions between product-location pairs into graph edges representing communications steps.

The second step characterizes the system input using probability density functions. It establishes the probability that an input of a particular size will be presented to the system. It characterizes both the computational nodes and communications edges of the graph as probability density functions. Node descriptions associate delay probabilities with times for computing a product. Edge descriptions associate delay probabilities with transferring information.

Step three combines component and connector performance descriptions into system performance descriptions by searching the graph to identify all paths from the initial product state to the final product state. Each of these paths is evaluated (using the performance descriptions established in step two and the techniques identified in chapter five) to obtain a probability density function that characterizes each architecture option performance.

Step four makes performance comparisons between the architectural options by evaluating the performance probability integral (PPI) as described in chapter four

to calculate the probability that any one architectural option will perform better than any other. The architecture selection decision follows directly from this calculation. It is based on a combination of two factors that go into selecting the desired architectural alternative: 1) highest probability of performing better, and 2) the spread of performance that is predicted. The "best performing" option is identified after evaluation of the performance probability integral. The spread of possible performance estimates is determined by examining the probabilistic descriptions that result after component contributions have been calculated.

### **3.3 Defining an Experiment**

The uncertainties associated with architectures and the early design decisions that implement them are not routinely considered in the context of randomness. After all, the design process is intended to be structured and repeatable. Yet at some level, since many of the factors that go into making those decisions are not known and the outcome of the decisions are unknown too, it can be useful think of each design decision as the outcome or result of running a probabilistic experiment. In fact in this context, it can be useful to think of the architecture's implementation as being the random selection of one specific implementation from all those which are achievable given the architecture description.

At a high level, let an urn represent a candidate architecture and put balls in that urn to represent the possible performance values that an implementation built

from that architecture might have. In the context of Figure 1.2, each urn of balls represents an architecture, a sub-tree rooted at the "System Specification and Design." Each leaf in that sub-tree represents a single implementation of that architecture. After populating the urn this way, one could generate a performance histogram of the counts of balls that fall into particular performance ranges. Such a histogram would show the relative likelihood of any particular performance level being attained. If one makes the performance bins sufficiently narrow, and scales the graph appropriately, the result can be mapped directly to a probability density function describing the likelihood of an architecture producing an implementation that will perform its task in a specified time.

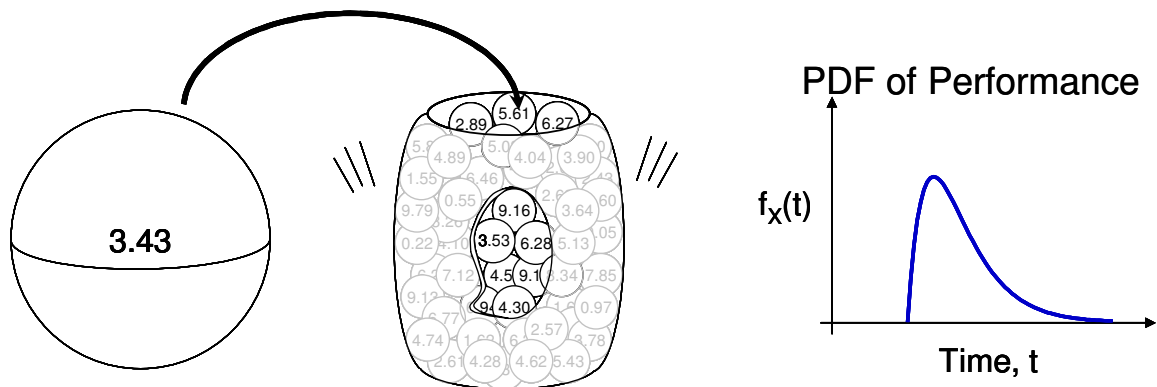


Figure 3.1 Visual description of the experiment

Consider the simplest case first. When there are just two architectures being considered, the experiment is modeled with just two urns, each holding a set of

balls. Each urn is labeled with the name of the architecture it represents. One would expect that each implementation could process a number of different input-data-sets. Since changing the input data results in different software execution paths being executed, one expects that the performance of a particular implementation depends on the input data that is applied. For each input-data-set implementation pair, construct a ball with the performance value written on the outside and add it to the urn. Although not practical, conceptually generate balls for all performance values for all of the possible input data sets that could be applied to each implementation of the architecture. This will generate a large number of balls for each urn since there are many combinations of input data sets, detailed designs, and capability ranges of implementers, etc.

Now define the architecture-selection decision as the following experiment. Randomly choose one ball from each of the two urns. Compare the two performance numbers (on the outside of each ball) and record which urn (architecture alternative) had the ball (implementation) with the smaller performance number, i.e., the winner. Replace each ball in its respective urn and shake up the urns. Repeat this process many times recording the winner each time. From this table of winners one can determine the likelihood that a particular architectural choice will perform better than its architectural alternative.



The relative frequency that each architectural choice is expected to perform with some specific amount of delay can be mapped into a probability density function (pdf). One can then evaluate which architecture of the two has the highest probability of performing better, with regard to the performance assumptions made, by using the techniques of chapter four.

### **3.4 Performance Estimation Fundamentals**

The experiment just described helps show how information known about the implementation strategy and the types of data to be processed by the system can influence a performance-based architecture selection. The approach can be made mathematically tractable. To be useful, the sources of uncertainty need to be examined and a method needs to be identified to develop the probability density functions that describe these uncertainties. This mapping of uncertainty to pdfs is routinely situation specific. The uncertainty model documented in section 3.5 can be used as a guide for finding sources of uncertainty and characterizing them.

Sources of performance uncertainty (the uncertainty in time associated with executing a computational activity) exist at every level of processing. At the hardware level, processor instructions often have data dependent execute cycle counts [ARM-05] as do assembly language implementations of “multiply” functions. Algorithm selection, data to be processed and initial algorithm starting

point can all affect delay by forcing the execution of alternative branching conditions. Even availability of resources can be uncertain when competing services make demands on the underlying hardware elements. Detailed examples of sources of uncertainty can be found in Appendix B.

### **3.5 Identifying Uncertainty Sources**

Once understood, each uncertainty contribution can be converted into a probabilistic performance description. This section describes six significant uncertainty sources routinely encountered during the system development process. These performance descriptions (in the form of probability density functions) can be combined to estimate the overall performance uncertainty of a likely implementation. Chapter five covers the operators needed to combine these descriptions into the system performance description. Since uncertainty contributions may change as the development evolves, the mapping of performance uncertainty to pdf should be done several times throughout the design process.

#### **3.5.1 Value or Data Based Performance Uncertainties**

Even after the detailed design is complete, there is uncertainty in the amount of time it takes to perform a computation. The time that it takes to execute any particular function or computation can be computed by combining three characterizing aspects: 1) the steps required to perform the function, 2) the

number of times that each such step is executed, and 3) the time that it takes to execute each step once. Knowing these three quantities, one can estimate the duration of the computation. For different instances of the computation however, these three quantities may have different values. Each may vary based on the data presented. As a result, each of these three quantities should be characterized in a probabilistic manner so that the calculation duration can be estimated over a range of expected input values and operating conditions. This characterization leads to the function execution duration being described by a probability density function that relates the function execution duration to the probability of that duration.

### **3.5.2 Algorithm-Based Performance Uncertainties**

The next uncertainty arises when the computational strategy (the set of algorithms and the sequence of their execution) is being defined. There is usually a set of alternate solutions that can provide the desired system functional behavior. Associated with each alternate solution are a performance value and a performance uncertainty. An analysis process is routinely executed to determine the characteristics of alternatives so the superior one can be selected.

Algorithm identification is essentially the development of a sequence of steps that change inputs into outputs. These steps would include not only system input and output as in the translation of key clicks into characters and the steps required to

change postscript strings into printed matter, but all intermediate process steps. Data moves through the system and changes form. Two fundamental activities occur, changing the form of the data and transferring the data from location to location. In this move-compute-move model, the sizes of intermediate data products are important as they affect a proposed solution's performance. The size of the data divided by the effective speed of the source-to-destination connection yields the time for each communications step (transmission time). Figure 3.2 characterizes the tradeoff between the computation time to reduce product size and the transmission time to move a product.

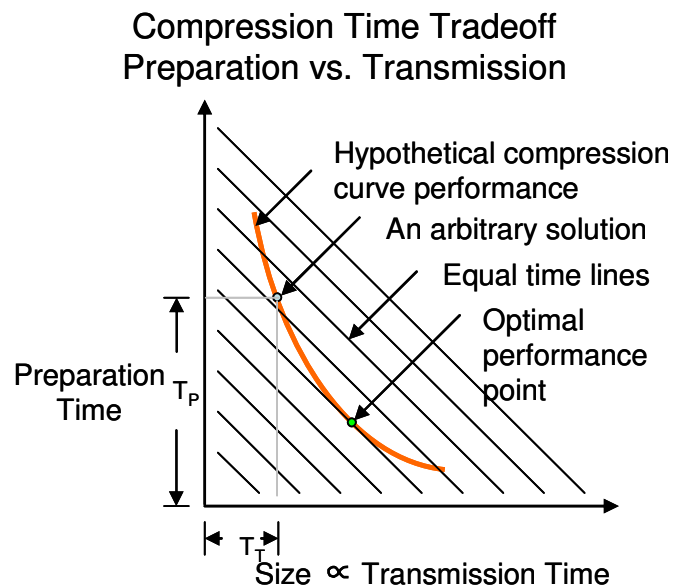


Figure 3.2 Curve characterizing processing and transmission time tradeoff

Since the goal is to evaluate relative overall system performance, the challenge is to analyze both operations (size change and location change) together without requiring absolute numbers. The connecting parameter between these two transformations is size-change-per-unit-compute-time. The fractional size-change-per-unit-of-compute-time is proportional to the effectiveness of the computing step and also represents the percent of data that no longer needs to be moved.

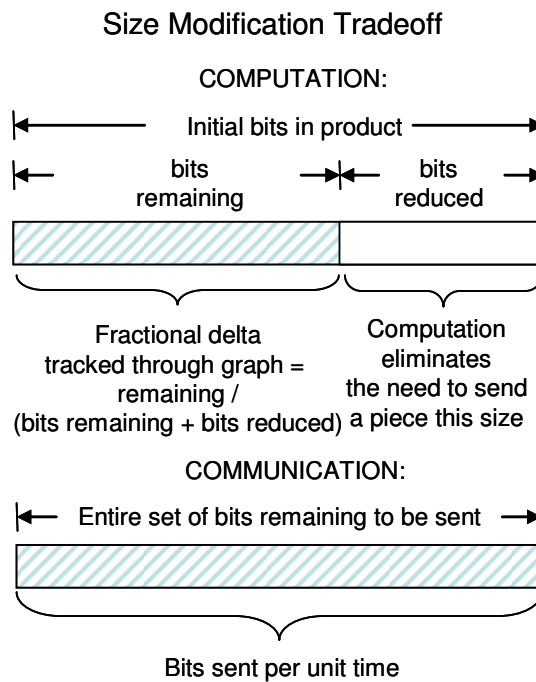


Figure 3.3 Computation and communications normalization

### **3.5.3 Topology-Based Performance Uncertainties**

Topology-based uncertainties are slightly different. This class of uncertainty addresses unknowns at a higher level in the design process where the topology or arrangement of data computations and data communications elements are being considered as they are combined into alternative data processing solutions. This uncertainty addresses the large-grain structural differences among proposed architectures. For any set of architecture options, there are two types of uncertainty to consider. The first type relates to the degree to which processing requirements are distributed across processing nodes, i.e., in what different ways can the computations required be distributed, and across which and how many processing elements. It is related to amount of parallelism that is desired in any particular computational solution. It addresses the uncertainty in the performance differences between different solutions in how the solution partitions the computation between parts that can be done in parallel and parts that must be done sequentially. The second uncertainty is similar in some ways to the algorithmic uncertainty of the last section but at a higher level. It is the uncertainty in time associated with the fact that a particular computation may take different paths through the computational nodes. It is concerned about the size of the performance differences to be experienced caused by proceeding down different paths at each decision point in the computation plan. These path decisions may be made based on either data elements associated with the actual computation, or based on characteristics of the operating environment, i.e., the

way that load is distributed based on how busy processing elements are at the time that a computation must be executed.

#### **3.5.4 Synchronization-Based Performance Uncertainties**

The next uncertainty type is the result of synchronization that is required to ensure correct computation. There are two sources. At the lowest level, there is the delay caused by constraining execution threads when accessing critical code sections. This type of performance degradation is required to ensure that multiple processes or threads do not corrupt the data used by each other during the execution of a computation. It results in making certain activities atomic. It as well causes delays as processes or threads may be required to wait for uncertain amounts of time to access a needed resource. At a higher level, synchronization addresses how computational progress is controlled ensuring that for any computational step, all of the inputs are present before a dependent output is generated. Often a computation requires a set of inputs and each of these may be generated from a different source. The computation may only proceed when all of the inputs are present. This uncertainty source accounts for differences in execution that are encountered when concurrent operations produce their results with differing delays before they approach synchronization points.

### **3.5.5 Load-Based Performance Uncertainties**

Load-based performance uncertainties are not intended to capture anecdotal evidence that suggests that some applications fail because of their success. Such situations might occur when a web application is unexpectedly successful and is deluged with requests that swamp the implementation causing long queues and unacceptable service times. Said performance uncertainty is difficult to predict, and as such difficult to estimate. Load based performance uncertainties are intended to capture the fact that systems loads can vary over a range of anticipated values. In the context of this analysis the assumption is made that the system is being designed for a specific range of loads. The additional assumption is that components and connections can be built that can exhibit the delay behaviors that are described in their performance descriptions. If the implemented component performance varies significantly from that planned, queues may grow and other system failures may occur. Those cases are not covered by this analysis. The presumption for this research is that components and connectors can be built which are consistent with the performance descriptions.

### **3.5.6 Sizing-Based Performance Uncertainties**

Sizing-based issues include two related uncertainties. The first is raw capacity. The second is available capacity.



Raw capacity addresses the actual processing capability of computational nodes and the bandwidth or data rate of the communications channels themselves. When absolute estimates are not known or cannot be projected with known uncertainty, it may be possible to pick an arbitrary computational standard, e.g., Bench Mark Units [SPEC09] and reference all computations (in a relative way) to this value. Similar estimates are applicable for available capacity.

Available capacity is a downward scaled value of raw capacity and also addresses the computational capability of nodes and link bandwidths. For communications links, it includes the performance degradation caused by errors generated during transmission as well as the consumption of resources by other channel users. Similarly, computational performance is a downward scaled "raw power" of the processor and includes the anticipated performance degradation caused by the competition for CPU resources from other processes. In both cases, processor cycles and channel bandwidth are often shared with other subsystems and only a fraction is useful to the calculation of interest. There is an uncertainty associated with the number and type of competing activities that leads to uncertainty in the availability of needed resources.

### **3.6 Summary**

This chapter begins by observing that there are many uncertainties in the development of systems, and at the beginning of the development process these

uncertainties can significantly affect performance estimation. These ideas lead directly to the benefit of characterizing the performance of the implementation of a system as a random variable, or outcome of a probabilistic experiment. The section then considers that the implementation of any actual system is the analysis and design of two types of activities: the transformation of data from one form to another and the communications of that data from one place to another. There can be design tradeoffs made between the processing time required to transform data (leading to an associated change in size) and the time required to transmit that data to another location. When this tradeoff is normalized by considering the ratio of bits-to-be-sent-reduction to compute-time-required to make that reduction, a performance characterization is possible.

The actual development of a system can be considered as an experiment since the values of the uncertainties encountered lead to a non-determinism in the outcome. These uncertainties arise from a number of sources. Some come from the fact that many computations take an amount of time that is related to the values manipulated. Others come from the fact that different algorithms have different efficiencies. Still others come from the need for computational synchronization and the sharing of resources on computing platforms and communications links. The chapter concludes with a summary of the steps needed to perform a comparative analysis of the expected performance of an implementation considering these uncertainties.

## **Chapter 4**

### **Comparing Architecture Performance Potential**

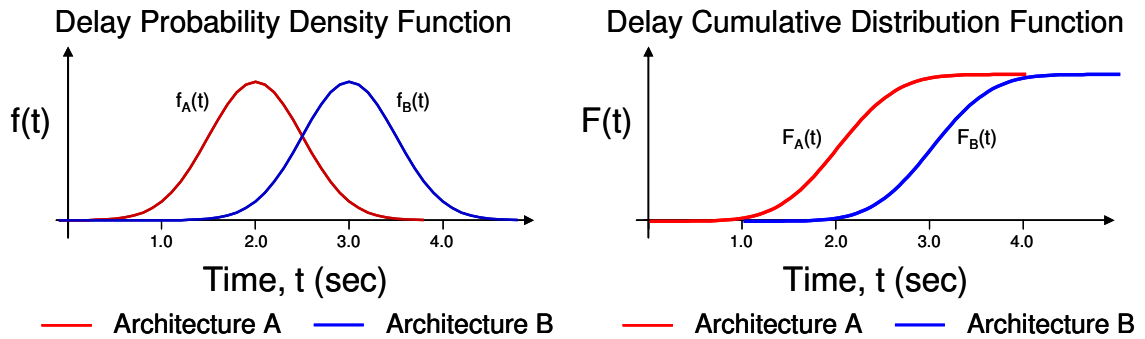
#### **4.1 Introduction**

In this chapter, assume that it is possible to create a complete performance description (in the form of a probability density function) of an architectural alternative that includes the established understanding of development uncertainties. This performance description characterizes the likely performance of the system and accounts for the cumulative affect of all that is understood about attributes of the development process (team maturity, requirements clarity, etc.) and the uncertainties associated with the proposed system implementations (input data sets allowed, algorithms to be used, system topology, etc.). This performance description can be built from performance descriptions describing subcomponents. Chapter five will discuss how to combine component and connector performance descriptions to generate a single system performance description. This system performance description characterizes the experiment that models the architecture-to-implementation performance mapping that is intended to represent the development process being used.

This chapter derives the performance probability integral (PPI) and presents several examples demonstrating its use. Applying the PPI calculation to a pair of performance descriptions yields the probability that a randomly selected implementation of one architecture will perform better than a randomly selected implementation of a second architecture. The illustrative examples in the chapter have been chosen to be simple and intuitive. A more interesting but still simple realistic example will be discussed in chapter six. Chapter seven will demonstrate the use of the PPI on two more real world examples.

## **4.2 Deriving a Performance Probability Integral**

The performance probability integral is a function that maps two performance descriptions into a real number (a probability). The first part of this section demonstrates the mechanics of making performance comparisons. The second part derives the actual integral to be evaluated. While the demonstration examples used to show the comparison mechanics assume normal distributions as argument performance descriptions, the approach is general as will be demonstrated in the verification examples that follow the integral derivation. Figure 4.1 shows performance descriptions for two architectures (left) and the cumulative distribution functions associated with these descriptions (right). Both functions will be used in the PPI derivation and explanation that follow.



Definition: The **probability density function** (PDF),  $f_X(t)$ , is the relative likelihood that system X will perform in time t.

Definition: The **cumulative distribution function** (CDF),  $F_X(t)$  is the relative likelihood that system X will perform in less than time t.

Figure 4.1 Delay density function and cumulative distribution function

Consider an architecture called “X.” Let  $f_X(t)$  be the architecture X performance description, that is the probability density function describing the relative likelihood that an implementation from architecture X will perform its task in some amount of time close to t. Let  $F_X(t)$  be a cumulative distribution function for system X. The cumulative distribution function,  $F_X(t)$ , is the probability that the system X will perform its function in *less than* time t. Density and distribution functions are related in the standard probabilistic sense. When it is clear to which system a performance description applies, or when there are more than one density function or cumulative distribution function on the axis of a graph, the subscript will be dropped unless confusion would result.

Consider two systems where the overall architecture performance description is represented by the sum of a large number of independent contributions. This situation is shown in the top left corner of Figure 4.1

The question that needs to be answered is "What is the probability that a randomly selected implementation from architecture A (red), will perform better than a randomly selected implementation of architecture B (blue)?" In this question, "better" means that a system completes its task in less time.

This calculation can be accomplished by breaking the problem into pieces. The probability that architecture A will produce an implementation that will perform better than the implementation produced from architecture B for a specific but arbitrary value  $t'$ , is the probability that the random variable representing the A implementation takes on a value less than  $t'$  while at the same time, the independently selected value representing the B implementation is greater than  $t'$ . The shaded regions of Figure 4.2 show this case. Note that the selection of the A implementation value and the selection of the B implementation value are two independent events. Since these selections are independent the probability of the combined event is just the product of the probabilities of the individual events. Expressed mathematically, this is the product of the integral over the area where the implementation from A is less than  $t'$  and the integral over the area where the implementation from B is greater than  $t'$ .

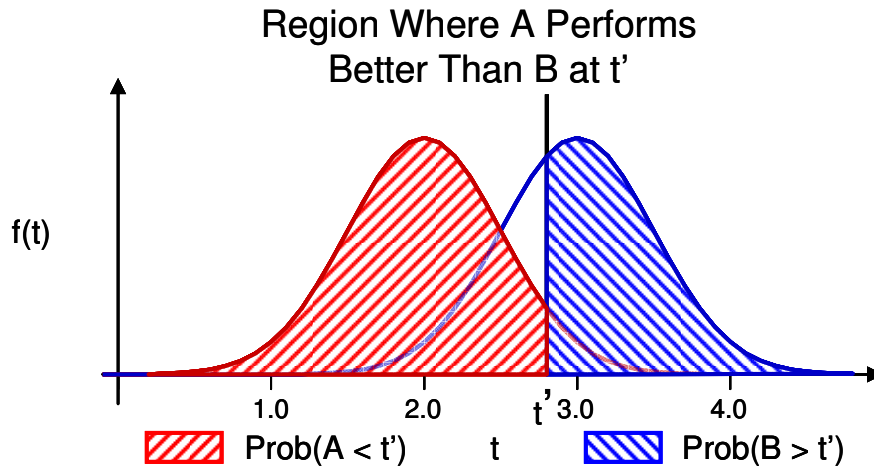


Figure 4.2 Calculating the probability System A outperforms System B

In equation form this would be:

$$P(A(t') < B(t')) = \int_{-\infty}^{t'} f_A(x) dx \cdot \int_{t'}^{\infty} f_B(x) dx$$

where  $f_A$  describes system A's performance and similarly for  $f_B$ .

Now generalize to the case where  $t'$  can take on any value. The probability of the union of mutually exclusive events is the sum of the probabilities of the individual events. Hence divide the above problem into a set of mutually exclusive events. Base the events on the cases where the random variable (representing B's performance) is to take on a value in the small interval greater than  $t'$ . Now

calculate the probability that A's performance will be less than that of B for an arbitrary but specific value of  $t = t'$  when B's performance is between  $t'$  and  $t' + \Delta t$ . Figure 4.2 shows this partition of the time axis into small time intervals,  $\Delta t$ .

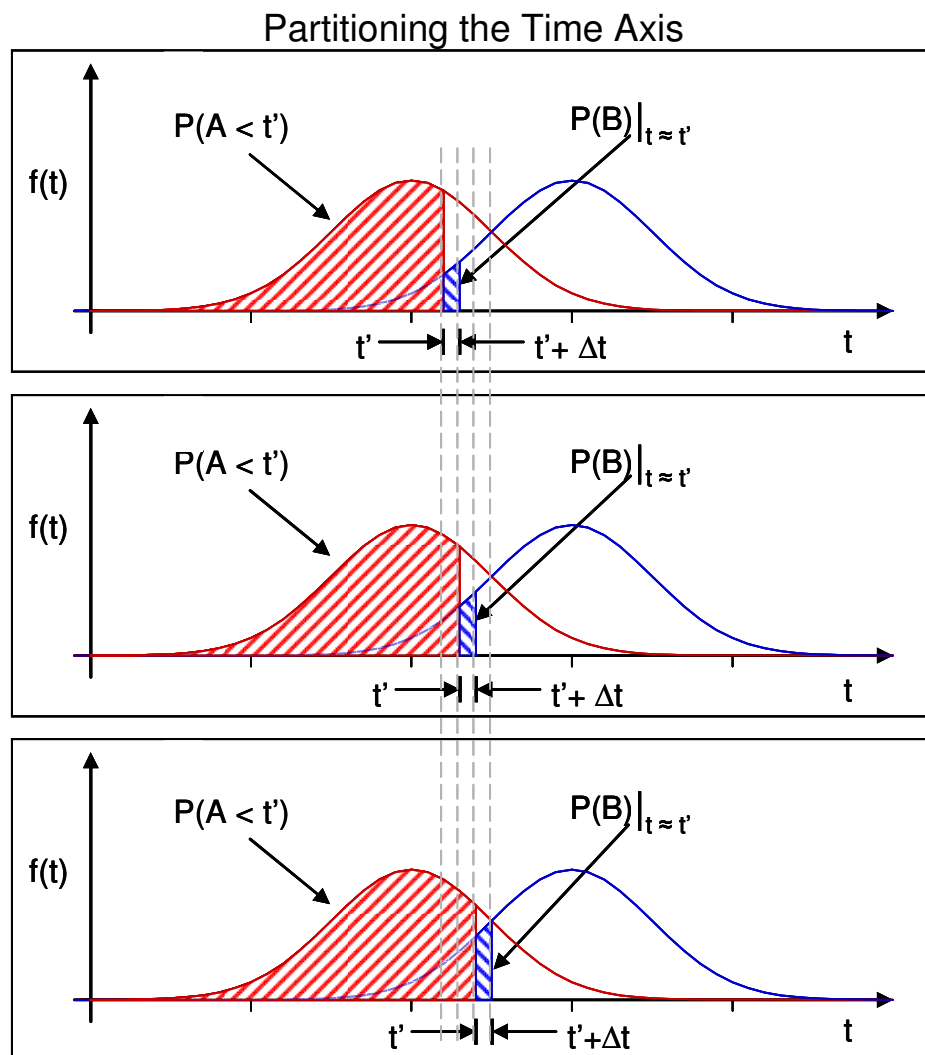


Figure 4.3 Evaluation of the  $P(A < t')$  for increasing values of  $t'$



The red shaded area is  $F_A(n\Delta t)$  and the thin slice to the right ( $\text{Prob}(A \approx t' \text{ or } t' \leq B < t' + \Delta t)$ ) is the difference between two successive cumulative distribution values for system B.

$$P(a < b) \approx \sum_{n=0}^{\infty} F_A(n\Delta t) \cdot [F_B((n+1)\Delta t) - F_B(n\Delta t)] \quad (2)$$

Figure 4.3 shows the evaluation of the  $P(A < B)$  for several successive  $\Delta t$  intervals. The summation accounts for all of the  $t'$  values since these are mutually exclusive events.

Note that the approximate probability that B will perform near  $t'$  is

$$F_B((n+1)\Delta t) - F_B(n\Delta t) = P\{(n\Delta t) < t' \leq (n+1)\Delta t\} \quad (3)$$

Substituting the right hand side of this equation into the one above yields:

$$\sum_{n=0}^{\infty} F_A(n\Delta t) \cdot [P\{(n\Delta t) < t' \leq (n+1)\Delta t\}] \quad (4)$$

Factoring out a non-zero  $\Delta t$  yields:

$$\sum_{n=0}^{\infty} F_A(n\Delta t) \cdot \left[ \frac{P \{ (n\Delta t) < t' \leq (n+1)\Delta t \}}{\Delta t} \right] \Delta t \quad (5)$$

Now for "sensible" functions the probability density function is defined as:

$$f(t') = \lim_{\Delta t \rightarrow 0} \frac{P \{ (n\Delta t) < t' \leq (n+1)\Delta t \}}{\Delta t} \quad (6)$$

Taking the limit as  $\Delta t$  goes to zero leads to:  $n\Delta t \rightarrow t$  and  $\Delta t \rightarrow dt$ .

Upon substitution this leads to the final result, the performance probability integral (PPI):

$$P(A < B) = \int_0^{\infty} F_A(t) \cdot f_B(t) dt \quad (7)$$

This equation enables the calculation of the probability that implementation A will perform better than implementation B given probabilistic descriptions of the two Architectures involved. In the remainder of this work, this result will be referred to as the performance probability integral (PPI).

### **4.3 Verifying the Probabilistic Analysis**

To verify the PPI result, four different example cases were analyzed in detail. Even though some of the examples are simple, they help build intuition about the PPI calculation. In selecting the examples, an attempt was made to span the space of likely relationships between the two performance descriptions used as arguments in calculating the PPI. For each example, the PPI was calculated using the CAPE tool, described in Chapter 8. The same example was then simulated using a discrete event simulation (DES). The discrete event simulation was run a large number of times to collect data for a goodness-of-fit test, since there is known to be variability in the simulation result. For the first example, a hypothesis test was done to determine if there was a difference between the CAPE result and the simulation result. For the remaining examples, a simpler test was executed. Since it is known that the sample mean of a population (the varying values obtained from the DES) should be normally distributed around the true mean a large number of sample means was generated and their distribution compared to a normal distribution. In all cases, the CAPE result is consistent with the DES result.

### **4.4 Example 1 – Comparing Two Normal Performance Descriptions**

To make the example specific, consider architecture performance descriptions in Figure 4.2 again. A's performance (red) has been estimated as normal with a mean of 2.0 and a standard deviation of 0.5. B's performance (blue) is similarly

approximated with a normal distribution having mean of 3.0 and standard deviation of 0.5.<sup>1</sup>

When these performance descriptions are substituted into the PPI as shown in Figure 4.3 the result indicates that about 92% of the time A (red) will perform better than B (blue). Since these performance descriptions are easily represented mathematically, numerical methods were applied to evaluate the PPI as well. The numerical integration of the PPI for these specific input functions yields a result of 0.921350 using a trapezoidal rule approximation with step size of 1E-7, and integration limits of minus six sigma to plus six sigma. According to standard error analysis [Krey99] this result is good to five significant figures. That error bound does not include the small area in the tails beyond six sigma. The result was further verified by simulation. Two normally distributed random variables representing implementations from architectures A and B were

---

<sup>1</sup> There is a non-zero probability that either system can perform in negative time. This error is small and can be ignored.

generated. One million samples were taken from each of these distributions and compared.

The results were tabulated, and the probability of A's performing better than B was assessed using a frequency interpretation of the experiment results, i.e., the proportion where the A value was less than the B value was assigned as the probability that A will perform better than B. The following histogram, Figure 4.4, shows the results of the one-million-sample experiment being run 1000 times. The sample mean from the simulation is 0.921351 and is indicated by the black vertical line. The integration result is indicated by the blue vertical line in bin four. Since the histogram was expected to be normal in distribution, the figure shows the comparison of expected value count per bin (in red italics) and the actual count (in black) just below the expected value.

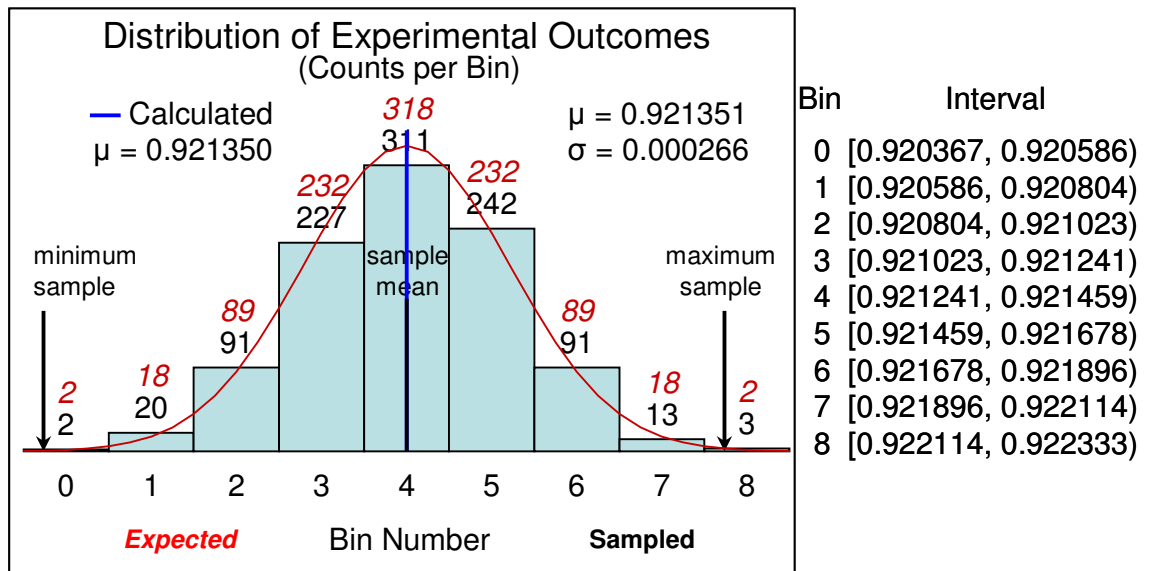


Figure 4.4 Simulation results for comparing systems from Figure 4.2

Applying hypothesis testing to this result, leads to the following two cases:

$$H_0: \mu = 0.921350 \quad \text{and} \quad H_1: \mu \neq 0.921350$$

Using a non-directional test and a level of significance  $\alpha = 0.05$ , the rejection region in the upper and lower tail is  $\alpha = 0.025$ . Bins zero and one are combined as are bins seven and eight so that each of these cases will have more than five expected elements [LeRS01]. This makes the number of degrees of freedom six. From tables of the t-distribution (since  $\sigma$  is unknown):

$$\text{Reject } H_0 \text{ if } t < t_4 = -2.4469 \text{ or } t > t_4 = +2.4469$$

Using differences between the expected and actual bin counts,  $t$  is computed to be 1.782769 and so the hypothesis can not be rejected based on the testing evidence.

The relative performance of two such architecture performance descriptions depends on the difference between the means, as well as the standard deviations of the respective normal distributions. Keeping the standard deviations the same, but varying the means yields the expected result displayed in Figure 4.5. When  $A$  has a mean far to the left of  $B$ , the  $P(A < B)$  is approximately one. When  $A$  has a mean that is far to the right of  $B$ ,  $P(A < B)$  is approximately zero. When the mean of  $A$  is the same as the mean of  $B$ , the  $P(A < B)$  is approximately 0.5.

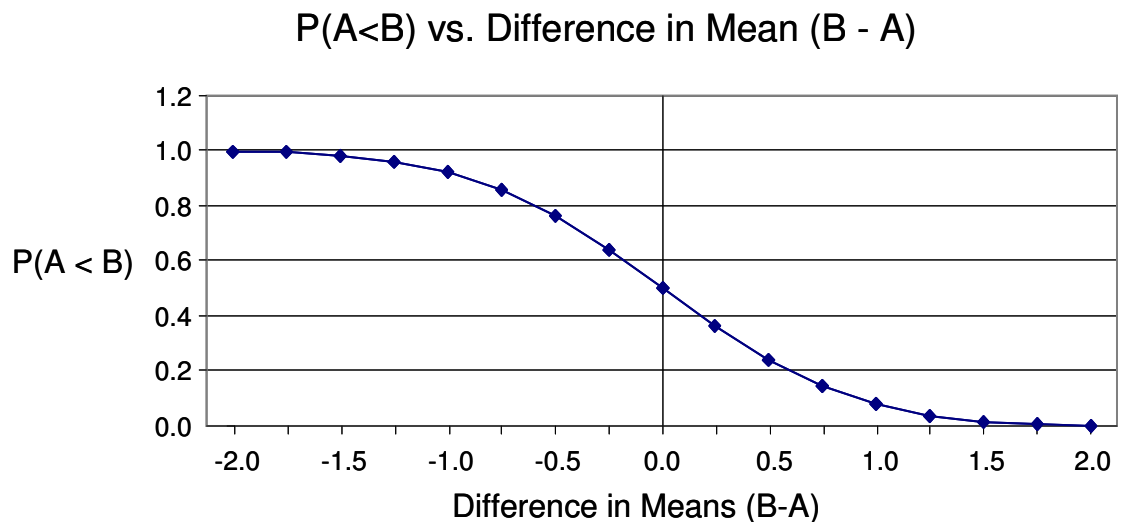


Figure 4.5 PPI values vs. difference in mean ( $B - A$ )

#### 4.5 Example 2 –Possibly Overlapping Performance Descriptions

Performance descriptions can be related in three ways: they may not overlap, they may partially overlap, or one may be included within the other. When using two uniformly distributed ranges of performance values in each of these three cases, the problem is simple enough to be analyzed by hand. Figure 4.6 shows the case where two uniform and non-overlapping probability density functions (1.0 to 2.0 and 3.0 to 4.0) are compared. The  $P(A < B) = 1.0$ .

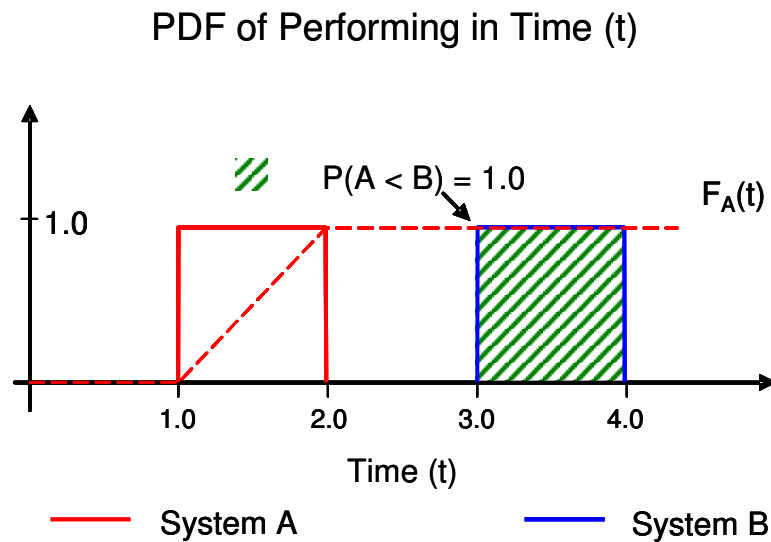


Figure 4.6 PPI calculation: non-overlapping performance descriptions



An example using overlapping uniform density descriptions (1.0 to 2.0 and 1.5 to 2.5) is shown in Figure 4.7. The calculation of the performance probability integral for this case yields that  $P(A < B) = 0.875$ . Results from simulation verification are shown in Figure 4.8

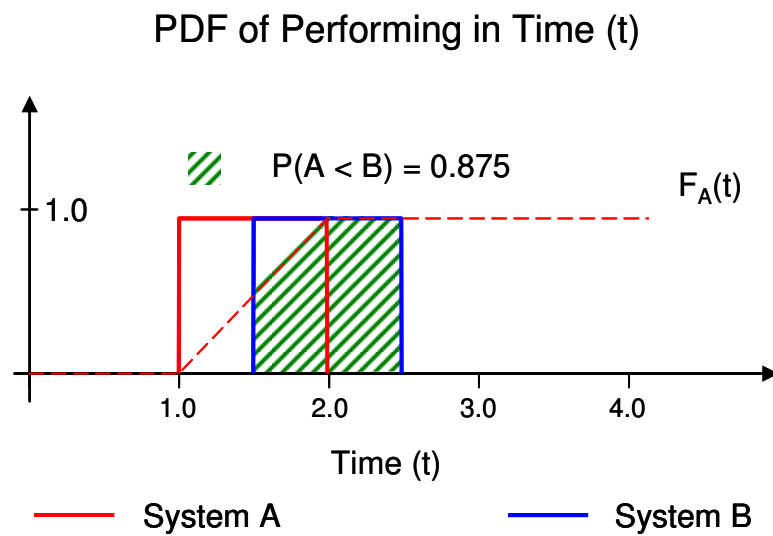


Figure 4.7 PPI calculation: overlapping performance descriptions

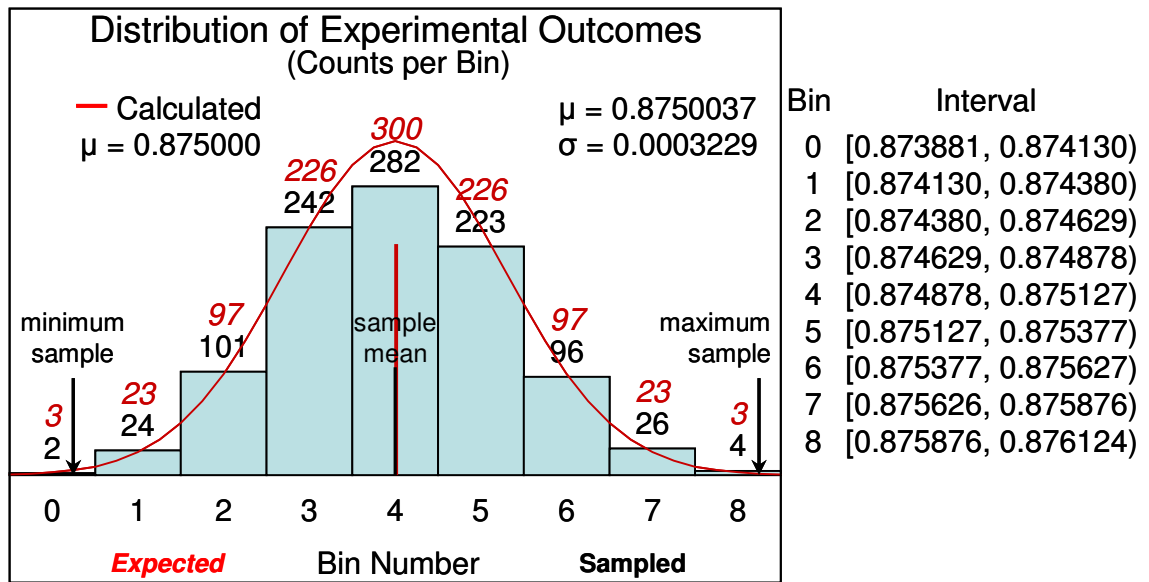


Figure 4.8 PPI simulation: overlapping performance descriptions

In the third case, A's performance range completely includes that of B's as seen in Figure 4.9. The evaluation process is similar. The simulation results that verify this calculation are shown in Figure 4.10.

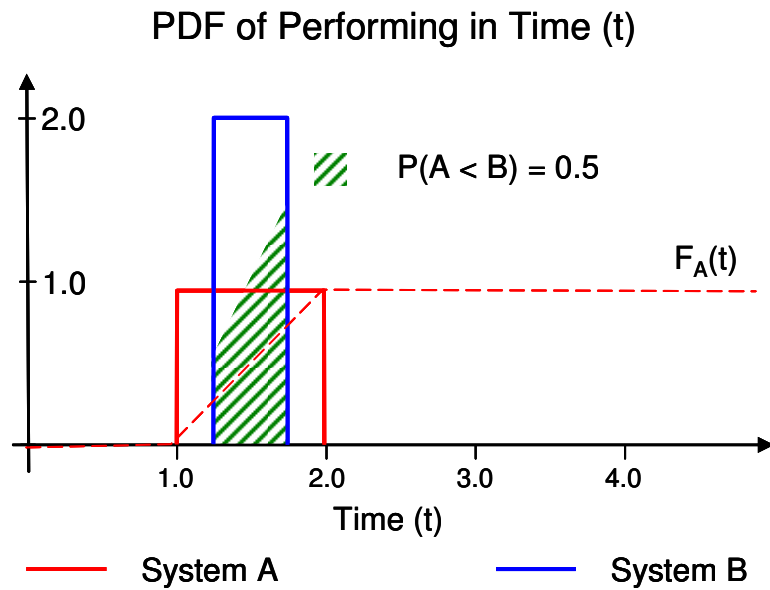


Figure 4.9 PPI calculation: complete overlap of performance descriptions

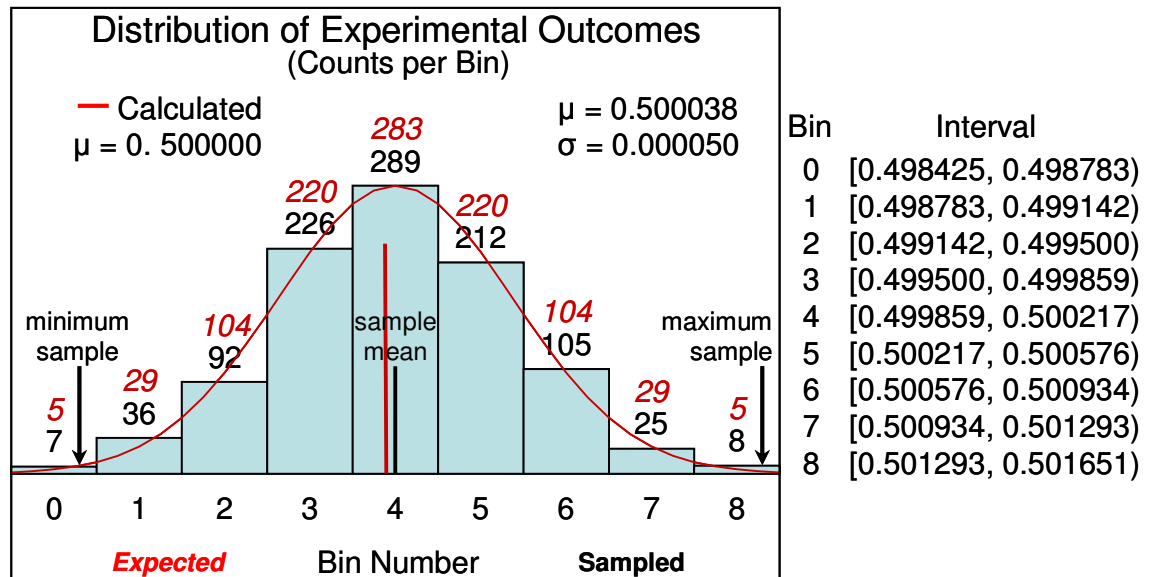


Figure 4.10 PPI simulation: narrow overlapping architecture

#### **4.6 Example 3 - Comparing Discrete Performance Descriptions**

There are cases where the architectural elements can take on discrete performance values rather than any value over a range. In this circumstance, a discrete probability model is appropriate. Similar to the deterministic performance case, the actual value for each performance mode is one number; the randomness comes from not knowing which fixed value from a set of possible values will occur. This situation might happen when considering the use of one of a set of Commercial Off-the-Shelf products. The performance of each alternative may be well known, but the ultimate selection from among the alternative products may not have been completed. The PPI analysis handles this situation similarly.

Consider the following two architecture performance descriptions. A takes one of six equally likely performance values. B takes one of three values with middle value twice as likely. This configuration is shown in Figure 4.11.

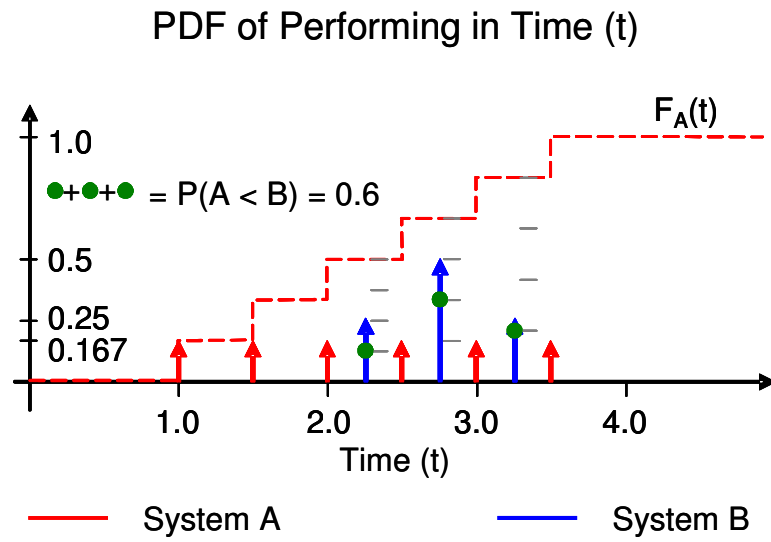


Figure 4.11 PPI calculation: discrete example

The performance probability integral can be evaluated by hand in this case:

$$P(A < B) = (0.25)(3/6) + (0.5)(4/6) + (0.25)(5/6) = 0.6667.$$

Figure 4.12 shows the simulation results verifying the evaluation of the performance integral.

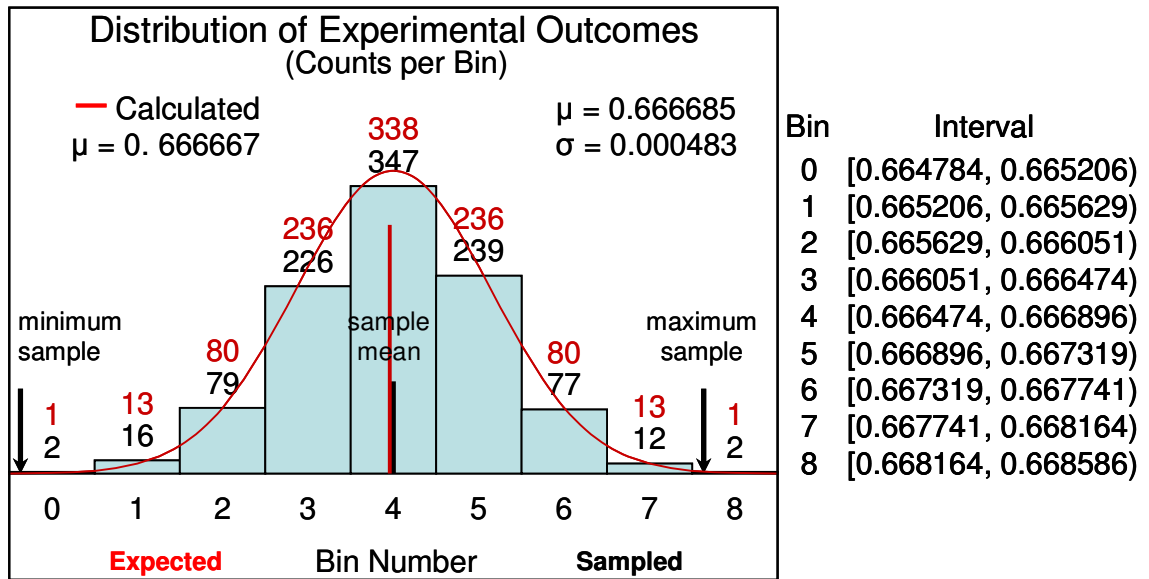


Figure 4.12 PPI simulation: discrete example

#### 4.7 Example 4 – Asymmetric Performance Descriptions

The form of the performance descriptions can be relatively general. The performance descriptions shown below are dissimilar and non-symmetrical. The calculation of the PPI for the example performance descriptions shown in Figure 4.13 yields the probability of A performing better than B as being approximately 0.45. The computations are done as discussed above.

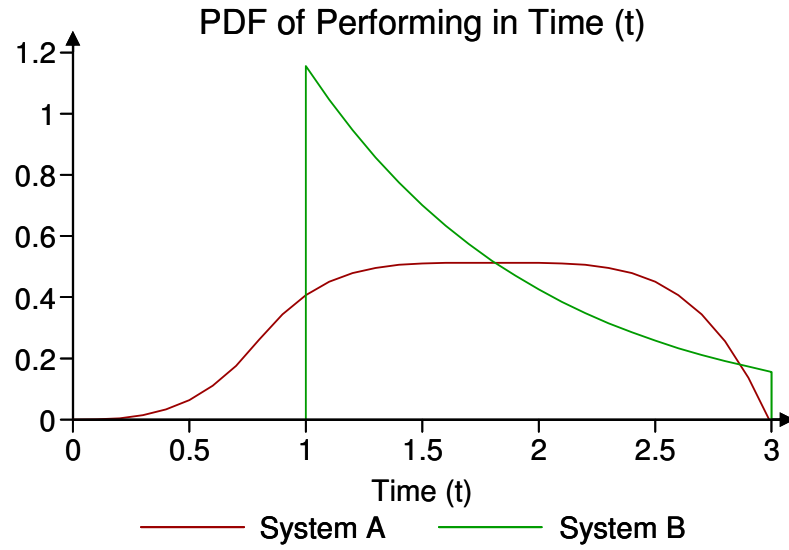


Figure 4.13 Non-symmetric performance description comparison

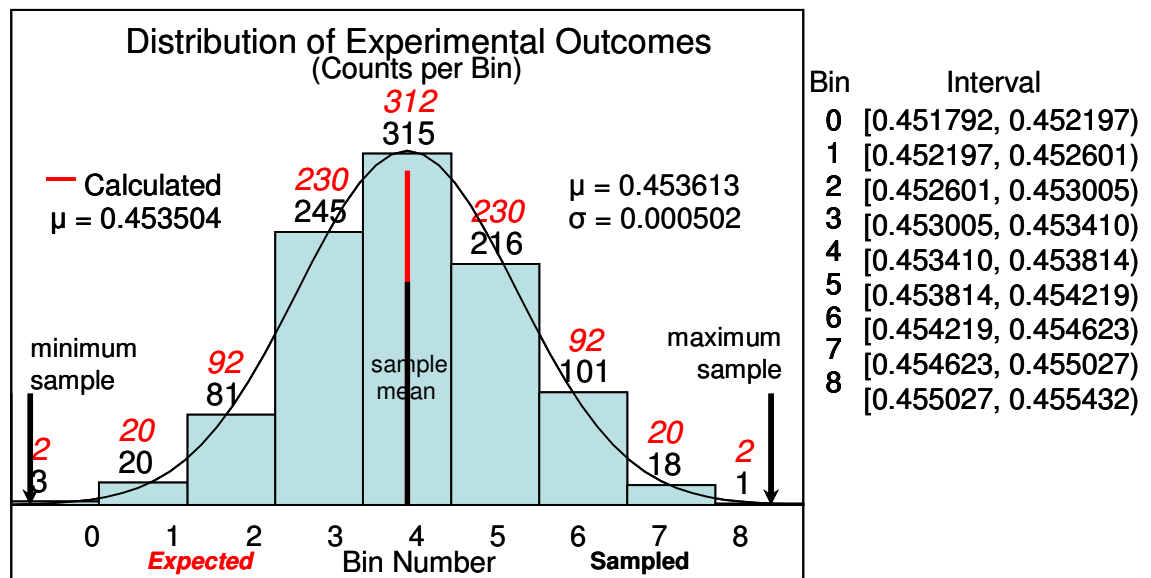


Figure 4.14 PPI simulation: Non-symmetric performance description

The simulation results and performance probability integral calculations agree. A has a 55% chance of doing *worse* than B.

#### 4.8 Summary

There are many uncertainties early in the development process that are unresolved when the architecture selection decision is made. When the uncertainties can be mapped to performance descriptions taking the form of probability density functions for each of the proposed architectures, a comparison is possible and one can assert that a random implementation, system A, from architecture A will perform better than a similarly implemented system B from architecture B. This section derived the form of the performance probability integral (PPI) which quantifies this comparison and provided verification for values generated by it use. The PPI is:

$$P(A < B) = \int_0^{\infty} F_A(t) \cdot f_B(t) dt$$



## **Chapter 5**

### **Constructing Architecture Performance Descriptions**

#### **5.1 Introduction**

The six uncertainty classes described in chapter three form a natural hierarchy that is parallel to a hierarchical model of system construction. In the development process, high level components and connections are successively broken down into smaller and smaller pieces. The reverse process is used when defining and characterizing the uncertainties of systems. Smaller elements with available performance descriptions are combined to construct higher and higher level system components' performance descriptions until the end-to-end performance description for the entire system is developed. A system performance description will routinely be generated in two steps: 1) components are arranged according to alternative topologies and then 2) probabilities are associated with execution paths once synchronization requirements have been applied. This chapter will describe the five basic functions that are used to combine lower level performance descriptions into a high level performance model. In the literature [Papo65], there are calculus based methods for generating the functions which result from combining low level performance descriptions (pdfs). These methods could be applied directly in this situation but

routinely the mathematics becomes difficult and closed form solutions are rarely available. For this work, numerical methods are used to combine performance descriptions (pdfs). Numerical methods make the approach practical and broadly accessible to analysts. In addition, use of these methods does not restrict the probabilistic system or component performance descriptions to simple mathematical functions. Finally, numerical methods are easier to implement than equivalent symbolic manipulation techniques.

This work uses five fundamental functions to combine the performance descriptions of sub-system components: summation, quotient, minimum, maximum and composite. Three of these functions (summation, minimum, and maximum) have already been documented as being useful in computing the time that it takes for a Command and Control organization to respond to an incoming task [Andr88]. There is no claim that these five functions represent a complete set for capturing the performance characteristics of all architectures. However, they are likely to be a sufficient set for the static, acyclic architectures covered by this research since those architectures can only have splits and joins in the directed graph by which the architectures are described. Systems constrained to be acyclic graphs can only split tasks into subtasks, wait for a set of inputs to perform a task, or allow the independent execution of multiple threads of activity.

The details of the techniques for implementing these functions within the context of Comparative Architecture Performance Evaluation (CAPE) can be found in Appendix C. In most cases, verification is done by building random number generators that deliver random values consistent with the specific density functions involved. Values produced by these generators are then combined according to the function being verified. The simulated result is then compared with a CAPE computed result. In the next sections, each of these functions is demonstrated in a practical example.

## **5.2 Summation**

The delay associated with executing a sequence of process steps involving no decisions or branches is computed by summing the contribution of each step. Below, Figure 5.1 shows an example of an individual searching for a web page, sending the result across a noisy link, and another person studying that web page. The time to accomplish all three of these serial activities is the sum of the three activity performance descriptions.

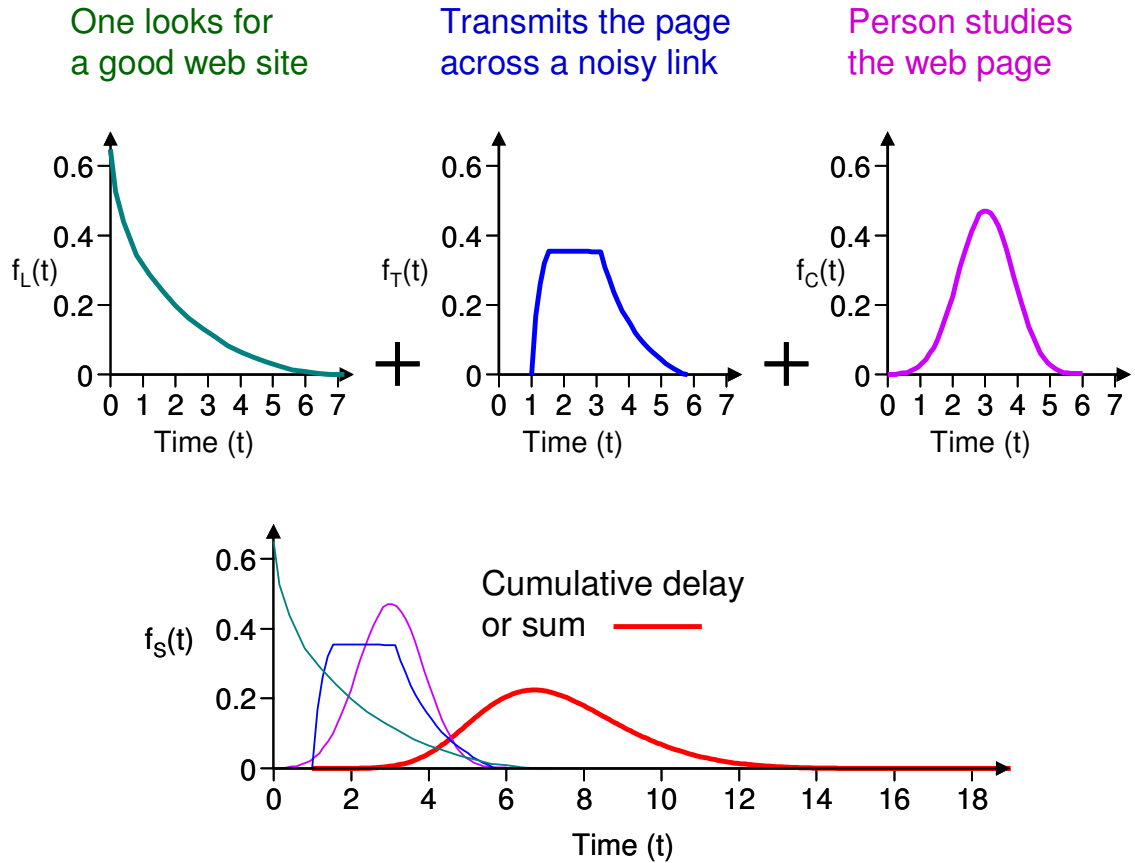


Figure 5.1 Summing process for activity performance descriptions

Since performance descriptions take the form of probability density functions, the summing function will take the form of the convolution of those probability density functions. The performance model defined here associates elementary system delays (those of components and connectors) with random variables and their defining probability density functions. Total system delay is the sum of these random variables (described by these pdfs) and calculated using a method

derived from the convolution definition. The convolution operator (\*) for two pdfs  $f_A(\cdot)$  and  $f_B(\cdot)$  is frequently defined [LaSh79] as:

$$f_A(t) * f_B(t) = \int_{-\infty}^{\infty} f_A(\tau) \cdot f_B(t - \tau) d\tau$$

### 5.3 Quotient

The quotient probability density function is used when one of the sequential processing pieces in an architecture requires the movement of data to a new location over a communications link. The basic calculation is similar to the deterministic case. The amount of data to be moved, divided by the data rate of the communications link, yields the time that the transmission should take.

At a basic level, data packages of some size get transmitted through communications links of some size. The sizes of the data packages presented to the system can be described probabilistically with a density function. There are several factors that impact communications link performance. In the situation where the link is error free, the transmit time is inversely proportional to the link data rate and directly proportional to the size of the packets being transmitted. If the communications link protocols provide error recovery through data retransmission or if the link can be congested by other users, the effective data rate will be reduced. It is the effective data rate of the communications link that

should be used for the random delay calculation. The time to traverse the link can be modeled as a random variable which depends on link error characteristics, link congestion and data packet sizes.

Both the amount of data presented and the rate at which it is transmitted can be modeled as random variables. The pdf for the communications delay can be computed from the quotient of the random data package size and a random effective data rate for the link, when these pdfs are known or can be estimated. The delay,  $z$ , is the size ( $x$ ) divided by the rate ( $y$ ). The equation for the quotient pdf is well known [Curt41]:

$$f_Z(z) = \int_{-\infty}^{+\infty} |y| f_{X,Y}(zy, y) dy$$

The joint probability density function  $f_{X,Y}$  representing the data size elements ( $f_X$ ) and link data rate ( $f_Y$ ) can normally be generated as the product of  $f_X$  and  $f_Y$  as these two densities can routinely be expected to be independent. The calculation has a closed form solution for some simple cases. The process described in this research can provide estimates of the result for more general cases.

## Connection Delay Calculation

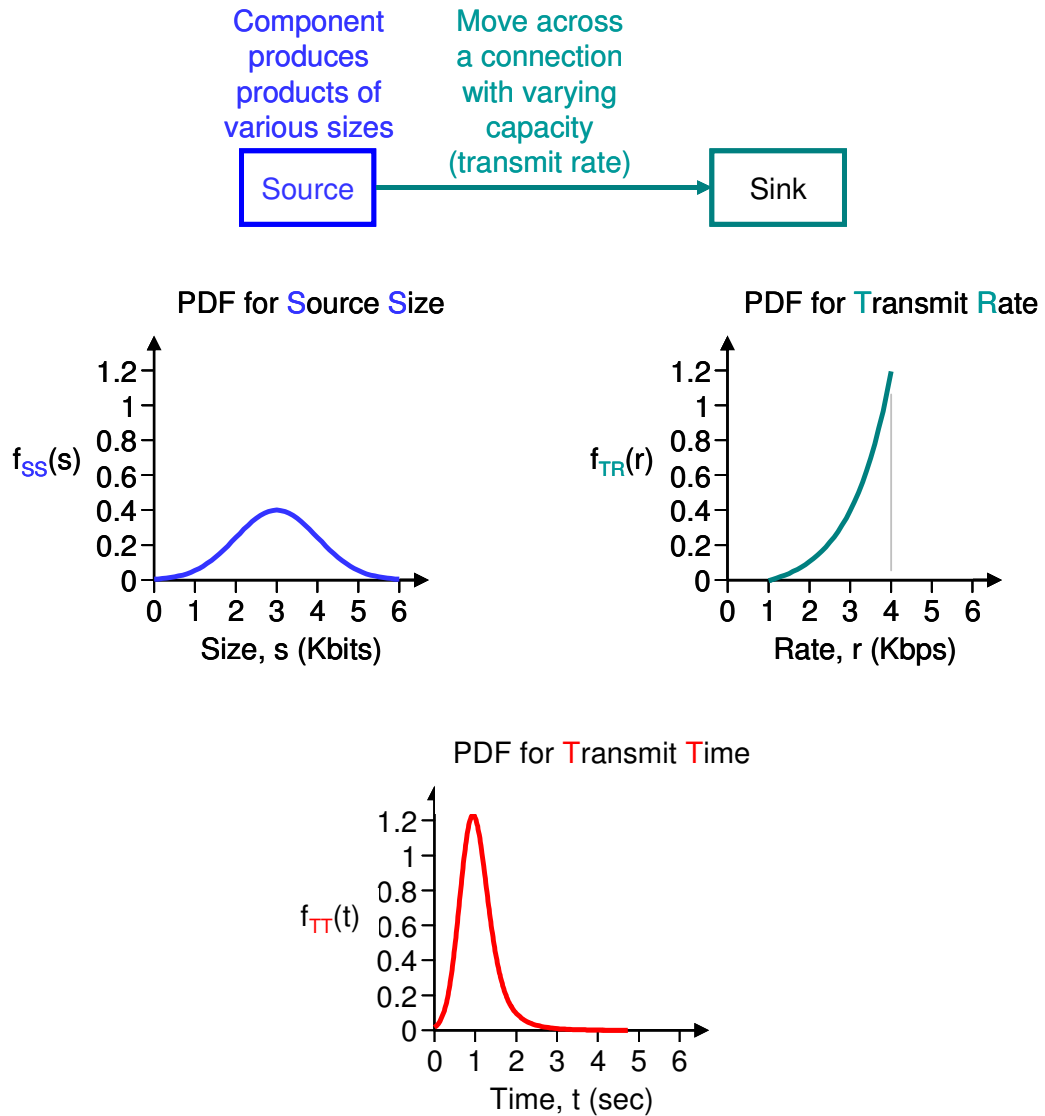


Figure 5.2 Source-sink performance descriptions for quotient calculation

## 5.4 MIN

The minimum function (MIN) is needed in the case where the architecture stipulates that a piece of information may be obtained from any one of a number of places (each of which may respond at a different time). The formula [Andr88] for computing the minimum  $f_Z(t)$  of two random variables, “g” and “h” is:

$$f_Z(t) = h(t)[1 - G(t)] + [1 - H(t)]g(t)$$

Where as usual the uppercase functions ( $H(t)$  and  $G(t)$ ) are the cumulative distribution functions associated with the lower case probability density functions ( $f(t)$  and  $g(t)$ ). When multiple minimum functions need to be calculated, the minimum function is applied a number of time using the first result with each of the next performance descriptions dictated by the problem statement.

In the following example, a user wishes to obtain the phone number from the web using a service. The goal is to understand how long it will take to get a response. The service is designed to simultaneously query three different and independent service providers. The query can be considered to be satisfied when the first of the three independent services responds with an answer.



Get the phone number for:

John Doe  
123 Maple St.  
Hometown, USA 12345

## Calculation Duration

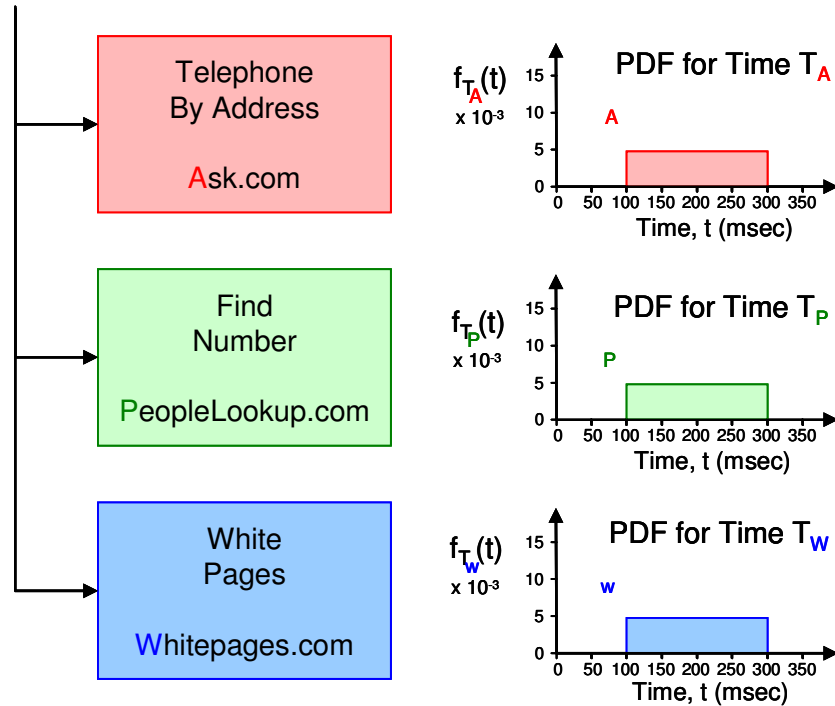


Figure 5.3 Example case where first result is sufficient

For the purpose of this example, each service is presumed to return an answer in a time that is uniformly distributed between 100 and 300 milliseconds. The result of the minimum function being applied to this case is shown in Figure 5.4.

$$T_{req} = \text{Min}(T_A, T_P, T_W)$$

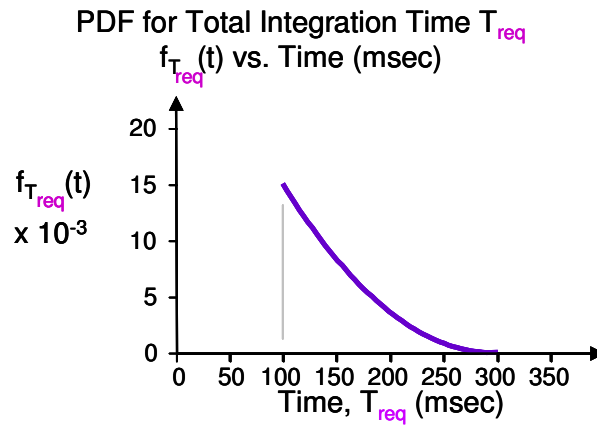


Figure 5.4 Minimum function computation result

This situation occurs any time that a product is requested from multiple independent sources, or a result is calculated in multiple independent different ways and the first answer is considered acceptable.

## 5.5 MAX

The maximum function (MAX) can be encountered in a number of ways as well. The most commonly encountered occurrence is when there are several input values required for a computation. These values may be generated in different places or at different times using different methods. They may be generated in parallel as well. Usually the computation can not proceed until all inputs are

available. Alternatively the situation can occur where a computation is done in number of different ways to ensure consistency in the result. The maximum function is useful to characterize the time it takes to be ready to continue with the comparison after a consistency check is done on these computed values. The formula [Andr88] for computing the maximum  $f_Z(t)$  of two random variables, “g” and “h” is:

$$f_Z(t) = h(t)G(t) + H(t)g(t)$$

Where as usual the uppercase functions ( $H(t)$  and  $G(t)$ ) are the cumulative distribution functions associated with the lower case probability density functions ( $f(t)$  and  $g(t)$ ). When multiple maximum functions need to be calculated, the maximum function is applied a number of times using the first result with each of the next performance descriptions dictated by the problem statement.

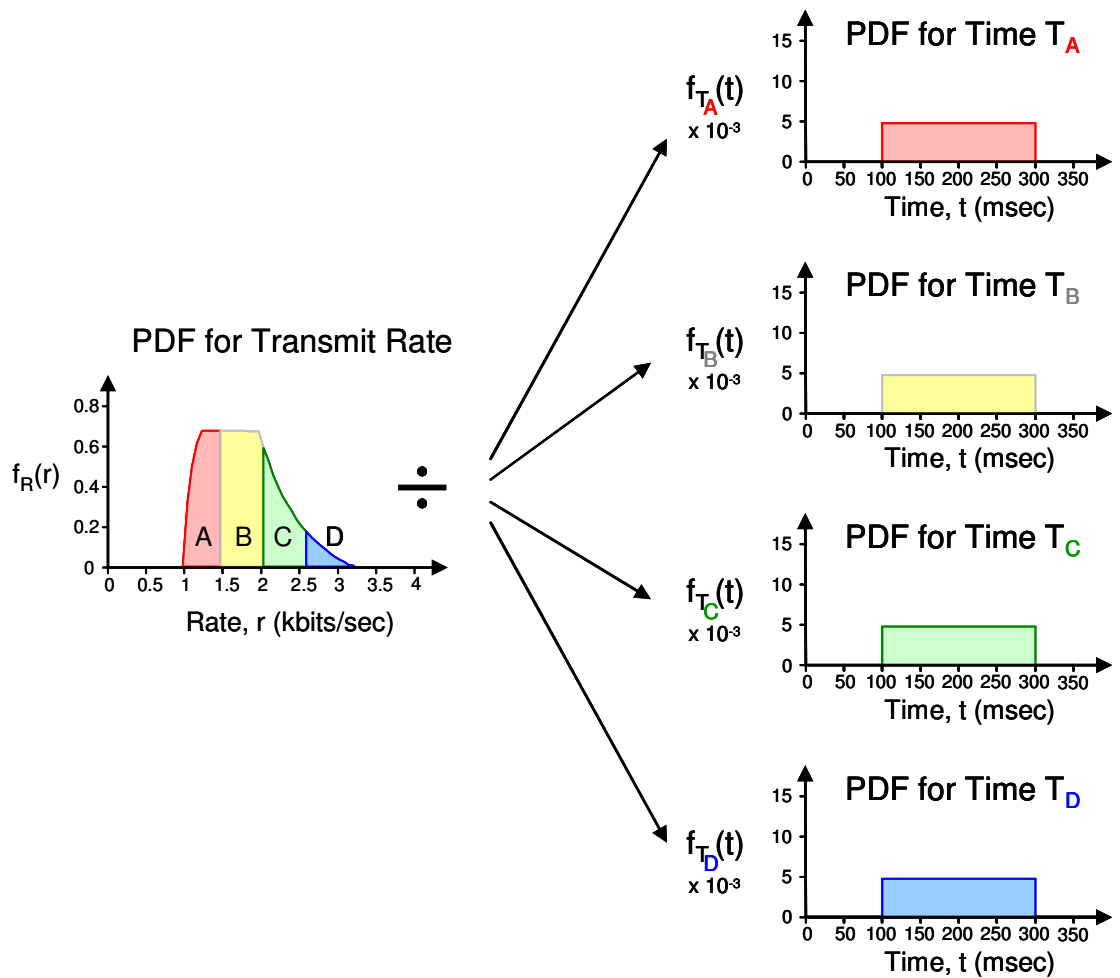


Figure 5.5 Timing for a subtasks processed in parallel

Figure 5.5 shows the calculation plan for integrating the area under the curve. The idea behind the example is that the calculation of the area under the curve can be broken up into four problems. Each of the colored bands represents a portion of the curve's area calculation that can be assigned to a separate integrator. Each of these integrators will perform their sub-tasks in a time

duration described by a uniformly distributed random variable with duration between 100 and 300 milliseconds. The final answer is not available until the last of these four computations completes and the four sub-task answers are added together. This addition time is assumed to be very small when compared with the delay associated with the rest of the calculation. Figure 5.6 shows the result of the maximum function being applied to these four performance descriptions.

$$T_{\text{Tot}} = \text{Max}(T_A, T_B, T_C, T_D)$$

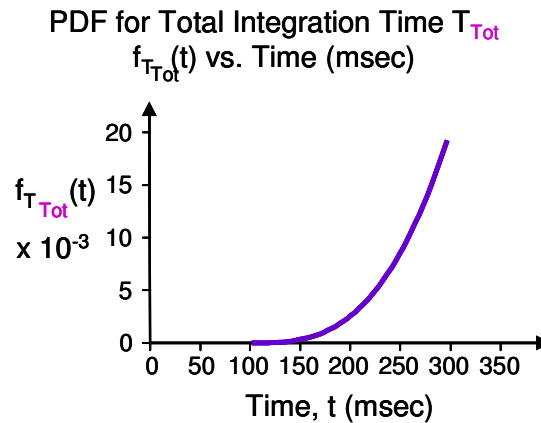


Figure 5.6 Maximum function computation result

## 5.6 Composite

There are times when different amounts of data may be moved across different paths, either in parallel or based on a decision. These different paths terminate on the same source and sink but traverse different intermediate points. In these situations, a composite delay density function characterizes the data movement. Normalized weighting factors are computed based on the probability of data traversing each path. The individual density functions are then multiplied by these normalized weighting factors and summed to produce the composite density function.

The composite calculation is useful in two separate situations. Figure 5.7, shows the situation where two classes of data are applied to a channel. The weighting factor for each class presented is expressed as a percentage of the total communications load applied. In this example, both classes are assumed to be of equal magnitude. The class distributed like a semi-circle (gray) and that distributed as a trapezoid (also gray) are scaled to the green and blue shapes before being combined in the communications channel. This scaling results from the weighting of the two sources and when combined creates the result shown as the red line. The vertical beige bars are the simulation confirming the resulting computation.

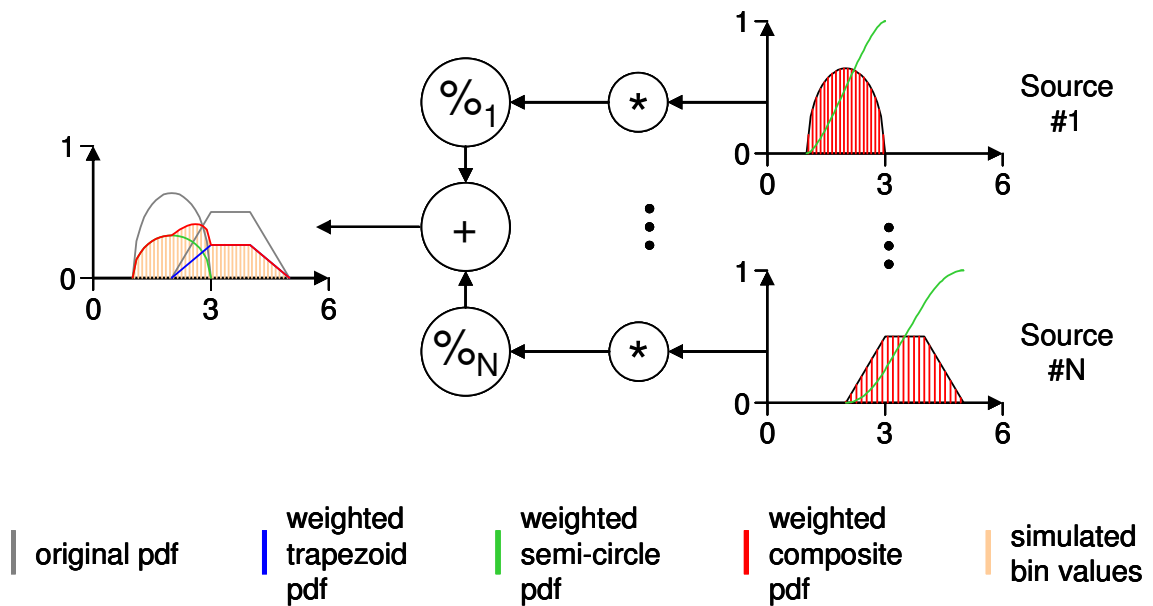


Figure 5.7 PDF result of combining two data classes in a common channel

In the second case, Figure 5.8, the data classes may be transmitted over different channels. The result is similar. Each channel changes the input data (size) pdf and these pdfs are added after being weighted proportionately by the amount transmitted. In the figure, the original pdfs (gray) are scaled according to the amount transmitted (blue, yellow, green) and combined into the red weighted composite description of the combined data. Again the beige vertical lines are the values captured with a discrete even simulation verifying the computation.

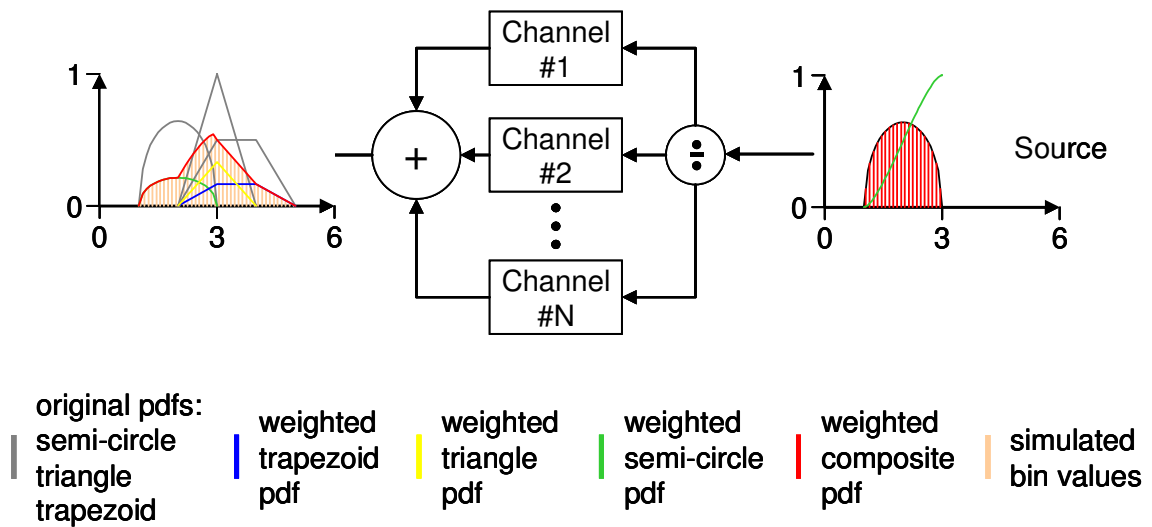


Figure 5.8 A single class of data is transmitted over any of multiple paths

Figure 5.9 shows examples of different weights applied to data passing through the channel. The  $p_C$  and  $p_T$  values represent the probability or percentage of the semi-Circular pdf and Trapezoidal pdf respectively as submitted to the channel. The red curve represents the aggregate. The beige lines show the simulation results confirming the computed channel behavior using random number generators matched to the input pdfs.



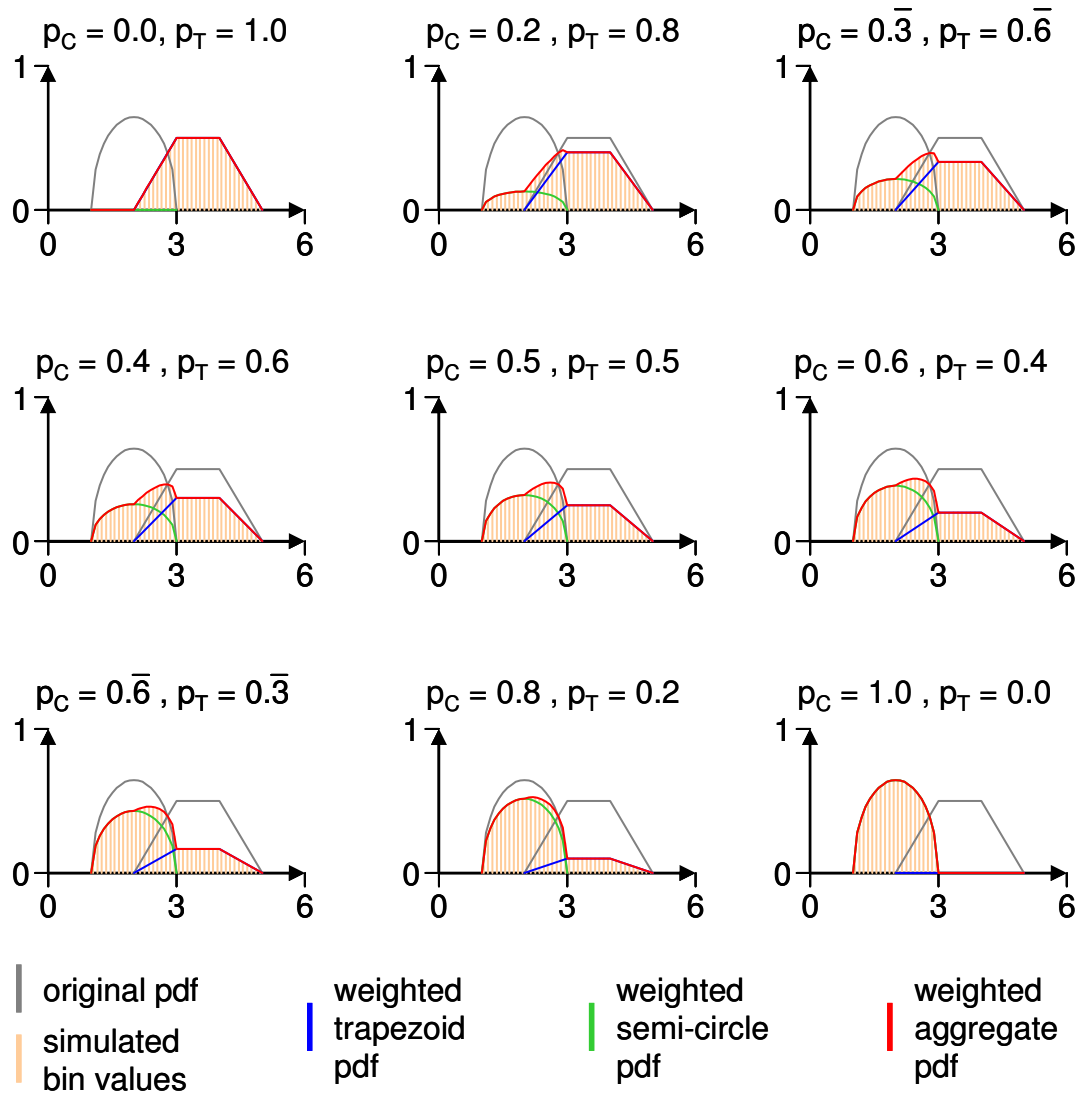


Figure 5.9 Simulation confirmation of weighted composite calculations

A simpler example of the composite function is shown below. In Figure 5.10 there are two possible paths through the system. The composite delay

generated in this situation is the probabilistically weighted combination of each of the system paths.

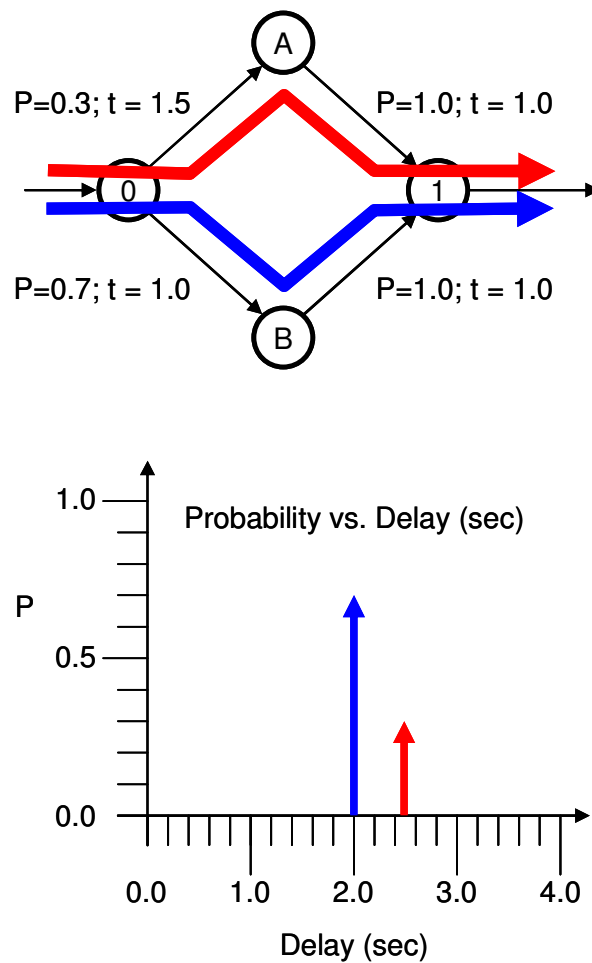


Figure 5.10 Composite delay result for example two path calculation

## 5.7 Summary

This section has described the five functions (sum, quotient, minimum, maximum, and composite) required to combine performance descriptions and the

methods used to calculate the performance description composing smaller elements into larger elements. The sum function is applicable when there is a sequence of elements to be combined. The quotient function is used when both the rate of the channel and the size of the data being put on the channel are random in size. The minimum and maximum functions are used for process synchronization. The composite function is used when there are either different classes of data traversing a path, or different paths being traversed by a single class of data.

## **Chapter 6**

### **An Illustrative Practical Architecture Example**

#### **6.1 Introduction**

To show how the preceding probabilistic analysis can be used for evaluating the performance potential of architectural alternatives, it is helpful to consider a simple example. The example chosen follows [DoWh07] and represents a classic web application with several different alternative architectures. To compare the performance potential of these alternative architectures, the architecture components and connectors are first characterized individually and then these descriptions are combined using the techniques of chapter five. After several alternative architectures have been analyzed, the results are compared to identify the relative performance potential of these architecture alternatives.

To perform the analysis, four steps are executed. These steps are: 1) identify the alternative architectures to be examined, 2) characterize the performance of architectural elements based on an appropriate set of assumptions that address scaling issues to ensure that the results are comparable architecture-to-architecture, 3) combine the delay contributions for each alternative, and 4) compare the delay results across architecture alternatives.

For the example, consider a small web application designed to provide a user with the current version of a requested XML document. Based on constraints associated with the operational environment, the high-level form of the solution is already agreed to be a three-tier architecture as shown in Figure 6.1. There is consensus that there will be some processing available at each tier, and that these tiers will be connected by communications links. The specifics describing the expected load, the available computational capability at each tier, and the sizes of the connecting communications links are initially unspecified.

In this application, an XML document request is made at the User tier and transferred to the Cache tier. A search is then performed first at the Cache (and then at the Server if required) to determine if an update is needed. If the user version of the requested document is found to be stale, an update is processed resulting in the User accessing the current document locally. There are many ways to execute the involved steps, but the expectation is that by caching documents between the User and the Server, the overall system response will be improved.

The diagram illustrates the User-Cache-Server architecture and its components. At the top, the architecture is shown as a sequence of components: User, Cache, and Server. The User component consists of a stick figure and a circle labeled 'COMP'. The Cache component consists of a circle labeled 'COMP' and a square labeled 'XMT'. The Server component consists of a circle labeled 'COMP'. Bidirectional arrows connect the User 'COMP' to the Cache 'COMP', and the Cache 'COMP' to the Server 'COMP'. Bidirectional arrows also connect the 'XMT' square in the Cache to the 'COMP' circle in the Cache, and the 'COMP' circle in the Cache to the 'XMT' square in the Cache. Below the architecture, a legend defines the components: a circle labeled 'COMP' is a 'Computational Node', a bidirectional arrow represents an 'Ideal Communications Link' with 'no delay, no loss, no errors', and a square labeled 'XMT' represents 'Practical Communications Parameters'.

Occasionally environmental factors force an update to documents located on the server. Rather than redistributing the complete current XML document to the user after each change, a processing improvement is adopted to decrease the size of the information sent between the tiers. The architect presented with this problem has decided to consider including any of a set of a compression mechanisms in the design to potentially speed transfer of data between the tiers. The chosen compression technique, shown in Figure 6.2, takes advantage of the fact that XML documents are tree-structured. Classical tree-difference algorithms [ZhSh89] can be used to generate difference scripts which are smaller than the original documents. These difference scripts minimize the editing distance

between the initial and updated trees (documents) by defining a series of additions and deletions or changes to the original document to produce the updated document. In the common case, these difference scripts are much smaller than the web pages themselves, and this can be seen to be a form of compression. A difference script can be converted into a XSLT document that describes how to transform the original (old) XML document into the updated XML document [KayM00]. The size of the generated XSLT document is proportional to the size of the difference script. These processing steps could be executed at different tiers in different architectures. The overall transformation process is shown in Figure 6.2.

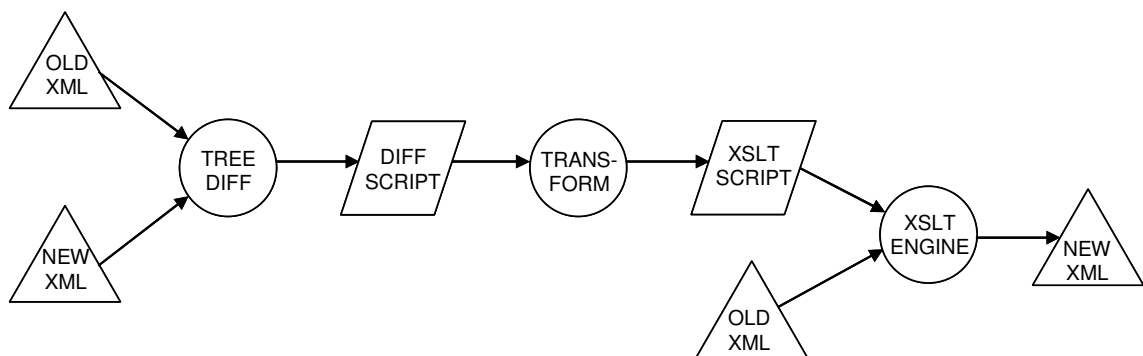


Figure 6.2 XML Tree Transform Process Example

### 6.1.1 Selecting Specific Architectures for Demonstration

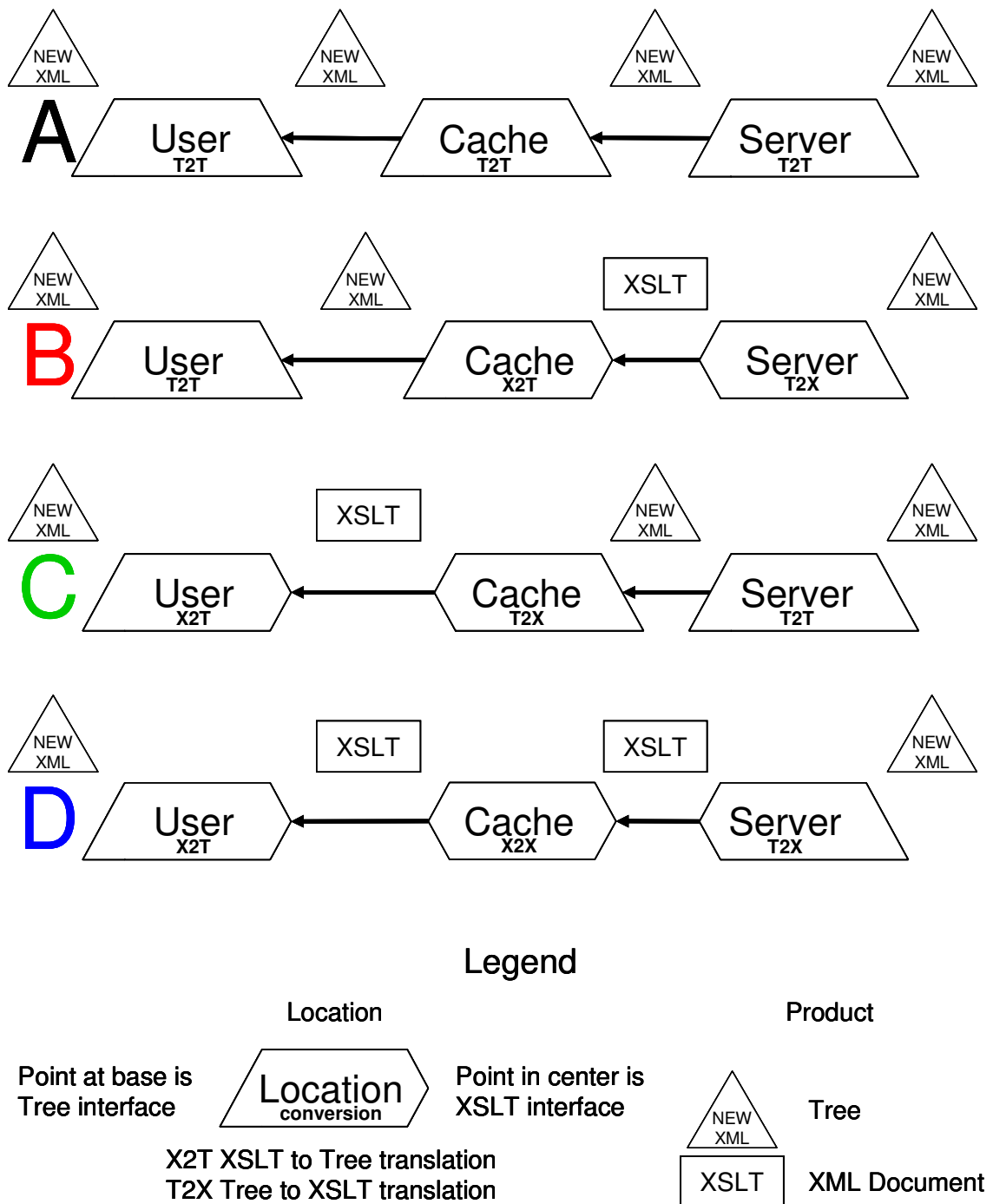


Figure 6.3 Alternative example architectures



The approach researched is broadly applicable. To make it easy to correlate the architecture differences with the performance differences in this example, the communications infrastructure will remain fixed for all four architectures considered. What will change is the configuration of compression servers. Were the communications links to be changed either by modifying the data rate, error rate, or locations between architecture nodes, the results would change accordingly.

Figure 6.3 shows four potential architectures for this application. All four architectures have three nodes connected sequentially and the different component shapes in the figure represent different versions of the components. Architecture A uses no compression; an updated XML document is sent from the Server to the User on request. Architecture B uses the compression techniques described earlier in the Server so that it outputs an XSLT script that is sent to the Cache. The Cache uses this script on its local copy of the document to create an updated version which is forwarded to the User. The Cache compresses this updated document into a XSLT script which the User will use to convert the old XML document into the updated XML document. Architecture C again uses these compression techniques but it creates an XSLT script at the Cache and sends it the User. The User similarly uses this script with its local copy of the document to create an updated version. The Cache compresses this updated document into a XSLT script which the User will use to convert the old XML

document into the updated XML document. Finally, in Architecture D, the compression is at the Server (as in Architecture B); in this case, the Cache forwards the compressed document to the User for use in converting the old XML document into the updated XML document (as in Architecture C).

## 6.2 Assigning Delay Descriptions to Architecture Elements

The information needed to compute the performance of each architecture computation or communications activity has the same basic form for any type of problem where these techniques can be applied. Data for these examples is given in table form. Table 6.1 describes the structure of these tables. This table structure will be used in subsequent analysis to keep track of the various delay contributions, both those which are assumptions associated with architecture component performance, and some which are derived as is the description of transmit delay. The top line (E) of the table holds the estimated value descriptions. It includes both the data transformation processing delay and the communications delay. The bottom line (D) holds the delays to be combined and represents the inputs to the functions that combine delays as in chapter five.

The boxes in this table are small. To keep the graphs readable, Table 6.2 shows the abbreviated versions that will be used for each graph in the following analysis sections. The table is formulated in three columns. The first column shows the location of the graph being explained (shadowed block). The center column

shows the graph labeled as it would be in a larger picture. The right column shows the abbreviated labeling of the graphs which will be used in the following analysis.

Table 6.1 Standard data table for comparing transition delays.

Delay Contribution					
Delay		From Data	To Data	Propagation	Transmit
D <sub>xy</sub>	E	A pdf description of data size <b>before</b> node processing	A pdf description of data size <b>after</b> node processing	A pdf description of the data leading edge delay	A pdf delay description for putting data on the connector
	D	A pdf description of the time needed for node processing to convert <b>before</b> pdf to <b>after</b> pdf		Same as above	Computed (pdf) transmit time

Table 6.2 Graph abbreviations used in analysis tables that follow

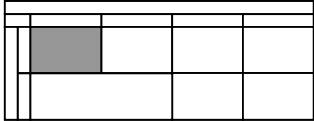
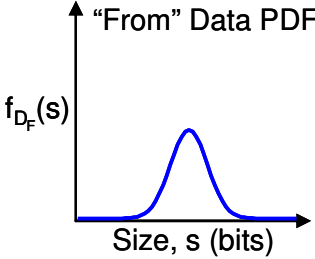
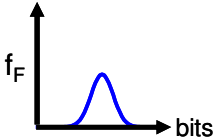
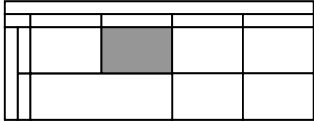
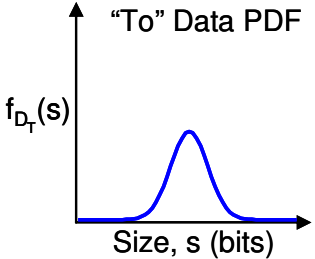
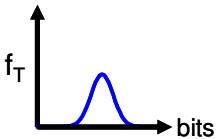
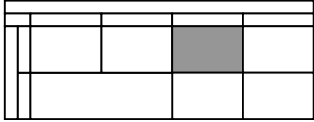
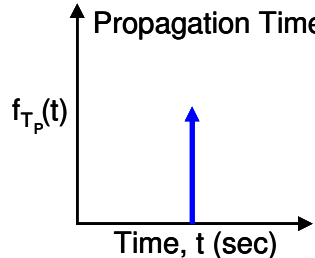
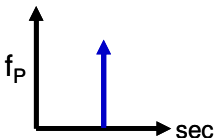
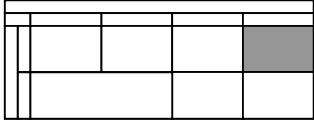
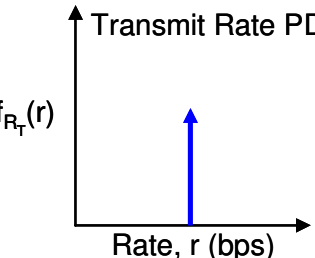
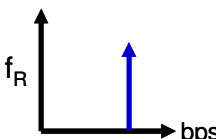
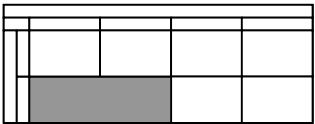
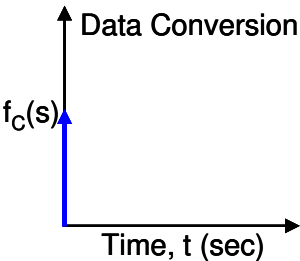
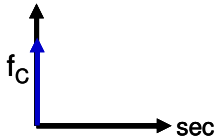
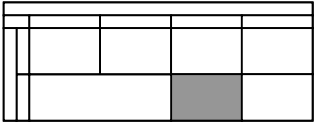
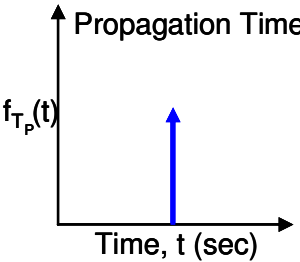
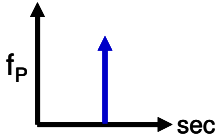
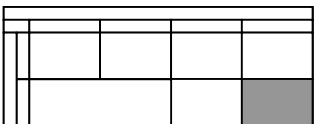
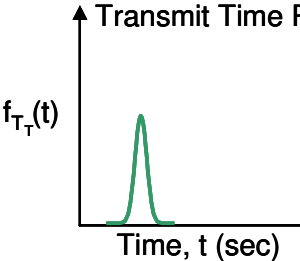
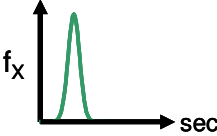
Table Location	Actual Graph	Abbreviated Graph
	<p>“From” Data PDF</p> 	
	<p>“To” Data PDF</p> 	
	<p>Propagation Time PDF</p> 	
	<p>Transmit Rate PDF</p> 	

Table 6.2 Graph abbreviations continued

Table Location	Actual Graph	Abbreviated Graph
	<p>Data Conversion PDF</p> 	
	<p>Propagation Time PDF</p> 	
	<p>Transmit Time PDF</p> 	

### 6.2.1 Characterizing the Architectural Elements

For analysis purposes, it is necessary to make some assumptions about the likely implementations and use of the system. The analysis that follows is based on the following assumptions:

- A single file is studied, but that file will have a size that is normally distributed.
- The one way (up and down) trip geostationary satellite delay is roughly twice that of a terrestrial trip across the continental United States, that presumed distance traversed on a terrestrial (fiber) link.
- The data rate for the satellite link is assumed to be twice that of the allocated portion of the terrestrial (fiber) link.
- The user to cache distance is approximately equal to the cache to server distance.

As with other values, characterizing estimates should be tailored based on an understanding of the specifics of the actual problem being solved.

### **6.2.2 Architecture A**

Starting with the simplest "just-send-it" architecture, architecture A, consider the factors that go into populating Table 6.1. Architecture A takes the documents that exist at the server and transfers them in total (without any compression) to the cache, and then again to the user. Delays associated with this process are labeled with a "D" that is subscripted by the source and destination nodes. In this

case, architecture A has two delays one from the Server to the Cache ( $D_{sc}$ ) and one from the Cache to the User ( $D_{cu}$ ). To evaluate architecture A using this methodology, compute those two delays and combine them.

To execute the calculation, the delay tables for both  $D_{sc}$  and  $D_{cu}$  are populated as shown in Figure 6.4. The descriptions that go into the table are specific to architecture A. The values have been taken from the characterizing assumptions listed above.

Since there is no processing involved at the Server, the transform time is zero. The communications time required to move the data from the Server to the Cache can be calculated by applying the assumptions regarding the throughput characteristics of the links connecting those nodes. Since there are few specifics at this early step of design, further assume that the transmit delay will dominate the propagation delay. Both delays are considered reference values and will be used to assign values later to  $D_{cu}$ .

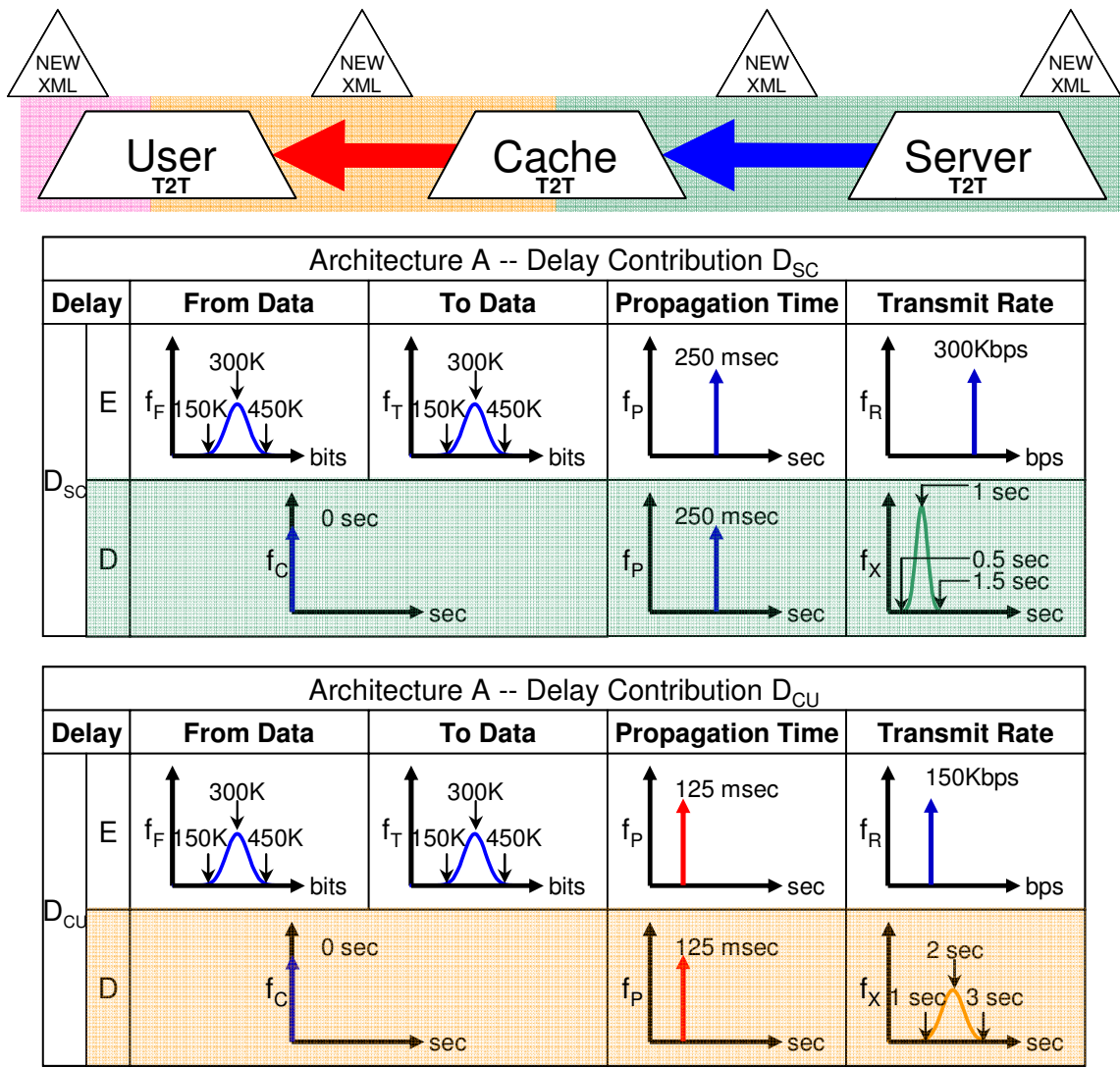


Figure 6.4 Architecture A and associated component delays

For this example, specific values have been selected for “Transmit Rate” and “Propagation Time” to make the example easy to follow. This is not however a requirement. As long as the ratios of all delays are preserved, any values could



be used. The final PPI result will be unaffected by scaling of the time axis. The goal of the methodology development was to make comparative architecture performance evaluations. The assignment of specific values is not a significant restriction to the method's use.

The total delay for the architecture is the sum of the six graphs in the “D” rows of Figure 6.5. This aggregate delay is computed using the techniques of chapter five.

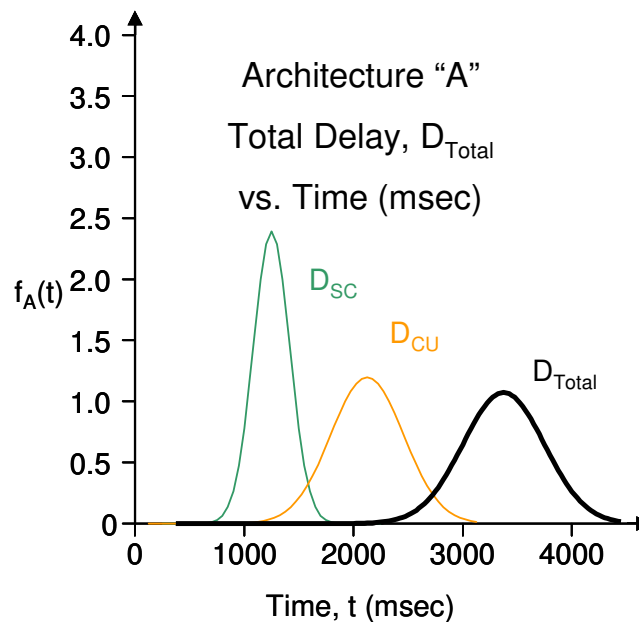
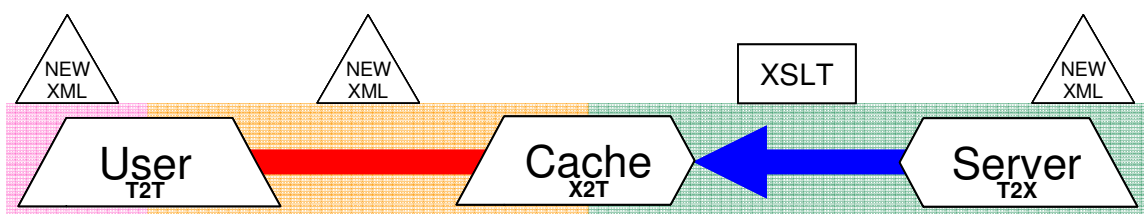


Figure 6.5 Performance description for architecture A

### 6.2.3 Architecture B

Figure 6.6 shows architecture B and the associated delay tables. In this situation, the product from the Server is an XSLT script generated from both the new and old XML files. Once this XSLT script is transferred to the Cache, it is used to convert the old XML file into the updated file. The total delay for the architecture is the sum of the six graphs in the “D” rows of Figure 6.6. The gray curves on Figure 6.6 show the performance values of architecture A and are included for comparison purposes. The result of combining all of these delays is shown in Figure 6.7.



Architecture B -- Delay Contribution $D_{SC}$				
Delay	From Data	To Data	Propagation Time	Transmit Rate
$D_{SC}$	E 			
	D 			

Architecture B -- Delay Contribution $D_{CU}$				
Delay	From Data	To Data	Propagation Time	Transmit Rate
$D_{CU}$	E 			
	D 			

Figure 6.6 Architecture B and associated delay tables

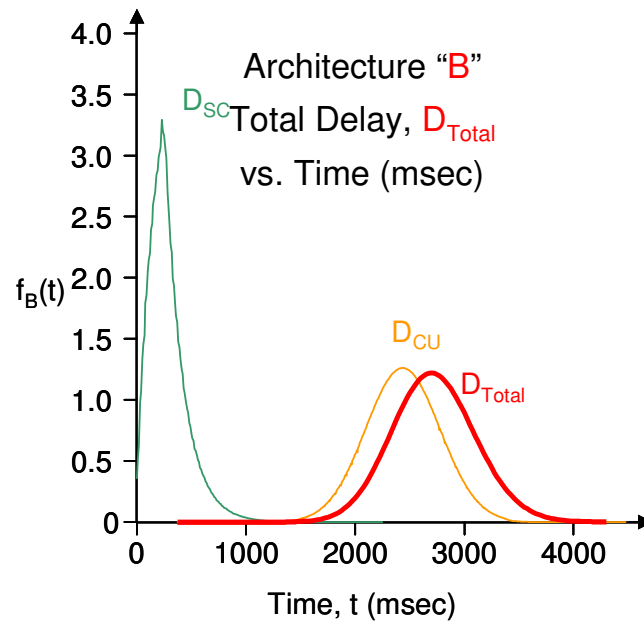
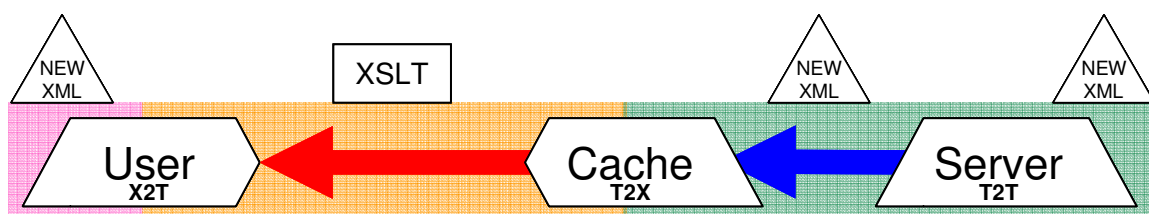


Figure 6.7 Performance description for architecture B

#### 6.2.4 Architecture C

Next architecture C is considered. Here the conversion from both the old and new XML trees to the XSLT script occurs in the Cache. The remainder of the architecture remains the same. The XSLT script once transferred to the User is used to convert the old XML document into the new XML document. The total delay for architecture C is made up of the six contributions in the "D" rows shown in Figure 6.9.



Architecture C -- Delay Contribution $D_{CU}$				
Delay	From Data	To Data	Propagation Time	Transmit Rate
$D_{CU}$	E 			
	D 			

Architecture C -- Delay Contribution $D_{UU}$				
Delay	From Data	To Data	Propagation Time	Transmit Rate
$D_{UU}$	E 			
	D 			

Figure 6.8 Architecture C and associated delay tables

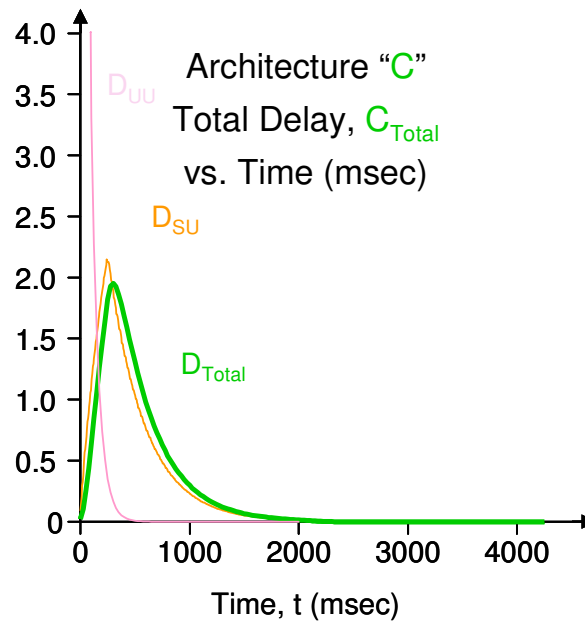
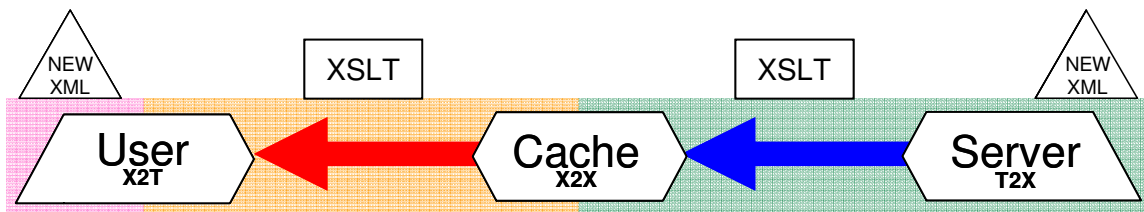


Figure 6.9 Performance description for architecture C

### 6.2.5 Architecture D

In the final architecture to be considered, D, the generation of the XSLT script is done in the Server and it is transferred through the Cache to the User where it is combined with a copy of the old document to produce the desired new document. The architecture and delay descriptions for this case are illustrated in Figure 6.10. The total delay accumulated in the process from the six contributions in rows "D" of Figure 6.10 is shown in Figure 6.11.



Architecture D -- Delay Contribution $D_{SU}$					
Delay		From Data	To Data	Propagation Rate	Transmit Time
$D_{SU}$	E				
	D				

Architecture D -- Delay Contribution $D_{UU}$					
Delay		From Data	To Data	Propagation Time	Transmit Rate
$D_{UU}$	E				
	D				

Figure 6.10 Architecture D and associated delay tables

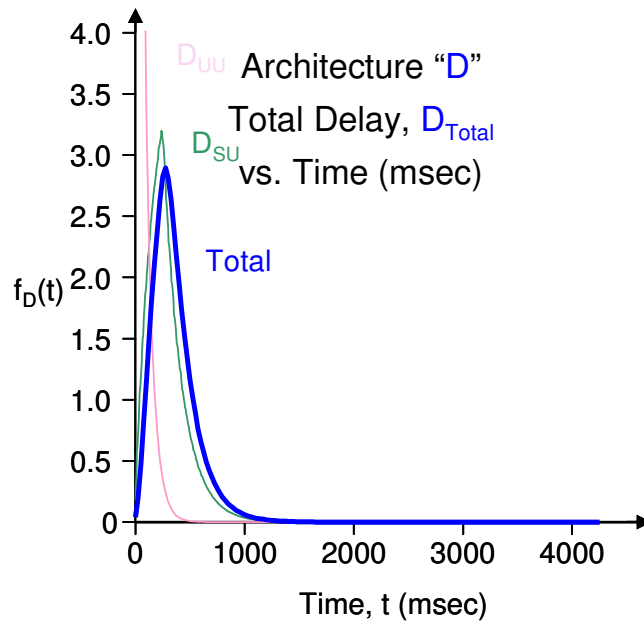


Figure 6.11 Performance description for architecture D

### 6.3 Comparing Results Across Architectures

Tabulating the evaluation of the performance probability integral for all pair-wise cases simplifies identifying the best performing architecture. Figure 6.13 shows the relative performance for all architectures on a pair-wise basis. When comparing architectures A and B it is seen that B is better than A 88.8% of the time. When comparing architectures C and D, D is better by 65% of the time. When comparing B and D, it is seen that D is better 100% of the time. Clearly the order in which these are compared says something about by how much the best is better. One can note that the probability that  $C_x$  will perform better than



$C_Y$  is just 1.0 minus the probability that  $C_Y$  will perform better than  $C_X$ . This is clear as there are only two cases.  $C_Y$  or  $C_X$  must perform better so the total of both performing better must equal one.

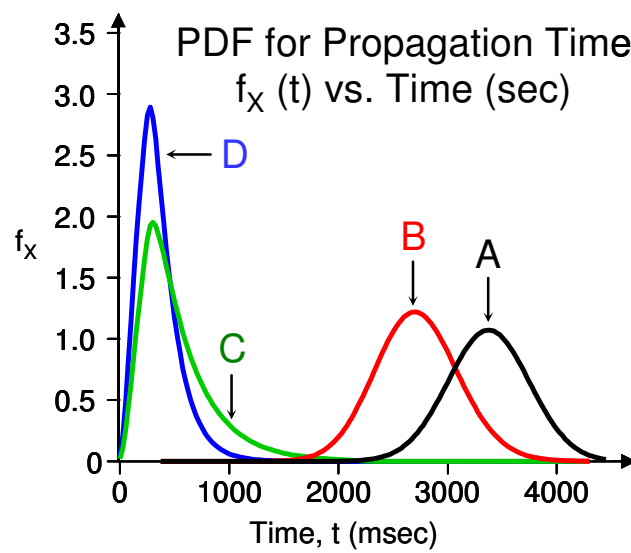


Figure 6.12 Plot of all four architecture delay descriptions

Arch	<b>A</b>	<b>B</b>	<b>C</b>	<b>D</b>
<b>A</b>	----	0.112	0.000	0.000
<b>B</b>	0.888	----	0.000	0.000
<b>C</b>	1.000	1.000	----	0.350
<b>D</b>	1.000	1.000	0.650	----

Prob(row < col)

Figure 6.13 Summarizing all architectural performance results

### 6.3.1 Scaling Issues

It is desirable to make comparisons between architectures on a relative basis, but this goal can be taken only so far. Ultimately, there must be a constant scale factor or fixed relationship maintained between the delays described for each architectural element and the others. As long as this delay-to-delay relationship can be preserved, then all architectures (and architecture element performance values) can be scaled without changing the outcome of the analysis. In this example, the geostationary satellite round trip delay could be chosen to be  $\frac{1}{2}$  a time unit in seconds. Based on the assumptions above, the terrestrial delay would be half that. The assumptions state that the satellite data rate is twice that of the terrestrial link. Hence after choosing the satellite data rate arbitrarily at two

bit/sec units, then the terrestrial data rate had to be four bit/sec units. Figure 6.14 documents the application of these assumptions to the example.

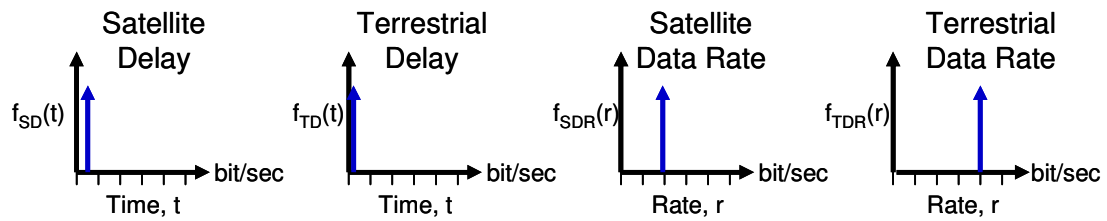


Figure 6.14 Establishing related propagation delays and data rates

Once the data rates have been established, the computed times for transmission must be inversely proportional to those rates. In this case, since the terrestrial data rate is twice that of the satellite link and the size of the data is the same in each case, the width of the pdf describing the time required to transmit a file is  $\frac{1}{2}$  as shown in Figure 6.15.

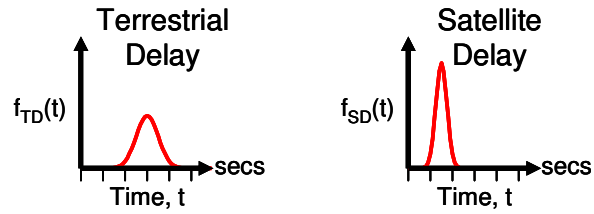


Figure 6.15 Result of scaling the satellite delay to the terrestrial delay

In more general cases, an arbitrary reference can be selected and applied consecutively to the graphs being specified. Consider two density functions  $f_A$  and  $f_B$ . With the arbitrary scale factor, compute the scale of  $f_A$  to be some constant  $C_a$ . Similarly compute the scale factor  $C_b$  for  $f_B$ . Then scale  $f_B$  by the factor  $C_b/C_a$ .

## 6.4 Summary

This chapter has taken a simple three tier example architecture and walked through the steps to identify the computational and communications processes involved with generating a solution. Next, the performance descriptions of the architecture elements were estimated and these descriptions were combined into an end-to-end estimate for each of the potential architectural alternatives. Finally, the performance probability integral was evaluated for each of the cases considered, to identify which of the architectural choices performed the best.

## **Chapter 7**

### **Two Larger Examples**

The previous chapter discussed performance analysis for a classical three tier database application. This chapter applies these concepts to a pair of problems which more closely represents real-world situations and therefore exposes issues commonly encountered in real applications. The benefits of these two examples are three-fold: they demonstrate different functions used in the performance analysis process, they give insight into some of the approximations that can be made to either simplify the analysis or to continue the analysis when pieces of information are unavailable and finally they demonstrate the utility of specifying performance parameters in a way that can account for variations in the use of the system. The first example represents a class of problems where the primary performance uncertainties are associated with link characteristics. This example considers exfiltrating data from a clandestine source to an unknown location. The second example is representative of a performance analysis that would usually be encountered later in the development lifecycle. At that time, more information is known about the design and performance requirements are likely better specified. This second example addresses the development of a service in a Service Oriented Architecture framework. It demonstrates that these performance analysis techniques can be used to refine performance estimates.

One interesting feature of this example is that it is possible to determine at architecture selection time that a specific architecture cannot meet the performance specification and that an alternative architecture can correct this situation.

## **7.1 Example One: Data Exfiltration**

The data exfiltration example is representative of a class of problems where many of the performance unknowns are delays associated with distances between the elements that make up the architecture. This type of uncertainty can be an advantage. There is value in designing systems to accommodate ranges of values rather than specific values. Design processes based on exact values for characterizing parameters are often less robust. Their performance can be sensitive to changes in the environment in which they run. Systems designed to accommodate ranges of critical values are normally more robust and can often be used for purposes other than those for which they are initially designed.

### **7.1.1 Problem Definition**

For this example, assume that there is a news reporter in a remote location that needs to exfiltrate data back from her location to a news room where it can be correlated with other pieces of information. Due to the remoteness of the source location, the reporter has only two communications alternatives. She may use a

radio which can establish a satellite link to a nearby town, or she may communicate through a local network provided by military assets. Once her data reaches the nearest town, there are two transmission options as well. The data may be interleaved with other users' data on a covert network connection to make it inconspicuous, or the data may be relayed to another satellite where it is forwarded to its destination for consolidation. Since the system is being designed for general purpose use, there is little known about the relative distances between the reporter, the town, and the collection location. Further, in an effort to be inconspicuous, the radio is designed to be hand-held. It has only a small antenna so the reporter's satellite link must be established with a low earth orbiting (LEO) satellite.

### **7.1.2 Large-Grain Delay Descriptions**

As with many early lifecycle architecture analysis questions, one needs to make some preliminary assumptions about the way components and connections perform when the solution is not entirely specified. The numbers shown in Table 7.1 are generated as approximations to actual system performance and represent a number of different deployment situations. This will almost always be true when the deployment locations are not known in advance of the system's being built. This situation is not uncommon since there can be a variety of options for implementing systems. Different options implies that the performance descriptions need to cover a range of values. For example, not all LEO satellites

orbit at the same altitude; hence the performance description for a satellite connection should be formulated as a range of values. Since the problem indicates that the reporter is trying to be inconspicuous, it is reasonable to assume that she is relatively close to but not in the town. Since the local satellite is in low earth orbit, the delay is likely to be approximately five to ten milliseconds. Assume the town is likely to be a far distance from the news room (as it would be in cases of an international disaster or war zone). The second satellite is likely to be in geosynchronous (GEO) orbit (250 ms one way delay). The terrestrial network delays are each likely to be a fraction of the geostationary satellite delay. The allocated terrestrial bandwidth for the tactical network is likely to be much less than that of either satellite link. At architecture design time, none of the specifics about either satellite or terrestrial hardware is known. It is reasonable then to estimate that the tactical data rate will be about half that of the geostationary SATCOM link and the same as the LEO satellite. These component and connection characteristics are summarized in Table 7.1.



Table 7.1 Summary of data exfiltration system performance estimates

	LEO-SAT (L1)			GEO-SAT (L3)		Tac-Net (L2)		Glob-Net (L4)	
	Prop	Rate	Vis Del	Prop	Rate	Prop	Rate	Prop	Rate
min	5 (ms)	128 (Kbps)	0 (min)	250 (ms)	256 (Kbps)	200 (ms)	128 (Kbps)	50 (ms)	256 (Kbps)
max	10 (ms)	256 (Kbps)	95.2 (min)	250 (ms)	512 (Kbps)	1000 (ms)	128 (Kbps)	280 (ms)	256 (Kbps)

### 7.1.3 Alternative Problem Architectures

There are three nodes and two connections in this architecture. Each pair of connected nodes has two alternative connection implementations resulting in four possible architectures. Figure 7.1 shows the options for the architectures and defines labels that will be used in the following analysis to establish the various delays on the connections. The problem of selecting the best performing data exfiltration architecture is solved by building the performance descriptions for each of the alternatives. The four options are listed in Table 7.2

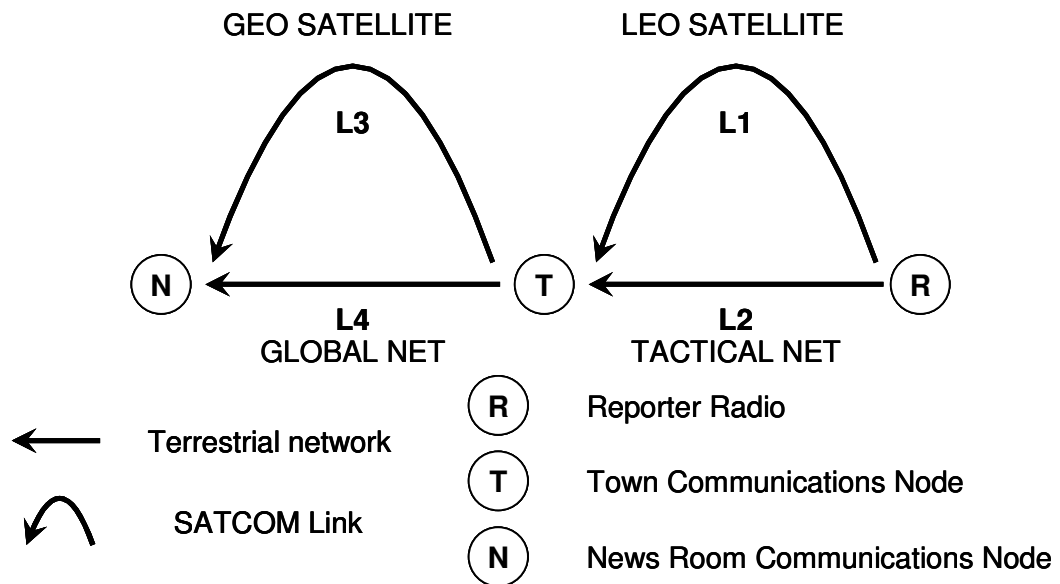


Figure 7.1 Data exfiltration architecture and labeling convention

Table 7.2 Table of architectures considered

Name	Components	Links
$A_{LG}$	LEO-SAT→GEO-SAT	L1, L3
$A_{LB}$	LEO-SAT→GLOB-NET	L1, L4
$A_{TG}$	TAC-NET→GEO-SAT	L2, L3
$A_{TB}$	TAC-NET→GLOB-NET	L2, L4

#### **7.1.4 Delay Characterization**

To characterize the end-to-end delays associated with each of the architecture alternatives, a performance description for each component and connection must be generated using estimates. The amount of data to be processed, the data rates and error characteristics of the links need to be described in order to build the performance descriptions. The propagation delay of each link is needed to calculate the overall transport delay. In situations where the satellite is not always available due to masking, the satellite visibility times need to be considered.

#### **7.1.5 Data Characterization**

To analyze the performance of the different possible architectures, one starts by characterizing the data that the reporter must transmit: text, audio clips, and store-and-forward video. Text messages will be assumed to be several hundred characters. Pre-requested voice questions will be answered with 4 - 15 second of perhaps 600-bytes-per-second compressed phone quality audio yielding 3000 to 9000 bytes per response. The video is likely to be 30 - 60 seconds at 30 frames per second, 640 scan lines per frame, 480 pixels per scan line, and three bytes per pixel with a compression ratio of 8 - 12 yielding about 99 MB to 132 MB per transmission. In each of the performance characterization diagrams that follow, only the video data is depicted. Since the processing and communications delays are directly proportional to the amount of data moved,

performance diagrams for the other data types would be scaled, time-shifted versions of these video diagrams.

#### **7.1.6 Characterizing Delay $D_{L1}$**

Figure 7.1 identifies the nomenclature that will be used to label nodes and links in the data exfiltration architecture performance tables. Each link option can be characterized using the format of Table 7.1 and the techniques described in chapter six. In this example, there are no delays associated with the transformation of data from one form to another and it is assumed that any data compression for voice and video occurs in the recording device.

Since there is no widely agreed altitude for low earth orbit satellites, assume that the satellite to be used has an altitude somewhere between 800 and 2000 km. The orbital angular velocity associated with the satellite will be 3.569 to 2.830 degrees per minute [WeLa99]. This velocity yields an orbital period of 101 - 127 minutes. Assume a nominal viewing angle from the ground of 90 - 180 degrees; the satellite is in view for roughly 25.2 - 63.5 minutes and not in view for about 50.5 - 95.2 minutes. The amount of time a satellite is in view is a function of the orbital period and the viewing angle from the ground. A summary of the possible satellite in-view and out-of-view times is shown as Figure 7.2.

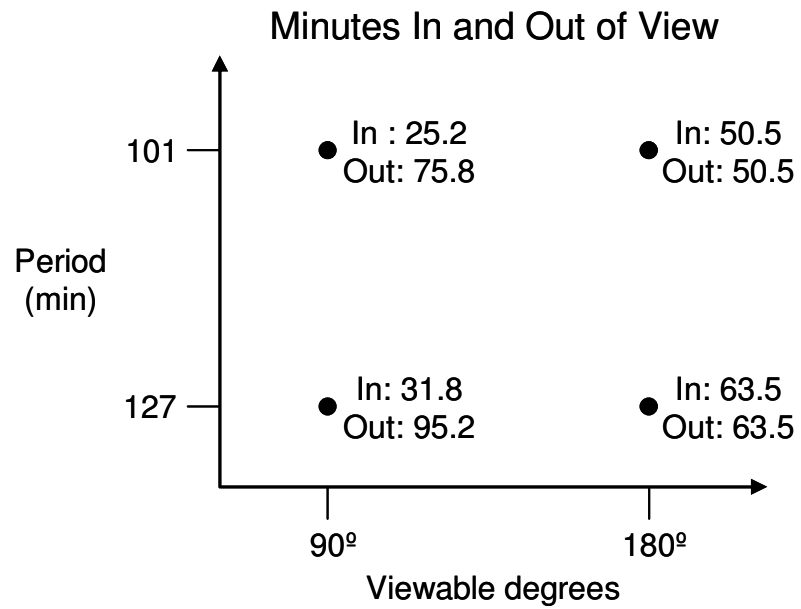


Figure 7.2 LEO satellite accessibility times

The probability density function describing the LEO connection transmission rate can be determined using the uniform quotient analysis example from chapter five. Only the corner values are shown in Figure 7.3.

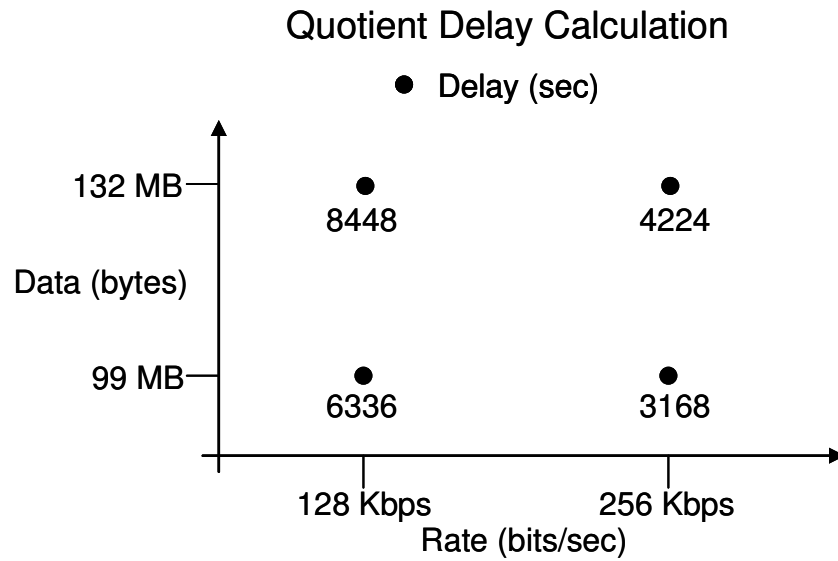


Figure 7.3 Critical points for LEO satellite transmit delay

Figure 7.4 shows the mapping of the data parameters to probability density functions as before with the “E” or expected performance row characterizing the link, and the “D” or delay row showing the connector delay results. The size of the transmitted data is assumed to be uniformly distributed between the bounds specified in the data characterization section above. There is no additional processing done by the system so the processing delay is zero. The transmit data rate over the satellite was not specified so is assumed to be uniformly distributed between 128 Kbps and 256 Kbps. Since both the data and the data rate are uniformly distributed, the link throughput can be calculated using the uniform quotient distribution calculation.

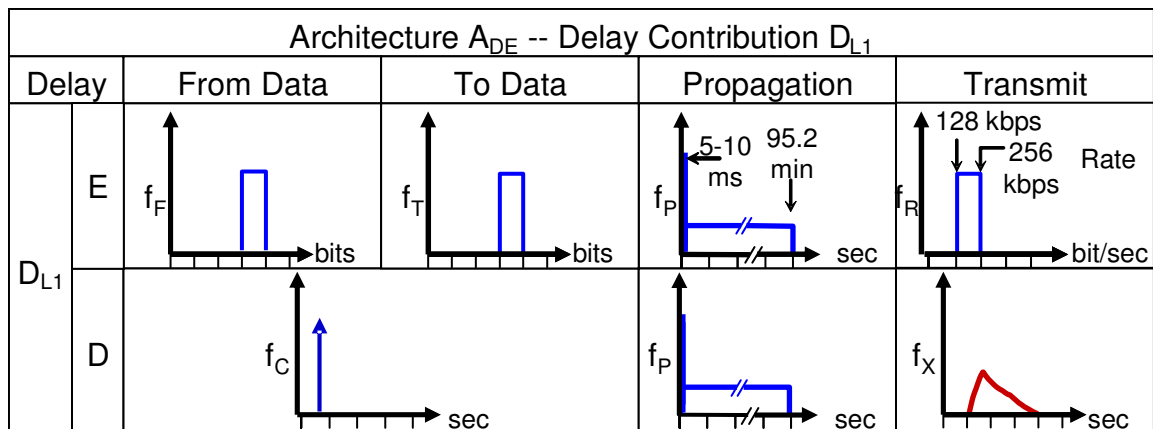


Figure 7.4 Architecture  $A_{DE}$ , characterization of delay  $D_{L1}$

An unusual aspect of this problem is that regarding the visibility of the satellite. The satellite must be in view to close the communications link. For that portion of the time when the satellite is in view, the propagation delay is characterized by that of the low earth orbiting satellite. For the satellite-not-in-view case, the waiting time is modeled as uniformly distributed over the time that the satellite is obscured by the earth. This model is reasonable since the satellite location and the time of the transmission request are both unknown. An exact analysis result is the combination of these two distributions, one uniform to account for the propagation delay when satellite is visible (altitude is still unknown) and the other uniform distribution describing the time that the user has to wait for the satellite to become visible. These characteristics are summarized in Figure 7.4 and the total delay is shown in Figure 7.5. Note that here and in the situations that follow, the

same diagram represents each of the three content types: text, voice, and data. The actual diagrams differ only in scale.

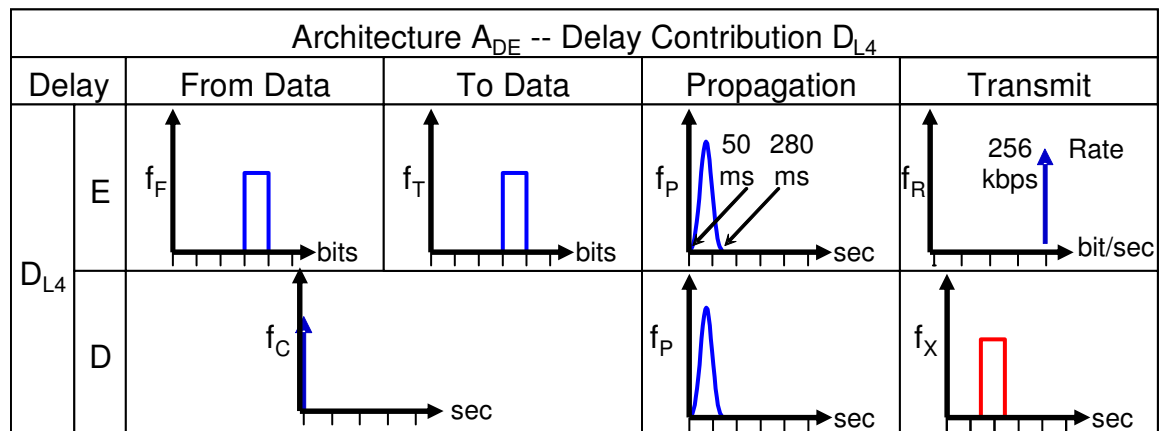


Figure 7.5 Combined delay contributions for  $D_{L1}$

### 7.1.7 Characterizing Delay $D_{L2}$

The process for analyzing the other links delays is similar. Link  $D_{L2}$ , a tactical terrestrial network link between the reporter and the town, will have longer delays when the source and destination are near the network diameter and shorter delays for nearby node pairs. There are more intermediate values than either extreme so a normal distribution is selected to represent this delay. Tactical networks are subject to high error rates and intermittent link outages caused by



multipath interference and terrain masking. Hence, the network is assumed to be a store-and-forward architecture when in data mode. The per-hop delay will be assumed to be around 100 ms, and the network diameter will be assumed to be ten. Since this is a terrestrial network, links will be short. The propagation delay (aside from the store-and-forward aspect) will be considered to be inconsequential. While the allocatable data rate would normally be larger on this type of network, it is being intentionally restricted at the source to maintain the clandestine nature of that source hence the effective data rate will be low.

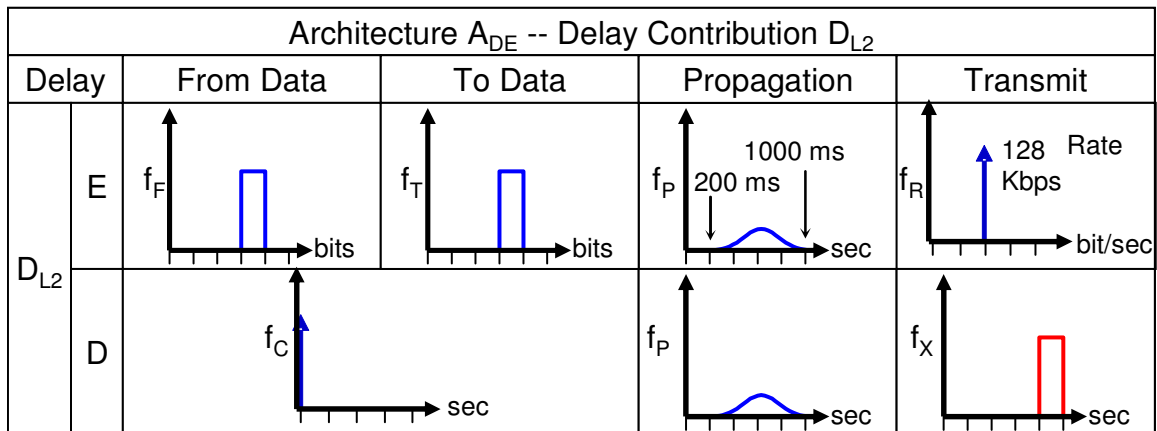


Figure 7.6 Architecture  $A_{DE}$ , characterization of delay  $D_{L2}$

### 7.1.8 Characterizing Delay $D_{L3}$

The characteristics of link  $D_{L3}$  between the town and the news room (Figure 7.7), will be similar to those of  $D_{L1}$ , however since this is a geosynchronous satellite, it is always in view (if ever in view), and the propagation delay (excluding visibility concerns) is significantly longer. Since the data structure does not change before traversing the links, the processing delay is again set to zero. Since geostationary satellite locations are limited, and antennas are costly links are routinely run at high data rates. When the anticipated data rate is between 256 Kbps and 512 Kbps and the viewing constraints are removed, the delay contributions look as depicted in Figure 7.7.

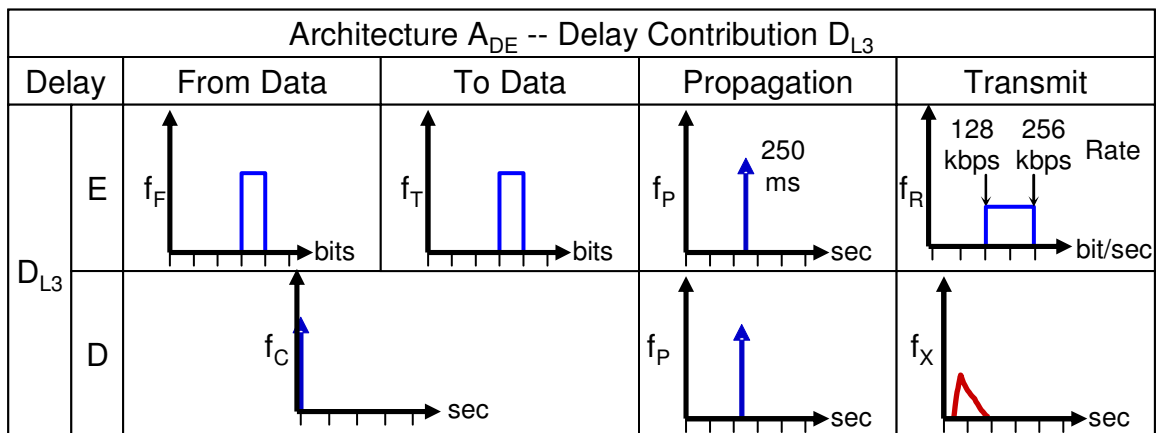


Figure 7.7 Architecture  $A_{DE}$ , characterization of delay  $D_{L3}$

### 7.1.9 Characterizing Delay $D_{L4}$

Link  $D_{L4}$  is similar to the tactical terrestrial network, but has several important differences. The global network will not be store-and-forward. It will rely on link layer retransmissions and TCP windowing to manage correction of the occasional error or link fault, and hence the per-hop delay will be small. The network however will likely be much larger and hence the number of intermediate nodes will be larger. Even though there will be more nodes, the delay at each will be small, and the aggregate will result in both a smaller uncertainty in delay and a smaller mean delay when compared with the tactical network. Figure 7.8 shows this characterization.

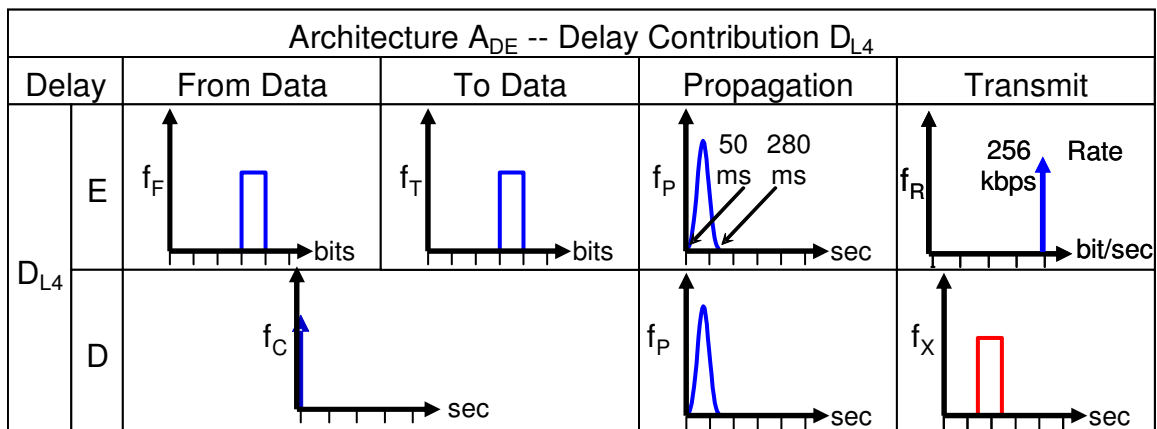


Figure 7.8 Architecture  $A_{DE}$ , characterization of delay  $D_{L4}$

### 7.1.10 Combining Component Performance Descriptions

The problem of selecting the best performing data exfiltration architecture ( $A_{DE}$ ) is solved by applying the PPI to the summed performance descriptions for each of the alternatives as listed in Table 7.2.

Since each of these architecture options is composed of two elements which are connected sequentially, the summation function or convolution discussed in chapter five is appropriate for determining the combined delay. Applying that technique to each architecture option yields the results shown in Figure 7.9

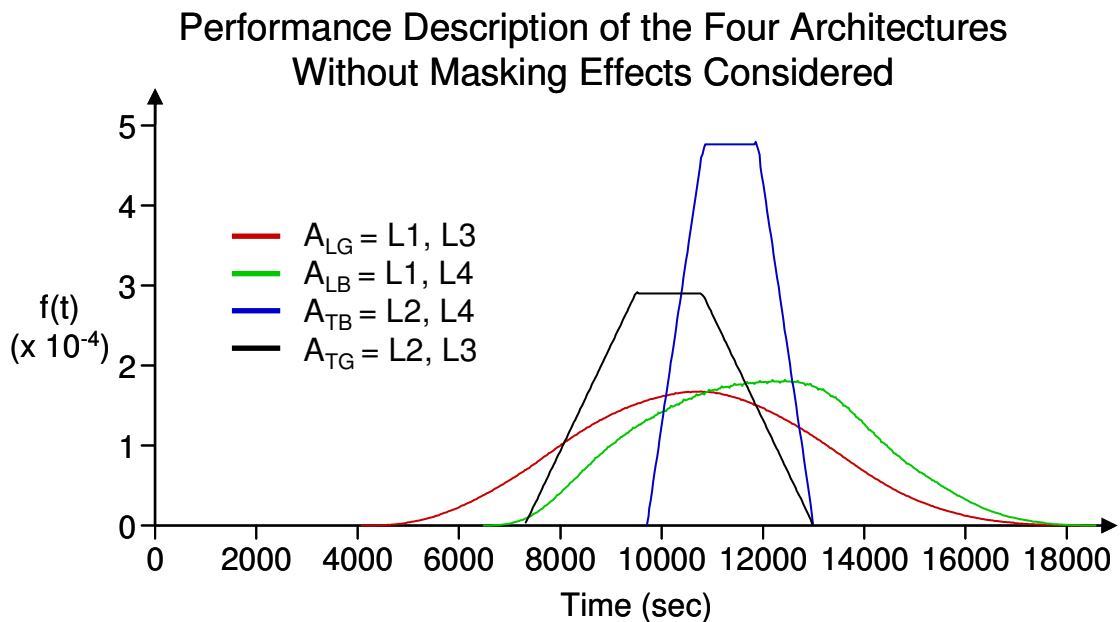


Figure 7.9 Video architecture performance descriptions

### 7.1.11 Comparing the VIDEO Performance of the Alternatives

For each pair of architectures in Figure 7.9, the PPI can be calculated.

$$PPI = \int_a^b F_A(t) \cdot f_B(t) dt$$

Here the limits “a” to “b” identify that portion of the pdf for which the function product has non-zero value. “ $F_A$ ” is the cdf of the architecture A performance description, and “ $f_B$ ” is the actual performance descriptions of architecture B. So the PPI takes two functions and maps them into a real number, the PPI which is the probability that architecture A will produce a better forming system than architecture B. The resulting probabilities are listed in Table 7.3. There the probability is shown that the architecture listed on the left edge of each row performs better than the associated column architecture in that row. In this case,  $A_{TG}$  outperforms the others as it has the highest value in its row for all three of the alternatives.

Table 7.3 PPI Results for Data Exfiltration

### PPI Results for Data Exfiltration

$\diagdown$	$A_{LB}$	$A_{LG}$	$A_{TB}$	$A_{TG}$
$A_{LB}$	---	0.351	0.413	0.244
$A_{LG}$	0.649	---	0.607	0.423
$A_{TB}$	0.587	0.392	---	0.204
$A_{TG}$	0.756	0.577	0.796	---

Prob(row < col)

The performance probability integral provides the set of architectures being analyzed with a partial ordering based on time (delay). Consider a subset of the architecture ordered pairs,  $(A_R, A_C)$ . Architecture  $A_R$  is the architecture listed on the left edge of a table **Row** and is paired with  $A_C$ , the architecture listed at the top of a table **Column**. The subset of pairs to be considered is that is that where the probability listed at the intersection cell is greater than 0.5. When these architecture pairs are mapped onto the positive real line (preserving this partial ordering) then the architecture which ends up on the left end of the so-aligned set will be the best performing. Using the table above and this partial ordering, the architectures will be ordered as:  $A_{TG}$   $A_{LG}$   $A_{TB}$   $A_{LB}$ ,  $A_{TG}$  being the best performing.

### **7.1.12 Example One Summary**

This research's method has been applied to the data exfiltration example. The architectural alternatives were identified. The component performance characteristics were defined. Uncertainties inherent in the problem were incorporated into the analysis at the component pdf generation stage, where performance descriptions are developed for all significant delays. Delays were combined according to the components in each architecture alternative. The PPI was calculated for each performance description pair and determined the probability that tactical-net to geostationary satellite option will out-perform the alternatives.

### **7.2 Example Two: A Service Oriented Architecture Based Service**

A Service Oriented Architecture can be useful in developing applications that require gathering data from a number of different sources. Its flexibility is particularly apparent when the needed data sources might change over time. This example highlights a case where there is a demonstrable relationship between a pair of architectures and the associated performance differences to be expected. The design statement that follows includes a performance description in the requirements specification. This is not necessary for comparing architecture performance on a pair-wise basis. The PPI was developed for those cases where there is insufficient data, or inadequate requirements details to allow for a more specific conclusion to be drawn. The detailed requirements

included in this example have been added since performance criteria are often specified in the requirements process. When system performance criteria are appropriately defined, the techniques developed in this work can be applied to determine whether the initial architecture considered meets that performance specification. In this example, analysis shows that the initial proposed architecture cannot meet its performance requirements. The architecture is modified by adding a redundant Photo Server (to be defined later). After this modification, the performance requirement can be met. While the addition of a redundant information source could be expected to improve performance, the benefit here is that the amount of performance improvement can sometimes be quantified in advance of implementation.

### **7.2.1 Problem Description**

This example assumes that the task is to provide helpful information to a traveler looking for real estate. This traveler needs information to assess different real estate offerings. The information requirements include access to an internet web page that describes potential land or houses to be viewed, a set of driving instructions to get to those places, and an aerial view of the destination neighborhood in order to assess life-style issues like the availability of walking paths or the existence of unsightly landmarks. Since there are a number of locations to be investigated, a web search engine is used to identify probable information sites for real estate to be visited. Photos and directions will be



provided by two other servers. An activity diagram depicting the initial processing approach to accomplishing this task is shown in Figure 7.10. This diagram establishes that the photos and directions are requested after a Search Engine result has been selected. The results from the Photo Server and the Directions Server are returned in an unspecified order. To establish a performance goal, assume that the system requirements specification states that at least 90 to 95 percent of the actions initiated by this system will return their results in less than 500 ms. The activity diagram of Figure 7.10 can be turned into a graph representing the architecture as shown in Figure 7.11.

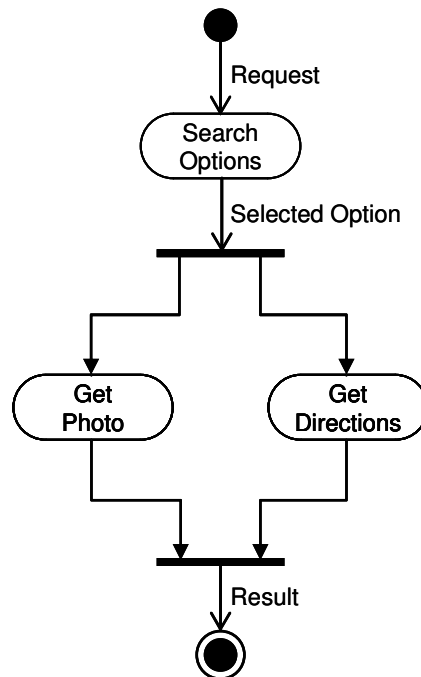


Figure 7.10 Initial activity diagram for SOA real estate service

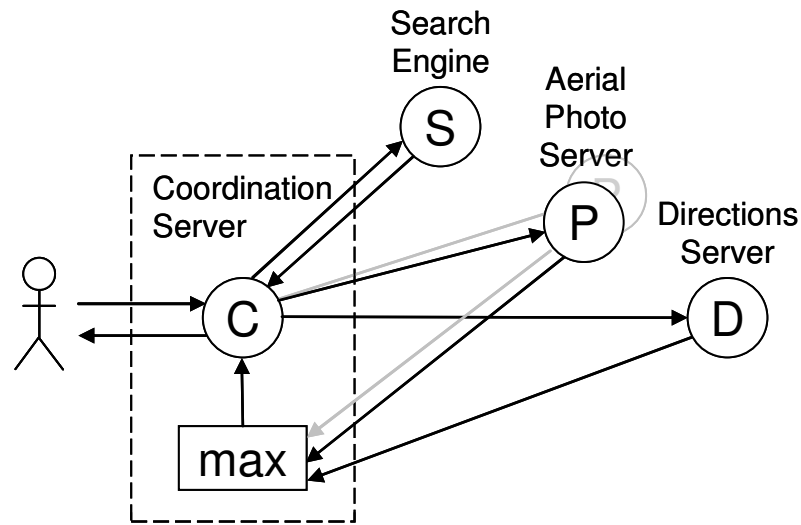


Figure 7.11 Symbolic SOA real estate service architecture graph

The “max” function depicted explicitly in Figure 7.11 is part of the Coordination Server. It ensures that the responses from both the Photo and Directions Servers have arrived before the Coordination Server sends the results back to the requestor. It is called out explicitly in the architecture because the architect wants to rule out implementations that allow the photos and directions to be delivered at different times. This synchronization requirement is important since it influences the performance of the delivered system, and requires additional analysis when compared with an architecture that does not include it. The grayed out P server is a redundant photo server that will be added to the architecture as an improvement when the analysis shows that the initial requirements specification can not be met. In the discussion, delay P is from the

Coordination Server to the Photo Server, and back to the Coordinating Server again (the input to the max function). Similarly, delay D is from the coordination server to the Directions server and back to the coordination server. The equation that describes the delay of the single is:

$$f(t) = S + \text{Max}(D, P)$$

### **7.2.2 Large Grain Delay Descriptions**

Some simplifying assumptions can be applied to this example to reduce its complexity without significantly affecting its usefulness. Assume that the three information servers are approximately the same distance from the Coordination Server so that propagation delay differentials can be ignored. To help build the performance estimates, assume that all of the content providers are located on a terrestrial network, and that the location of each is known. Assume further that the required servers exist so that performance descriptions can be measured. Alternatively assume that server performance has been characterized in a performance specification so that there is a basis for constructing server performance descriptions.

### **7.2.3 Delay Characterization**

In view of the assumptions made above, approximate the inter-server communications delays to be 50 ms. Were those assumption not appropriate,

(i.e., if the locations were unknown), or if known but different distances from the Coordination Server, the analysis would proceed as it did for the delay analysis conducted in the first example of this chapter.

#### **7.2.4 Data Characterization**

The request that initiates each system sequence of activities is considered to be a small string of about 100 bytes. Since the Directions Server is providing directions as an HTML page, assume page length to be uniformly distributed from 400 - 600 bytes. Assume that the size of the web page returned from the search engine is uniformly distributed from 2000 - 5000 bytes, and assume that the size of the compressed aerial photo is uniformly distributed between 5000 and 8000 bytes. As with all of the examples presented, these numeric characterizations of sizes are estimates. When using these techniques in a true analysis, they would be adjusted to fit the actual circumstances of the problem being solved.

#### **7.2.5 Delay Characterization to the Photo Server, P**

There are a number of standards for internet photo formats, and they vary largely in size, detail included, and compression techniques. If there were preferred servers to be accessed or there was specific information available regarding the size of the pictures to be returned this information would be included here. Lacking that information, consider that a range of values was selected. The time

to convert each photo request into a result is assumed to be uniformly distributed from 100 - 200 ms. The one-way delay to the Photo Server is 50 ms so the total propagation delay is 100 ms. The transmit rate for the entire network is assumed to be 256 Kbps. Based on these assumptions, the important delays are characterized in Figure 7.12.

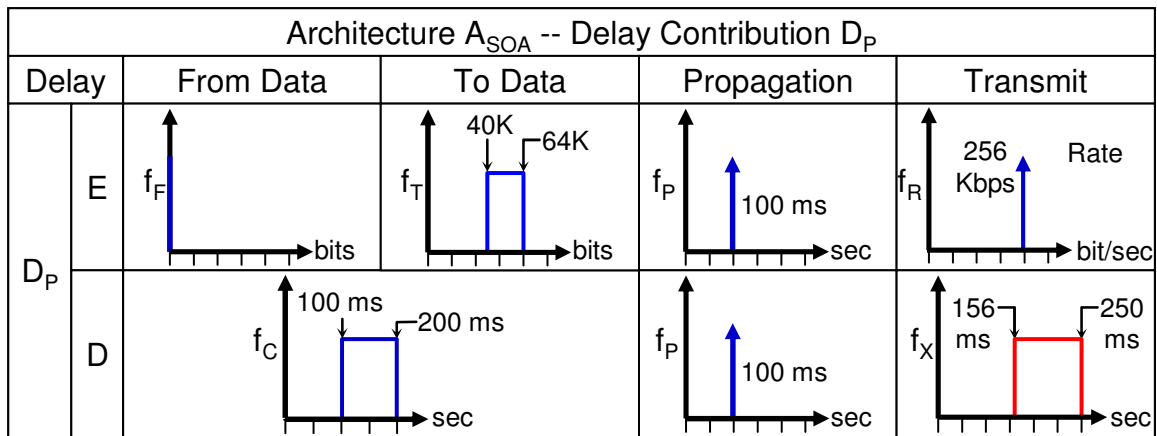


Figure 7.12 Architecture SOA, Photo Server delay

### 7.2.6 Delay Characterization to the Search Engine, S

Assumptions for the performance of the search engine are made in a similar manner. The anecdotal information available about the time to process search requests indicates that there may be some caching of frequently requested

queries so there is likely a lower limit on the time to produce a result. At the same time, it will take longer to gather the requested information on a complex search. As a result, the data conversion time for the search engine is modeled as an Erlang distribution with a mean of 100ms. Since the data rate of the network is assumed constant, the transmit delay is deterministically related to the size of the data being sent.

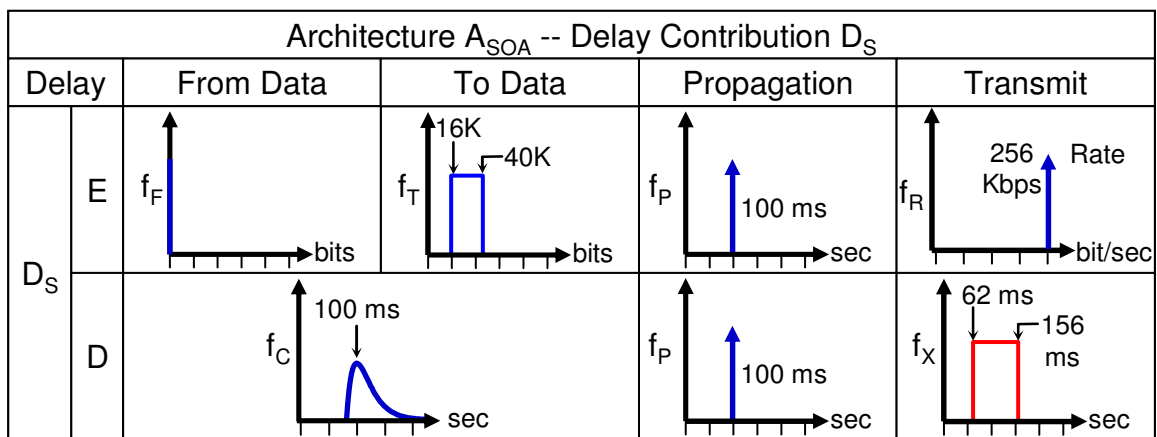


Figure 7.13 Architecture SOA, Aerial Search Engine delay

### 7.2.7 Delay Characterization to the Directions Server, D

The situation with the Directions Server is similar. There is no specific information to help generate better performance characteristics at this time.

What is known about directions is that sometimes they are simple and short, and sometimes long and complex. Routinely however there are a set of a dozen or so steps in navigating the roads. Since few descriptions will be very short and similarly only a few very long, yet many of intermediate length, a normal distribution was selected to describe the Directions Server performance. To keep the analysis simple, the inclusion of other information like businesses in the area, etc. has been assumed to be filtered out before transmission.

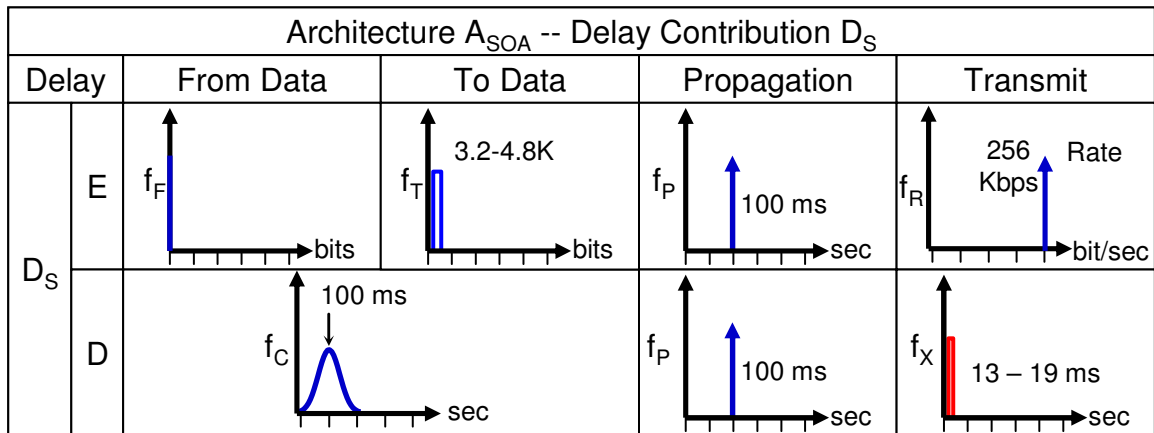


Figure 7.14 Architecture SOA, Directions Server delay

### 7.2.8 Combining Server Delays

The lower half of each of the Figure 7.12 – Figure 7.14 depicts the individual contributions for the total processing for each server. In each case, these three contributions are summed to generate the individual cumulative server delays. These cumulative delays that result are shown in Figure 7.15. These would be the performance descriptions (red – Directions, green – Photo, and blue – Search) anticipated before encountering the “max” function. Note that in each case the cumulative delay is the convolution of the three contributions. The equation for the system delay is:

$$f(t) = S + \text{Max}(D, \text{Min}(P_1, P_2))$$



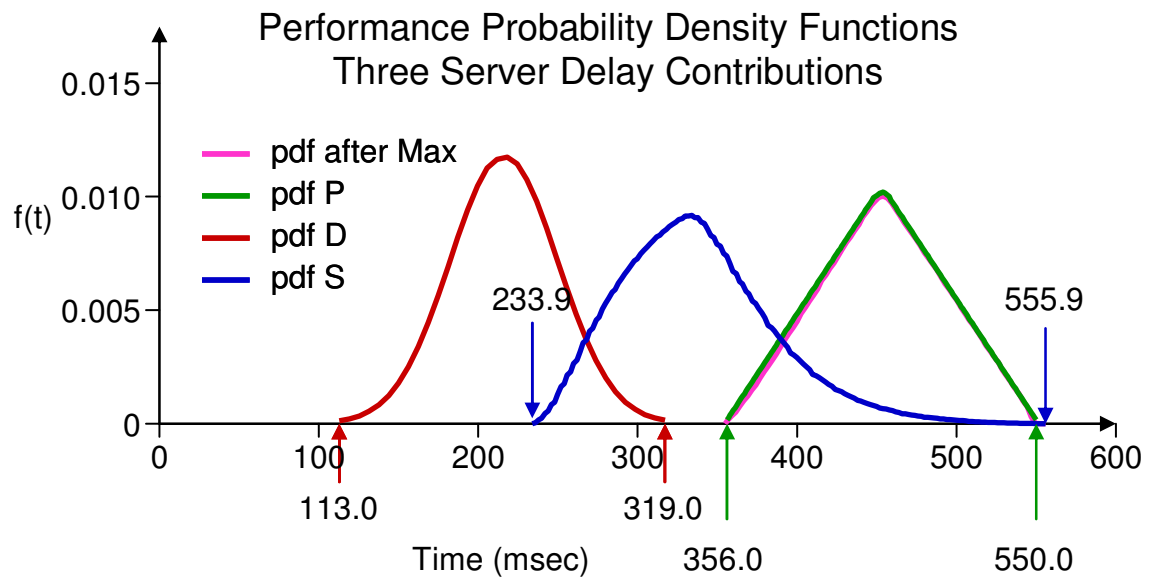


Figure 7.15 Cumulative results for describing the delays of each server

The partially obscured pink trace describes the result after applying the maximum function of the Coordination Server. This obscuring is due to the fact that the Photo Server is significantly slower than the Directions server. It is this pdf that describes the overall performance of the complete architecture described. The computation that generates the pdf of the combined system is actually generated from the cumulative distribution function (cdf) of the combined system. This cdf for the total delay can be used to assess compliance with the performance requirements specification. When comparing the expected performance of this architecture with that in the performance specification stated up front, the original architecture can not achieve the desired goals, as shown in Figure 7.16.

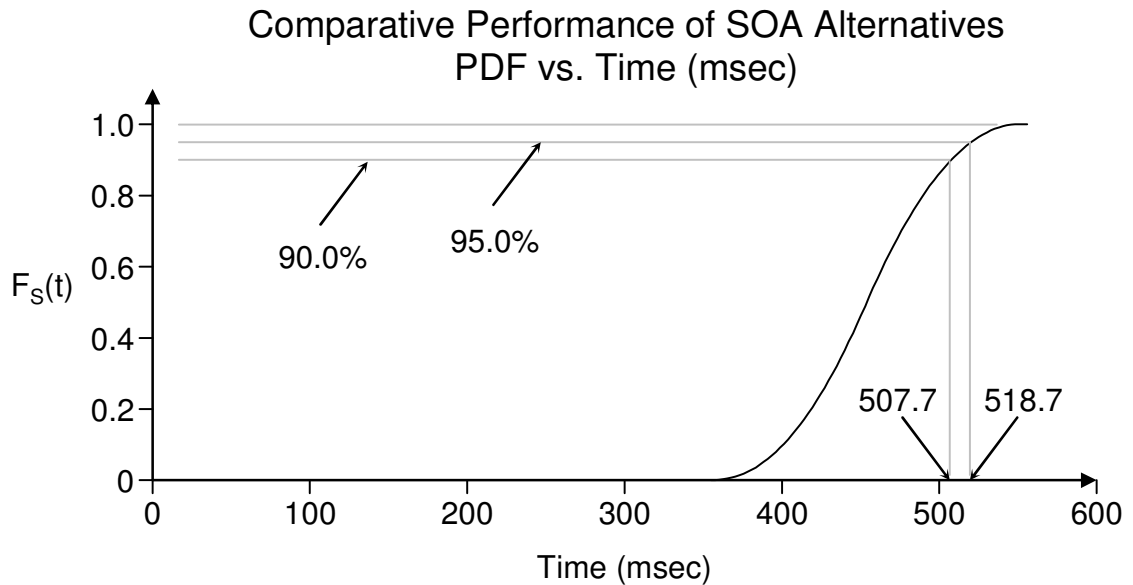


Figure 7.16 Projected performance of original example SOA

One option for correcting this failure to meet the performance goals is to change the architecture. While there are many ways that such a change could be made, the one selected here is based on observing that the Photo Server performs least well. While any of the servers could be changed to potentially improve performance, replicating the Photo Server with another that works with the same performance description appears promising. The max function of the Coordination Server can proceed when either of the two Photo Servers returns its value. In essence, the parallel combination of Photo Servers will be combined with the “min” function and the execution continues as previously described. When two independent but similarly performing Photo Servers are used in this

way, the combination performance of the pair is as reflected in Figure 7.17. The dual Photo Server architecture is shown to improve system performance.

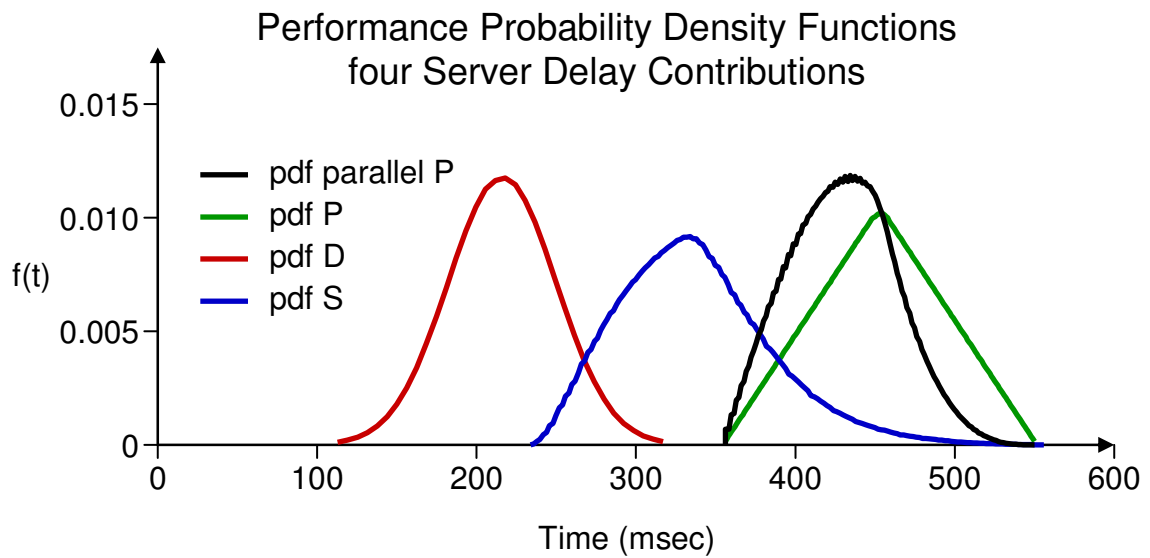


Figure 7.17 Server contributions with parallel Photo Server

Combining the performance of all four servers now leads to the performance demonstrated in Figure 7.18.

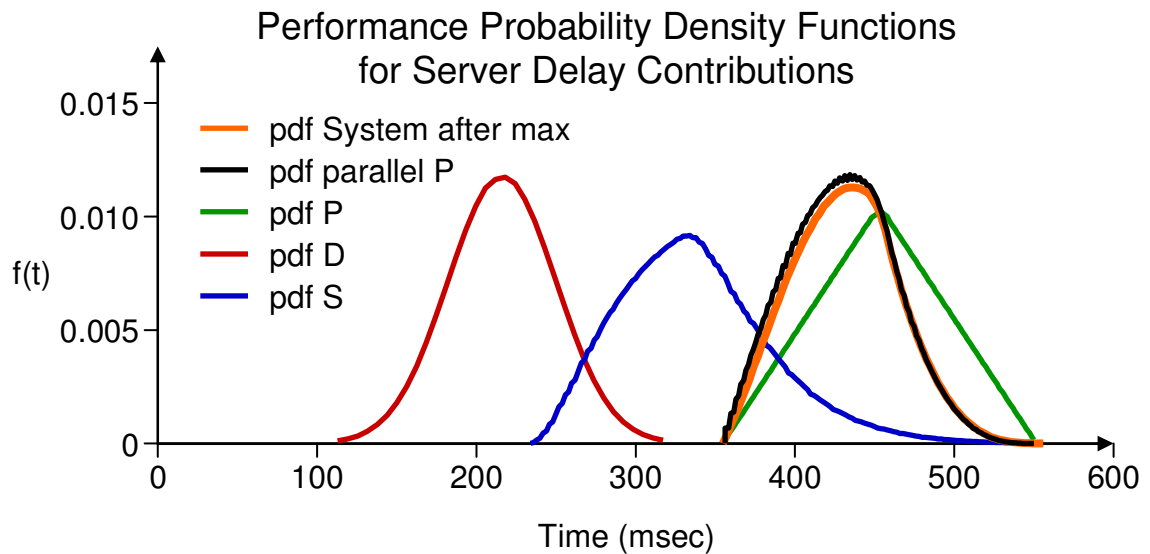


Figure 7.18 Performance of parallel Photo Server architecture

In a manner similar to above, the cumulative distribution of the combined architecture can be used to assess compliance with the performance requirements specification, as shown in Figure 7.19. The performance specification is expected to be met. It should be noted that there are some uncertainties in the values that are projected by this method. Future research would be appropriate to get a better understanding of the magnitudes of these uncertainties. When the PPI is calculated for these two systems, the probability that the parallel D version will perform better than the non-parallel version is 0.664 or the redundant version will perform better almost two thirds of the time.

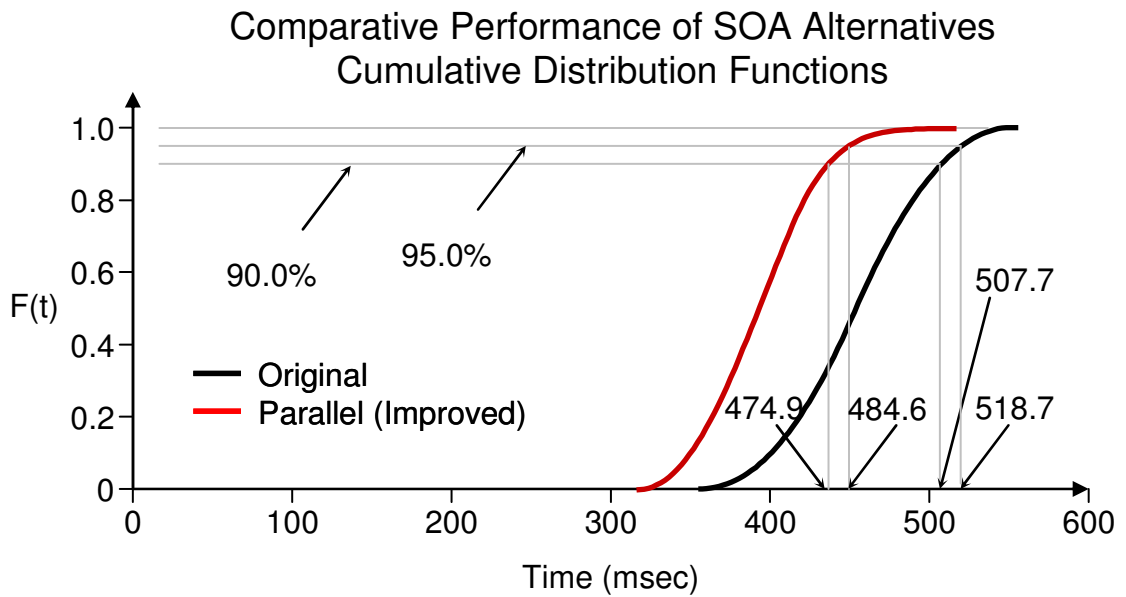


Figure 7.19 Projected performance of modified example SOA

### 7.2.9 Example Two Summary

Example two followed the same methodology as example one. In this case however, the analysis showed that the initial performance specification would likely not be met. After modification of the architecture by adding in a parallel Photo server, a recalculation of the expected performance was done; the improvement was quantified and an acceptable performance was achieved.

### **7.3 Summary**

This chapter has presented a detailed analysis of two simple but still realistic problems. The analysis techniques developed earlier in this work were applied to proposed architecture solutions. Conclusions were obtained for the relative performance capabilities of alternative architectures in each case. In many cases, there was no performance information available for the estimation of a component's performance. Best available knowledge was used as an estimate. The use of assumptions is recommended in the absence of concrete information. This process was designed to be simple to execute so that as knowledge about components and connections matures it can be easily rerun to provide improved insight into the likely performance of "to be" generated implementations.

## **Chapter 8**

### **The CAPE Tool**

#### **8.1 Introduction**

In the previous two chapters, examples were presented to demonstrate the analysis method. Without computational support, such analysis is laborious and error prone. As part of this research effort, a tool was developed to assist the analyst in the manipulation and visualization of probability density functions describing the performance of architecture alternatives. The tool provides three useful capabilities: 1) it provides implementations for the methods (sum, minimum, maximum, quotient, and composite) for combining the probability density functions describing component and connection performance into a probability density function describing the system architecture's anticipated end-to-end performance, 2) it provides a method for evaluating the PPI for pairs of architectures using the pdf descriptions developed, and 3) it generates Microsoft PowerPoint macros which can be used to create figures and diagrams for explaining the performance behavior of the systems being evaluated.

## **8.2 The CAPE Tool Design**

The CAPE tool-development strategy was guided by two objectives. The first was to leverage commercial off-the-shelf and open source software artifacts. The second was to provide the analyst with a flexible tool that could easily incorporate new functionality and permit modification of existing code templates. The outcome of following this approach is that CAPE is built on the capabilities of the Sun NetBeans® IDE in combination with Microsoft PowerPoint®.

A set of analyst-editable code snippets or templates constitute the largest part of the tool's user interface. The analyst uses the NetBeans IDE to modify and combine these code snippets with two libraries, FUNC, and PLOT (developed as part of this research) to exercise the functions described in chapter five and support visualization of architecture performance analysis. The specific performance and topology characteristics of each architecture are edited into the templates. The compiled set of modified templates is then executed to produce MS PowerPoint (PPT) macros which run in PowerPoint to generate the graph portion of analysis figures.

The conceptual output of the tool is an object of class "Figure." The Figure class incorporates many conventional diagram and figure attributes. The actual output of the tool is a macro which will draw a Figure object in MS PowerPoint. Figure



class attributes include size, color, axes, freeform object, and lines, well as others one would expect in modeling a traditional document figure.

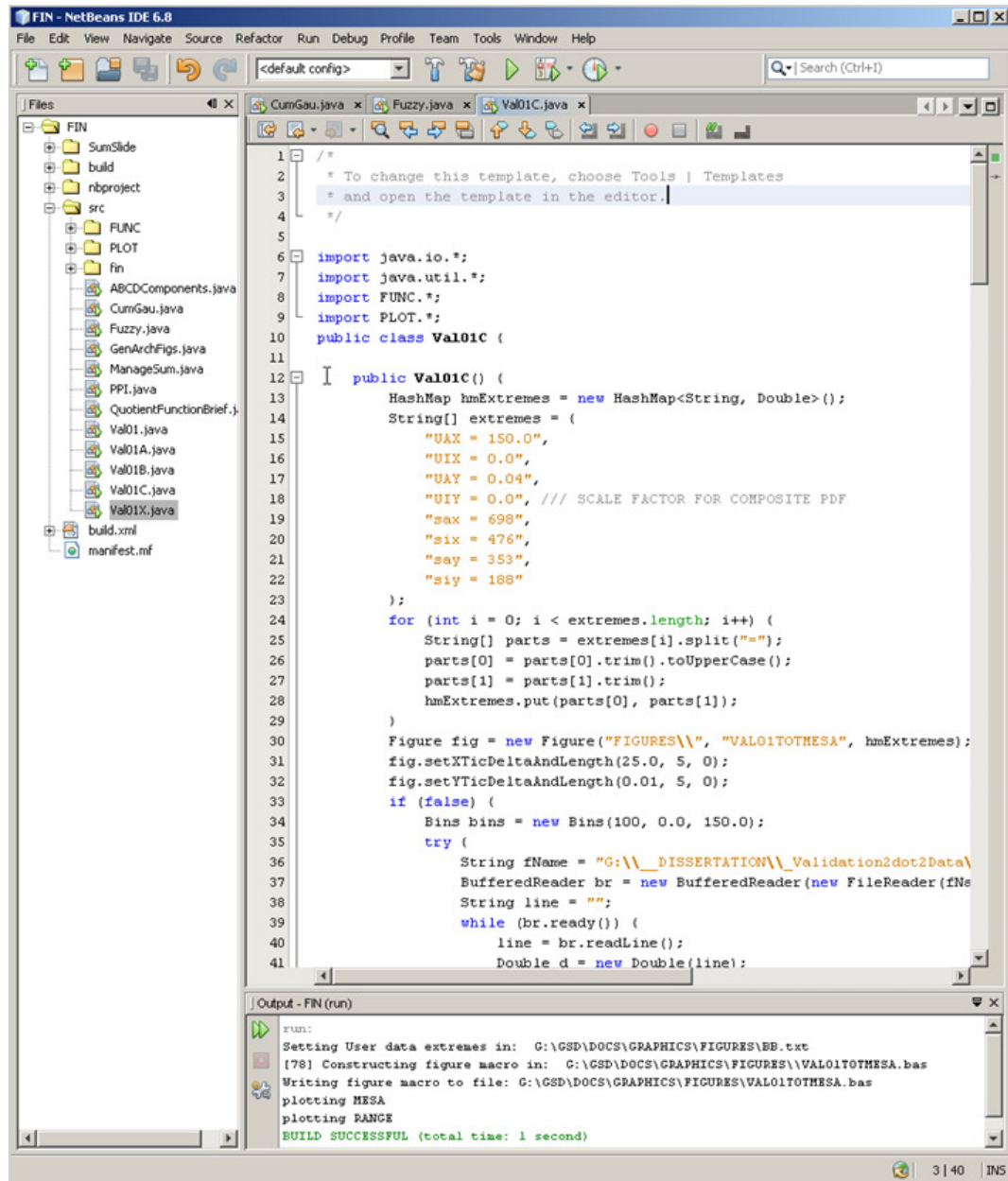


Figure 8.1 NetBeans IDE with partial code templates

A top level IDE screen shot is shown as Figure 8.1. The code templates that are shown in the editor window will be explained later in this chapter. The figure demonstrates with a partial example how code templates are combined into an executable program which can be run within the NetBeans IDE to produce the desired results. Intermediate results are displayed in the lower window labeled “Output.” This “Output” window logs error conditions that arise, and for longer computations displays a progress status as the computations proceed.

CAPE provides two analysis capabilities: 1) it can generate end-to-end architecture performance descriptions, and 2) it can compute the PPI given a set of architecture performance descriptions. The figures below show two analysis capabilities of the CAPE architecture. Different code snippets are incorporated into the IDE depending on the analysis capability being exercised. Figure 8.2 shows the structure of architecture performance description analysis. Component and connector performance descriptions are specified in terms of “Functions,” a class that can be used to characterize known uncertainties. The architecture topology is represented by code that uses the five combination functions described in chapter five to merge the component and connector performance descriptions into an end-to-end architecture performance description.

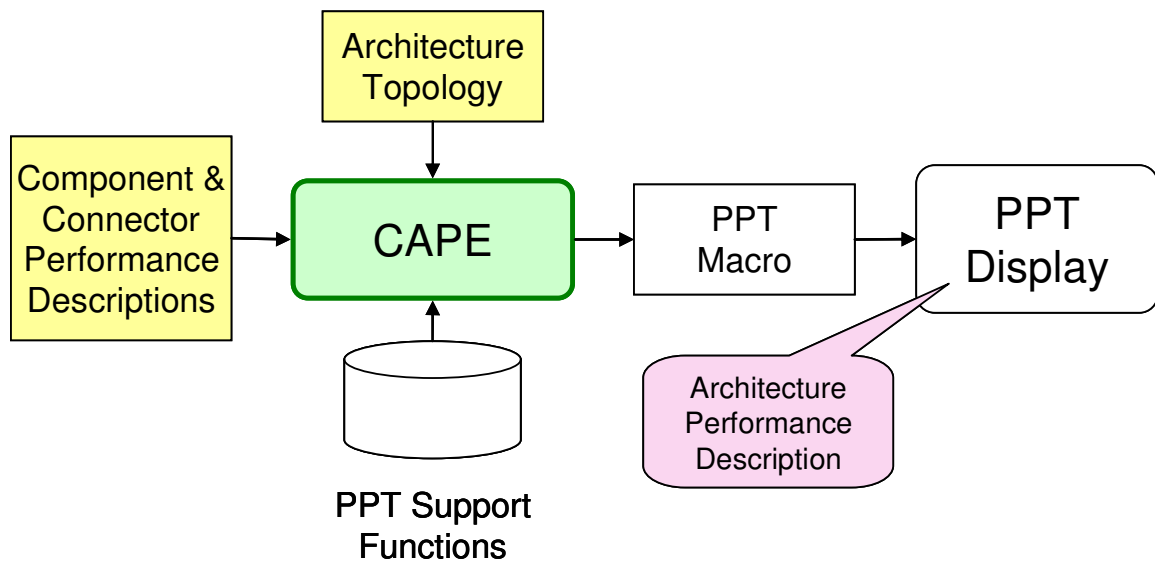


Figure 8.2 CAPE tool structure - PDF analysis

Figure 8.3 shows the architecture shows the structure of PPI analysis. The architecture performance descriptions identified on the figure are those generated with CAPE from component and connector performance descriptions. The “Comparative Descriptors” are implemented as sets of names of architecture performance descriptions that are to be pair-wise compared. Pairs of architecture performance descriptions are input to CAPE methods to produce a table of probabilities indicating the likelihood that one architecture will produce an implementation that will perform better than an implementation of the second.

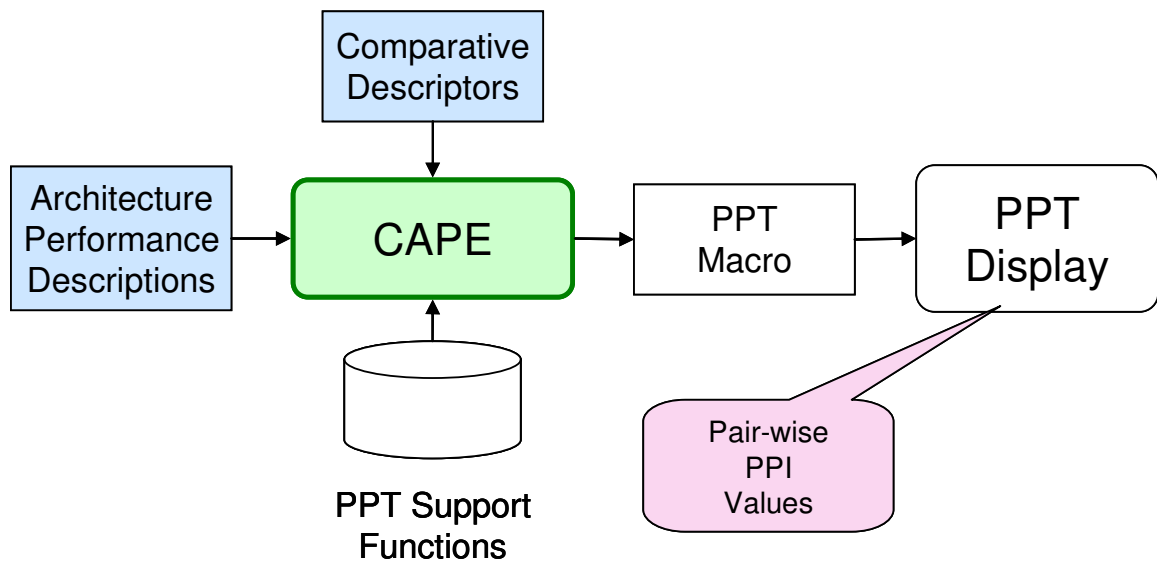


Figure 8.3 CAPE tool structure - PPI analysis

### 8.3 Code Snippet Functionality

Each performance analysis done with CAPE is structured in essentially the same manner. Routinely, five code snippets are needed. One of the code snippets is relate to the establishment of screen and user coordinate spaces. A second is associated with the formatting of diagrams and figures for incorporation into document. A description of the other three snippet types follows next.

### 8.3.1 Function Definitions

The second code snippet usually defines the probability density functions to be used in the analysis. These functions can be generated in either of two ways: 1) by reading stored file descriptions from persistent storage or 2) generating them directly within the code segment by enumerating function values point by point. Figure 8.4 shows how to construct a function point-wise within code. In this case, the constructor takes a single argument, the name to be assigned to the function. Line five of Figure 8.5 shows how to read the description from a file. In this case, the Function constructor takes two arguments, the file name containing the function description, and the name to be assigned to the function

```
Function fun = new Function("MESA");  
fun.addFP(45.75, 0); // Adding function points here  
fun.addFP(47.25, 2);  
fun.addFP(48.75, 13);  
// ... Many data points deleted  
fun.addFP(129.75, 6);  
fun.addFP(131.25, 2);  
fun.addFP(132.75, 0);
```

Figure 8.4 Building a Function point-wise within the code

```

0 // Setting of extremes deleted for brevity
1 String funName[] = {
2     "D12", "D23", "D34", "D45", "D54", "D43", "D32", "D21"
3 };
4 Function fun[] = new Function[8];
5 for (int i = 0; i < funName.length; i++){
6     fun[i] = new Function("G:\\GSD\\DOCS\\GRAPHICS\\"
7         +"FIGURES\\"+funName[i] + ".f()", funName[i]);
8 }
9 Figure fig = new Figure("FIGURES\\", "VAL01TOT",
10     hmExtremes);
11 fig.setXTicDeltaAndLength(25.0, 5, 0);
12 fig.setYTicDeltaAndLength(0.01, 5, 0);
13
14 Function fX = new Function("F5");
15 fX.addFP(0.2, 0.0);
16 fX.addFP(0.4, 2.5);
17 fX.addFP(0.6, 0.0);
18
19 Function f01 = Function.convolve(fun[0], fun[1], 0.1);
20 Function f23 = Function.convolve(fun[2], fun[3], 0.1);
21 // ... other function colvolutions deleted
22 Function f0123X4567 = Function.convolve(f0123X, f4567,
23     0.5);
24 f0123X4567.name = "F0123X4567";
25
26 f0123X4567.setTau(-1.5); // Set delays = 0.25 * 6.0
27 double area = f0123X4567.integFmTo(0.0, 150.0, 0.25);
28 f0123X4567.setASF(1.0/area);
29 f0123X4567.makeThisConnonical();
30
31 fig.addFun(f0123X4567);
32 fig.close();

```

Figure 8.5 CAPE pdf functions validating the chapter nine Army example

### 8.3.2 Modeling Topological Aspects of the Architecture

The topology of the architecture defines two aspects of the problem solution. It describes the sequence of components and connectors through which data

passes to transform data from a system input into an output. It further identifies the synchronization points of the algorithms used to realize the architecture. Both the relationships between the elements (components and connectors) of the architecture and the architecture synchronization requirements are represented with sequences the code functions discussed in chapter five. Quotient, sum, and composite realize data transformations, minimum and maximum accomplish process synchronization. All of these functions are provided in the FUNC library. For example, to combine sequential connector and component delay probabilistic descriptions into a combined description the sum function would be used. Figure 8.5 shows an example where a suitably defined array of Functions is convolved together to generate the probabilistic sum. This sum is then shifted by a fixed delay (`setTau()`) and scaled to ensure it conforms to the requirements of a pdf.

This example is taken in part from the Army communications example used as methodology validation in chapter nine. The code lines are numbered for reference and used in the following description. Lines one through five define names for the functions to be manipulated and read these functions from files. Lines six through eight define the figure which will result from the execution of the code and include both X and Y axes. Lines nine through 12 define the pdf describing function F5 and build this triangular pdf in a point-wise manner. Lines 15 and 16 generate a new function as the sum of two others. Line 17 shifts that pdf by a fixed delay. Lines 18-20 scale the function to ensure it is compliant with

the definition of a pdf, and put the function in standard form. Line 21 adds the function just created to the figure, and finally line 22 causes the figure macros to be generated to file. More complex examples would likely incorporate other functions as well to describe a more complex architecture topology.

### 8.3.3 CAPE Evaluation of the PPI

The fifth code snippet is related to the computation of the PPI. The evaluation of the PPI is straightforward. When many function comparisons are needed it is useful to put functions in an array and build the cumulative distributions functions similarly. The PPI can then be calculated by iterating the formula values through the arrays. A non-iterated version is shown in Figure 8.6.

```
// Set the extremes data structure
// Construct Figure fig and add axes
double area = 0.0;
// Define the two functions
Function A = new
    Function("G:\\__DISSERTATION\\_Brief\\aRes.f()", "A");
Function B = new
    Function("G:\\__DISSERTATION\\_Brief\\bRes.f()", "B");
// Compute cumulative distribution functions from the pdfs
Function Acum = A.getCumulative(25.0);
// Perform the PPI calculations
Function fun = Function.multiply(Acum, B, 25.0);
Double ppiVal = fun.integFmTo(0.0, 4500.0, 25.0);
System.out.println("PPI: " + ppiVal);
fig.close();
```

Figure 8.6 CAPE used to compute PPI



## 8.4 Library Support

There are two libraries associated with CAPE. The first is the FUNC library. The evaluation techniques that evolved to implement the library functions were based on representing the functions to be manipulated as sets of piece-wise linear approximations to real functions, and then performing the calculations on a point by point basis. To handle even the simple cases, the end points representing these linear segments approximating the true functions had to be general enough to handle finite discontinuities. A representation was established that associated with each domain value, a set of range values, and properties that indicated whether each was closed or open. With this data structure to define segment end points, the function was represented by an ordered list of segments. Mathematical operations between functions were then implemented on a point by point basis. Integrations were performed by application of the trapezoidal rule for area computation. Division is not used. Addition is straightforward as is multiplication by a constant (positive or negative constant to represent subtraction). Multiplication of functions as needed for convolution and is implemented as an exact quadratic using to the end point of linear piecewise approximations of the functions being used as arguments. The library also provides the capability to manipulate functions (shift in time, scale in amplitude, generate cumulative distribution function from pdf, generate pdf from cumulative distribution function, put functions in standard or canonical form, etc.). It provides methods that can be put together to implement the topological aspects of the

architecture, for example, convolve to implement sum, and two argument implementations of minimum, maximum and quotient. The composite function is implemented as a weighted “add” of probability density functions. More details of the FUNC library are included in Appendix A.

The second library is PLOT. This library provides all of the support routines required to generate PowerPoint graphs without the need to understand the underlying PowerPoint Visual Basic for Applications. The plotting process identifies which PPT support functions and subroutines are needed to perform the plot and includes these automatically in the macro generation portion of the computation. The macros are generated in the file that was identified in the Figure constructor. This file is then imported into PPT and executed from within the VBA Project pane of the VBA code editing window. More PLOT library details can be found in Appendix D.

#### **8.4.1 Results**

The code templates report progress information about the computational processes to the Output window of the NetBeans IDE. The CAPE output-box content from the calculation of Figure 8.5 is shown in Figure 8.7.

```

run:
Setting User data extremes in:
    G:\GSD\DOCS\GRAPHICS\FIGURES\BB.txt
[78] Constructing figure macro in:
    G:\GSD\DOCS\GRAPHICS\FIGURES\Ch8EX.bas
[1336]      TRI      REC
Convolution iterations = 300
TRI      REC      39 --> 300    Five progress indicators
TRI      REC      94 --> 300
TRI      REC     168 --> 300
TRI      REC     237 --> 300
TRI      REC     283 --> 300
Writing figure macro to file:
    G:\GSD\DOCS\GRAPHICS\FIGURES\Ch8EX.bas
plotting TRI
plotting REC
plotting CONV_TRI_REC
BUILD SUCCESSFUL (total time: 29 seconds)

```

Figure 8.7 CAPE text output from example

#### 8.4.2 A Simple Complete Example

The example that follows, Figure 8.8, generates two functions. The first comes from a stored file on disk; the second is generated point-wise within the code. These two functions are then added probabilistically (convolved) to show the sum, and then all three functions are plotted on a common graph. The graph in Figure 8.12 is exactly as it comes out of the PPT macros. Axis labels and graph labels have to be added separately in PPT.

```

// First section - Defining the Bounding Boxes
HashMap hmExtremes = new HashMap<String, Double>();
String[] extremes = {
    "UAX = 5.0", "UIX = 0.0",
    "UAY = 1.3", "UIY = 0.0",
    "sax = 600",
    "six = 400",
    "say = 500",
    "siy = 300"
};
for (int i = 0; i < extremes.length; i++) {
    String[] parts = extremes[i].split("=");
    parts[0] = parts[0].trim().toUpperCase();
    parts[1] = parts[1].trim();
    hmExtremes.put(parts[0], parts[1]);
}
// Second section - Defining the Functions used
Function tri = new Function("TRI");
tri.addFP(1.5, 0.0);
tri.addFP(2.5, 1.0);
tri.addFP(3.5, 0.0);
tri.setColor(255, 0, 0);
Function rec = new Function("REC");
rec.addFP(0.5, 1.0);
rec.addFP(1.5, 1.0);
rec.setColor(0, 0, 255);
Function sum = new Function("SUM");
// Third section - Defining the figure to hold the
// visualization
Figure fig = new Figure("FIGURES\\", "Ch8EX", hmExtremes);
fig.setXTicDeltaAndLength(1.0, 5, 0);
fig.setYTicDeltaAndLength(0.2, 5, 0);
fig.addFun(tri);
fig.addFun(rec);
// Fourth section - pdf manipulation, generate the sum
// (convolve)
sum = Function.convolve(tri, rec, 0.01); // 0.01 =
// resolution
sum.setColor(51, 153, 102); // DARK GREEN
fig.addFun(sum);
// Fifth section - Generate the PPT macros
fig.close();

```

Figure 8.8 CAPE input for example analysis

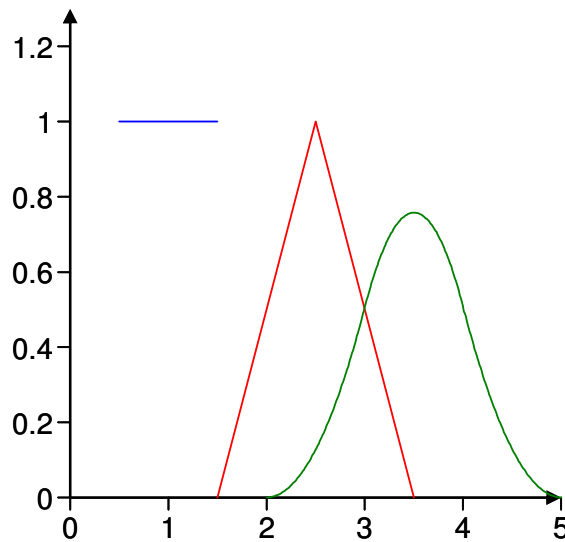


Figure 8.9 Raw macro result once executed within PPT

### 8.4.3 Summary

The CAPE tool provides three types of calculation capability: 1) support for the generation of and manipulation of probability density functions that represent the delay characterizations of components and connections, and 2) a set of plotting support macros written in Visual Basic that simplify the plotting of figures describing the performance graphs generated, and 3) support for evaluating the PPI for a set of architectures. The CAPE tool is an integration of five major pieces. Two are commercial or open source: Microsoft's PowerPoint, and Sun's NetBeans. The other three pieces were developed specifically within this research effort. The function library (FUNC) provides class descriptions for

representing probability density functions, described in more detail in Annex A. This includes implementations of manipulation functions needed to generate end-to-end delay performance descriptions. A plotting library (PLOT), described in Annex D, which is a set of Visual Basic macros to manipulate the plotting of the graphs generated by CAPE. And finally the set of code templates which define the user and screen coordinates, generate functions from files or point-wise in code, manipulate pdf delay descriptions, and guide the computation of the PPI from end-to-end architecture delay descriptions.

## **Chapter 9**

### **Methodology Validation**

#### **9.1 Introduction**

The ideal technique for validating CAPE, the methodology presented in this research, is that of selecting a large set of architectures and instantiating each of those architectures a large number of times. One could then make performance measurements on each implementation and combine them (grouped by parent architecture) into architecture specific performance descriptions. Once the architecture performance descriptions were determined, the PPI could be validated by comparing the PPI computed result with the probabilistic (frequency based) individual implementation performance measurements. Unfortunately however, this approach is not feasible. Implementation costs are too high. Reduction to a small number of implementations does not help either. The implementations selected might not well represent the spread of possible performance values that could be achieved with a large number of implementations. Taking measurements of real world systems is even more restrictive. There would be only a single instantiation to consider. An alternative validation approach must be developed.

Since the PPI results were extensively validated in chapter four and the implementation of the functions are validated in Appendix C, what remains to be validated is the combination of functions that are used to build-up architecture-performance descriptions. The heart of such a validation effort centers on confirming that results generated from the CAPE methodology agree with results from other methods known (or assumed) to be valid. The Department of Defense (DoD) often uses, a modeling and simulation technique based on MESA [MITR05] for examining performance issues under a broad set of circumstances. For this validation effort, two problems are evaluated using the MESA tool to validate CAPE's results. The first problem is taken from an Army communications program and exercises the sum and quotient functions. I do not have access to the data of any other real-world problem to validate the other functions. As a result, the example problem defined in chapter seven of this work is modeled in MESA for results comparison. It may be noted that the first example is dominated by the communications delays in the architecture. This is not surprising as it is a communications system. However there is no loss of generality in the validation as communications and computational delays are treated in exactly the same manner within the analysis method. The second example (while not based on real-world data) does demonstrate the incorporation of significant computational delays.



Within this chapter, section 9.2 describes the MESA modeling technique and the next section introduces two result comparison techniques. The first technique is based on hypothesis testing, and the second leverages the concept of a norm from vector space analysis. Section 9.4 validates two examples, the Army communications problem, and the real estate problem discussed in chapter seven. CAPE and MESA model results are compared. The chapter closes with an explanation of why hypothesis testing is not more useful in this situation and explains the normed vector result.

## **9.2 Modeling Technique**

Each of the two validation examples that follow are treated in a similar manner. A discrete event simulation was built using the graphical model construction capability of Extend 7 [Rive98] coupled with the MESA discrete event simulation engine libraries.

Extend is a customizable graphically-oriented general purpose modeling and simulation environment which can handle both discrete and continuous modeling tasks. Simulation elements are “dropped” onto the screen and connected graphically in a manner similar to Lab View®. Random processing times can be allocated to simulation elements. Resources are tracked through queues and plots are available showing queue lengths, queuing times, etc. There is a

scripting language associated with MESA that manages sensitivity analysis by rerunning simulations with varying parameters.

MESA is a toolkit designed to run on Extend and is specifically designed to support end-to-end performance analysis of services being developed in a Service Oriented Architecture. MESA provides ready-to-run library components that are specifically designed for SOA analysis. The combination of MESA components running on the Extend simulation system infrastructure provides a unified, flexible, verifiable performance estimate of the problems being modeled.

### **9.3 Result Comparison Techniques**

Statistical testing is routinely performed on experiment sample data to decide whether to accept or reject a hypothesis. For this validation effort, the appropriate hypothesis to be tested is whether or not the two models suggest the same performance for the architecture being analyzed. After collecting data on the two models, a Chi Squared Goodness of Fit test can detect differences. Hence an alternative technique based on normed vector spaces is used to gauge the amount of difference between the two performance estimates.

#### **9.3.1 Chi Squared Goodness of Fit Testing**

One of the techniques routinely used to determine whether or not two densities are the same is the Chi Squared Goodness of Fit test. This test is formulated to

decide which of two hypotheses is likely to be correct. Alternative  $H_0$ , usually captures the hypothesis that there is no difference between the two densities and  $H_1$ , represents that there is a difference. To perform the actual test, samples from the two densities are collected and a statistic is first computed then compared with a threshold to establish whether or not there is statistical evidence to reject  $H_0$ .

The basic Chi Squared goodness-of-fit-test has three steps. The first step breaks down the range of values to be processed into sub-ranges (bins) and compares the number of observed values in respective bins to the number of expected values in those bins. To ensure a correct result, a set of “rules of thumb” are applied to structure the test. First, no bin should have fewer than five values; second, there should be 50 or more values in the total sample [LeRS01], and third, for best results, the bins should be configured so that each bin has approximately the same number of expected values [FiHe10]. In this case, there is no known-to-be-true set of values so the observed values from the CAPE description could be compared with the observed values of the MESA description. The formula for doing so is as follows.

$$\chi^2 = \sum_{i=1}^k \frac{(o_k - e_k)^2}{e_k}$$

The  $k$  summation is over the set of bins chosen for the test.

The second step is to define the “level of significance” of the test. The level of significance of a test is often represented as alpha ( $\alpha$ ) and represents the probability of making a Type I error. A Type I error is the probability of falsely rejecting  $H_0$ . Associated with a specific value for alpha, and the number of degrees of freedom (related to the number of bins) is a tabulated threshold value. When the Chi Squared value calculated from the formula exceeds the tabulated threshold,  $H_0$  is rejected.

In a more typical test, the number of sample values taken is usually limited by cost, i.e., the time to make the measurements or the dollars needed to be spent to collect the data. Here samples can be generated quickly and cheaply. The question remains how many data points to use in the comparison. The power of the test is the ability of the test to reject a false null hypothesis. This power improves by taking a larger number of samples. At least one thing is clear. Since there are some differences between the model results, the test will eventually be able to differentiate between the two when enough samples are generated

### 9.3.2 Norm Based Difference Measurement - MESA vs. CAPE

The study of normed vector spaces provides an approach for describing the size of the difference between two performance estimates. Given a vector space,  $X$ , with elements,  $x$ , any function that maps an element of that vector space into a real number can be called a norm,  $\|x\|$ , if it satisfies the following axioms:

1.  $\|x\| \geq 0$  for all  $x \in X$ ,  $\|x\| = 0$  if and only if  $x = \theta$ .
2.  $\|x + y\| \leq \|x\| + \|y\|$  for each  $x, y \in X$ .
3.  $\|\alpha x\| = |\alpha| \|x\|$  for all scalars  $\alpha$  and each  $x \in X$  [Lune69].

The set of all continuous functions on a closed interval  $[a, b]$  of the real line is a vector space and one of the useful norms on that vector space is:

$$\|x\| = \int_a^b |x(t)| dt$$

The types of probability density functions that are generated by CAPE and MESA are both in this space. This norm can be used to measure the difference between the performance descriptions.

## **9.4 Validation Examples**

For the purpose of validating this work, two examples have been selected which cover a substantial part of the interesting architecture modeling space. The Army example takes unclassified data from a real program office. The Real Estate Service example follows chapter seven.

### **9.4.1 Example One – The Army Tactical Environment**

The first example is taken from an engineering study done for the Defense Information Systems Agency by MITRE Corporation. It demonstrates the most commonly used functions of the model (sum and quotient) in the context of developing services in a Service Oriented Architecture environment. The two functions, sum and quotient, dominate most service performance analysis done within the DoD since current SOA services do not broadly support dynamic service discovery. Most complex services are developed by identifying a set of statically discovered functional services and executing them sequentially.

This particular engineering study was done as part of a recent Enterprise Wide Systems Engineering (EWSE) task that established guidelines for service delivery in a SOA environment and used the Army tactical communications infrastructure as the specific architecture for the analysis. The goal was to generate guidelines that program developers and evaluators could use to assess the performance of new and proposed services. These results would support

tradeoff analysis between cost, performance and sometimes schedule when developing program alternatives. Previously such performance analysis work had been done in an ad hoc manner. This task developed a structured analysis approach that could be consistently applied in many environments. Consistency in analysis is valued as implementing services in the SOA structure is seen as an important step in moving the Department of Defense to a net-centric communications environment.

While the CAPE method presented in this work provides for including many types of quantifiable performance uncertainty, e.g., the capability of the development team, uncertainties about component location, etc., the MESA tool does not. To make the results of both models comparable, these more general types of uncertainties were not included in this validation work.

A specific architecture was selected as a subset of the broader EWSE task and examined in detail. That subset of the larger architecture analyzed in detail is identified and highlighted in yellow in Figure 9.1. That figure shows the communications path that connects a Squad element to a Combatant Commander's (COCOM) location using a sequence of devices: local tactical vehicle, adjacent tactical mobile communications center, intermediate satellite, and finally the COCOM termination equipment. The path to return the service result is similar using the same communications elements in the reverse order.

The analysis steps identified in chapter five were applied to this sequence of equipment elements. The performance characteristics are unclassified values.

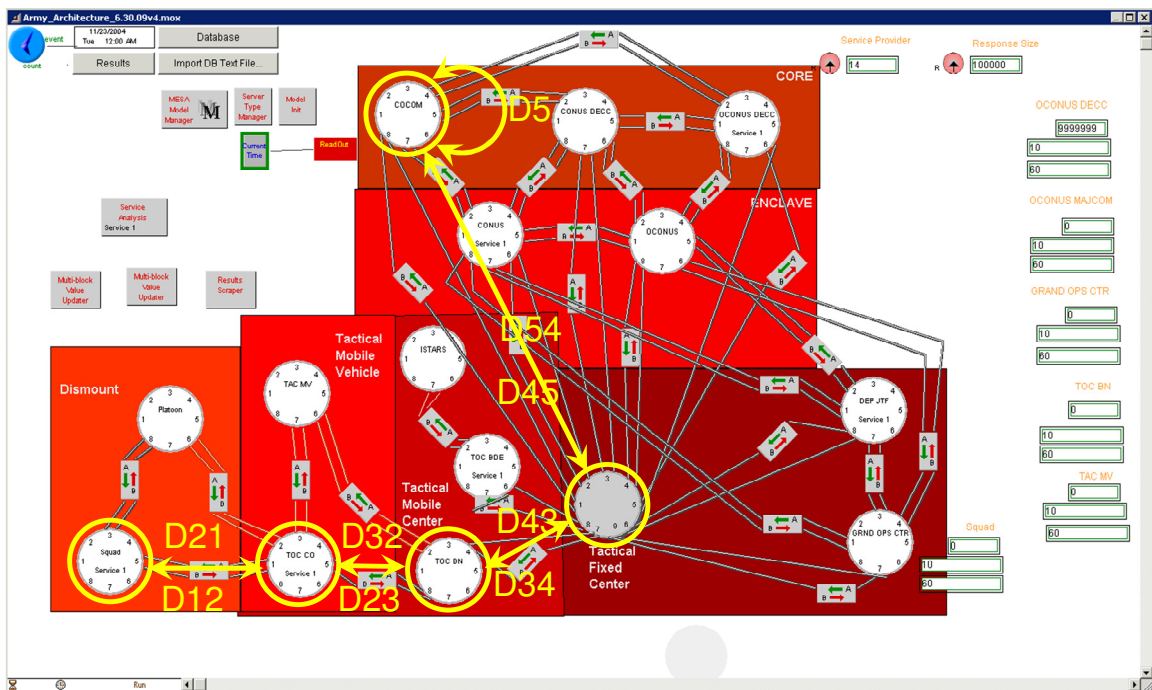


Figure 9.1 Top level MESA model for Army SOA example

The performance description for this system can be calculated from the following:

$$DelayTotal = D_{12} + D_{23} + D_{34} + D_5 + D_{54} + D_{43} + D_{32} + D_{21}$$



Each of the delays identified in Figure 9.1 was translated into a component or connection and the delays were modeled using the CAPE methodology.

### Architecture Transmission Delay Contributions


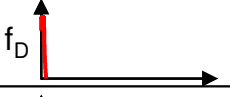
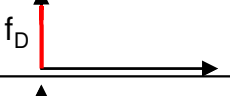
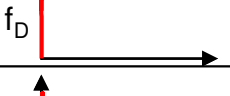

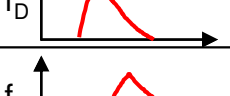
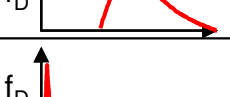
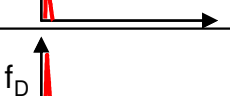
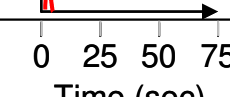
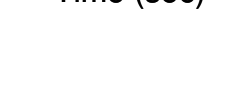
Comp/Conn	Min (sec)	Max (sec)	Scaled PDF
D12	0.008	1.280	
D23	0.013	2.000	
D34	0.001	0.128	
D45	0.001	0.128	
D5	0.200	0.600	
D54	16.000	48.000	
D43	25.000	75.000	
D32	1.600	4.800	
D21	1.600	4.800	
CUMPROP	1.440	1.440	

Figure 9.2 Detailed CAPE delays with bounding values

To simplify the comparison of delays estimated by CAPE and MESA analysis techniques, the fixed propagation delays of each connection were separated out from the associated transmission delays (as they do not change the transmission delay pdf shape) and were added back in to generate the final result in the bottom row of Figure 9.2 under the heading of CUMPROP.

Figure 9.2 shows the CAPE delays. The CAPE result is generated by summing the nine element delays (D12-D21) with the sum function and adding in the CUMPROP value. Figure 9.3 shows the MESA and CAPE results.

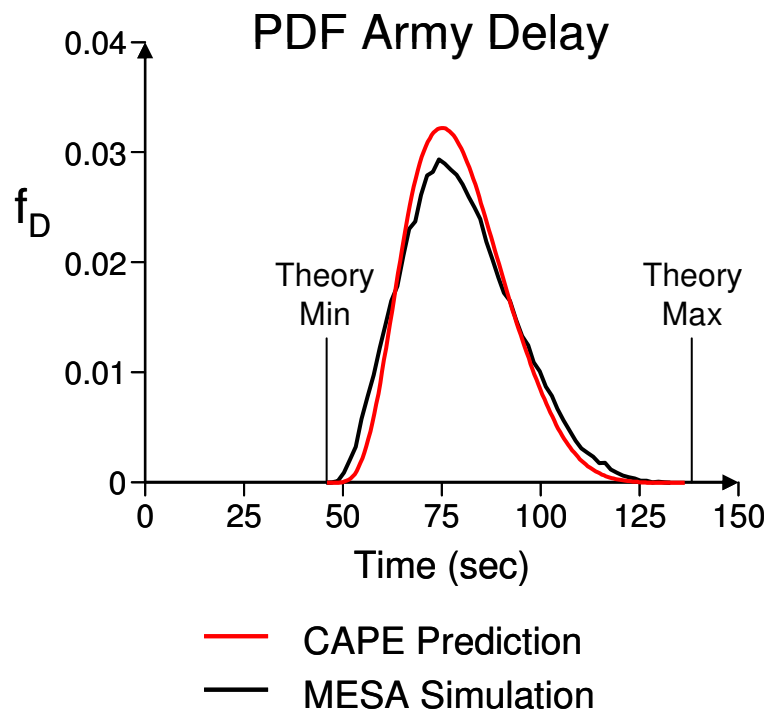


Figure 9.3 PDF comparison of CAPE and MESA results Army example

While the two predicted performance density functions are very similar, they can be distinguished with hypothesis testing (section 9.5.1). Using vector analysis techniques (section 9.3.2) the difference can be approximated as 7%. Since the MESA model has been validated against other models before taking on the EWSE engineering study, this consistency in shape and location indicates that CAPE has produced a correct result for this example.

#### **9.4.2 Example Two – Real Estate Service**

The second example, a real estate service, demonstrates some of the different CAPE functions that are used when the topology is not completely sequential. The performance characteristics for this validation were taken directly from chapter seven.

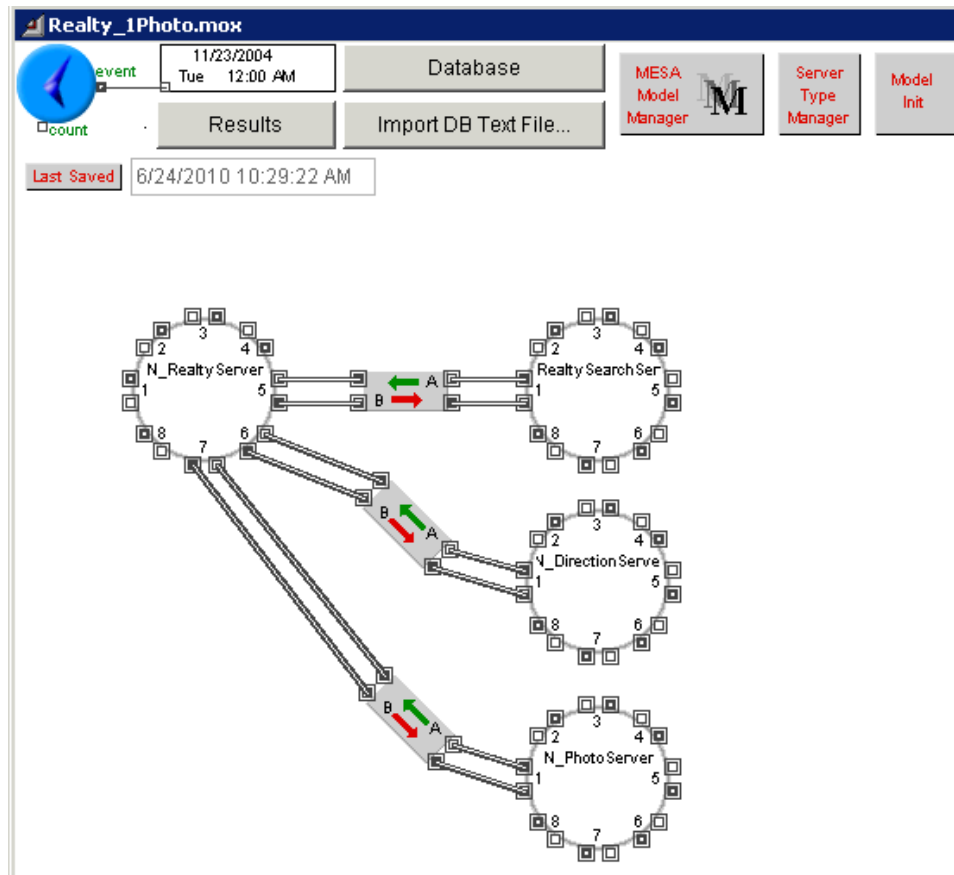


Figure 9.4 MESA model for the real estate service (single photo server)

The performance description for this system can be calculated form the following:

$$DelayTotal = D_{RealtySearch} + MAX(D_{DirectionDelay}, D_{PhotoDelay})$$

In a manner similar to example one, two MESA/Extend models were developed. Figure 9.4 shows the single photo server case. The dual photo server case is analogous (but not shown). The CAPE calculated architecture delay descriptions for the single photo server case are shown in Figure 9.5 and the total or cumulative delay is shown in Figure 9.6.

### Real Estate Example with **Single** Photo Server Delays

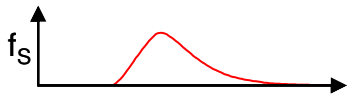
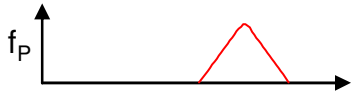
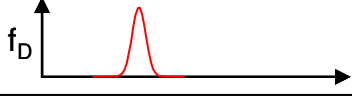
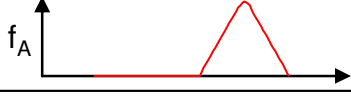
Delays Comp/Conn	Theoretical Min (msec)	Theoretical Max (msec)	PDF (different scales)
Search Function	165.625	1054.375	
Single-Photo Function	359.375	553.125	
Directions Function	115.625	321.875	
Max Direction & Single Photo	359.375	553.125	

Figure 9.5 Single photo server delay contributions

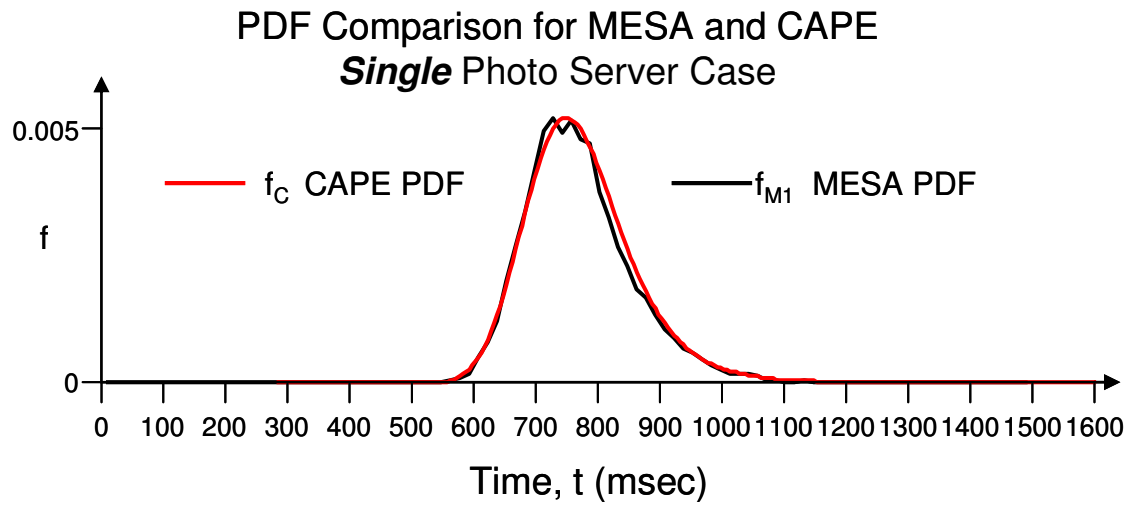


Figure 9.6 Total delay for single photo server

Similarly, the transmission delay contributions and the total delay for the two photo server case are shown in Figure 9.7 and Figure 9.8 respectively.

### Real Estate Example with **Dual** Photo Server Delays

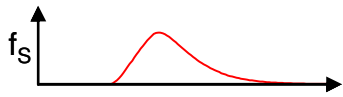

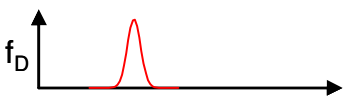

Delays Comp/Conn	Theoretical Min (msec)	Theoretical Max (msec)	PDF (different scales)
Search Function	165.625	1054.375	$f_S$ 
Dual-Photo Function	359.375	553.125	$f_{DP}$ 
Directions Function	115.625	321.875	$f_D$ 
Max Direction & Dual Photo	359.375	553.125	$f_A$ 

Figure 9.7 Transmission delay contribution for the two photo server case

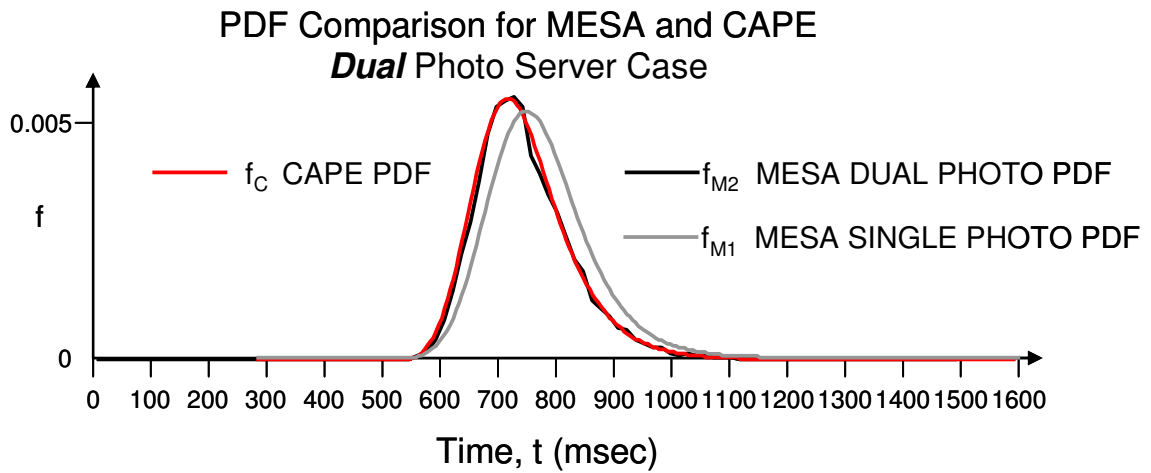


Figure 9.8 Cumulative delays for the two photo server case

In each case, the CAPE predictions reasonably match the MESA simulation estimates of the performance of the real estate service, and the service as modified with the additional photo server. This closeness of predictions again tends to validate that the CAPE methodology makes a reasonable estimate of the anticipated implementation performance.

## **9.5 Comparing MESA and CAPE Results Quantitatively**

When comparing the results obtained from the MESA simulation and the CAPE calculation, one clearly sees that the two have a strong resemblance. However, a more quantifiable comparison may be possible. There are at least two possibilities: statistical comparison (hypothesis test) and difference measurement with a suitable norm.

### **9.5.1 Hypothesis Testing - MESA vs. CAPE**

The validation process conducted for this research is a little bit different than for a routine Chi Squared testing circumstance. In comparing the results here, we are not cost constrained with regard to how many sample data points are collected. Section 9.3.1 identified commonly used rules of thumb for performing Chi Squared testing. To be consistent with these rules, the test process collects sample values until the rules of thumb are satisfied, i.e., more than 50 samples, and no fewer than 5 in a bin, etc. After these rules have been satisfied, there is a choice, to be made: a) evaluate the Chi Squared statistic, or b) gather more



sample points and then evaluate the statistic. Figure 9.9 shows the results of an experiment where each of the 1000 dots represents the selection of a random seed followed by the production of random samples according to the CAPE pdf and the MESA pdf until the rules of thumb are satisfied. Once the rules of thumb are satisfied, the  $\chi^2$  value is plotted against the sample count.

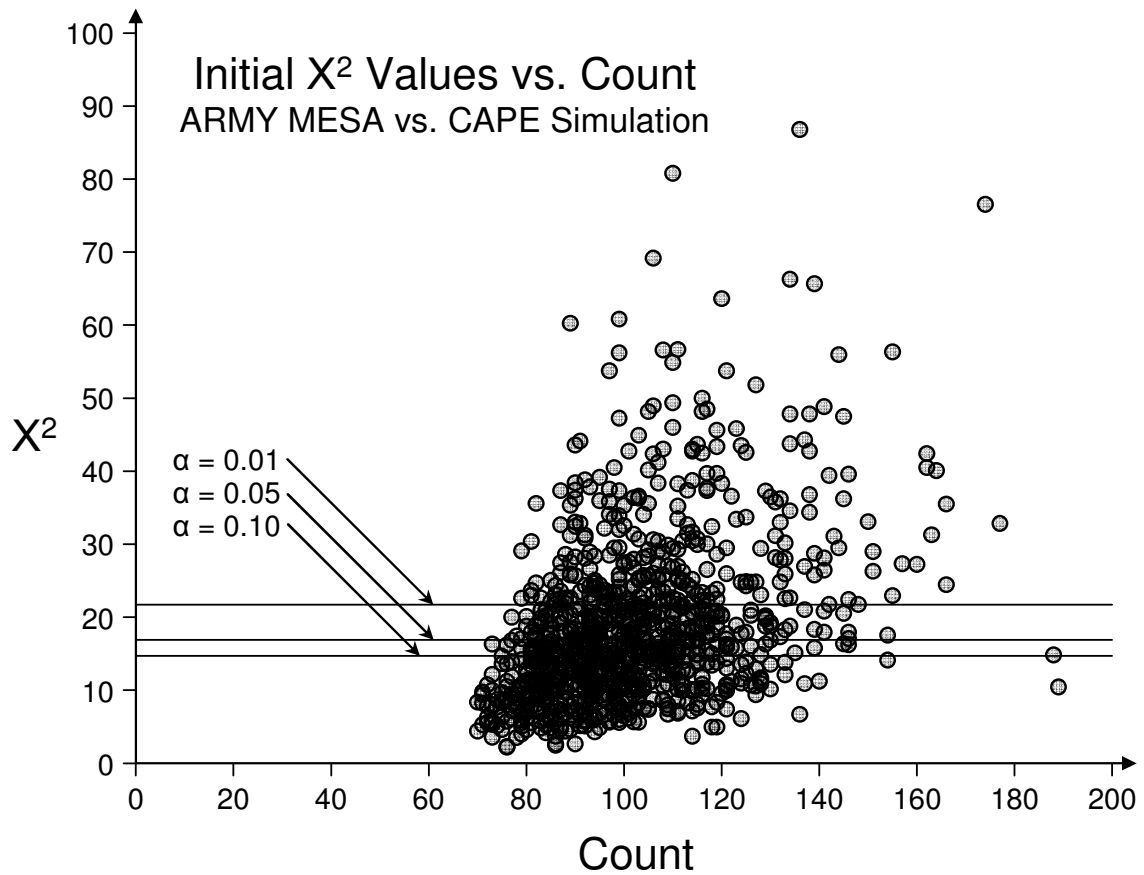


Figure 9.9 Initial  $\chi^2$  values vs. count needed to meet rules of thumb

Of the 1000 initial points plotted, 59% exceed the threshold for rejecting  $H_0$  at the 10% level of significance, 47% exceed the threshold for the 5% level, and 29% exceed the threshold for the 1% level.  $H_0$  should be rejected. The result is the same for the Real Estate Service example.  $H_0$  should be rejected. Differences between MESA and CAPE pdfs drive this behavior. More samples worsens the situation, Figure 9.1. Traces near the extremes highlight the variability.

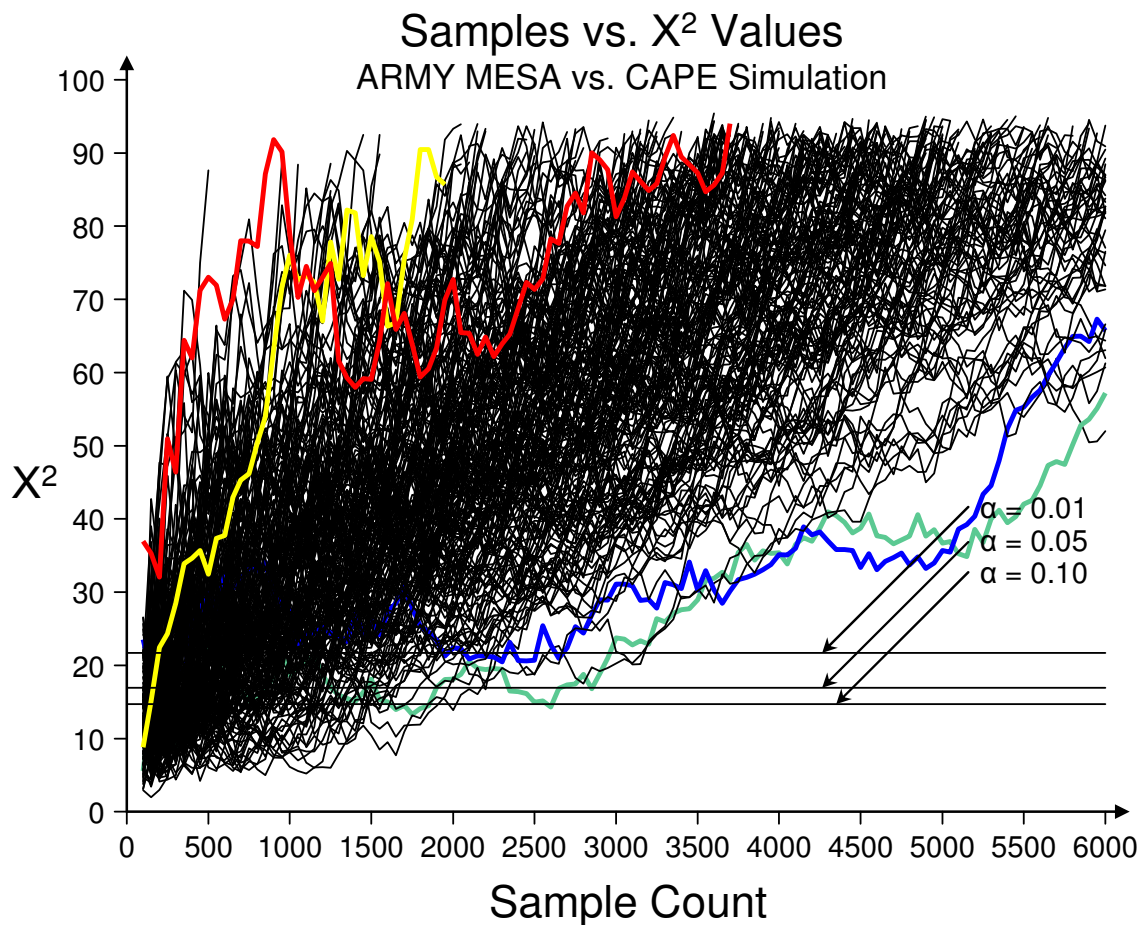


Figure 9.10 Plot of  $\chi^2$  values over time

When the number of available comparison points is large, the Chi Squared Goodness of Fit test does not help very much in establishing the near equality of the two models. As a result, an alternative approach is used based on normed vector spaces as discussed in section 9.3.2.

The probability density functions generated by MESA and CAPE in this situation are continuous and of finite range so they do constitute a vector space. Using the norm identified in section 9.3.2 one can take the absolute value of the difference between the MESA performance estimate and the CAPE estimate in the Army communications example. The result is plotted in Figure 9.11 (in blue).

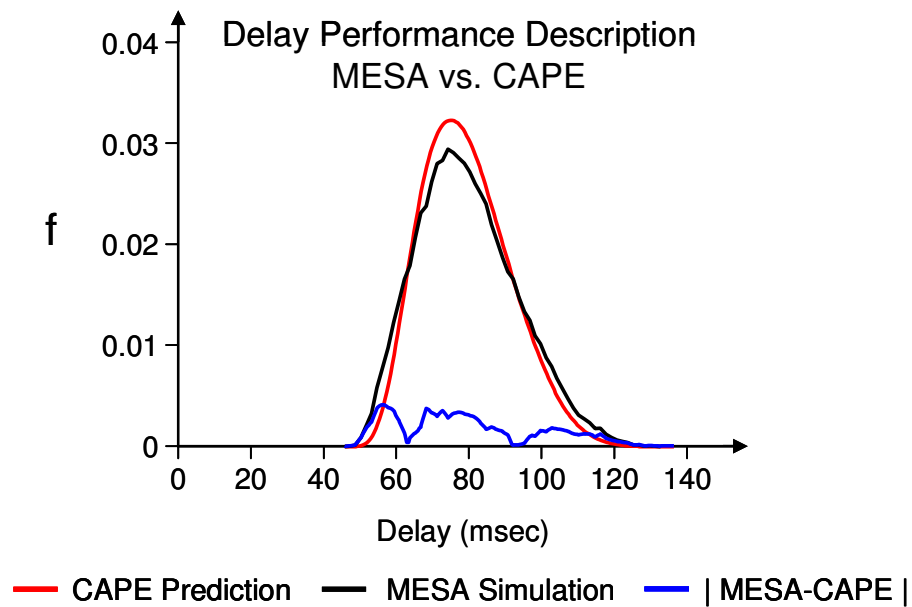


Figure 9.11 Absolute MESA-CAPE estimate difference— Army example

When the blue curve is integrated over its range, the value is 0.1375. Now since the two original curves were probability density functions whose area must integrate to one, the maximum difference possible for these two curves would be 2.0. The minimum difference would be 0.0 if the two curves were the same. Hence the difference between the MESA and CAPE performance estimates differ by approximately 7% for this example. Neither performance description is truly known to be the correct value, as both are performance estimate approximations of the same system (yet to be completed). Using this method, the performance estimate difference for the Real Estate Service with single photo server, Figure

9.6 is only 3.1%, and the difference between the two estimates for the Real Estate Service with dual photo server, Figure 9.8, is 3.3%.

In absolute terms, a 3% or 7% difference between models could be considered large or small depending on the concern of the moment. A portion of this comparison difference is generated by the “wiggles” in the performance estimate function generated by MESA. MESA is a discrete event simulation, and as such, the shape of the curve will change over time as more values are added. There is nothing in the underlying physics of the systems that would suggest that such “wiggles” are an artifact of the systems, but rather they are generated as part of the modeling technique. As such the “wiggles” can be ignored, and a smoothed MESA prediction would be even closer to the CAPE estimate.

The Army example however is a bit different. The MESA function is slightly broader (and consequently shorter) than the CAPE function. The difference is not huge, but it is observable. Yet even this difference might be considered insignificant in view of the fact that the current version of the PPI only considers the probability that one implementation will likely perform better than another. It does not yet assert that it will be better by some significant amount, nor by a specifiable amount. Further, the PPI and the CAPE analysis technique was to help identify performance differences between architectures. When the projected

differences are small, it is highly likely that the actual selection should be made on another quality criterion, i.e., portability, security, etc.

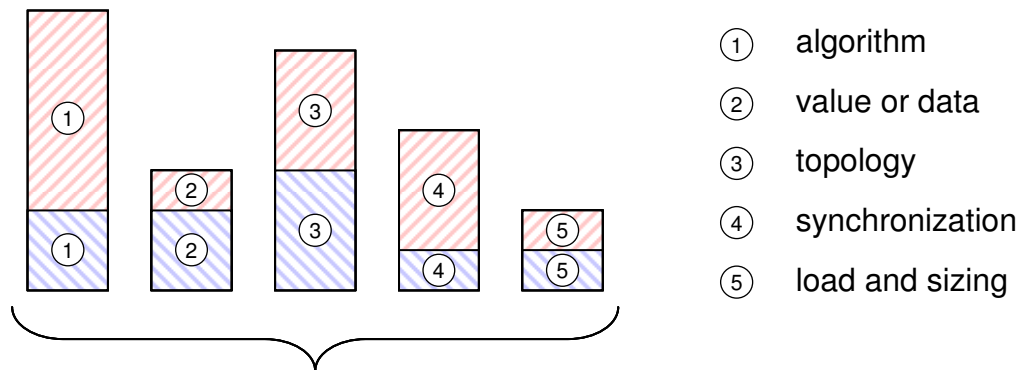
The two approaches described for evaluating the closeness of the MESA and CAPE predictions show that *in this case* the normed vector space method is more helpful. This will not always be the case. When other methods of estimating performance are used, it is likely that the cost of gathering data points will be more expensive, and as such, there may be many fewer. In such cases, the Chi Squared testing will likely be more effective. It will however still only tell whether or not there is a detectable difference, not how big that difference is (as does the normed vector approach).

## **9.6 Conclusion**

The result of the detailed analysis validating CAPE predictions against the MESA predictions needs to be put into perspective. It is clear that these two models produce slightly different results and when sufficient numbers of data points can be gathered, the Chi Squared test can distinguish between them. Differences between models are to be expected, as models are just approximations of reality and it is the approximation aspect of the system reality that generates these differences. In the case of modeling the performance of the “to be built” system, true reality can not be known in advance. Differences between models will likely exist for all but the simplest cases.

To appreciate the importance of these differences, one must revert back to the intent of the analysis. The principal reason for generating performance models at the earliest stages of design is to formulate a technical basis for filtering out less well performing architectures. It is an attempt to narrow the field of plausible architectures so that more detailed analysis can be done on only those which look most promising. Weeding out less promising alternatives is a cost saving measure; eliminating some possibilities before time and money are invested in more detailed analysis. This down-selection process only makes sense where the architecture performance potential differences are considered significant. Small model differences will not hide significant differences between alternative architecture performance potential estimates.

These differences in model performance estimates can be viewed as a benefit. Consider some of the sources of uncertainty enumerated in section 3.5: algorithm, value or data, topology, synchronization, load and sizing. One can observe both the initial estimate and evolution of the uncertainty sizes involved in generating the proposed architecture's performance potential as design decisions are made. Graphing each of these uncertainties in size (using a simplified minimum, maximum estimate) the individual contributions can be seen in Figure 9.12. Here each of the decisions has been allocated two uncertainties, one for the minimum performance value, and another for the maximum.



Initial estimates for five proposed design decisions

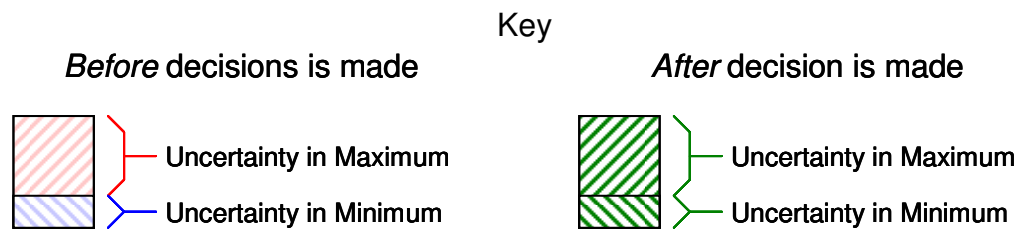


Figure 9.12 Uncertainty contributions before design decisions are made

When these uncertainty estimates are regrouped placing maximums and minimums together the result is a simplified estimate of the system's performance bounds. Figure 9.13 shows the change in the estimate of those bounds can be viewed over time as various design decisions are made.



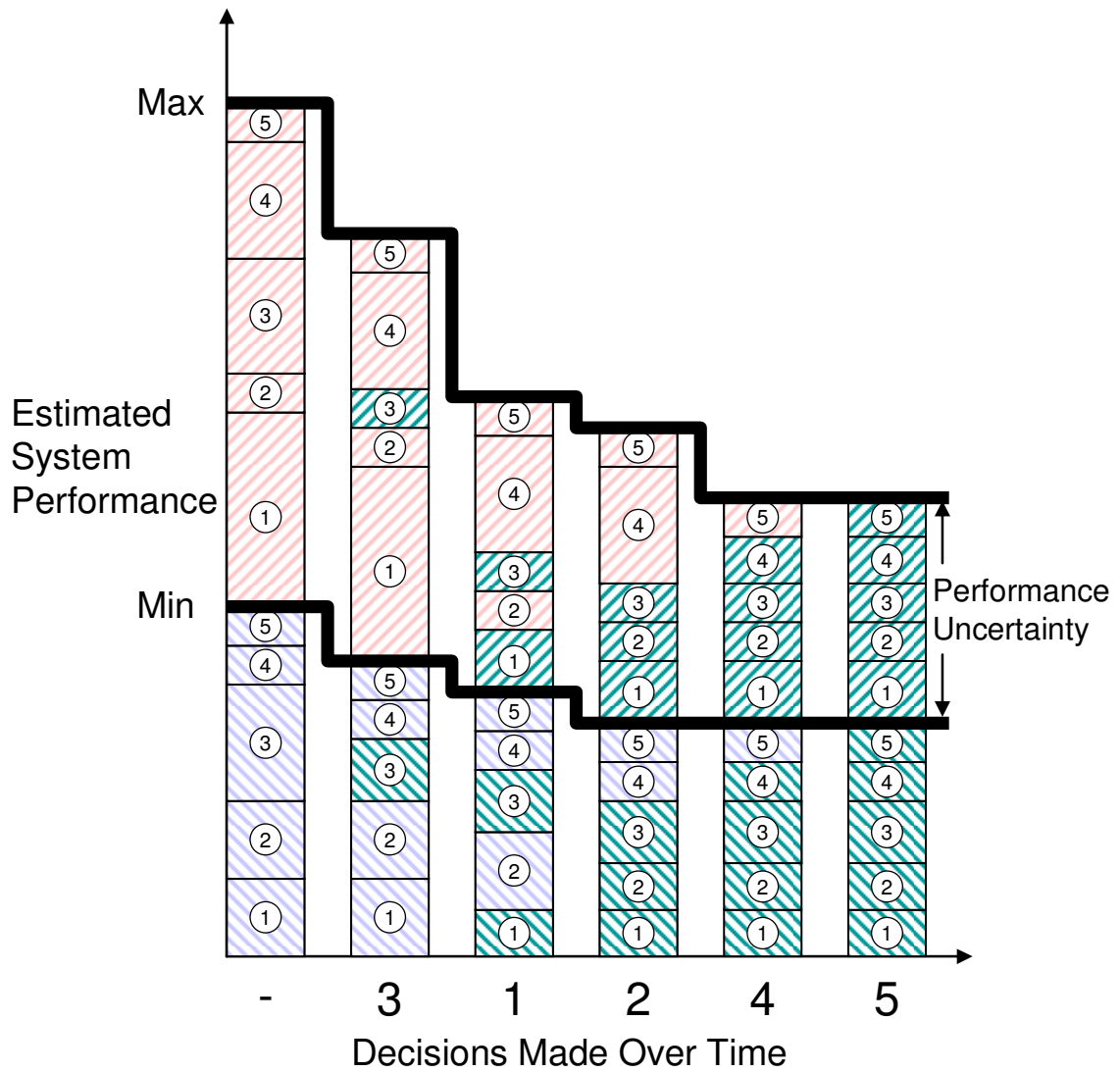


Figure 9.13 Changing performance bounds as decisions are made

One of the benefits to knowing that there are differences in model performance estimates and how those differences are related is that they help to establish the

sizes of the uncertainties being considered in analyses as shown in Figure 9.13. From the validation example earlier, it is known that different models will predict slightly different performance ranges. As a result, if the Army Communications System is used as an element of a larger performance estimation problem, this model uncertainty should be included in the uncertainty estimates and calculations used to perform the analysis. Model differences add to our understanding of the performance of system elements, and as such tend to improve the confidence that the modeled composite system will behave as predicted.

A similar uncertainty reduction over the development life-cycle is seen in costing.

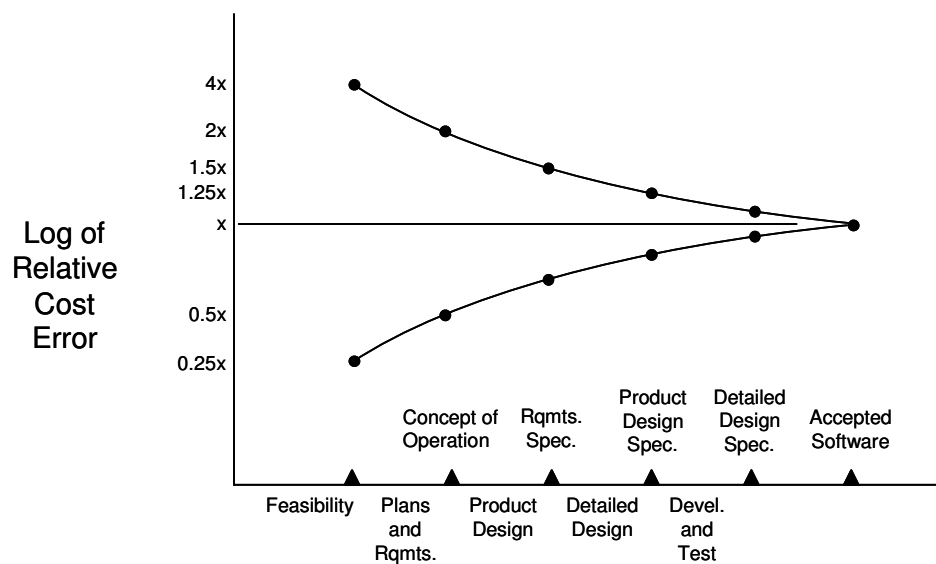


Figure 9.14 Boehm software development cost uncertainty description

Boehm [Boeh81] found that cost-to-complete estimates could be off by a factor of four early in the life-cycle. Vagueness and ambiguity in the software requirements lead to these uncertainties. This situation is similar to that found here in performance estimation. Various early models will approximate reality differently. Uncertainty in how components and prototypes will perform is greatest at the outset and decreases over time as design decisions are made reducing performance uncertainty. The modeling process is valid.

## **9.7 Summary**

This chapter has taken two CAPE modeling examples and validated the results against MESA/Extend models used within the Department of Defense. Hypothesis testing and normed vector analysis methods were described for comparing the performance descriptions of the two examples. In each case, the results from both models are in agreement. The chapter concludes with a discussion of the differences between the estimation methods and an explanation of the advantages and shortcomings of each.

## **Chapter 10**

### **Contributions and Future Research**

#### **10.1 Introduction**

In this dissertation, a method has been developed (chapter four and chapter five) for making performance comparisons between architecture alternatives. Using these techniques at architecture design time, the developer can compare the estimated likely performance of systems that might be built from an architecture description. The techniques described provide a systematic means for quantitatively assessing the probability that one particular architecture will produce an implemented system which will perform better than another through the evaluation of what is called the "performance probability integral." The approach is applicable at different levels of abstraction of the problem solution description, and provides for the inclusion for varied sources of uncertainty in how the implementation will actually be instantiated. It allows for refinement of the comparative performance estimates as detailed design characteristics are specified and provides a method for managing the uncertainties of those characteristics not quantitatively defined. To demonstrate the procedure, the classical three-tier architecture was examined (chapter six) for two architectures and two implementation choices for communications links. A proof-of-concept

prototype has been developed to perform the probabilistic computations. This allows statistical methods to be applied by non-experts. Further, the technique allows for description of element characteristics in a simplified manner, i.e., without needing to generate complex equations to describe the probability density functions that describe delays, sizes, and processing times. This simplified interface expands the utility of the approach by making it usable to a broader audience.

A variety of methods can be used to evaluate system performance. Many are most easily applied to systems that are partially implemented or where detailed component and connector information is available. Particularly popular are approaches based on queuing simulations. Queuing approaches can produce precise results by taking advantage of mature system design parameters sets. When the design parameters are not well established, a queuing approach often resorts to checking combinations of sets of values for each of the unspecified or large uncertainty specifications. This results in a larger number of cases being run to cover the design space and takes additional time. When the queuing models are evaluated using simulation based techniques, the solution time can be significant as well. Queuing network models solved using other approaches are often less accessible to the broad research audience.

CAPE takes advantage of the early design uncertainties associated with component and connection performance descriptions, parameterizes them and manages them. As such, CAPE routinely does not require multiple runs for systems of known topology. CAPE was intended to address performance related design issues earlier in the system development than other techniques. It informs the decision processes when it is the topology or combining of different components and connections that is being resolved. While this type of performance analysis can also be done using queuing models, using queuing models is likely to take longer than when using the CAPE approach. One of the primary advantages of using CAPE is its simplicity. It allows researchers and analysts from a broad set of disciplines to do productive work in architecture performance analysis without substantial start-up training. The results that it generates are quickly computed and the examples accumulated to date show good performance correlation with the systems that are being built.

## **10.2 Research Contributions**

Specifically the contributions of this research are:

- a) This research identifies the underlying uncertainties that exist at system architecture design time that likely do not exist later in the development life-cycle. These uncertainties are categorized so that the system designer can appropriately characterize them individually.

- b) Five functions necessary to combine system architecture level uncertainties are identified.
- c) For each of the five functions needed to combine the uncertainties at the system architecture level, a computation method is demonstrated and verified.
- d) The Performance Probability Integral is derived to calculate the probability that the expected performance of one architectural option will exceed the expected performance of another architectural option.
- e) A method is defined that characterizes architectural uncertainties and, maps them into probabilistic descriptions, then combines them for application of the performance probability integral.
- f) A tool was developed that assists in performing the calculations for combining component and connection performance descriptions into system architecture performance descriptions.
- g) A simple three tier data application and two quasi-real world examples are used to demonstrate the use of the developed technique.

### **10.3 Future Research**

There are a number of additional research areas which could enhance this work. They represent both extensions to and refinements of that presented here and would increase its utility of the methods described previously.

#### **10.3.1 Expand Offered Workload Analysis**

Current CAPE analysis techniques capture the static aspects of data transformation and movement. CAPE assumes that the sizes of data elements posed to the architectures being analyzed as well as the behavior characteristics of the components and connectors that make up those architectures are correct. The consequence of assuming correct performance descriptions is that 'bottlenecks' can not happen unless they are described in the performance description. This is a "chicken and egg" problem. If one does not know the correct performance description, or can not build a component or connector which exhibits that performance, there will be an issue as bottlenecks could exist.

CAPE does not currently address the rate at which those input data objects arrive. This choice was made believing that the sizes of queues and the flow-rate of data elements was a detailed design consideration. Whether detailed design or not, there is value in incorporating into CAPE the input considerations associated with the arrival rates of data elements, and how those rates effect the later portions of the design process. Supplementing CAPE with an probabilistic



arrival rate feature would provide both architecture and detailed designers with improved insight into end-to-end system performance.

### **10.3.2 Improve Model Implementation Efficiency**

A number of data structures are required to support the computations that take place when this model is executed. For the purpose of this work, emphasis was placed on achieving correct results. There is value in generalizing and formalizing both the data representations and function implementations that manipulate this data so that a broader class of problems can be considered easily. Such extensions would include better incorporation of Dirac delta functions into the function definitions and the associated improvements in operations used on these functions once so extended. The redefinition of such data structures and functions should consider primarily efficiency of execution and representation while taking into account appropriate numerical accuracy.

### **10.3.3 Applying CAPE to the Design of Software Architectures**

While CAPE is expected to be of assistance in developing distributed computation systems, there may be refinements which would be beneficial for analyzing software architectures too. Software architectures are likely to address a lower level of architecture design than that at a system level. Delays that could be ignored at a system level may become significant to the performance of a software architecture. Further study would be beneficial to identify how software

architecture designs differ from system designs, and how CAPE could be modified to extend to this area of architecture performance analysis.

#### **10.3.4 Simplify Performance Specifications Graphical User Interface**

The current implementation of these algorithms works with functions that are either numerically defined as sequences of linear approximations, or in some cases defined by code snippets that can provide such approximations. This interface is awkward for researchers interested in the performance of the architecture of the to-be system. While the exact form of a more desirable interface is not clear right now, it would appear that a graphical interface allowing the researcher to define functions with a point-and-click approach would be desirable. Such an interface would have to include appropriate transformations or warnings to ensure that the probability density functions so constructed were mathematically correct.

#### **10.3.5 Generate Parameterized Pre-built Architectural Entity Models**

The set of interesting architectures which are routinely considered for performance analysis have a number of common components. Such components include satellite links, other transport links, aggregations of such links, i.e., networks, as well as processing models that transform data from one form to another. The performance of each of these is routinely associated with a set of parameters that characterize spatial considerations, error handling as well

as more fundamental attributes like data rates or available bandwidth allocations. Since these types of elements are likely to be incorporated into any number of analysis efforts, it would be valuable to pre-build models for sets of components that are routinely used in architecture performance analysis. This would not only reduce the analysis setup time for investigators, but help to ensure consistency between architecture specifications.

#### **10.3.6 Quantify Improvement Factor**

The method developed assesses the probability that one system is expected to perform better than another. It may be of use however to add an additional parameter, i.e., that one system will perform better than the other by more than X%. This would assist the Systems Engineer in identifying cases where such differences are likely to be more significant. Confidence intervals have not been considered in this work. Work to incorporate them would be valuable in the obvious way.

#### **10.3.7 Specify an Appropriate Multi-Attribute Utility Function**

In restricting the scope of this research effort to keep it manageable, a decision was made to address only the performance aspect of architecture analysis. There are a number of other attributes that other researchers will likely study. These might include security, reliability, and cost to name but a few. Once this broader analysis is possible, there needs to be a way to combine the findings of

each of these analyses together into a single view for comparing architectures against this broader combined criterion. A multi-attribute utility function would appear to be one method of combining these disparate views of an architecture, but there may be others. Some work should be done to provide guidelines or recommendations on how to combine architecture analysis products into a more general decision rule for selecting that which is most appropriate for a particular situation.

#### **10.3.8 Specify a Compatible Cost Model for a Bayes Decision**

Life-cycle cost is one of the principal concerns faced by any system builder. Routinely there is a desire to minimize the expected system cost while at the same time achieving critical performance parameters. While there may be a number of ways to project the expected cost of a development, one that must clearly be considered is Bayes Criterion. This criterion combines the probability of making a certain decision (architecture selection) with the cost associated with that decision (implementation) to calculate the expected cost to be absorbed. This research has only offered a method for computing the probabilities of one architecture performing better than another. This could be extended with Bayes Criteria to include cost data to be of more value to Systems Engineers.

### **10.3.9 Summary**

This research effort has made the comparison of architectures with respect to performance concerns viable. It has done so in a manner that is consistent with current best practices and is practical, useful, and broadly applicable. This chapter has summarized the contributions of this research and identified areas of proposed study to improve the usability and extend applicability of the methods proposed.

## Appendix A

### Function Model and Elementary Function Operations

#### A.1 Introduction

The results calculated for the examples in this work are based on the model of functions and function operations demonstrated here. All of the computations are carried out numerically based on graphical algorithms. This approach was selected since it is possible that the probability density functions encountered in practical calculations may not be easily approximated with closed form representations, e.g., the quotient probability density function discussed in chapter four.

#### A.2 Representing Functions of One Variable

Functions are implemented as Java objects of a general form with  $y$  as the dependent (range) variable and  $x$  as the independent (domain) variable. There is an amplitude scaling factor that simplifies multiplication by a scalar, and a  $\tau$  argument term which allows for the shifting of functions by amounts of the domain variable. The direction of this shift is controlled by the sign applied to the shift amount ( $\tau$ ). The function argument domain value is inverted through a sign variable applied to  $x$  to easily accomplish the function of convolution. The general form of functions is therefore:

$$y = \text{Amplitude} * \text{function} ( \text{argument} )$$

where argument is defined to be:

$$\text{argument} = \text{plusOrMinus} * x \text{ plusOrMinus } \tau$$

The implementation of these functions of one variable is done as a data structure of piecewise linear approximations of the exact function. The end points of the approximating line segments or domain values are implemented in the Function Domain Value (FDV) class. These domain or  $x$  values hold both the exact representation (Java based finite arithmetic approximation) and a rounded value.

The exact value is used for plotting while the rounded value is used for indexing the domain values in data structures like trees and hash maps where small accumulating errors due to floating point increments in “for” or “while” loops can occur causing difficulties in achieving the proper end-of-loop condition.

Points in the range, i.e., the y value of the function are represented by the Function Range Value (FRV) class. This class routinely holds three numeric values and three open-closed conditions. For each range value, there is a value for the point at the exact domain value, a continuous-to-the-left value, and a continuous-to-the-right value. There are also three Boolean values to indicate that the point is either open or closed the each of the associated magnitude values. There are consistency checks applied as clearly all combinations are not meaningful.

### **A.3 Operations on Functions**

There are seven fundamental operations implemented for functions: multiplication by a scalar as well as function shift, addition, multiplication, integration, differentiation and convolution. In each case, the result is an instance of the Function class and is available for further computation in subsequent steps.

### **A.4 Scalar Multiplication**

Multiplying a function by a scalar is done directly through the adjustment of the amplitude attribute of the function described above. When the function needs to be put into canonical form, i.e., where the amplitude is one, and the value of the shift is zero, all individual function points are adjusted.

### **A.5 Shift Operation**

The shift operation is useful for implementing convolution and is handled by changing the “tau” variable defined in the function definition. Positive tau values (after combining with the associated plusOrMinus prefix operator) shift the function to the left. Similarly negative values shift the function to the right

### **A.6 Integration Operation**

Integration is performed on each of the piecewise linear approximating segments and summed. Each linear segment is subdivided appropriately and the

integration is done by evaluating the formula for the integration of that segment. The integration function is used in largely three ways: to compute cumulative distribution functions from probability density functions, to calculate the normalization constant for functions that are proposed as probability density functions, and again in the convolution operation.

## **A.7 Differentiation Operation**

Differentiation is accomplished on a point by point basis for each function approximating line segment. The slope of each segment is taken to be the value of the function derivative. This value is assigned to the midpoint of that line segment.

## **A.8 Convolution Operation**

Convolution is a binary operation on functions often represented by the “star” operator and is defined as:

$$f_A(t) * f_B(t) = \int_{-\infty}^{\infty} f_A(\tau) \cdot f_B(t - \tau) d\tau$$

The implementing code first flips the first function by change the independent variable from plus to minus. Then for each shift value, “tau” where the function products have value, that product is formed point-wise and integrated across the domain of definition. This is a straight forward implementation of the definition.

## **A.9 makeCanonical() Operation**

The make Canonical function is a utility function that performs appropriate transformations of function points so that the function representation is changed in a way that makes the amplitude scaling factor 1.0, and the shift amount (tau) 0.0. This is useful in comparing functions to determine their differences.

## **A.10 Representing Functions of Two Variable**

Functions of two variables are represented in a similar manner with the exception that the approximating line segments become approximating planes defined by the four points at the corners of the grid which defined each of the two independent variables. The approximation that is used in the calculations assigns all values within the bounding rectangle of the four defining points as the



average value of those points. Most of the useful cases involve the joint density function of joint distribution function of two independent random variables. In either of these cases, the joint density function is related to the product of two one dimensional functions, and the associated cumulative distribution functions are generated from the appropriate integration over regions of this product.

### **A.11 Quasi-Arbitrary Random Variable Generation Functions**

For testing purposes, it is useful to be able to generate random variables for quasi-arbitrary probability density functions. The technique used here takes as input a function that is proportional to the desired pdf that describes the random variable needed. There are three steps required to instantiate a random number generator that will numbers consistent with the target pdf. First the proposed pdf is integrated over its domain of definition. One over this area is defined to be the scaling constant. The pdf is then scaled by this constant. Scaling is needed to ensure that the probability of all outcomes sum to one.

The cumulative distribution function (cdf) is then calculated by integrating incrementally from the minus infinity to plus infinity. This cumulative distribution function is then inverted, i.e., where  $y = f(x)$  in the cdf, the required function is  $y = f^{-1}(x)$ . Now random numbers are selected from the uniformly generated generator provided by Java on the interval  $[0,1)$  and applied as the argument of the  $f^{-1}()$  calculated.

### Example PDF and CDF with Bin Count

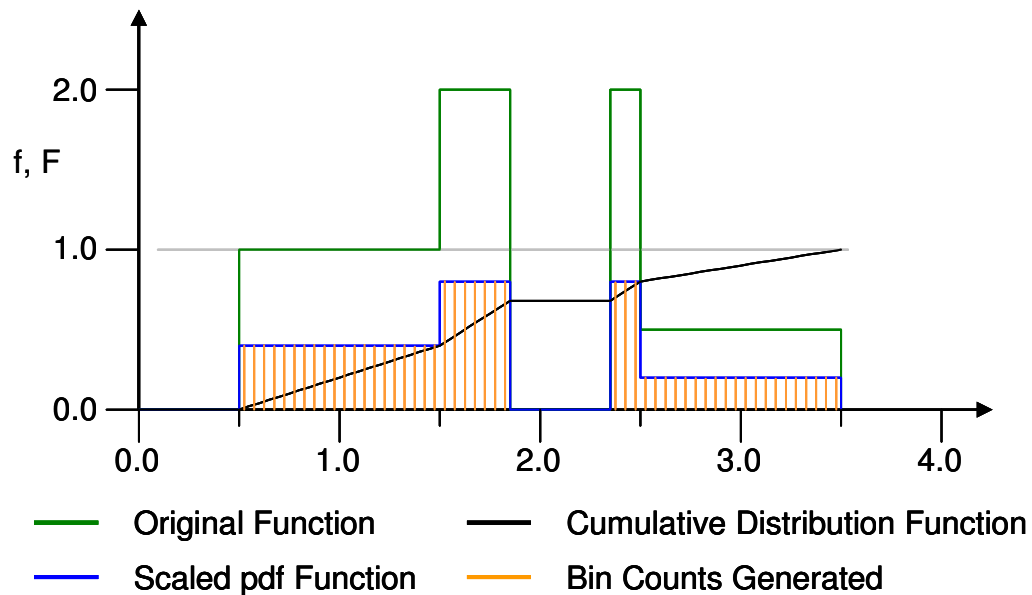


Figure A.5.1 Conversion of pdf to cdf.

Consider Figure A.5.1 The originally provided function is in green. After integration over its domain of definition, the area under the green curve is calculated to be 2.5. The blue curve is then generated from the green curve by multiplying it by the scalar  $(1.0 / 2.5) = 0.4$ . Then the blue curve is integrated over the interval to generate the black curve. This black curve is then inverted in Figure A.5.1.

Now values are selected at random from the uniform density function,  $[0,1)$  shown as the dark thick black line on the horizontal axis and mapped to the desired pdf, i.e.,  $a \rightarrow a'$ ,  $b \rightarrow b'$ ,  $c \rightarrow c'$  etc. The orange vertical lines in Figure A.5.1 portray the normalized count of 1,000,000 random numbers generated in this manner and divided into 70 equally spaced bins from zero to 3.5.

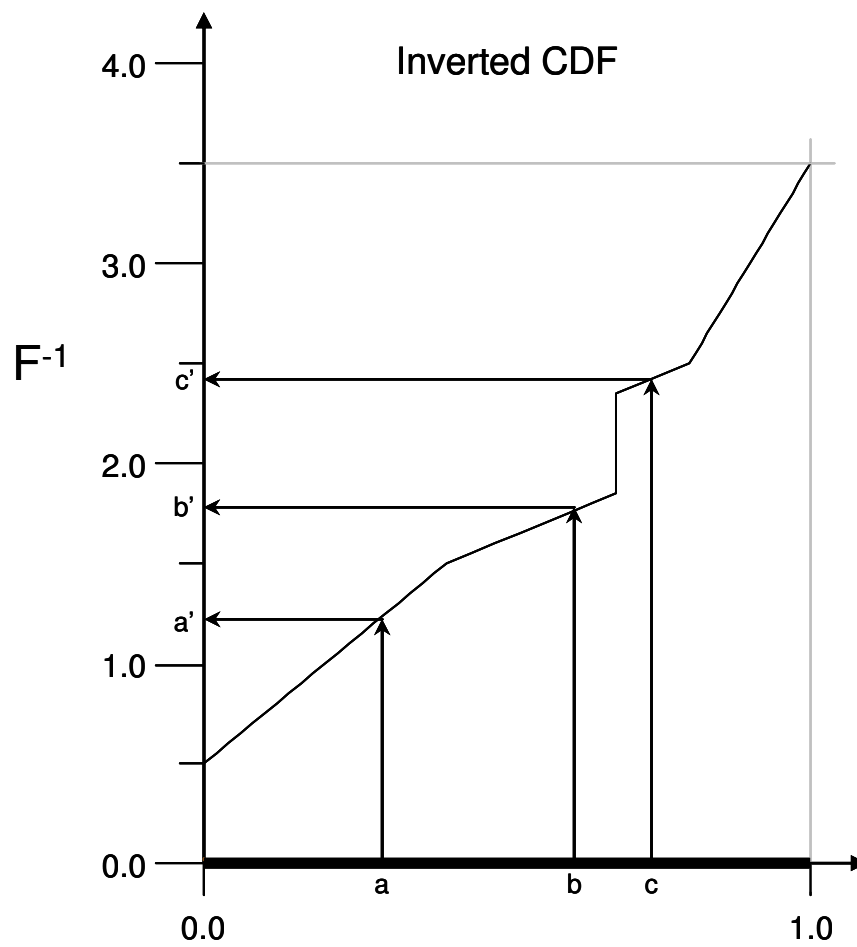


Figure A.5.2 Inverted CDF used to generate random variables

## Appendix B

### Detailed Examples of Uncertainty

Even at the lowest levels of computation, examples of uncertainty exist. In [Kobl00] there is a discussion of the multiplication of a k-bit binary number by an l-bit binary number. Routinely this type of calculation is accomplished with successive shift add operations. Even when the shifting and copying portions of the algorithm can be ignored as small when compared to the actual addition portions of the process, there exists a difference in execution duration of the computation due to differing numbers of 1's in the number being multiplied. Since the number of 1's in the multiplier are unknown until the computation is provided actual data, a tight estimate of the time to perform a general multiplication may be characterized in a probabilistic manner base on that number of multiplier 1's [Irvi09].

#### Binary Multiplication

```
      11101101
      x 100101
      -----
      11101101
      11101101
      11101101
      11101101
      -----
     10001001000001
```

Figure B.1 The number of additions is approximately equal to the number of multiplier 1's.

Algorithmic performance uncertainty arises largely because one may not know in advance how many times a particular looped action will be executed when it is coded as a "do until," "while," or perhaps even a "for" construct iterating over an array of run-time-determined size. In other situations, a result may not be produced until a certain exit criteria is met and as such, the number of iterations may not be known until certain values meet specified termination condition for the computation. Here both the number of iterations, and the precise time associated with executing each iteration may be unknown or may depend on the values of the computation being considered.

Next consider an example of algorithm selection. One option is shown in Figure B.2.

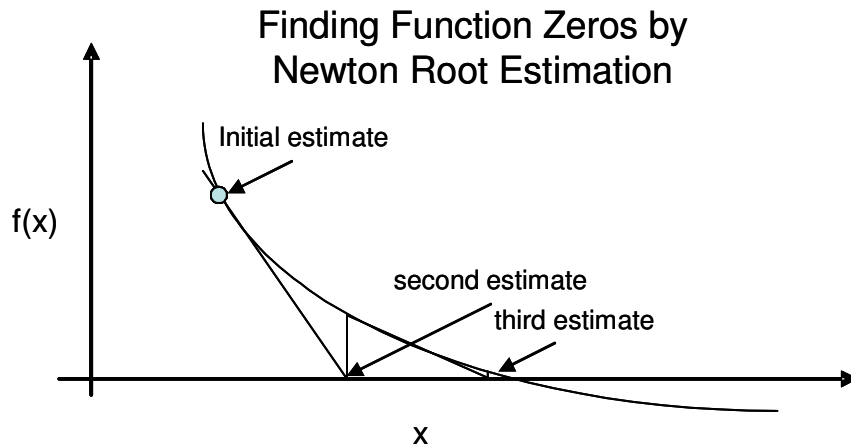


Figure B.2 First three steps in a Newton Root Approach

The goal here is to find a root of a specified polynomial. Figure B.2 shows the first three steps given a specific function, and initial starting estimate when using the Newton method. Yet when a different algorithm (secant, Figure B.3) is employed, an entirely different set of estimates is generated, and a different convergence time is likely.

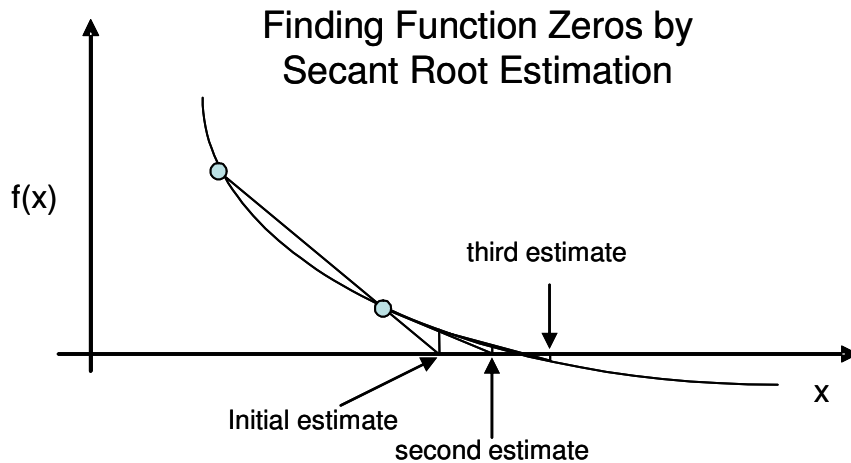


Figure B.3 First three steps in a secant root approach

In the algorithm uncertainty case, the time it takes to get to the answer is a function of the starting point(s), the shape of the function being examined, and the algorithm being used to compute the desired root. Each factor adds a different uncertainty to the overall time that the calculation requires.

At higher levels in the computation hierarchy different uncertainties exist. Consider the delay associated with retrieving information from a database. The delay experienced between the request submission and the result arrival may depend on the size of the data stored, the organization of the underlying data elements, the complexity of the query, the existence of synchronization locks for critical resource elements, or even the competition for CPU resources generated by other processes attempting to gain access. At even higher system levels, the time to move data may depend on the path that the data takes which in turn may change over time.

A probabilistic approach is appropriate for characterizing delays in this environment, and hence is appropriate for estimating performance at an architecture level of analysis. At each level of computation, there are uncertainties, and these must be characterized and then combined to produce a meaningful representation of an entire process.

## Appendix C

### Computational Methods and Verification

#### C.1 Introduction

The six uncertainty classes described in chapter five are computed using the techniques identified in the following sections.

#### C.2 Summation

Summation is one of the most often used combination functions. Two representative examples are provided as verification tests.

Consider a system with fifteen delays each characterized by a uniform density function with duration from zero to 0.5 time unit as shown in Figure C.0.1

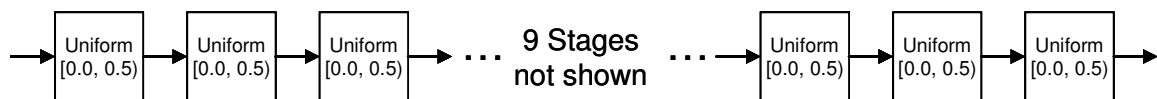


Figure C.0.1 Fifteen independent stage uniformly distributed delay line

From the definition of the uniform probability density function:

$$U(a, b) = \begin{cases} 1 / (b-a), & \text{for } a \leq x \leq b \\ 0, & \text{otherwise} \end{cases}$$

$$\text{Mean} = (a + b) / 2 \quad \text{Variance} = (b - a)^2 / 12$$

Using convolution to iteratively sum this density function 15 times, yields the result shown below

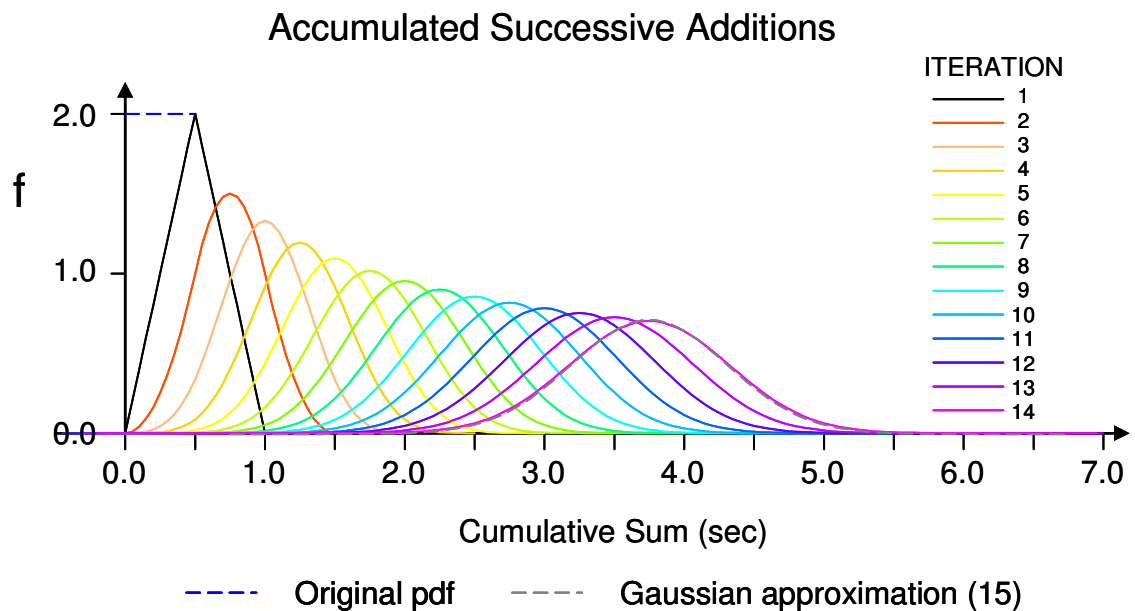


Figure C.0.2 Iterative summation of uniform densities



The law of large numbers states that the sum of a large number of independent identically distributed random variables approaches normal with a mean equal to the sum of the means and variance equal to the sum of the individual variances. In the limit the sum is exactly normal. Since 15 is significantly less than infinity there will be an error or difference between the convolution result and a true Normal distribution's shape. As verification of this result, superimposed over that final sum is a dashed black line of Gaussian shape with mean 15 times the uniform and variance of 15 times the uniform. The result however is close to ideal. The purple curve (iteration 14, sum 15) is known to be a probability density function since it was derived as the addition of independent identically distributed random variables. The integral under the curve should be 1.0. When the integral is calculated with the analysis tool the result is 1.00114. The integral under the Gaussian curve with mean of 3.75 and variance of 0.3125 (plotted gray) is 1.0 when calculated in the same manner. The maximum difference between the two functions occurs at the mean of 3.75 and is equal to 0.0125. The sum of the mean square differences between these two functions summed over all 121 points plotted is 0.01253429. The sum of the mean square error between the convolution approximation and the Gaussian is 0.002221959. The difference between the true Gaussian and the sum of the uniform densities is small. The calculation is useful, and is consistent with the fact that the resolution used for the graphical integration is 2% of the range of the original random variable. These errors are considered to be small when comparing them to the uncertainties associated with architecture performance manipulation in general. The graphical convolution result is reasonable for this purpose.

A second simple example helps to show that the implementation of this graphical convolution can also represent arbitrarily defined independent random variables. The first of these two pdfs was chosen because of its irregular shape. It has no specific symmetry. The second pdf could reasonably be encountered in a noisy communications channel. Figure C.0.3 and Figure C.0.4 show the pdf, the cdf, and the plotted verification by simulation of the bin counts of the two performance descriptions being combined.

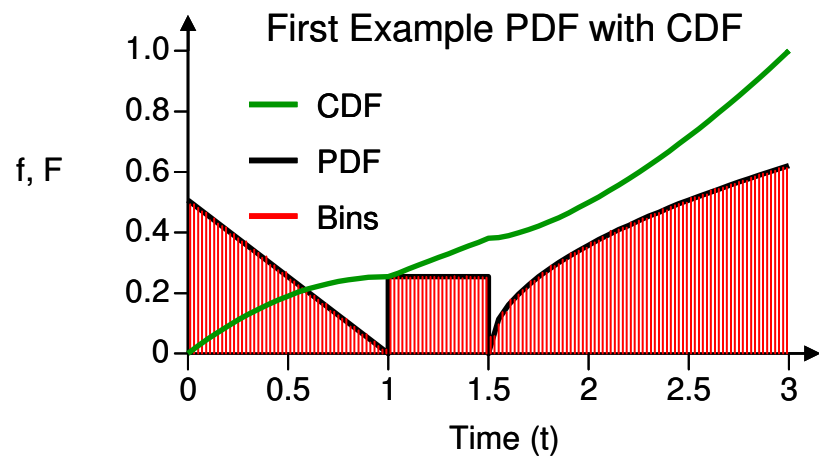


Figure C.0.3 An irregular pdf describing system performance

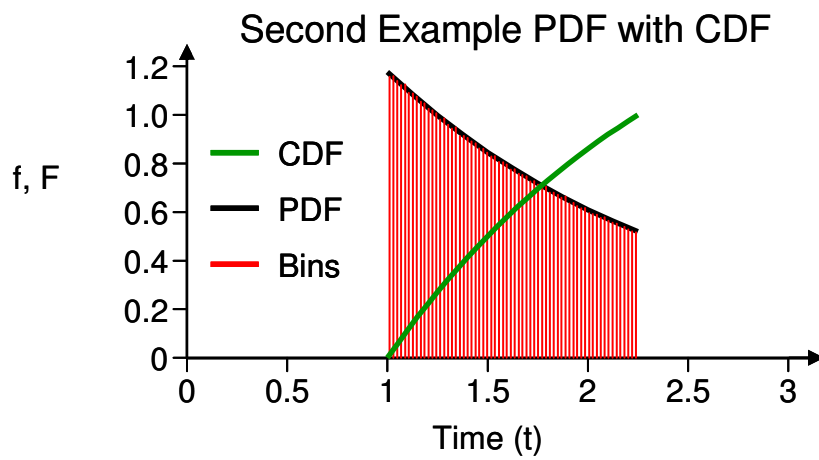


Figure C.0.4 Another pdf describing system performance

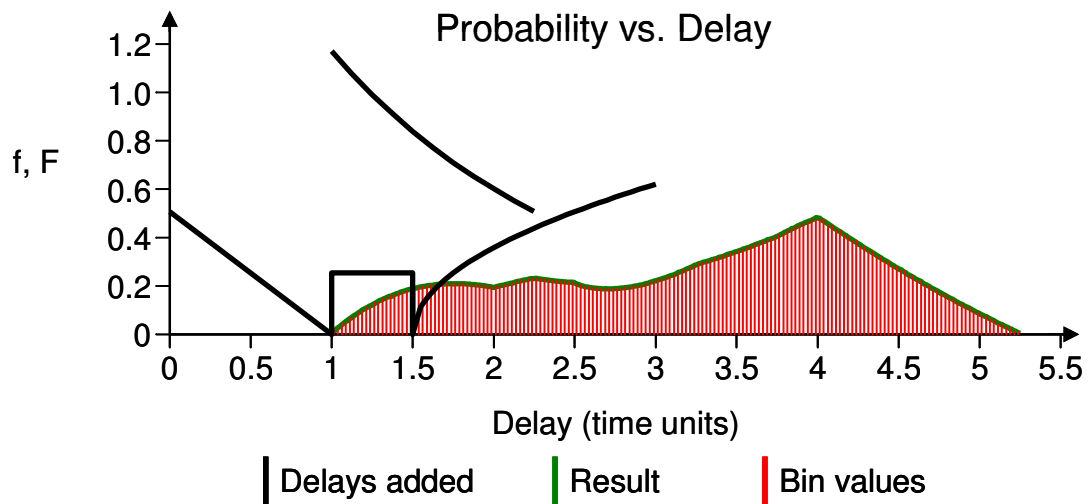


Figure C.0.5 Convolution of pdfs from Figure C.3 and Figure C.4

Figure C.0.5 results when these two performance descriptions are combined through convolution. The simulation result is consistent with the convolution result.

### C.3 Quotient

The basic calculation for the quotient probability density function is similar to the deterministic case

Consider the simple example defined in Figure C.0.6. The data package size is assumed to be uniformly distributed from 10 to 50 bytes. The data rate of the link is assumed to be uniformly distributed from 75 to 150 bytes per second.

Figure C.0.6 also shows the geometry associated with calculating the cumulative distribution function of the quotient from the joint density function of size and rate. The calculated cdf describes the probability that the time needed for a data packet transmission will be less than some time,  $t'$ .

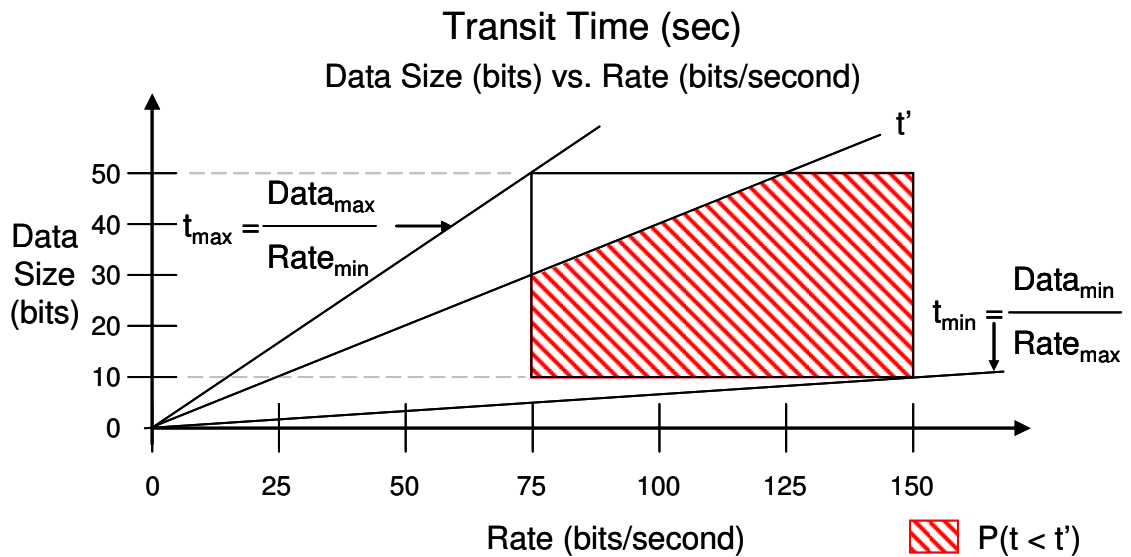


Figure C.0.6 Relationship of transmit time to data size and data rate

Consider first the two lines labeled  $t_{\max}$  and  $t_{\min}$ . The maximum time it will take to send a data packet ( $t_{\max}$ ) is that associated with the maximum size packet traversing the minimum data rate link. Conversely the minimum time (transmission delay) is experienced when the smallest size packet is sent over the highest speed the link can achieve. These values of  $t_{\max}$  and  $t_{\min}$  bound the quotient performance. Each of the positive slope diagonal lines ( $t_{\max}$ ,  $t'$  and  $t_{\min}$ ) represent the ratio of the data size to link data rate. These lines which represent time values divide the region of integration for the calculation. For each  $t'$  value between  $t_{\min}$  and  $t_{\max}$ , the red shaded fraction of the whole rectangle represents the probability that a delay will be less than the value stipulated by the  $t'$  slope defining the regions. Integrating  $t'$  from  $t_{\min}$  to  $t_{\max}$  must yield one, the probability that it will take some amount of time between  $t_{\min}$  and  $t_{\max}$  to send the data packet. Since the shaded area is the probability of  $t$  being less than  $t'$  it is the value of the cumulative distribution function (cdf) for the size-speed ratio (quotient probability) for that time  $t'$ . Using this graphical approach yields a graph of the cdf for the time associated with the distribution of times associated with the data packet size and link data rate densities, Figure C.0.7.

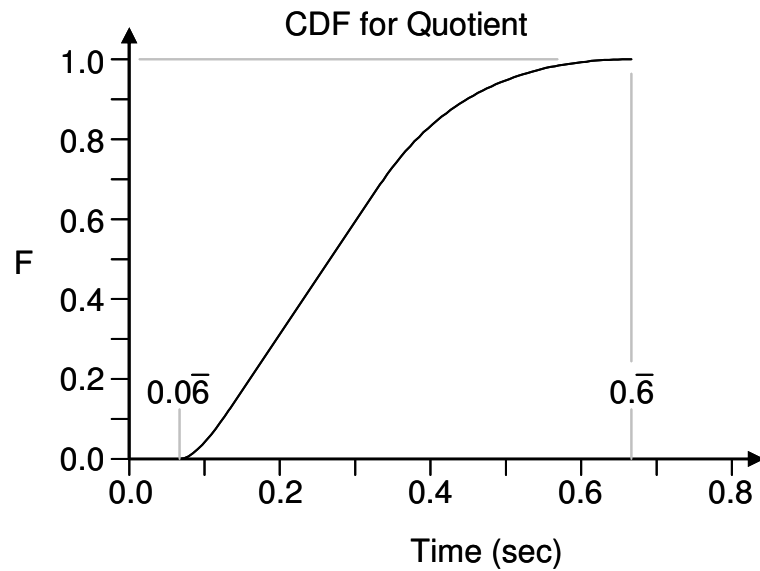


Figure C.0.7 Quotient cumulative distribution function for the example

A probability density function can be derived from the cumulative distribution function by differentiation in cases where the cdf function is well behaved. For this same example, the probability density function is numerically calculated. The computation yields Figure C.0.8. This result is confirmed by simulation in Figure C.0.9.

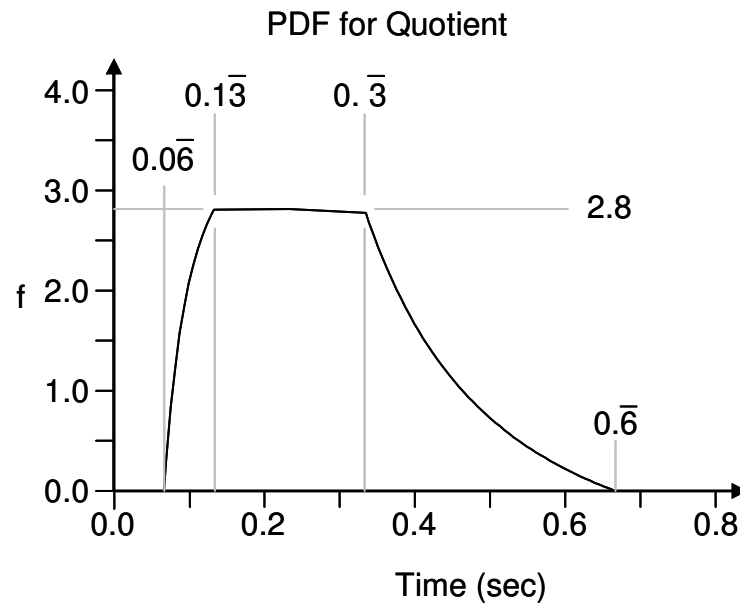


Figure C.0.8 Example probability density function

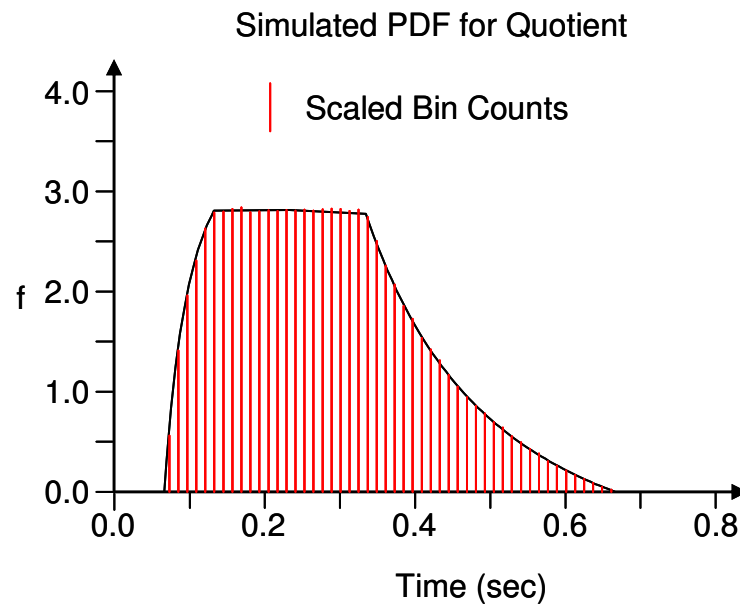


Figure C.0.9 Simulation of one million example quotients

The data-packet size and link data rates were simulated by uniformly distributed random variables as described in Figure C.0.6. One million sample quotients were calculated and counted in 50 bins spanning the domain of the pdf. The scaled bin values were then plotted in Figure C.0.9.

#### C.4 MIN

The minimum function (MIN) can be calculated graphically as well. Consider a two input example to examine how this computation is done. Two sources, A and B, generate and transmit the requested information. Source A is capable of returning the requested data in a time uniformly distributed between 10 and 30 time units. Source B similarly returns data in a time uniformly distributed between 25 and 35 time units. Since these two events are considered independent, their joint probability density function is the product of the two uniform pdfs. The shaded portion (both red and blue) shows the domain of the joint pdf for this situation.

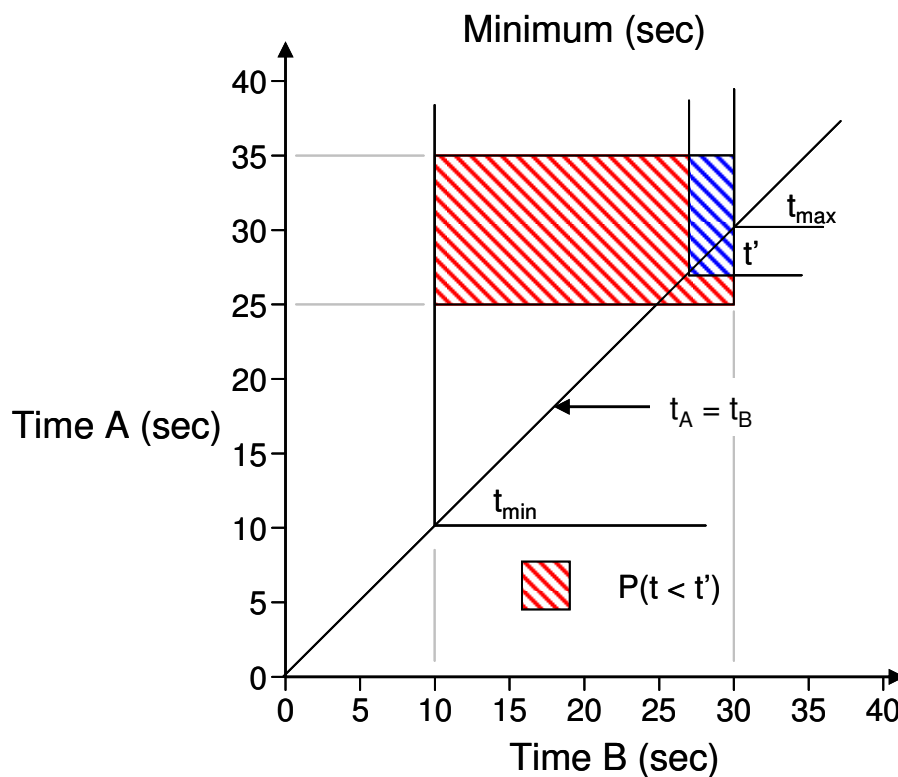


Figure C.0.10 Example two input minimum geometry calculation

The computation of the cdf follows the process described above. The right angle line marked  $t'$  separates the shaded region into those with values less than  $t'$  (red) and those greater than  $t'$  (blue). The shaded region below and to the left of the  $t'$  line (red) is that portion of the time where either  $t_A$  or  $t_B$  is less than  $t'$ . The bounding values of the possible times that can occur with the current description of the system performance are  $t_{\min}$  and  $t_{\max}$ . When this calculation is performed, the cumulative distribution function can be plotted as in Figure C.0.11.

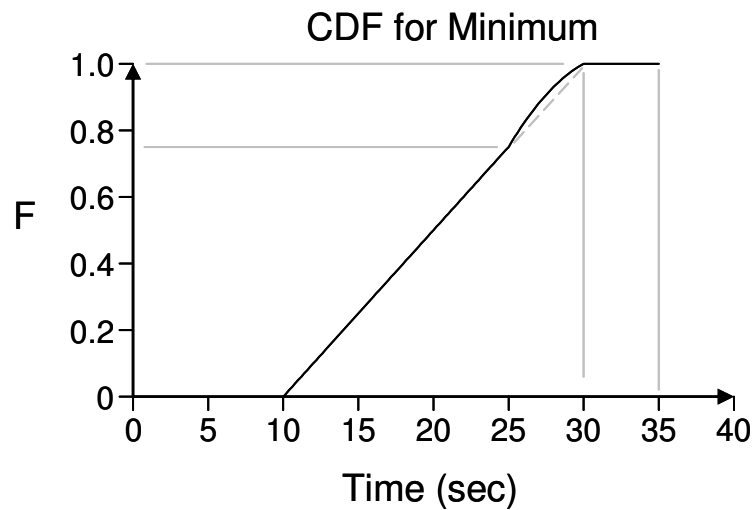


Figure C.0.11 Calculated cdf MIN for two input example

The pdf is calculated directly from this cdf through differentiation as the cdf is well behaved. Figure C.0.12 shows the pdf with a simulation of one million points binned and scaled, confirming correctness.

The solution for cases where there are more than two inputs is generated iteratively. Let the  $n$  inputs be labeled  $i_1, i_2, i_3, \dots, i_n$ , and define the initial result,  $r$ , to be the min of  $i_1$  and  $i_2$ . Then for  $i^*$  from  $i_3$  to  $i_n$ ,  $r$  is replaced by the min of  $r$  and  $i^*$ . Alternatively, this can be viewed as:  $\text{Min}(i_1, \text{Min}(i_2, \text{Min}(i_3, \text{Min}(\dots, i_n)))$



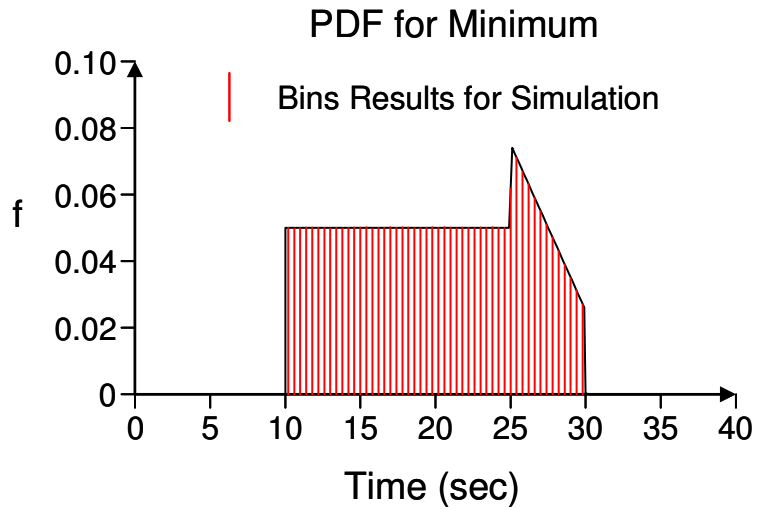


Figure C.0.12 PDF for minimum of the uniform joint pdf

## C.5 MAX

The maximum function (MAX) is calculated in a manner similar to that of MIN. Consider the two input case and discussed in section 5.4. The geometry of the maximum calculation is shown in Figure C.0.13. The domain of the joint density function remains the same (both red and blue shaded areas). The red shaded area shows where the maximum values reside. The process is similar to the minimum calculation done earlier. Integration takes place from  $t_{\min}$  to  $t_{\max}$ . Each interim point is represented by the red shaded area bounded by the domain of the joint density and the angled line marked  $t'$ . The cdf is shown in Figure C.0.14.

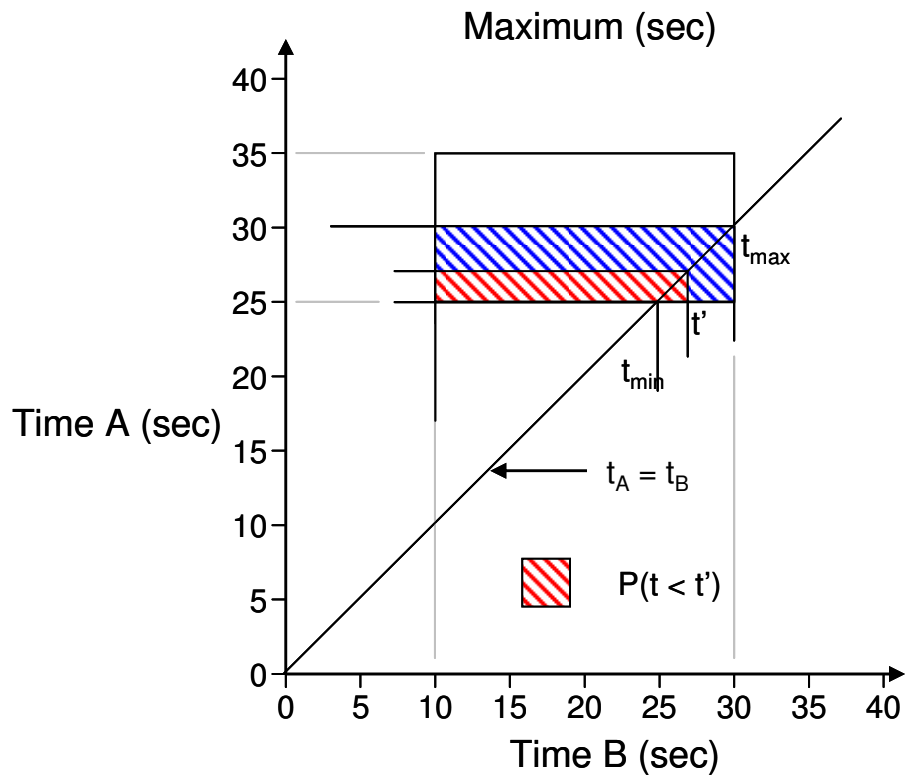


Figure C.0.13 Example two input maximum pdf geometry calculation

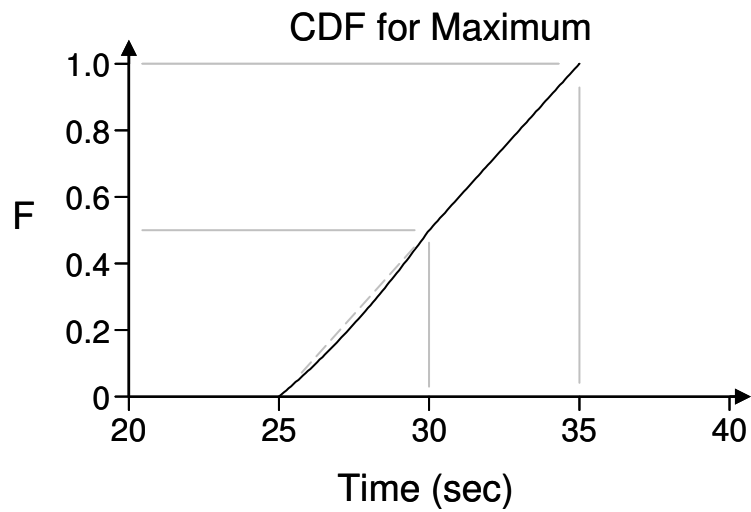


Figure C.0.14 Example cumulative distribution – two input example

This cdf can be graphically differentiated to determine the associated pdf as shown in Figure C.0.15. The vertical red lines represent the simulation of one million points counted in 25 bins across the domain of the pdf.

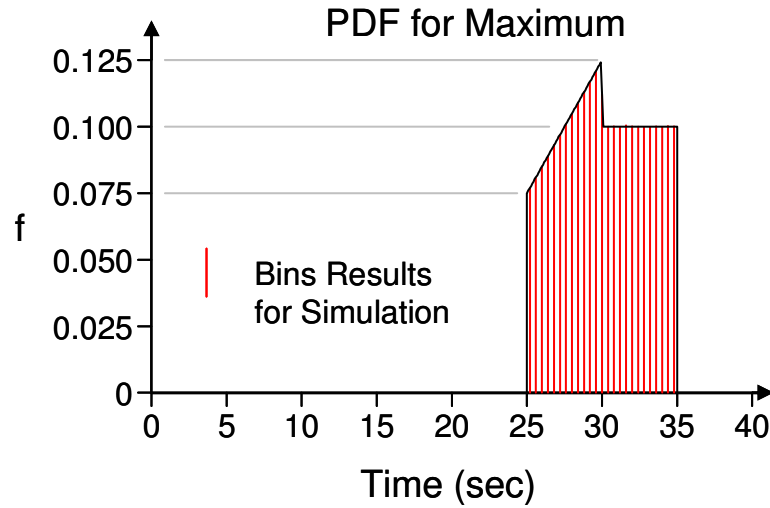


Figure C.0.15 Example two input maximum pdf calculation

For cases where there are more than two inputs, the result is generated iteratively similar to the case of MIN. Given  $n$  inputs the pseudo-code algorithm is as follows: Let the  $n$  inputs be labeled  $i_1, i_2, i_3, \dots, i_n$ , and define the result  $r$  initially to be the max of  $i_1$  and  $i_2$ . Then for  $i^*$  from  $i_3$  to  $i_n$ , replace  $r$  with the max of  $r$  and  $i^*$ . Alternatively, it can be viewed as:  $\text{Max}(i_1, \text{Max}(i_2, \text{Max}(i_3, \text{Max}(\dots, i_n)))$

## **Appendix D**

### **PLOT Library Functionality**

The PLOT function library that supplements CAPE was designed under Microsoft PowerPoint 2003, but since the object model has not changed in the newer versions, it is expected to work while yet untested. The concept behind the library is to provide the user with all of the essential functions needed to convert the PDF performance descriptions into a visual for easy analysis.

The underlying scripting language for PowerPoint is Visual Basic for Applications, and all of the supplementary subroutines and functions that support the PPT drawing of figure are coded in this language. To manage this set of functionality, there is a Java Figure class which does three things: 1) it maintains the attributes normally found in a minimal figure, e.g., coordinate axis, tick-marks, tick-mark labels, etc. 2) it maintains the scaling factors which relate the user coordinate system to the screen coordinate system, and 3) offers a source code manager which scans the requirements of the figure being produced and automatically copies into the output macro file, all of the supporting subroutines and functions.

The support routines that are provided have largely self explanatory names: addLabel, addSCLine, addSLLine, addTextBox, addUArrow, addUDot, addUFF, addULine, addURectangle, addUTextBox, alignLabelsWithTicks, clearAll, drawSBB, drawUXAxis, drawUYAxis, fixMargins, GOR (Get Object Reference), Header, lenThree, s2ux, s2uy, scaleX, scaleY, setColor, setShapeTextColor, setTBMarginsZero, setText, u2sx (User to Screen X), u2sy, and \_dependencies.txt. this final file is a hand coded dependency list of all functions and subfunctions used through the code module. Routinely, "U" represents "user," "S" represents "screen," X and Y have their traditional meanings.

## REFERENCES

## REFERENCES

- [AABI00] Andolfi, F., et al. "Deriving Performance Models of Software Architectures from Message Sequence Charts." *Proceedings of the 2<sup>nd</sup> International Workshop on Software and Performance* (2000). 47-57. Print.
- [AaHT02] van der Aalst, van Hee and van der Toorn, R. A. "Component-Based Software Architectures: a Framework Based on Inheritance of Behavior." 42.2-3 (2002). 129-171. Print.
- [AKLW02] Avritzer, A., et al. "Software Performance Testing Based on Workload Characterization." *Proceedings of the 3<sup>rd</sup> International Workshop on Software and Performance* (2002). 17-24. Print.
- [AlGa94] Allen and Garlan, D. "Formalizing Architectural Connection." *Proceedings of the 16<sup>th</sup> International Conference on Software Engineering* (1994). 71-80. Print.
- [AlKo05] Alzamil and Korel, B. "Application of Redundant Computation in Software Performance Analysis." *Proceedings of the 5<sup>th</sup> International Workshop on Software and Performance* (2005). 111-121. Print.
- [Alle97] Allen, R. "A Formal Approach to Software Architecture." *Ph.D. Thesis, CMU Technical Report CMU-CS-97-144* (1997). Print.
- [Alsa04a] Alsaadi, A. "A Performance Analysis Approach Based on the UML Class Diagram." *Proceedings of the 4<sup>th</sup> International Workshop on Software and Performance* (2004). 254-260. Print.
- [Alza04b] Alzamil, Z. "Application of the Operational Profile in Software Performance Analysis." *Proceedings of the 4<sup>th</sup> International Workshop on Software and Performance* (2004). 64-68. Print.
- [AmBo07] D'Ambrogio and Bocciarelli, P. "A Model-driven Approach to Describe and Predict the Performance of Composite Services." *Proceedings of the 6<sup>th</sup> International Workshop on Software and Performance* (2007). 78-89. Print.

- [Ambr05] D'Ambrogio, A. "A Model Transformation Framework for the Automated Building of Performance Models from UML Models." *Proceedings of the 5<sup>th</sup> International Workshop on Software and Performance* (2005). 75-86. Print.
- [AmCl01] Ammar, Cortellessa and Ibrahim, A. "Modeling Resources in a UML-based Simulative Environment." *Computer Systems and Applications, ACS/IEEE International Conference* (2001). 405-410. Print.
- [Andr88] Andreadakis, S.. "Analysis and Synthesis of Decision-Making Organizations." *PhD dissertation, Massachusetts Institute of Technology* (1988). Print.
- [ARM-05] "ARM1156T2-S Revision: r0p0 Technical Reference Manual." *ARM The Architecture for the Digital Age*. ARM Limited, 2005. Web. 10 Jan 2010.
- [ArSp00] Arief and Speirs, N. A. "A UML Tool for an Automatic Generation of Simulation Programs." *Proceedings of the 2<sup>nd</sup> International Workshop on Software and Performance* (2000). 71-76. Print.
- [BaBS02] Balsamo, Bernardo and Simeoni, M. "Combining Stochastic Process Algebras and Queueing Networks for Software Architecture Analysis." *Proceedings of the 3<sup>d</sup> International Workshop on Software and Performance* (2002). 190-202. Print.
- [BaCK98] Bass, Clements and Kazman, R. "Software Architecture in Practice." New York: Addison Wesley, 1998. Print.
- [BaGo09] Babar and Gorton, I. "Software Architecture Review: The State of Practice." *Computer*. (2009). 26-32. Print.
- [BaIm98] Balsamo, Inverardi and Mangano, C. "An Approach to Performance Evaluation of Software Architectures." *Proceedings of the First International Workshop on Software and Performance* (1998). 178-190. Print.
- [BaMa03] Balsamo and Marzolla, M. "A Simulation Based Approach to Software Performance Modeling." *Proceedings of the 9<sup>th</sup> European Software Engineering Conference Held Jointly with 11<sup>th</sup> ACM SIGSOFT International Symposium on Foundations of Software Engineering* (2003). 363-366. Print.
- [BaMa05] Balsamo and Marzolla, M. "Performance Evaluation of UML Software Architectures with Multiclass Queueing Network Models." *Proceedings of the 5<sup>th</sup> International Workshop on Software and Performance* (2005). 37-42. Print.

- [BaMI04] Balsamo, DiMarco and Inverardi, P. "Model-Based Performance Prediction in Software Development: A Survey." *IEEE Transactions on Software Engineering* 30.5 (2004). 295-310. Print.
- [BaSi01] Balsamo and Simeoni, M. "Deriving Performance Models from Software Architecture Specifications." *European Simulation Multiconference 2001* (2001). 65-89. Print.
- [BBKV08] Bause, F., et al. "A Framework for Simulation Models of Service-Oriented Architectures." *Proceedings of the SPEC International Workshop on Performance Evaluation: Metrics, Models and Benchmarks* (2008). 208-227. Print.
- [BCR+09] Bozzono, M. et al. "Verification and Performance Evaluation of AADL Models." *Proceedings of the 7th joint meeting of the European Software Engineering Conference and the ACM.* (2009). 285-286. Print.
- [BeCD00] Bernardo, Ciancarini and Donatiello, L. "AEMPA: A Process Algebraic Description Language for the Performance Analysis of Software Architectures." *Proceedings of the 2<sup>nd</sup> International Workshop on Software and Performance* (2000). 1-11. Print.
- [BeDM02] Bernardi, Donatelli and Merseguer, J. "From UML Sequence Diagrams and Statecharts to Analysable Petri Net Models." *Proceedings of the 3<sup>rd</sup> International Workshop on Software and Performance* (2002). 35-45. Print.
- [BeKR07] Becker, Koziolk and Reussner, R. "Model-Based Performance Prediction with the Palladio Component Model." *Proceedings of the 6<sup>th</sup> International Workshop on Software and Performance* (2007). 54-65. Print.
- [BeKR09] Becker, Koziolk and Reussner, R.. "The Palladio Component Model for Model-Driven Performance Prediction." *Journal of Systems and Software* 82.1 (2009). 3-22. Print.
- [BeMi04] Bertolino and Mirandola, R. "Software Performance Engineering of Component-based Systems." *Proceedings of the 4<sup>th</sup> International Workshop on Software and Performance* (2004). 238-242. Print.
- [BFG+04] Becker, S., et al. "Towards a Generic Framework for Evaluating Component-Based Software Architectures." *Proceedings zur 1. Verbundtagung Architekturen, Komponenten, Anwendungen* 57. (2004). 163-180. Print.



- [BMMI04] Balsamo, S., et al. "Experimenting Different Software Architectures Performance Techniques: a Case Study." *Proceedings of the 4<sup>th</sup> International Workshop on Software and Performance* (2004). 115-119. Print.
- [Boeh81] Boehm, B. *Software Engineering Economics*. Prentice-Hall. (1981). 311. Print.
- [Boro04] Borowski, J. "Thesis: Software Architecture Simulation - Performance Evaluation During the Design Phase." Karlskrona: Institute fur programvaruteknik och datavetenskap Dept. of Software Engineering and Computer Science, 2004. 1-30.
- [Bulk00] Bulka, D. "Server-Side Programming Techniques, Java<sup>(TM)</sup> Performance and Scalability." Vol. 1. Boston: Addison-Wesley, 2000. Print.
- [CaMe03] Campos and Merseguer, J. "Exploring Roles for the UML Diagrams in Software Performance Engineering." *Proceedings of the International Conference on Software Engineering Research and Practice* 1. (2003). 43-47. Print.
- [CaMe06] Campos and Merseguer, J. "On the Integration of UML and Petri Nets in Software Development." *Petri Nets and Other Models of Concurrency-ICATPN 2006* 4024. (2006). 19-36. Print.
- [CCIP06] Colangelo, D., et al. "Reducing Software Architecture Models Complexity: A Slicing and Abstraction Approach." *Formal Techniques for Networked and Distributed Systems (FORTE)* 4229. (2006). 25-32. Print.
- [CGH+04] Canevet, C., et al. "Analysing UML 2.0 Activity Diagrams in the Software Performance Engineering Process." *ACM SIGSOFT Software Engineering Notes* 29.1 (2004). 74-78. Print.
- [CGHT99] Clark, G., et al. "Experiences with the PEPA Performance Modelling Tools." *IEE Proceedings Software* 146.1 (1999). 11-19. Print.
- [ChCo05] Chung and Cooper, K. "COTS-Aware Requirements Engineering and Software Architecting." *Proceedings of the Internal Workshop on Systems/Software Architectures (IWSSA04)* 57.1 (2005). 100-111. Print.
- [CHLS09] Coste, N. et al. "Towards Performance Prediction of Compositional Models in Industrial GALS Designs." *Proceedings of the 21st International Conference on Computer Aided Verification*. 204-218. Print.

- [CIDB98] Clark, Devnani-Chulani and Boehm, B. "Calibrating the COCOMO II Post-Architecture Model." *Proceedings of the 1998 International Conference on Software Engineering* (1998). 477-480. Print.
- [Clem96] Clements, P. "A Survey of Architecture Description Languages." *Proceedings of the 8<sup>th</sup> International Workshop on Software Specifications & Design* (1996). 16-25. Print.
- [CLGL05] Chen, S., et al. "Performance Prediction of Component-based Applications." *Journal of Systems and Software* 74.1 (2005). 35-43. Print.
- [CINo96] Clements and Northrop, L. "Software Architecture: An Executive Overview, CMU/SEI-96-TR-003, ESC-TR-96-003." (1996). Print.
- [CoGM08] Cortellessa, Di Gregorio and Di Marco, A. "Using ATL for Transformations in Software Performance Engineering: a Step Ahead of Java-based Transformations?" *Proceedings of the 7<sup>th</sup> International Workshop on Software and Performance* (2008). 127-132. Print.
- [CoIW99] Compare, Inverardi and Wolf, A. "Uncovering Architectural Mismatch in Component Behavior." *Science of Computer Programming* 33.2 (1999). 101-131. Print.
- [CoMi00] Cortellessa and Mirandola, R. "Deriving a Queueing Network Based Performance Model from UML Diagrams." *Proceedings of the 2<sup>nd</sup> International Workshop on Software and Performance* (2000). 58-70. Print.
- [CoMI03] Cortellessa, Di Marco and Inverardi, P. "Three Performance Models at Work: A Software Designer Perspective." *Proceedings of the 2<sup>nd</sup> International Workshop on Foundations of Coordination Languages and Software Architectures* 97. (2003). 219-239. Print.
- [Cort05] Cortellessa, V. "How Far Are We from the Definition of a Common Software Performance Ontology?" *Proceedings of the 5<sup>th</sup> International Workshop on Software and Performance* (2005). 195-204. Print.
- [CoTr08] Cortellessa and Trubiani, C. "Towards a Library of Composable Models to Estimate the Performance of Security Solutions." *Proceedings of the 7<sup>th</sup> International Workshop on Software and Performance* (2008). 145-156. Print.
- [Curt41] Curtiss, J. "On the Distribution of the Quotient of Two Chance Variables." *Annals of Mathematical Statistics* (1941). 477-480. Print.

- [Darw94] Dabrowski, Mills and Elder, J. "Understanding Consistency Maintenance in Service Discovery Architectures During Communication Failure." *Proceedings of the Third International Workshop on Software and Performance* (2002). 168-178. Print.
- [DeAM08] Deshpande, Apte and Marathe, S. "PerfCenter: a Performance Modeling Tool for Application Hosting Centers." *Proceedings of the 7<sup>th</sup> International Workshop on Software and Performance* (2008). 79-90. Print.
- [DePE04] Denaro, Polini and Emmerich, W. "Early Performance Testing of Distributed Software Applications." *ACM SIGSOFT Software Engineering Notes* 29.1 (2004). 94-103. Print.
- [DODA97] C4ISR Architecture Working Group. "C4ISR Architecture Framework Version 2.0." *Armed Forces Communications and Electronics Association*. US Department of Defense, 1997. Web. 11 Jan 2010.
- [DODI08] "Defense Acquisition Guidebook, Chapter 4 Systems Engineering." *Defense Acquisition University*. Defense Acquisition University, 2009. Web. 12 Jan 210.
- [DoWh07] Doyle and White, E. "Comparative Architecture Performance Analysis at Design Time." *Proceedings of the 33<sup>rd</sup> International Computer Measurement Group Conference* (2007). 231-242. Print.
- [Duga04] Dugan, R. F. "Performance Lies My Professor Told Me: the Case for Teaching Software Performance Engineering to Undergraduates." *Proceedings of the 4<sup>th</sup> International Workshop on Software and Performance* 29.1 (2004). 37-48. Print.
- [EdKa03] Eden and Kazman, R. "Architecture, Design, Implementation." *Proceedings of the 25<sup>th</sup> International Conference on Software Engineering* (2003). 149-159. Print.
- [EsWi01] Eshuis and Wieringa, R. "An Execution Algorithm for UML Activity Graphs." *Proceedings of the 4<sup>th</sup> International Conference on the Unified Modeling Language, Modeling Languages, Concepts, and Tools* (2001). 47-61. Print.
- [FaSS00] Fahringer, Scholz and Sun, X. "Execution-driven Performance Analysis for Distributed and Parallel Systems." *Proceedings of the 2<sup>nd</sup> International Workshop on Software and Performance* (2000). 204-215. Print.

- [FiHe10] Filliben, and Heckert, A. (2010). NIST/SEMATECH e-Handbook of Statistical Methods, Chapter 1.3.5.15. Retrieved from <http://www.itl.nist.gov/div898/handbook/eda/section3/eda35f.htm>. Web.
- [FrBo98] Franch and Botella, P. "Putting Non-Functional Requirements Into Software Architecture." *Proceedings of the 9<sup>th</sup> International Workshop on Software Specification and Design* (1998). 60-67. Print.
- [GaAO94] Garlan, Allen and Ockerbloom, J. "Exploiting Style in Architectural Design Environments." *Proceedings of the ACM SIGSOFT'94 Symposium on the Foundations of Software Engineering* 19.5 (1994). 175-188. Print.
- [GaAO95] Garlan, Allen and Ockerbloom, J. "Architectural Mismatch or Why It's Hard to Build systems Out of Existing Parts." *Proceedings of the 17<sup>th</sup> International Conference on Software Engineering* (1995). 179-185. Print.
- [GaMW97] Garlan, Monroe and Wile, D. "ACME: An Architecture Description Interchange Language." *Proceedings of Centers for Advanced Studies Conference, CASCON'97* (1997). 169-183. Print.
- [GaPZ94] Gannon, Purtilo and Zelkowitz, M. "Software Specification, A Comparison of Formal Methods." (1994). Print.
- [Garl95] Garlan, D. "An Introduction to the Aesop System." (1995). Print.
- [GaSh96] Shaw and Garlan, D. "Software Architecture, Perspectives on an Emerging Discipline." Englewood Cliffs: Prentice Hall, 1996. Print.
- [GHKR04] Gilmore, S., et al. "Software Performance Modelling Using PEPA Nets." *Proceedings of the 4<sup>th</sup> International Workshop on Software and Performance* (2004). 13-23. Print.
- [GoMe00] Gomaa and Menasce, D. "Design and Performance Modeling of Component Interconnection Patterns for Distributed Software Architectures", *Proceedings ACM Workshop on Software Performance, ACM Press*, (2000). 117-126. Print.
- [GoMK95] Gomaa, Menasce, and Kerschberg, L. "A Performance-Oriented Design Methodology for Large-Scale Distributed Data-Intensive Information Systems", *Proceedings IEEE International Conference on the Engineering of Complex Computer Systems*, (1995). 72-79. Print.
- [GoMS01] Gomaa, Menasce, Shin E., "Reusable Component Interconnection Patterns for Distributed Software Architectures," *Proceedings ACM Symposium on Software Reusability*, (2001). 69-77. Print.

- [GoRa91] Gorlick and Razouk, R. R. "Using Weaves for Software Construction and Analysis." *Proceedings of the 13<sup>th</sup> International Conference on Software Engineering* (1991). 23-34. Print.
- [GrMi04] Grassi and Mirandola, R. "Towards Automatic Compositional Performance Analysis of Component-based Systems." *Proceedings of the 4<sup>th</sup> International Workshop on Software and Performance* 29.1 (2004). 59-63. Print.
- [Gunt98] Gunther, N. "The Practical Performance Analyst." Boston: McGraw-Hill, 1998. Print.
- [GuPe02] Gu and Petriu, D. C. "XSLT Transformation from UML Models to LQN Performance Models." *Proceedings of the 3<sup>rd</sup> International Workshop on Software and Performance* (2002). 227-234. Print.
- [GuPe05] Gu and Petriu, D. C. "From UML to LQN by XML Algebra-based Model Transformations." *Proceedings of the 5<sup>th</sup> International Workshop on Software and Performance* (2005). 99-110. Print.
- [GuSh08] Gupta and Shirole, J. V. "Architecting, Developing and Testing for Performance of Tiered Collaboration Products." *Proceedings of the 7<sup>th</sup> International Workshop on Software and Performance* (2008). 25-32. Print.
- [HeHK02] Hermanns, Herzog and Katoen, J. "Process Algebra for Performance Evaluation." *Theoretical Computer Science* 274.1-2 (2002). 43-87. Print.
- [HHK+00] Hermanns, H., et al. "Compositional Performance Modelling With the TIPTool." *Performance Evaluation* 39.1-4 (2000). 5-35. Print.
- [Hoeb00] Hoeben, F. "Using UML Models for Performance Calculation." *Proceedings of the 2<sup>nd</sup> International Workshop on Software and Performance* (2000). 77-82. Print.
- [HoSK02] Hopkins, Smith and King, P. J. B. "Two Approaches to Integrating UML and Performance Models." *Proceedings of the 3<sup>rd</sup> International Workshop on Software and Performance* (2002). 91-92. Print.
- [HoWi07] Ho and Williams, L. "Developing Software Performance with the Performance Refinement and Evolution Model." *Proceedings of the 6<sup>th</sup> International Workshop on Software and Performance* (2007). 133-136. Print.

- [Hugh00] Hughes, P. H. "Toward a Common Process Model for Systems Development and Performance Engineering." *Proceedings of the 2<sup>nd</sup> International Workshop on Software and Performance* (2000). 115-116. Print.
- [ICSE07] "INCOSE Systems Engineering Handbook." *International Council on Systems Engineering*. 3.1 ed. 2007. Print.
- [Irvi09] Irvine, K. R. "Assembly Language for Intel Based Computers." Upper Saddle River: Prentice-Hall, 2009. Print.
- [IyRo02] Iyengar and Rosu, D. "Architecting Web Sites for High Performance." *Scientific Programming* 10.1 (2002). 75-89. Print.
- [JaBo05] Jansen and Bosch, J. "Software Architecture as a Set of Architectural Design Decisions." *Proceedings of the 5<sup>th</sup> Working IEEE/IFIP Conference on Software Architecture* (2005). 109-120. Print.
- [KayM00] Kay, K. "XSLT Programmer's Reference." Wrox Press, LTD., 2000. Print.
- [KiPo00] King and Pooley, R. "Derivation of Petri Net Performance Models from UML Specifications of Communications Software." *Proceedings of the 11<sup>th</sup> International Conference on Computer Performance Evaluation: Modelling Techniques and Tools* (2000). 262-276. Print.
- [Kobl00] Koyblitz, N. "A Course in Number Theory and Cryptography." Berlin: Springer, 2000. Print.
- [KoRe08] Koziolok and Reussner, R. "A Model Transformation from the Palladio Component Model to Layered Queueing Networks." *Proceedings of the SPEC international workshop on Performance Evaluation: Metrics, Models and Benchmarks* (2008). 58-78. Print.
- [Krey99] Kreyszig, E. "Advanced Engineering Mathematics." New York: John Wiley & Sons, Inc., 1999. Print.
- [Lako96] Lakos, J. "Large Scale C++ Software Design." New York: Addison-Wesley, 1996. Print.
- [LaSh79] Larson and Shubert, B. "Probabilistic Models in Engineering Sciences, Volume I, Random Variables and Stochastic Processes." New York: John Wiley & Sons, 1979. Print.

- [LeRS01] Levine, Smidt and Ramsey, P. P. "Applied Statistics for Engineers and Scientists: Using Microsoft Excel and Minitab." Englewood Cliffs: Prentice Hall, 2000. Print.
- [LiLW02] Liu, Law and Wiederhold, G. "Analysis of Integration Models for Service Composition." *Proceedings of the 3<sup>rd</sup> International Workshop on Software and Performance* (2002). 158-165. Print.
- [LKA+95] Luckham, D., et al. "Specification and Analysis of System Architecture using Rapide." *IEEE Transactions on Software Engineering* 21.4 (1995). 336-354. Print.
- [LoMC04] Lopez-Grao, Merseguer and Campos, J. "From UML Activity Diagrams to Stochastic Petri Nets: Application to Software Performance Engineering." *Proceedings of the 4<sup>th</sup> International Workshop on Software and Performance* (2004). 25-36. Print.
- [LTK+02] Lindemann, C., et al. "Performance Analysis of Time-enhanced UML Diagrams Based on Stochastic Processes." *Proceedings of the 3<sup>rd</sup> International Workshop on Software and Performance* (2002). 25-34. Print.
- [LuHa98] Luthi and Haring, G. "Mean Value Analysis for Queueing Network Models with Intervals as Input Parameters." *Performance Evaluation* (1998). Print.
- [LuVe95] Luckham and Vera, J. "An Event-Based Architecture Definition Language." *IEEE Transactions on Software Engineering* 21.9 (1995). 717-734. Print.
- [MaDK94] Magee, Dulay and Kramer, J. "Regis: A Constructive Development Environment for Distributed Programs." *IEEE/IOP/BCS Distributed Systems Engineering* 1.5 (1994). 304-312. Print.
- [MaHe01] Malony and Helm, B. R. "A Theory and Architecture for Automating Performance Diagnosis." *Future Generation Computer Systems* 18.1 (2001). 189-200. Print.
- [MaIn03] Di Marco and Inverardi, P. "Starting from Message Sequence Chart for Software Architecture Early Performance Analysis." *Proceedings of the 2<sup>nd</sup> International Workshop on Scenarios and State Machines: Models, Algorithms, and Tools* (2003). Print.

- [MBKR08] Martens, A. et al. "An Empirical Investigation of the Effort of Creating Reusable, Component-Based Models for Performance Prediction." *Proceedings of the 11th International Symposium on Component-Based Software Engineering* (2008). 16-31. Print.
- [MeAl96] Menascé and Almeida, V. "*Capacity Planning for Web Performance: Metrics, Models, and Methods.*" Upper Saddle River, Prentice-Hall, Inc., 1996. Print.
- [MeCM00] Merseguer, Campos and Mena, E. "A Pattern-based Approach to Model Software Performance." *Proceedings of the 2<sup>nd</sup> International Workshop on Software and Performance* (2000). 137-142. Print.
- [MeCo78] Melsa and Cohn, D. L. "Decision and Estimation Theory." New York: McGraw-Hill, 1978. Print.
- [MITR05] "MESA – Modeling Environment for Service-oriented-architecture (SOA) Analysis User Manual" Ver. 1.0. MITRE Corporation, (2005). Print.
- [MKUM09] Maia, P. et al. "Towards Accurate Probabilistic Models Using State Refinement". (2009) 281-284. Print.
- [MLH+00] de Miguel, M., et al. "UML Extensions for the Specification and Evaluation of Latency Constraints in Architectural Models." *Proceedings of the 2<sup>nd</sup> International Workshop on Software and Performance* (2000). 83-88. Print.
- [MMPT10] Malavolta, I., et al. "Providing Architectural Languages and Tools Interoperability Through Model Transformation Technologies." *IEEE Transactions on Software Engineering*. 36.1 (2010). 119-140. Print.
- [MoMu02] Mos and Murphy, J. "A Framework for Performance Monitoring, Modelling and Prediction of Component Oriented Distributed Systems." *Proceedings of the 3<sup>rd</sup> International Workshop on Software and Performance* (2002). 235-236. Print.
- [MeGo00] Menasce and Goma H. "A Method for Design and Performance Modeling of Client/Server Systems," *IEEE Transactions on Software Engineering* 26.11, (2000). 1066-1085. Print.
- [MEG+10] Menascé, D. et. al. "A Framework for Utility-Based Service Oriented Design in SASSY", *Proceedings of the First Joint WOSP/SIPEW International Conference on Performance Engineering*, (2010). 27-36. Print.



- [Mena08] Menascé, D. "Computing Missing Service Demand Parameters for Performance Models". *Proceedings of the Thirty-fourth International Computer Measurement Group Conference*, (2008). 241-247. Print.
- [MoRi97] Moriconi and Riemenschneider, R. A. "Introduction to SADL 1.0: A Language for Specifying Software Architecture Hierarchies, Technical Report SRI-CSL-97-01." (1997). Print.
- [MORT96] Medvidovic, N., et al. "Using Object-Oriented Typing to Support Architectural Design in the C2 Style." *Proceedings of the ACM SIGSOFT'96 Fourth Symposium on the Foundations of Software Engineering* 21.6 (1996). 24-32. Print.
- [MoSW08] Moreno, Smith and Williams, L. "Performance Analysis of Real-time Component Architectures: Aa Model Interchange Approach." *Proceedings of the 7<sup>th</sup> International Workshop on Software and Performance* (2008). 115-126. Print.
- [NaBG09] Naumovich, Bernardi and Gribaudo, M. "ITPN-PerfBound: a Performance Bound Tool for Interval Time Petri Nets." *Fifteenth International Conference on Tools and Algorithms for the Construction and Analysis of Systems* 5.2 (2009). 50-53. Print.
- [NPSH01] Nord, R., et al. "Effective Software Architecture Design: from Global Analysis to UML Descriptions." *Proceedings of the 23<sup>rd</sup> International Conference on Software Engineering* (2001). 741-742. Print.
- [OMBC09] "Assessing Risks and Returns: A Guide for Evaluating Federal Agencies' IT Investment Decision-making." *U.S. Government Accountability Office*. United States General Accounting Office, 1997. Web. 10 Jan 2010.
- [Papo65] Papoulis, A. "Probability, Random Variables, and Stochastic Processes." New York: McGraw-Hill, 1965. Print.
- [PBXB08] Pillana S. et al. "Automatic Performance Model Transformation from UML to C++." *Proceedings of the 2008 International Conference on Parallel Processing* (2008). 228-235. Print.
- [PeGo04] Pettit and Gomaa H., "Modeling Behavioral Patterns of Concurrent Software Architectures Using Petri Nets", *Proceedings of the Fourth Working IEEE/IFIP Conference on Software Architecture*. (2004) 57-68. Print.
- [PeGo06a] Pettit and Gomaa H., "Modeling Behavioral Design Patterns of Concurrent Objects", *Proceedings of the International Conference on Software Engineering*, (2006). 202-211. Print.

- [PeGo06b] Pettit and Gomaa H., "Modeling Behavioral Patterns of Concurrent Objects Using Petri Nets", *Proceedings of the Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing*. (2006). 303-312. Print.
- [PeSh02] Petriu and Shen, H. "Applying the UML Performance Profile: Graph Grammar-based Derivation of LQN Models from UML Specifications." *Proceedings of the 12<sup>th</sup> International Conference on Computer Performance Evaluation, Modelling Techniques and Tools* (2002). 159-177. Print.
- [PeWo02] Petriu and Woodside, M. "Performance Analysis in the Software Lifecycle, Analysing Software Requirements Specifications for Performance." *Proceedings of the 3<sup>rd</sup> International Workshop on Software and Performance* (2002). 1-9. Print.
- [PeWo05] Petriu and Woodside, M. "Software Performance Models from System Scenarios." *Performance Evaluation* 61.1 (2005). 65-89. Print.
- [PeWo92] Perry and Wolf, A. L. "Foundations for the Study of Software Architecture." *ACM SIGSOFT Software Engineering Notes* 17.4 (1992). Print.
- [PIFa02] Pllana and Fahringer, T. "On Customizing the UML for Modeling Performance-Oriented Applications." *Proceedings of the 5<sup>th</sup> International Conference on the Unified Modeling Language* 2460. (2002). 259-274. Print.
- [PSG+09] Pustina, L., et al. "A Practical Approach for Performance-driven UML Modelling of Handheld Devices-a Case Study." *Journal of Systems and Software* 82.1 (2009). 75-88. Print.
- [Rive98] Rivera J., "Modeling with Extend," *Proceedings of the 1998 Winter Simulation Conference*, (1998), 256-252, Print.
- [SaJP05] Sancho, Juiz and Puigjaner, R. "Automatic Performance Evaluation and Feedback for MASCOT Designs." *Proceedings of the 5<sup>th</sup> International Workshop on Software and Performance* (2005). 193-194. Print.
- [ScMS02] Schefczik, Mitschele-Thiel and Soellner, M. "On MSC-based Performance Simulation." *Proceedings of the 3<sup>rd</sup> International Workshop on Software and Performance* (2002). 166-167. Print.
- [ScSR00] Schmietendorf, Scholz and Rautenstrauch, C. "Evaluating the Performance Engineering Process." *Proceedings of the 2<sup>nd</sup> International Workshop on Software and Performance* (2000). 89-95. Print.

- [SDK+95] Shaw, M., et al. "Abstractions for Software Architecture and Tools to Support Them." *ACM SIGSOFT Software Engineering Notes* 21.4 (1995). 314-335. Print.
- [ShJT05] Sharma, Jalote and Trivedi, K. S. "Evaluating Performance Attributes of Layered Software Architecture." *Component-Based Software Engineering, 8<sup>th</sup> International Symposium, CBSE 2005* 3489. (2005). 66-81. Print.
- [ShTr05] Sharma and Trivedi, K. S. "Architecture Based Analysis of Performance, Reliability and Security of Software Systems." *Proceedings of the 5<sup>th</sup> International Workshop on Software and Performance* 30.5 (2005). 217-227. Print.
- [ShVN08] Shen, Virani and Niu, J. "Formalize UML 2 Sequence Diagrams." *Proceedings of the 2008 11<sup>th</sup> IEEE High Assurance Systems Engineering Symposium* (2008). 437-440. Print.
- [SiWo02] Siddiqui and Woodside, C. "Performance Aware Software Development (PASD) Using Resource Demand Budgets." *Proceedings of the 3<sup>d</sup> International Workshop on Software and Performance* (2002). 275-285. Print.
- [SLG+05] Smith, C. U., et al. "From UML Models to Software Performance Results: an SPE Process Based on XML Interchange Formats." *Proceedings of the 5<sup>th</sup> International Workshop on Software and Performance* (2005). 87-98. Print.
- [SmWi02a] Smith and Williams, L. "PASA<sup>SM</sup>: a Method for the Performance Assessment of Software Architectures." *Proceedings of the 28<sup>th</sup> International Computer Measurement Group Conference* (2002). 179-189. Print.
- [SmWi02b] Smith and Williams, L. "PASA<sup>SM</sup>: an Architectural Approach to Fixing Software Performance Problems." *Proceedings of the 28<sup>th</sup> International Computer Measurement Group Conference* (2002). 307-320. Print.
- [SmWi03] Smith and Williams, L. "Best Practices for Software Performance Engineering." *Proceedings of the 29<sup>th</sup> International Computer Measurement Group Conference* (2003). 83-92. Print.
- [Somm04] Sommerville, I. "Software Engineering." Pearson Addison Wesley, 2004. Print.

- [SPEC09] "SPEC CPU2006 Results." *Standard Performance Evaluation Corporation*. N.p., n.d. Web. 12 Jan 2010.
- [TaPe08] Tawhid and Petriu, D. "Towards Automatic Derivation of a Product Performance Model from a UML Software Product Line Model." *Proceedings of the 7<sup>th</sup> International Workshop on Software and Performance* (2008). 91-102. Print.
- [TKMG08] Teilans, A., et al. "Design of UML Models and Their Simulation Using ARENA." *WSEAS Transactions on Computer Research* 3.1 (2008). 67-73. Print.
- [Trac93] Tracz, W. "LILEANNA: a Parameterized Programming Language." *Proceedings of the Second International Workshop on Software Reuse* (1993). 66-78. Print.
- [TrGi08] Tribastone and Gilmore, S. "Automatic Extraction of PEPA Performance Models from UML Activity Diagrams Annotated with the MARTE Profile." *Proceedings of the 7<sup>th</sup> International Workshop on Software and Performance* (2008). 67-78. Print.
- [UcYa00] Uchitel and Yankelevich, D. "Enhancing Architectural Mismatch Detection with Assumptions." *Proceedings of the 7<sup>th</sup> IEEE International Conference and Workshop on the Engineering of Computer-Based Systems* (2000). 138-146. Print.
- [VDTD07] Verdickt, T., et al. "Hybrid Performance Modeling Approach for Network Intensive Distributed Software." *Proceedings of the 6<sup>th</sup> International Workshop on Software and Performance* (2007). 189-200. Print.
- [VeAp06] Verlekar and Apte, V. "A Methodology and Tool for Performance Analysis of Distributed Server Systems." *Proceedings of the 28<sup>th</sup> International Conference on Software Engineering* (2006). 913-916. Print.
- [Vest96] Vestal, S. "MetaH Programmer's Manual, Version 1.09 Technical Report." Plymouth: Honeywell Technology Center, 1996. Print.
- [WeLa99] Eds. Wertz, J. R. and Larson, W. "Space Mission Analysis and Design." Torrance: Microcosm Press, 1999. Print.
- [WHKT08] Wang, Z., et al. "Automatic Generation of SystemC Models from Component-based Designs for Early Design Validation and Performance Analysis." *Proceedings of the 7<sup>th</sup> International Workshop on Software and Performance* (2008). 139-144. Print.

- [WiKe00] Wilson and Kesselman, J. "Java Platform Performance Strategies and Tactics." Boston: Addison-Wesley, 2000. Print.
- [WPP+05] Woodside, M., et al. "Performance by Unified Model Analysis (PUMA)." *Proceedings of the 5<sup>th</sup> International Workshop on Software and Performance* (2005). 1-12. Print.
- [WuWo04] Wu and Woodside, M. "Performance Modeling from Software Components." *Proceedings of the 4<sup>th</sup> International Workshop on Software and Performance* 29.1 (2004). 290-301. Print.
- [WuXi04] Wu and Woodside, M. "Performance Modeling From Software Components." *Proceedings of the Fourth International Workshop on Software and Performance* 29.1 (2004). 290-301. Print.
- [XuJi08] Xu, J. "Rule-based Automatic Software Performance Diagnosis and Improvement." *Proceedings of the 7<sup>th</sup> International Workshop on Software and Performance* (2008). 1-12. Print.
- [XuKu98] Xu and Kuusela, J. "Modeling Execution Architecture of Software System Using Colored Petri Nets." *Proceedings of the 1<sup>st</sup> International Workshop on Software and Performance* (1998). 70-75. Print.
- [ZhSh89] Zhang and Shasta, D. "Simple Fast Algorithms for the Editing Distance Between Trees and Related Problems." *SIAM Journal of Computing* 18.6 (1989). 1245-1262. Print.
- [ZWR+01] Ziegenbein, D., et al. "Interval-Based Analysis of Software Processes." *Proceedings of the 2001 ACM SIGPLAN Workshop on Optimization of middleware and distributed systems* 36.8 (2001). 94-101. Print.

## **CURRICULUM VITAE**

Gerald S. Doyle was born in Morristown, NJ. He was awarded a Bachelor of Science degree from the United States Military Academy in 1973, a Masters Degree in Electrical Engineering from the Naval Postgraduate School in 1980, and a Masters Degree in Computer Science from George Mason University in 2000.

After receiving his commission in 1973, he served in the United States Army, during which time he commanded a Signal Company in Korea and two Artillery Batteries in Fort Sill before attending the Signal Officer's Advanced course in Fort Gordon, Georgia. After a short assignment with the Defense Data Network Program Management Office, he was appointed to teach physics, electromagnetism and the statistical experimentation for the Department of Physics at the United States Military Academy. He completed a program at the Armed Forces Staff College in Norfolk, Virginia and was then assigned in 1988 as the Deputy Commander of NATO's largest Communications Logistics Depot, the AFCENT Central Region. In 1991 he rejoined DISA as the chief of the Advanced Technologies and Network Management Division.

He entered the federal civil service with the Defense Information Systems Agency in 1993. During his service in a variety of satellite and space related assignments, he helped engineer technical standards for communication and network control for the SATCOM environment. As part of DISA's most recent transformational effort he was selected to be the Chief Engineer and Deputy of the Transport Division. In October of 2004, he was chosen to be the Chief of the DISA Systems Engineering Architecture and Integration Center. In June 2005, Mr. Doyle was appointed to the Senior Executive Service as the DISA Systems Engineering Transformational Executive where he currently heads the Systems Engineering Center. In October 2010, he was appointed as the Principal Director for Engineering for the Defense Information Systems Agency.