Steps in an Architecture Tradeoff Analysis Method: Quality Attribute Models and Analysis

Mario R. Barbacci
S. Jeromy Carriere
Peter H. Feiler
Rick Kazman
Mark H. Klein
Howard F. Lipson
Thomas A. Longstaff
Charles B. Weinstock

May 1998

TECHICAL REPORT
CMU/SEI-97-TR-029
ESC-TR-97-029

CMU/SEI-97-TR-029
ESC-TR-97-029

Steps in an Architecture Tradeoff Analysis Method: Quality Attribute Models and Analysis

Mario R. Barbacci
S. Jeromy Carriere
Peter H. Feiler
Rick Kazman
Mark H. Klein
Howard F. Lipson
Thomas A. Longstaff
Charles B. Weinstock

May 1998

Architecture Tradeoff Analysis Initiative

Carnegie Mellon University
Software Engineering Institute

Pittsburgh, PA
15213-3890

This report was prepared for the SEI Joint Program Office

HQ ESC/DIB
5 Eglin Street
Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER

(signature on file)

Mario Moya, Maj, USAF
SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number F19628-95-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 52.227-7013.

This document is available through Asset Source for Software Engineering Technology (ASSET): 1350 Earl L. Core Road; PO Box 3305; Morgantown, West Virginia 26505 / Phone: (304) 284-9000 or toll-free in the U.S. 1-800-547-8306 / FAX: (304) 284-9001 World Wide Web: http://www.asset.com / e-mail: sei@asset.com

Copies of this document are available through the National Technical Information Service (NTIS). For information on ordering, please contact NTIS directly: National Technical Information Service, U.S. Department of Commerce, Springfield, VA 22161. Phone: (703) 487-4600.

This document is also available through the Defense Technical Information Center (DTIC). DTIC provides access to and transfer of scientific and technical information for DoD personnel, DoD contractors and potential contractors, and other U.S. Government agency personnel and their contractors. To obtain a copy, please contact DTIC directly: Defense Technical Information Center / Attn: BRR / 8725 John J. Kingman Road / Suite 0944 / Ft. Belvoir, VA 22060-6218 / Phone: (703) 767-8274 or toll-free in the U.S.: 1-800 225-3842.

Table of Contents

1 Why Architecture Tradeoff Analysis? 1

2 Requirements and Architectural Views 5
2.1 Problem Description 5
2.2 Identification of Requirements and Constraints 6
2.3 Structural View of the Architecture 8

3 Attribute Models 11
3.1 Performance Model 11
3.1.1 Execution Path Components 11
3.1.2 Latency and Jitter Equations 13
3.2 Availability Model 14
3.3 Security Model 15

4 Attribute Analysis 17
4.1 Performance Analysis 17
4.2 Availability Analysis 19
4.3 Security Analysis 20
4.3.1 Man-in-the-Middle Attack Scenario 21
4.3.2 Spoof-the-Server Attack Scenario 21
4.3.3 Parameters for Security Scenarios 22

5 Model Sensitivity and Tradeoffs 25
5.1 Comparison Against Requirements 25
5.2 Sensitivity and Tradeoffs 26

6 Conclusions 29

References 31

List of Figures

Figure 1-1	The Architecture Business Cycle [Bass 98]	2
Figure 1-2	Information in the Blackboard	3
Figure 2-1	Structural View of the Architecture	8
Figure 3-1	Performance Model Paths and Components	12
Figure 4-1	Man-in-the-Middle Attack	21
Figure 4-2	Spoof-the-Server Attack	22

List of Tables

Table 2-1	Requirements and Attributes	7
Table 4-1	Performance Summary	19
Table 4-2	Availability Summary	20
Table 4-3	Security Summary	23
Table 5-1	Sensitivity to the Number of Servers and Server Failure Rates	27

Abstract

This paper presents some of the steps in an emerging architecture tradeoff analysis method (ATAM). The objective of the method is to provide a principled way to understand a software architecture's fitness with respect to multiple competing quality attributes: modifiability, security, performance, availability, and so forth. These attributes can interact or conflict—improving one often comes at the price of worsening one or more of the others, thus it is necessary to trade off among multiple software quality attributes at the time the software architecture of a system is specified, and before the system is developed. This report illustrates typical quality attribute models, analyses, and tradeoffs using a small real-time industrial application.

1 Why Architecture Tradeoff Analysis?

In large software systems, the achievement of qualities such as performance, availability, security, and modifiability is dependent not only upon code-level practices (e.g., language choice, detailed design, algorithms, data structures, and testing), but also upon the overall software architecture. Quality attributes of large systems can be highly constrained by a system's software architecture. Thus, it is in our best interest to try and determine at the time a system's software architecture is specified whether the system will have the desired qualities.

A variety of qualitative and quantitative techniques are used for analyzing specific quality attributes [Barbacci 95]. These techniques have evolved in separate communities, each with its own vernacular and point of view and have typically been performed in isolation. However, the attribute-specific analyses are *inter*dependent, for example, performance affects modifiability, availability affects safety, security affects performance, and everything affects cost. In other words, each quality attribute has interfaces to other attributes. These interfaces represent dependencies between attributes and are defined by parameters that are shared among attribute models. If we can identify these interfaces, the results from one analysis can feed into the others. This is the principal difference between an architecture tradeoff analysis and other software analysis techniques—that it explicitly considers the *interfaces* between multiple attributes, and permits principled reasoning about the tradeoffs that inevitably result from such connections. Other analysis frameworks, if they consider connections at all, do so only in an informal fashion, or at a high level of abstraction (see [McCall 94, Smith 93]).

In addition to the technical factors represented by the quality attribute's models and analysis, a software architecture is influenced by business and social forces from multiple stakeholders. Thus, design decisions are often made for non-technical reasons: strategic business concerns, meeting the constraints of cost and schedule, using available personnel, and so forth. "[The message] is that the relationships among business goals, product requirements, practitioner's experience, architectures, and fielded systems form a cycle with feedback loops that a business can manage" [Bass 98]. This "architecture business cycle" is illustrated in Figure 1-1.

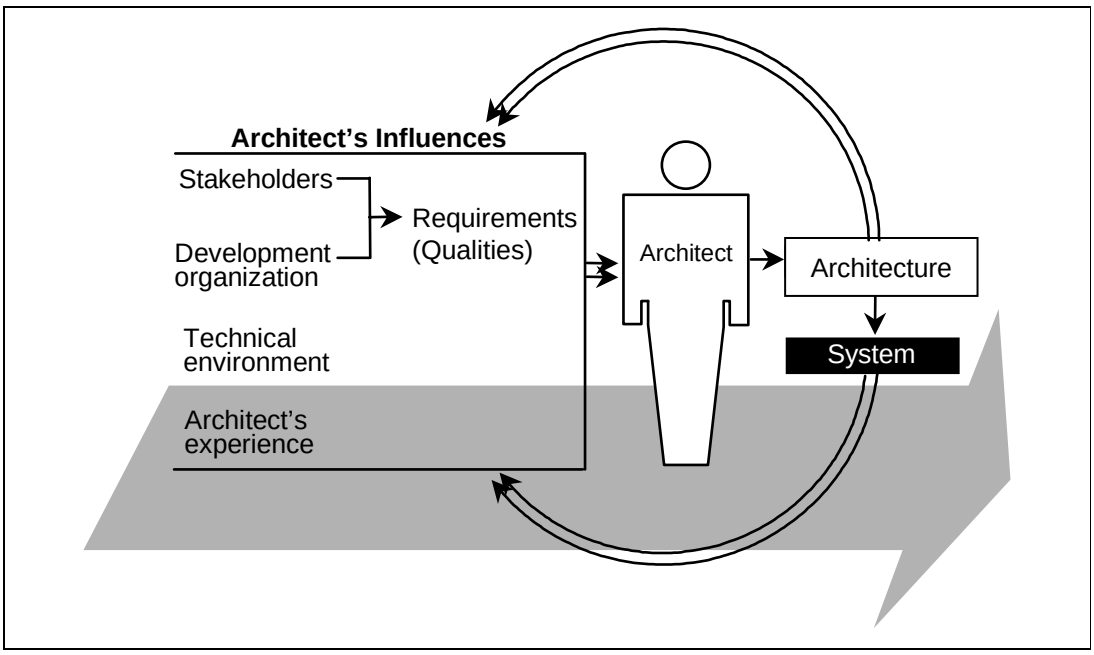

Figure 1-1: The Architecture Business Cycle [Bass 98]

There are multiple activities involved in the architecture business cycle:

- creating the business case for the system
- understanding the requirements
- creating or selecting the architecture
- representing and communicating the architecture
- analyzing or evaluating the architecture
- implementing the system based on the architecture
- ensuring that the implementation conforms to the architecture

These activities do not take place in a strict sequence and there are many feedback loops as the multiple stakeholders negotiate among themselves, striving for some consensus. To visualize the process, imagine a blackboard, Figure 1-2, in which the participants read and write various types of information (e.g., requirements, constraints, evaluation results) without necessarily following a set order or rank. The implication is that potentially any of the stakeholders (architects, attribute experts, developers, etc.) can make use of information developed by any other stakeholder and can introduce information that could be of interest to anyone else.

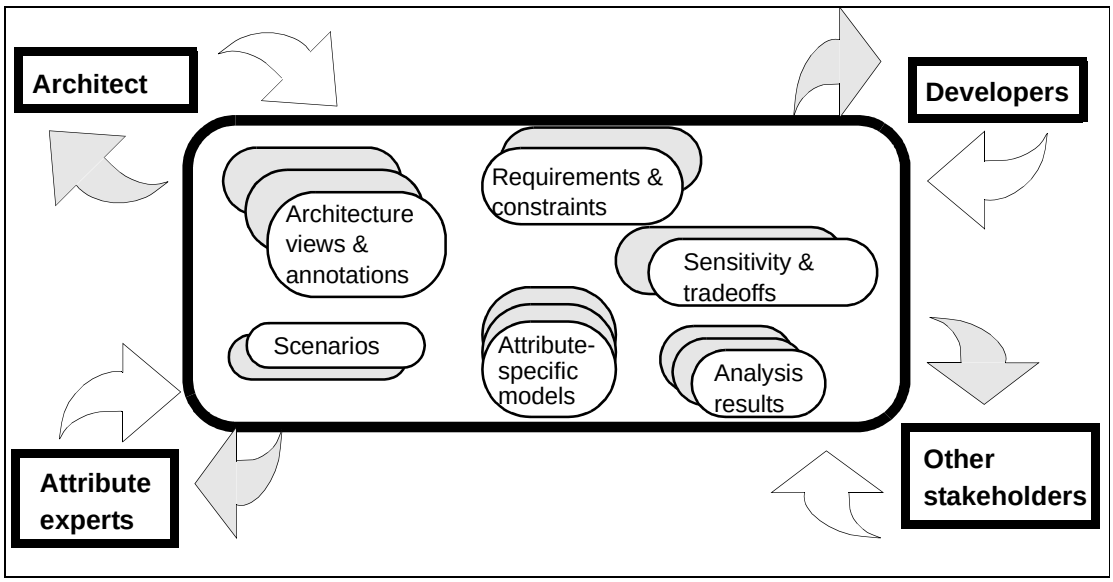

Figure 1-2: Information in the Blackboard

The purpose of this report is to illustrate, through a simple problem, the attribute modeling, analysis, and tradeoff activities suggested by the blackboard "boxes." We have chosen a "model system" that has been used in several case studies [Nielsen 87, Sanden 89, Smith 93]. We have generalized the problem description by requiring support for multiple host computers acting as clients — the single host computer in the original problem becomes a special case. This extension provides a richer basis for exploring a variety of quality attribute issues.

In Chapter 2 we present the problem, identify the requirements and constraints, and present a structural view of an architecture to be analyzed. In Chapter 3 we present attribute-specific models for performance, availability, and security. The models to be considered in an architecture tradeoff analysis are determined by the system requirements, the architectural views, and the experience of the attribute specialists and the architect.

In Chapter 4 we apply the attribute models developed in the previous chapter and carry out the analyses. To conduct the analyses we must identify values for the parameters of the models. As we shall see, these parameters can be explicit in the requirements or architectural view, discovered in the models, or assumed in scenarios used to carry out the analyses. In all cases, the values are either known from prior experience or are assumed to be so, but subject to confirmation during development. The discovered and assumed parameters have to be added to the

architecture as refinements or annotations. Since any assumptions, constraints, etc. needed for a model could potentially affect other models, the annotation of the architectural views—together with the scenarios—must be exposed or communicated to all stakeholders.

Finally, Chapter 5 discusses the sensitivity of the results to the information used in the analyses and the tradeoff points between attributes.

2 Requirements and Architectural Views

2.1 Problem Description

The system is a simple industrial application consisting of a remote temperature sensor (RTS) and a number of remote host computers (operator stations). The RTS controls a battery of furnaces and a digital thermometer. The function of the RTS is to periodically query the thermometer for the temperature of a specified furnace and to report the temperatures to the host computers. Safe operation of the plant requires that it operates with minimal downtime and that the temperature readings are timely and accurate.

The RTS and the host computers communicate by passing messages on a local area network. To simplify the exposition, we assume that the RTS is a simple device, capable of doing one thing at a time. Thus, there is no overlap between internal operations and transmission of reports over the network.

The hosts specify the frequency of temperature readings for each furnace of interest by sending occasional control requests specifying a furnace number and the interval (10 to 99 seconds) between temperature readings. Control requests are stored in a FIFO queue until the RTS is free to process the requests.

Processing a control request involves updating the schedule of periodic readings, requesting an initial temperature reading from the thermometer, and storing the temperature reading in a FIFO output queue until the RTS is free to transmit temperature reports to the requesting hosts.

Processing periodic reports involves keeping track of time, requesting a temperature reading from the thermometer at the scheduled reading time, and storing the temperature reading in a FIFO output queue until the RTS is free to transmit temperature reports to the requesting hosts.

To obtain a temperature reading, the RTS calls the thermometer task with a furnace number as a parameter and the thermometer returns the temperature of the furnace, a value in the range of 0-1000^{o}C.

2.2 Identification of Requirements and Constraints

There are a number of formal and informal techniques for requirements elicitation and validation, including introspection, questionnaires, interviews, and protocol analysis; any number of techniques or combinations thereof could be used. However, independently of how we derive the requirements, we must link them to the architecture views, attribute models, analysis, and other information in the blackboard. That is, the information written in the requirements box of the blackboard has to be meaningful to all the stakeholders. It would be useful, for example, to categorize requirements according to attribute-specific concerns and use this to guide the refinement of requirements.[1]

Although the problem description is not very detailed and does not provide a great deal of quantitative information, we can nevertheless identify several attributes of concern as a starting point. It does not matter if we miss a hidden requirement or if we can not assign a value to a requirement at this time. The development of the attribute models and their analysis will provide grounds for quantifying or identifying missing requirements.

Table 2-1 quotes the word(s) in the system description that suggest that the particular attribute/concern is important (i.e., that there is a requirement to address the concern). In general, the requirement will fall into several classes:

- explicit: appears in the original problem description
  - bound or specified (e.g., FIFO queueing, range of reading intervals)
  - unbound or unspecified (e.g., acceptable RTS downtime)
- discovered: identified by the models and added to the requirements as either bound or unbound requirements (e.g., RTS failure and repair rates)
- assumed: identified by the models and postulated in scenarios as values for unbound requirements (e.g., types of attacks the system must resist)

We use scenarios to explore the space of requirements and constraints. Scenarios help put in concrete terms otherwise vague or unquantified requirements and constraints. They also facilitate communication between stakeholders because it forces them to agree on their perception of the requirement or constraint.

1. This is a somewhat recursive argument: We need to identify the relevant attributes and them map the requirements to these attributes but we don't know what the relevant attributes are until we have the requirements. This is one of the reasons why a fixed sequence of steps is hopeless and a shared blackboard works better than a pipeline.

Statement in problem description	Suggested attribute	Attribute specific concern	Example of requirement statement
Minimal downtime (requirement)	RTS availability	RTS downtime (system not operational)	The RTS must be operational W% of the time.
Timely readings (requirement) Periodic readings (requirement) Reading intervals range from 10 to 99 seconds. (constraint)	RTS and network performance	latency (delay of temperature reports) jitter (variability in delay)	The hosts must receive periodic reports within R ± D (or ΔD) seconds of their scheduled reading. The hosts must receive an initial report within R ± D (or ΔD) seconds of sending the request.
Accurate readings (requirements) Furnace temperature ranges from 0 to 1000^{o}C (constraint)	RTS and network security	integrity of the reports	The reports must be protected against attacks of type A with probability P.
	thermometer accuracy	deviation from true value	The temperature reading must be with R% of its true value.
Sequential operation (constraint)	RTS and network performance	latency and throughput (number of reports per minute)	The system must process at least R reports per minute.
Multiple furnaces and hosts (requirement)	RTS and network performance	throughput	The system must support up to F furnaces, H hosts, and a load distribution L.
Occasional control requests (constraint)	RTS and network performance	throughput	A host issues a control request every C minutes. The control requests constitute X% of the traffic.

Table 2-1: Requirements and Attributes

2.3 Structural View of the Architecture

Figure 2-1 presents a structural view of the architecture, in which multiple servers implement the functionality of the RTS and where the host computers act as clients, evenly allocated to the servers. This is the architecture view that will be the focus of the modeling and analysis This particular architecture was chosen because it is a direct map of the problem description.[1]

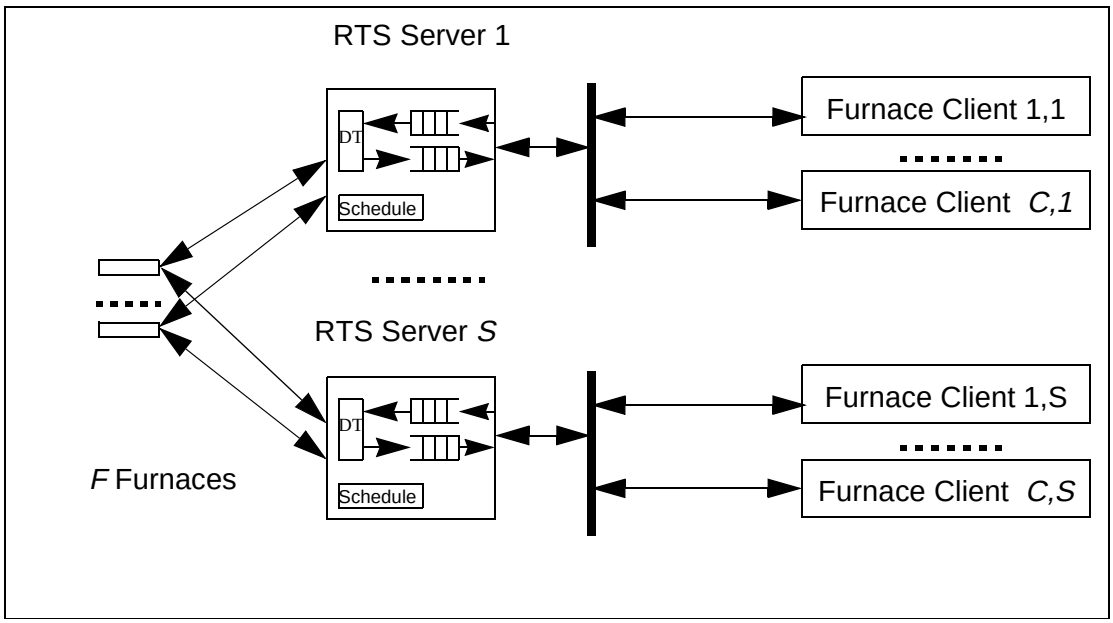

Figure 2-1: Structural View of the Architecture

In this structure, the path for a control request generated by a client and the corresponding temperature report sent back by a server consists of the following steps:

1. A control request message is transmitted via the LAN and stored in the input queue.
2. The control request eventually arrives at the head of the input queue and is processed:
   a. update the schedule for temperature readings
   b. request a temperature reading from thermometer, as if a scheduled event time had arrived
3. The temperature reading is stored in the output queue.
4. The temperature report eventually arrives at the head of the output queue and is transmitted via the LAN to the client.

1. This begs several questions: Who did it? How did it happen? Is it always a structural view? The fact is, there is always a creative step at the beginning and the initial candidate just happens. An alternative way to phrase this is that every analysis starts with a little synthesis.

The path for a periodic temperature report generated by a server consists of the following steps:

1. A periodic temperature reading is performed:
    a. wait for the next scheduled reading
    b. request a temperature reading from the thermometer
2. The temperature reading is queued in the output queue.
3. The periodic temperature report eventually arrives at the head of the output queue and is transmitted via the LAN to the client.

As in the case of the requirements, the structural view of the architecture must be annotated with information or parameters needed for the attribute specific models. These parameters fall into several classes:

- explicit: appear in the initial architecture view
    - bound or specified (e.g., execution paths)
    - unbound or unspecified (e.g., S servers, C clients)
- discovered: identified by the models and added to the architecture view as either bound or unbound parameters (e.g., operation times, transmission times, failure rates)
- assumed: identified by the models and postulated in scenarios as values for unbound parameters of the architecture (e.g., minimal, maximal, and average loads; probability of success of an attack)

We use scenarios to explore the space defined by the attribute models. Scenarios help put in concrete terms parameters of the models that are not part of the architecture. They also facilitate communication between stakeholders because it forces them to agree on their perception of the architecture.

3 Attribute Models

In this section we illustrate models for performance, availability, and security—the attributes suggested by the requirements in the problem description. Notice that all of these attributes are observable at runtime (i.e., are exhibited by the system during execution). Other attributes not observable at runtime (e.g., modifiability) could be included but the level of detail in the problem description, requirements, and architectural view is insufficient to make this a useful or interesting exercise.

3.1 Performance Model

In a real-time system latency and jitter are frequent concerns and these will be the focus of our performance modeling. Latency and jitter of the temperature reports are important because the information might be used by the client to perform operations that could have disastrous consequences if the temperature reports arrive late or at unpredictable times.

The performance analysis involves computing latency (in simple cases) by tracing the execution path of a function and assigning latencies to each portion of the path. More complex cases involving throughput and capacity can be analyzed using either scheduling, simulation, or queueing models [Audsley 95, Conway 67, Klein 93, Lehoczky 94, Smith 90, Stankovic 95]. To carry out the performance analysis we can use direct execution scenarios to identify the execution paths, complemented by checklists to identify or assign arrival rates and distributions to each component and connector in the path. Given the arrival rates and distributions for each component, we can compute the latency, throughput, and capacity of the system as a whole.

3.1.1 Execution Path Components

To build the latency models, we need to identify the paths, the components, and the execution times to be added, and all of these must be identified in the architecture. From the structure of the architecture, we identify the components involved in the execution path of an activity and for each component we must determine an execution time. These execution times, unless already specified in the architecture, are discovered parameters needed by the performance model.[1]

The activities of interest are the control requests and the generation of periodic reports; their execution paths are shown in Figure 3-1. The components involved in the operations are

- Local area network. Network transmission time for control request messages is C_{net}. Network transmission time for periodic reports is C_{net}.
- Input and output queues. Queues for control requests received by the server and temperature reports ready for transmission to the clients. Queueing time for control packets is C_{dq}. Queueing time for temperature reports is C_{dq}.
- Digital thermometer. Temperature reading time is included in the caller's operation.
- Task for control requests (calls thermometer). Control request processing time, including thermometer invocation, is C_{fnc}.
- Task for periodic reports (calls thermometer). Scheduler processing time, including thermometer invocation, is C_{fnc}.

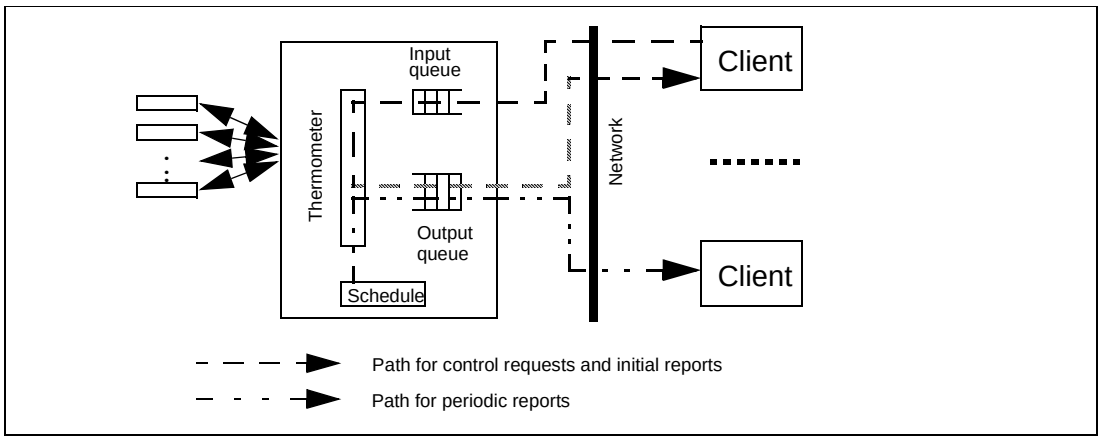

Figure 3-1: Performance Model Paths and Components

We are making two simplifying assumptions. First, queueing operations for control requests (input queue) and temperature reports (output queue) take the same amount of time, C_{dq}. The second assumption is that the processing time to generate temperature reports in response to a control request (initial reports) or in response to a scheduled reading (periodic reports) take the same amount of time, C_{fnc}. The times could be different due to different message sizes but the difference is negligible. In any event, as we shall see, different queueing and processing times could be easily included in the models. With these path definitions, we can now express the latency and jitter models.

1. The models here measure latency between the client sending the request and the client receiving the result. Another interesting latency that might be considered is the time between the operator issuing the request and the operator seeing the result (i.e., We would need to add the computing time of the client software.).

3.1.2 Latency and Jitter Equations

We distinguish the latency of an initial temperature report (i.e., the first report in response to a control request) from the latency of a periodic report. These are the two activities of interest identified from the problem description and the requirements.

The worst case control latency (WCCL) occurs when all clients assigned to a server simultaneously send control requests for readings from all furnaces. The worst latency affects the client whose request is the last one to be received by the server because requests are processed in FIFO order (i.e., its initial reading will also be the last to be sent by the server). If we assume that the clients (C) are evenly allocated to the servers (S), and each client wants to engage all F furnaces, WCCL is given by Equation 3-1:

$$\mathbf{WCCL = C/S * F * (2 * C_{net} + 2 * C_{dq} + C_{fnc})} \qquad \textbf{[Equation 3-1]}$$

Since control requests are infrequent, additional models (i.e., average and best case) might not tell us much. This is the most stressful case. If the queueing, processing, and network times for control requests and periodic reports are different, the times in the equation above should be those of the control request, and the times in the equations below should be those of the periodic reports.

In general, the latency of a periodic report (PL) is given by Equation 3-2:

$$\mathbf{PL = (Q+1) * (C_{dq} + C_{fnc} + C_{net})} \qquad \textbf{[Equation 3-2]}$$

Q is the number of temperature reports scheduled to take place at the same time as the report of interest. The new report is placed at the end of the queue and has to wait until its turn arrives (i.e., until there are no reports ahead of it). Notice that the formula assumes no concurrency between network transmission and server operation time.

The worst case periodic latency (WCPL) occurs when all clients assigned to a server have scheduled readings from all furnaces and these are all due at the same time (**Q = C/S*F**). Obviously, the best case periodic latency (BCPL) occurs when there are no reports in the queue (**Q = 0**). Varying Q, we can bracket the latency affecting a periodic temperature report (see Equations 3-3 and 3-4).

$$\mathbf{WCPL = C/S * F * (C_{dq} + C_{fnc} + C_{net})} \qquad \textbf{[Equation 3-3]}$$

$$\mathbf{BCPL = (C_{dq} + C_{fnc} + C_{net})} \qquad \textbf{[Equation 3-4]}$$

Jitter is defined as the variation in latency from the ideal or best case (see Equation 3-5):

Jitter = PL - BCPL **[Equation 3-5]**

3.2 Availability Model

We focus on evaluating the availability of the servers (i.e., the amount of time the servers are working). Availability of the temperature reports is important because the information might be used by the client to perform operations that could have disastrous consequences if the temperature reports are missing.

In performing availability analysis, we rely on Markov modeling to calculate the reliability and availability of the system as a whole from the reliability of the system's parts. To conduct the availability analysis, we assign each component a failure rate and a repair rate—the rate at which this component recovers from a failure—obtained from questionnaires or checklists, depending on the maturity of the domain or prior experience with similar components. To understand the availability of the RTS, we use a machine repair model with S machines and one repairman (see for example [Ross 89]). The amount of time each machine operates before breaking down is exponentially distributed with mean $1/\lambda$ (the failure rate of machines is λ). The amount of time that it takes to repair is exponentially distributed with mean $1/\mu$ (the repair rate is μ). Both λ and μ are discovered parameters needed by the availability model.

In the machine repair model we say that the system is in state *n* whenever *n* machines are not it use. In our situation, we say that the system is operating (albeit with diminished capacity) whenever at least one server is operating and we say that the system is down whenever all S servers are down. The long-run proportion of time that the system is in state S (i.e., the system is down, by our definition) is given by Equation 3-6:

[Equation 3-6]

$$D_S = \frac{S! \cdot \left(\frac{\lambda}{\mu}\right)^S}{1 + \sum_{n=1}^{S} \left(\frac{\lambda}{\mu}\right)^n \times \frac{S!}{(S-n)!}}$$

Availability is simply the complement, the long-run proportion of time that the system is not in state S (where all the servers are down) (see Equation 3-7):

$A = 1\text{-}D_S$ **[Equation 3-7]**

3.3 Security Model

We focus on the accuracy of the temperature reports sent from the servers to the clients. This information might be used by the clients to perform operations that could have disastrous consequences if the temperature reports are altered by an attacker.[1] Thus we are concerned with the integrity of the information passing between the servers and clients.

There are several recognized techniques for a security analysis. The two major categories of these techniques are formal analysis and scenario-based analysis. In formal analysis, mathematical specifications of the requirements are used as goals to be verified from a rigorous specification of the behavior of the architecture. In scenario-based analysis, specific threats and objects to be protected in a system are identified, then the defensive measures are tested under expected behavior of the threat scenarios to determine if the goals of the security requirements have been met. Formal analysis is best suited for small, fully-specified systems that are designed around a detailed security model. The scenario-based analysis is more suited to a balanced approach of attribute analysis and will be used for the system under consideration.

The window of opportunity, W, for an intruder is the length of time the intruder can operate undetected. This means that after about W minutes, it is likely that the intruder would be detected by an operator and traced to a point of origin. Within the window, there is an attack rate (R) that the intruder is capable of — this activity includes target selection of a particular client/server connection to attack and investigation of the specific properties of this connection to attack. This rate, multiplied by the exposure time (W), gives the number of system attacks attempted by an intruder during the exposure window. For an attempted intrusion to be successful, the intruder must then successfully apply the attack (e.g., kill server) with some probability of success P. In general the attack will consist of several components with probabilities of success, P_1 ... P_n, and the general expression for the expected number of successful attacks is shown is Equation 3-8:

$$E = W * R * P_1 * ... * P_n \qquad \textbf{[Equation 3-8]}$$

To calculate the number of successful attacks within an acceptable window of opportunity for an intruder, we need to define various parameters associated with the environment, including probabilities of successful attacks using standard assumptions and other values reasonable for the design.[2] These are discovered parameters needed by the security models.

---

1. There are no requirements that temperature reports be hidden or kept secret. Thus we don't care if attackers can read the temperature reports, we just don't want them to modify the values.

2. In general, these probabilities are dependent on the operational environment of the delivered system, which includes such factors as operator training and patch management for the operating system. These dependencies are out of the scope of architectural analysis at this level of abstraction, but must be considered later in the design process.

4 Attribute Analysis

In this chapter we apply the attribute models previously developed. As we will see, not all the information needed for the analyses is available from the requirements and architectural views. To conduct the analyses we must assign values to the parameters in the various models.

4.1 Performance Analysis

For purposes of illustration let's assume the following values for the performance parameters:

- F = number of furnaces = 16
- S = number of servers = 2
- C = number of clients = 8
- C_{net} = Network transmission time = 120 ms
- C_{dq} = Queue operation time = 10 ms
- C_{fnc} = Processing time = 160 ms

There is one additional parameter in the equations that is not really part of the architecture but depends on the assumed behavior of the clients and their demand on the system. This is Q, the number of simultaneous periodic reports (Equation 3-2). This has to be postulated via scenarios.

The calculation of the worst case control latency (Equation 3-1) and best case periodic latency (Equation 3-4) are fairly straightforward since they do not depend on the length of the queue for periodic reports:[1]

$$WCCL = C/S * F * (2 * C_{net} + 2 * C_{dq} + C_{fnc}) = 8/2 * 16 * (2*120 + 2*10 + 160) = 26{,}880 \text{ ms}$$

$$BCPL = (C_{net} + C_{dq} + C_{fnc}) = (120 + 10 + 160) = 290 \text{ ms}$$

To calculate the worst case periodic latency, WCPL, we must postulate various combinations of how many clients have requested temperature readings from each furnace and the frequency

1. Well, not exactly. The case of the control request assumes that the server was idle and there were no pending periodic reports in the output queue when the control requests arrived. To be more realistic, we would add to WCCL the time to transmit any pending periodic reports and this could be extracted from one of the subsequent scenarios.

of these readings. If we had some idea of the arrival rates for control requests and the distribution of periodic reading intervals, we could use queueing theory to compute the queue lengths. This information might not be available and the queue length might have to be guessed using "back of the envelope" estimates. However, in this case we face a possible explosion of scenarios, given the millions of possible arrangements.[1] Rather than engage in any exhaustive exploration of the space, we will pick a heavy load and a medium load scenario. In actual practice, it would be up to the stakeholders to decide if these scenarios are representative of the workload or if additional scenarios are needed.

Heavy load scenario — The heaviest load occurs when all clients assigned to a server have scheduled all furnaces making a total of C/S*F = 8/2 * 16 = 64 scheduled temperature reports. The assumption is that all reports are synchronized and have the same period, thus they are all due at once, regardless of how much time elapses between the occurrences of the event. When the appointed time arrives, the server goes into overdrive, getting temperature readings from the thermometer, queueing them up as fast as it can, and finally transmitting them when all 64 readings are queued. The periodic latency (Equation 3-2) for the heavy load scenario is

$$PL_{HL} = Q * (C_{net} + C_{dq} + C_{fnc}) = 64 * (120 + 10 + 160) = 18{,}560 \text{ ms}$$

If the assumed common period is greater than 18,560 milliseconds the system can handle the load because it will be able to flush the queue before the next block of readings is due. If the common period is smaller that 18,560 milliseconds the system will never keep up.[2]

Moderate load scenario — A moderate load could have each client engaging only 8 of the 16 furnaces for a total of 32 (8/2*8) scheduled readings. If all readings are synchronized and due at once, the periodic latency (Equation 3-2) for the moderate load scenario is

$$PL_{ML} = Q * (C_{net} + C_{dq} + C_{fnc}) = 32 * (120 + 10 + 160) = 32*290 = 9{,}280 \text{ ms}$$

That is, even if all 32 scheduled readings had the minimal reading frequency (10 seconds), the system can handle the whole queue before the next group of readings is due. We could try any number of additional combinations of clients, furnaces, and reading intervals and repeat the calculations.

1. Rounding reading intervals to the nearest second, each client can engage up to 2^{16} combinations of furnaces and each furnace can be assigned 89 (i.e., 99-10) possible intervals, for a grand total of 46,661,632 cases.

2. The assumption that all reports are synchronized is not too farfetched. Even if all the scheduled reading intervals are evenly spread between 10 and 99 seconds, every once in a while (the least common multiple of the intervals) they will all again coincide and the queue will fill with 64 reports as described above. If the last report in the queue has a reading interval smaller than 18.560 seconds it will miss deadlines as often as the queue builds up beyond some limit.

The calculation of Jitter (Equation 3-5) is straightforward:

$Jitter_{HL} = PL_{HL} - BCPL = 18{,}560 - 290 = 18{,}270$ ms

$Jitter_{ML} = PL_{ML} - BCPL = 9{,}280 - 290 = 8{,}990$ ms

The performance results are summarized in Table 4-1.

WCCL	PL_{HL}	PL_{ML}	BCPL	$Jitter_{HL}$	$Jitter_{ML}$
26.880 seconds	18.560 seconds	9.280 seconds	0.290 seconds	PL_{HL} - BCPL= 18.270 seconds	PL_{ML} - BCPL = 8.990 seconds

Table 4-1: Performance Summary

4.2 Availability Analysis

The availability model required the specification of S, λ and μ, the number of servers, and server failure and repair rates respectively. Let's assume the same value of S (2) used in the performance scenarios. The fraction of downtime (Equation 3-6) becomes the following:

[Equation 4-1]

$$D_S = \frac{S!\left(\frac{\lambda}{\mu}\right)^s}{1 + \sum_{n=1}^{S}\left(\frac{\lambda}{\mu}\right)^n \times \frac{S!}{(S-n)!}} = \frac{2 \times \left(\frac{\lambda}{\mu}\right)^2}{1 + 2 \times \left(\frac{\lambda}{\mu}\right) + 2 \times \left(\frac{\lambda}{\mu}\right)^2} = \frac{2 \cdot \lambda^2}{\mu^2 + 2 \cdot \lambda \cdot \mu + 2 \cdot \lambda^2}$$

The failure and repair rates could be "discovered," if we had experience with previous systems, and simply taken from a product specification sheet. It is more likely that we have to assume failure and repair rates in some scenario; these rates would have to be tested or confirmed during development. For the purposes of illustration, assume that there are two types of faults that can bring the server down: hardware and software. For each of these fault scenarios we assume a rate of failure and a rate of repair.

Hardware failure scenario — The rate of a hardware failures, such as a burned-out power supply, range from 1 to 2 failures/year and require a visit by a technician to repair (1/2 day to repair). That is, λ = 1~2 failures/year and μ = 2 repairs/day (730 repairs/year).

Software failure scenario — The rate of software failures, such as an operating system crash, range from 8 to 24 failures/year and require restarting the server to repair (10 minutes to reboot). That is, λ = 8~24 failures/year and μ = 144 repairs/day (52,560 repairs/year).

Solving Equation 4-1 for the higher values of hardware (HF) and software (SF) fault scenarios yields

$$D_{HF} = (2\lambda^2)/(\mu^2 + 2\lambda\mu + 2\lambda^2) = (2*2^2)/(730^2 + 2*2*730 + 2*2^2) = 1.4930164157E\text{-}5$$

$$D_{SF} = (2\lambda^2)/(\mu^2 + 2\lambda\mu + 2\lambda^2) = (2*24^2)/(52{,}560^2 + 2*24*52560 + 2*24^2) = 4.1662483059E\text{-}7$$

We can not simply add the downtimes as if these failures scenarios were independent. Software failures can only occur when the hardware is operating. Thus, the combined fraction of downtime is:

$$D = D_{HF} + (1\text{-}D_{HF})*D_{SF} = 1.5346782767E\text{-}5$$

Converting these fractions of a year into hours, the availability results are summarized in Table 4-2.

Hardware Failures (1/2 day repair)			Software Failures (10 minute repair)			Combined Failures	
Failures per year	**Availability ($1\text{-}D_{HF}$)**	**Hours down per year**	**Failures per year**	**Availability ($1\text{-}D_{SF}$)**	**Hours down per year**	**Availability**	**Hours down per year**
2	0.99998507	0.1307	24	0.99999958	0.0036	0.99998465	0.1344

Table 4-2: Availability Summary

4.3 Security Analysis

We will consider two modes of operation for an attacker to distort the information sent by the server. In a man-in-the-middle (MIM) attack, the intruder gets between the server and its clients and modifies temperature reports sent to the clients. In a spoof-the-server attack, the intruder spoofs or takes over the role of a server and fakes temperature reports sent to its clients.

4.3.1 Man-in-the-Middle Attack Scenario

For the man-in-the-middle attack, Figure 4-1 shows the attack as it would take place on the client side of a network. In a MIM attack, the attacker uses a TCP intercept tool to modify the values of the temperatures during transmission. The window of opportunity (W) for an intruder is the length of time the intruder can operate undetected. Within the window, there is an attack rate (R) that the intruder is capable of. For an attempted intrusion to be successful, the intruder must then successfully apply the attack scenario with some probability of success $P_{TCP.}$ There are no specific defenses beyond the difficulty of matching the correct intrusion tool with the client/server versions of the hardware and software. The only barrier is the length of the window of opportunity and the probability of success for the TCP intercept tool. The expected number of successful MIM attacks within the window is derived from Equation 3-8:

$$E_{MIM} = W * R * P_{TCP} \qquad \textbf{[Equation 4-2]}$$

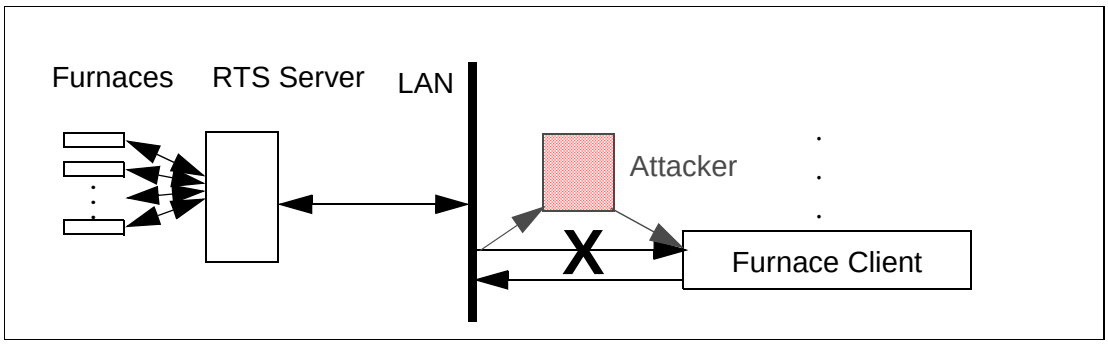

Figure 4-1: Man-in-the-Middle Attack

4.3.2 Spoof-the-Server Attack Scenario

For the spoof-the-server attack, Figure 4-2 shows the attack as it would take place on the server side of the network. To successfully spoof the server, there are three possible ways to succeed. The intruder could wait for the server to fail, then spoof the server's address and take over the client connections. This presumes that the intruder can determine when the server has failed and can take advantage of this before the clients time-out. Another successful method would be to cause the server to fail (the "kill server" attack), then take over the connections. A third is to disrupt the connections between the client and server, then establish new connections as the spoofed server (the "kill connection" attack). For this analysis, it is presumed that the intruder is equally likely to attempt any of these methods in a given attack.

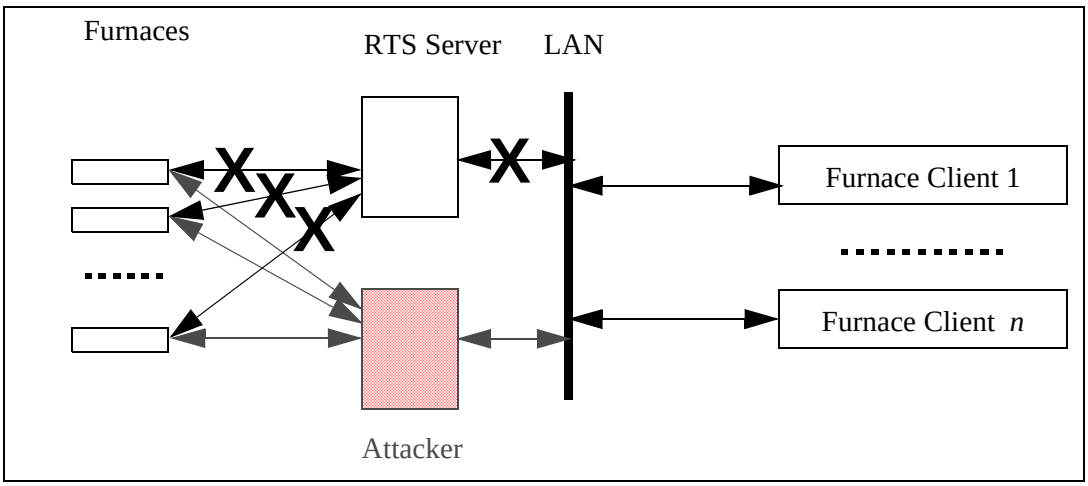

Figure 4-2: Spoof-the-Server Attack

As in the MIM attack, the window of opportunity (W) for an intruder is the length of time the intruder can operate undetected. Within the window, there is an attack rate (R) that the intruder is capable of. For an attempted intrusion to be successful, the intruder must then successfully apply the attack scenario with some probability of success P_{attack} and assume the identity of the server with some probability of success $P_{Spoof\text{-}IP}$. The expected number of successful spoof attacks within the window is derived from Equation 3-8:

$$E_{Spoof\text{-}failure} = W * R * P_{Server\text{-}failure} * P_{Spoof\text{-}IP} \quad \textbf{[Equation 4-3]}$$

$$E_{Spoof\text{-}Kill} = W * R * P_{Kill\text{-}Serverl} * P_{Spoof\text{-}IP} \quad \textbf{[Equation 4-4]}$$

$$E_{Spoof\text{-}Kill\text{-}connection} = W * R * P_{Kill\text{-}connection} * P_{Spoof\text{-}IP} \quad \textbf{[Equation 4-5]}$$

For a given server failure rate λ, the probability of a server failure within the window W is

$$P_{Server\text{-}failure} = 1 - e^{-\lambda SW} \quad \textbf{[Equation 4-6]}$$

4.3.3 Parameters for Security Scenarios

To evaluate the probability of successful attacks under either scenario we have to provide values for the various probabilities of success, windows, and rates of attack. Since these are only estimates based on experience of the security analyst's experience in the environment, they may be adjusted based on additional information or constraints on the environment. Additional training for the operators may, for example, reduce the exposure time for an intruder. Alternatively, the probability for a successful attack may increase as more information about the con-

figuration of the system is gained by the intruder. However, the estimates provided here are sufficient to illustrate the analysis method:

- attack exposure window, W = 60 minutes
- attack rate, R = 0.05 systems/minute (20 minutes per attack)
- server failure rate, λ = 10 failures/year
- number of servers, S = 2
- probability of success of a TCP intercept (MIM), $P_{TCP} = 0.5$
- probability of success of a spoof IP address (spoof), $P_{Spoof\text{-}IP} = 0.9$
- probability of success of a kill connection (spoof), $P_{Kill\text{-}Connection} = 0.75$
- probability of success of kill server (spoof), $P_{Kill\text{-}Server} = 0.25$

The expected number of successful attacks (Equation 4-2 through Equation 4-6) is

$$\mathbf{E_{MIM} = W * R * P_{TCP}} = 60 * 0.05 * 0.5 = 1.5$$

$$\mathbf{P_{Server\text{-}failure} = 1 - e^{-\lambda SW}} = 1 - e^{-(10 * 2 * 1/(24*365))} = 0.0023$$

$$\mathbf{E_{Spoof\text{-}Server\text{-}failure} = W * R * P_{Server\text{-}failure} * P_{Spoof\text{-}IP}} = 60 * 0.05 * 0.0023 * 0.9 = 0.0061$$

$$\mathbf{E_{Spoof\text{-}Kill\text{-}server} = W * R * P_{Kill\text{-}server} * P_{Spoof\text{-}IP}} = 60 * 0.05 * 0.25 * 0.9 = 0.66$$

$$\mathbf{E_{Spoof\text{-}Kill\text{-}connection} = W * R * P_{Kill\text{-}connection} * P_{Spoof\text{-}IP}} = 60 * 0.05 * 0.75 * 0.9 = 2.04$$

The security results are summarized in Table 4-3.

Man-in-the-Middle (MIM) Attack	Spoof-the-Server Attacks		
E_{MIM}	$E_{Spoof\text{-}failure}$	$E_{Spoof\text{-}Kill\text{-}server}$	$E_{Spoof\text{-}Kill\text{-}connection}$
1.5	0.0061	0.66	2.04

Table 4-3: Security Summary

5 Model Sensitivity and Tradeoffs

5.1 Comparison Against Requirements

The performance characteristics are summarized in Table 4-1. Since we did not specify explicit requirements for latency and jitter, we can not make any strong statements about met and unmet requirements. However, we know the context of the problem and we can still apply a “reasonableness” criteria in evaluating the performance of the system.

A worst case control latency (WCCL) response time of 26.88 seconds is questionable. Considering that this is almost three times the minimal reading interval (10 seconds), a client scheduling a furnace with short reading intervals will receive the first reading (the response to the control request) much later than it would have received the reading had it been a periodic update. Further, remember that the WCCL equation assumed that there was no other activity going on. If the server was also busy generating periodic reports, the WCCL would have been worse.

The latency for periodic reports (PL) is borderline. Under the heavy load scenario the latency (18.56 seconds) can be almost twice the minimal reading period (10 seconds) and under a moderate load scenario, the latency is barely (9.28 seconds) under the threshold. This should alert the participants that the periodic latency, specifically the heavy load periodic latency, must be an explicit requirement.

In the case of jitter, we must ask, “What is the cost of a missed update? Is it ever acceptable to violate this requirement?” In some safety-critical applications the answer would be “no.” In most applications, the answer could be “yes,” providing that this occurrence was infrequent. As in the worst case periodic latency, the high value of the jitter (18.27 seconds) compared to the shortest possible period indicates that the jitter must be an explicit requirement.

The vagueness of the requirement for minimal downtime has to be resolved. Depending on the domain in which the RTS is used, the downtime (0.13 hours/year) could be adequate or intolerable. If the stakeholders agree that this availability is adequate they could declare victory by codifying it into the requirements as an acceptable downtime before preceding.

The model only captures the server availability, and any improvements in availability coming from this model would look for improvements in the server. A more useful and realistic model would include the client and the network availability. Since all three components operate “in series,” the model would only be marginally more difficult to analyze.

The results of the security analysis show that within the 60-minute exposure window an attack is likely to succeed. For the man-in-the-middle scenario, the expected number of successful attacks is 1.5, indicating that an intruder would have more than enough time to complete the attack before detection. For the spoof attack, the number of successful attacks ranges from 0.006 to just over 2, again showing that a penetration using this technique is also likely.

There are two possible reactions to these results. First, the security requirements must be made explicit to decide if the results of this analysis are really good enough for the target environment. Alternatively, security features could be added to the components and the architecture that would make these attacks more difficult.

5.2 Sensitivity and Tradeoffs

For each of the attribute models we must identify those parameters that have a major effect on the results for that model. A sensitive parameter is one that has a great impact on the model. (Variations in the parameter correlate strongly with variations in the modeled or measured value.) Parameters that are common to more than one attribute model influence multiple attributes and can be used to tradeoff between attributes.

Sensitive parameters found in only one set may not have been considered by the other model experts, or may not be relevant or sensitive to that model. Sensitive parameters that affect more than one attribute can be positively correlated (i.e., a change in one direction has positive effects on all attributes (win-win)), or negatively correlated (i.e., an improvement in one attribute may result in negative impact on another attribute (win-loss)).

Without repeating the analyses, let’s examine the effect of the number of servers on the attribute models. Table 5-1 shows the effect of varying the number of servers from 1 to 3 (with all other parameters remaining the same).

	1 Server	2 Servers	3 Servers
WCCL	53.760 seconds	26.880 seconds	17.920 seconds
BCPL	0.290 seconds		
PL_{HL} (Q = C/S*F)	37.120 seconds	18.560 seconds	12.374 seconds
PL_{ML} (Q = (C/S)*(F/2))	18.560 seconds	9.280 seconds	6.187 seconds
Combined downtime hardware rates: $\lambda = 2$ per year $\mu = 730$ per year software rates: $\lambda = 24$ per year $\mu = 52560$/year	27.922 hours/year	0.1344 hours/year	0.0010 hours/year
$E_{Spoof\text{-}failure}$ $\lambda = 10$ per year	0.0030 successful attacks/hour	0.0061 successful attacks/hour	0.0092 successful attacks/hour

Table 5-1: Sensitivity to the Number of Servers and Server Failure Rates

The table shows that increasing the number of servers (S) decreases latencies at approximately a linear rate (actually, an exponential decay with a limiting value reached when there is one client per server and there is never a queueing delay); has a very large impact on availability (about 100 times less downtime with each additional server); and also increases the probability of a successful attack at approximately a linear rate (increasing servers provides more opportunities for a sever failure, needed for one of the spoof attack modes.) This is a case where the same parameter improves two attributes and lowers another attribute.

The server failure rate provides another case of an interface between attributes. The same parameter appears in the availability and security analysis and in both cases an increase in the failure rate lowers both attributes. A higher hardware failure rate increases downtime and the expected number of successful spoof attacks. A higher software failure rate increases downtime but it is unlikely to affect security because the repair time is short (the spoofing intruder would be caught when the real server restarts).

Notice however that the values assumed for λ are very different in the availability and security scenarios. If these are hardware failures, the security domain expert is more pessimistic than the availability domain expert; if these are software failures the security domain expert is more optimistic than the availability domain expert. These differences have to be reconciled by the stakeholders.

Sometimes the interdependence is not obvious: The number of clients affects performance (increasing load) and security (increasing points for MIM attack), although the number of clients is not explicit in the security model.

Using richer models would help uncover additional attribute interdependencies. For example, although the availability analysis presented in this report was only analyzing the availability of the servers, in a complete analysis we would look at potential failures of the clients and the networks, and we would look at various failure types. One type of failure is dropping a message. If we assume that the communication channel is not reliable, then we must plan for re-sending messages. To do this involves additional computation (to detect and re-send lost messages), storage (to store the messages until they have been successfully transmitted), and time (for a time-out interval and for message re-transmission). Thus one of the major implications of this enhanced availability model is that the performance model needs to be modified.

Similarly, adding new requirements will have implications on the models. For example, a security requirement for encryption of messages between the server and client might contribute a significant performance overhead.

The dependencies between attributes can work in both directions. For example, improved performance of the system, in the form of a more tightly bounded jitter, will have impacts for security. This is because many attempts at breaching the system's security will add a delay to message transmission time. If the jitter is small, it will be easy to detect intruders because they will cause message delays that will exceed the maximum jitter bound. Another way in which reducing jitter aids security is that it eliminates the possibility of a covert information channel.

6 Conclusions

Our motivation for developing an ATA method is the desire to make rational choices among competing architectures, based upon well-documented, well-reasoned analyses of system attributes at the architectural level. An important aspect of this method is that it serves as a vehicle for the early clarification of requirements. It does this by analyzing a system from many perspectives simultaneously, then comparing the assumptions made in those analyses, repairing the models whenever those assumptions do not coincide.

For every assumption that we make in a system design, we trade cost for knowledge. For example, how should we define jitter? If a periodic update is supposed to arrive every ten seconds, do we want it to arrive exactly every ten seconds, on average every ten seconds, or some time within each ten second window? To give another example, consider the requirement detailing the worst case latency of periodic reports. Is the worst case ever acceptable? If so, how frequently will we tolerate it?

The process of analyzing architectural attributes forces us to try to answer these questions. Either we understand our requirements precisely or we pay (by over-engineering or under-engineering the system). If we over-engineer, we pay by making the system needlessly expensive. If we under-engineer, we face system failures, losing customers or perhaps even lives.

Every analysis step that we take precipitates new questions. While this seems like a daunting, never-ending prospect, it is manageable because these questions are posed and answered within an *analytic* attribute framework. Within this framework, we stop iterating when we are comfortable with the results and can accept the risks of any residual error.

In addition to solidifying requirements, the ATA method helps to uncover implicit requirements. This occurs because attribute analyses are *inter*dependent. We discover the interfaces between attributes by examining the assumptions that we make for analysis A while performing analysis B. Not allowing for retransmission in the performance analysis is one example of such an interface based upon an implicit, *hidden* assumption (“no dropped packets”). This assumption, if false, may have implications for security or availability. A solution to the problem of dropping packets will have implications for performance.

The focus on attribute interfaces forces attribute experts and other stakeholders to communicate through a shared “blackboard.” Attribute experts independently create models, then exchange information (e.g., clarifying or creating new requirements), and on the basis of this

information, refine the models. The interaction of attribute-specific analyses has a greater effect on system understanding and stakeholder communication than any of these analyses could do on their own. As a result of performing the activities illustrated in this report, the stakeholders have an enhanced understanding of, and confidence in, the system's ability to meet its requirements. They also have a documented rationale for the architectural choices made. This documentation consists of attribute models, the scenarios used in the analyses, and the results of those analyses.

In addition to this information, we are studying the concept of attribute-based architectural styles (ABAS). An ABAS is a specific kind of architecture in which the components and their interactions, the attribute models and scenarios, and the attribute tradeoffs and interfaces have all been identified and analyzed. Collections of ABAS would make the method more efficient because they tell the architect and the other stakeholders what information is needed, what results will be available, and what tradeoffs are available. The analysis of architectures built entirely or mostly from ABAS would not have to be carried out ab initio.

References

Audsley 95 Audsley, N. C. et al. “Fixed Priority Pre-Emptive Scheduling: An Historical Perspective.” *Real-Time Systems 8*, 2-3 (March-May 1995): 173-198.

Bass 98 Bass, L.; Clements, P.; & Kazman, R. *Software Architecture in Practice.* Reading, MA: Addisson-Wesley Publishing Company, 1998.

Barbacci 95 Barbacci, M.; Klein, M.; Longstaff, T.; & Weinstock, C. *Quality Attributes* (CMU/SEI-95-TR-21 ADA307888). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1995.

Barbacci 96 Barbacci, M.; Klein, M.; Weinstock, C. *Principles for Evaluating the Quality Attributes of a Software Architecture* (CMU/SEI-96-TR-36 ADA324233). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1996.

Conway 67 Conway, R.; Maxwell, W.; & Miller, L. *Theory of Scheduling*. Reading, MA: Addison-Wesley Publishing Company, 1967.

Kazman 96 Kazman, R.; Abowd, G.; Bass, L.; & Clements, P. “Scenario-Based Analysis of Software Architecture.” *IEEE Software 13,* 6 (November 1996): 47-55.

Klein 93 Klein, M.; Ralya, T.; Pollak, B.; Obenza, R.; & Gonzales Harbour, M. *A Practitioner’s Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems*. Boston, MA: Kluwer Academic Publishers, 1993.

Lehoczky 94 Lehoczky, J.P. “Real-Time Resource Management Techniques,” 1011-1020. *Encyclopedia of Software Engineerin*g, Volume 2, Marciniak, J.J (ed.). New York, NY: J. Wiley, 1994.

McCall 94 McCall, J. “Quality Factors,” 958-969. *Encyclopedia of Software Engineering*, Volume 2, Marciniak, J.J (ed.). New York, NY: J. Wiley, 1994.

Nielsen 87 Nielsen, K. & Shumate, K. “Designing Large Real-Time Systems with Ada.” *Communications of the ACM 30*, 8 (August 1987): 695-715.

Ross 89 Ross, S. M. *Introduction to Probability Models*, 4th Edition. Boston, MA: Academic Press, 1989.

Sanden 89 Sanden, B. “Entity-Life Modeling and Structured Analysis in Real-Time Software Design—A Comparison.” *Communications of the ACM 32*,12 (December 1989):1458-1466.

Smith 90 Smith, C. U. *Performance Engineering of Software Systems*. Reading, MA: Addison-Wesley, 1990.

Smith 93 Smith, C. & Williams, L. “Software Performance Engineering: A Case Study Including Performance Comparison with Design Alternatives.” *IEEE Transactions on Software Engineering 19*, 7 (July 1993): 720-741.

Stankovic 95 Stankovic, J. A., et al. “Implications of Classical Scheduling Results for Real-Time Systems.” *IEEE Computer 28*, 6 (June 1995): 16-25.

Trivedi 82 Trivedi, K. S. *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*. Englewood Cliffs, N.J.: Prentice-Hall, 1982.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. **AGENCY USE ONLY** (leave blank)	2. **REPORT DATE** May1998	3. **REPORT TYPE AND DATES COVERED** Final
4. **TITLE AND SUBTITLE** Steps in an Architecture Tradeoff Analysis Method: Quality Attribute Models and Analysis		5. **FUNDING NUMBERS** C — F19628-95-C-0003
6. **AUTHOR(S)** Barbacci, M.; Carriere, S.; Feiler, P.; Kazman, R.; Klein, M.; Lipson, H.; Longstaff, T.; Weinstock, C.		
7. **PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)** Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213		8. **PERFORMING ORGANIZATION REPORT NUMBER** CMU/SEI-97-TR-029
9. **SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)** HQ ESC/DIB 5 Eglin Street Hanscom AFB, MA 01731-2116		10. **SPONSORING/MONITORING AGENCY REPORT NUMBER** ESC-TR-97-029
11. **SUPPLEMENTARY NOTES**		
12.a **DISTRIBUTION/AVAILABILITY STATEMENT** Unclassified/Unlimited, DTIC, NTIS		12.b **DISTRIBUTION CODE**

13. **ABSTRACT** (maximum 200 words)

This paper presents some of the steps in an emerging architecture tradeoff analysis method (ATAM). The objective of the method is to provide a principled way to understand a software architecture's fitness with respect to multiple competing quality attributes: modifiability, security, performance, availability, and so forth. These attributes can interact or conflict—improving one often comes at the price of worsening one or more of the others, thus it is necessary to trade off among multiple software quality attributes at the time the software architecture of a system is specified, and before the system is developed. This report illustrates typical quality attribute models, analyses, and tradeoffs using a small real-time industrial application.

14. **SUBJECT TERMS** architecture business cycle, architecture tradeoffs, attribute models, availability, performance, quality attributes, security, software architecture			15. **NUMBER OF PAGES** 32 16. **PRICE CODE**
17. **SECURITY CLASSIFICATION OF REPORT** UNCLASSIFIED	18. **SECURITY CLASSIFICATION OF THIS PAGE** UNCLASSIFIED	19. **SECURITY CLASSIFICATION OF ABSTRACT** UNCLASSIFIED	20. **LIMITATION OF ABSTRACT** UL