

Éric Dorion

On the Performance of Military Distributed Information Systems

Mémoire
présenté
à la Faculté des études supérieures
de l'Université Laval
pour l'obtention
du grade de maître ès sciences (M.Sc.)

Département de génie électrique
FACULTÉ DES SCIENCES ET DE GÉNIE
UNIVERSITÉ LAVAL

Avril 2003

Résumé

Ce mémoire réfléchit au problème de la garantie de performance des applications militaires réparties. Du problème général, quatre sous-problèmes seront énoncés et expliqués. Autant d'avenues de solutions seront proposées pour les adresser. Les sous-problèmes sont: les mesures de mérite pour le commandement et contrôle militaire, la définition de la qualité de service ainsi que des concepts qui s'y rattachent, l'architecture logicielle CORBA en support à la performance, et une méthodologie de la modélisation de la performance basée sur UML pour encadrer les efforts.

Summary

This thesis reflects on the problem of ensuring performance in military distributed applications. From the general problem, four sub-problems will be identified and explained. As many solutions to address these will be proposed. The sub-problems are: Military command and control measures of merit for system instrumentation, quality of service and related concepts for guaranteeing performance, CORBA architecture in support to performance and a UML-based methodology to encompass performance modelling efforts.

Éric Dorion
Étudiant/
Student

Dr. Denis Laurendeau
Directeur de recherche/
Research Director

Pour toi grand-maman...

Contents

Contents	iii
List of Figures	vi
List of Tables	vii
1 General Overview	1
1.1 Introduction	1
1.2 Performance of Military Information Systems	2
1.2.1 Command and Control	2
1.2.2 OODA Loop	3
1.3 Scoping the Problem	4
1.3.1 The Performance Vocabulary	5
1.3.2 The "Measuring Performance" Attitude	5
1.3.3 The "Hidden Behaviour" Impairment	5
1.3.4 The Lack of a Performance Methodology	5
1.4 Organization of the Thesis	6
2 Command and Control Measures of Merit	7
2.1 Background	7
2.2 Influences on the Choice of Measures	8
2.2.1 Scenario	8
2.2.2 Observer's Objectives	8
2.3 Definition of Measures	10
2.4 Characteristics and Properties of Measures	11
2.5 Types of Measures	12
2.6 Examples of Hierarchical Measures	13
2.7 Determining a Set of Metrics	15
2.7.1 Choosing the Metrics	15
2.7.2 Linking Metrics Between Them	15
2.7.3 Determining Variation Ranges	15
3 Quality of Service	18
3.1 Introduction	18
3.2 Definition of Quality of Service	18

3.2.1	The Notion of Predictability and Consistency	18
3.2.2	The Notion of Guarantee	19
3.2.3	The Notion of Management	19
3.2.4	The Notion of End-to-end	20
3.2.5	So, What is QoS Anyway?	20
3.3	QoS Concepts	21
3.3.1	QoS Requirements, QoS Parameters and QoS Context	21
3.3.2	QoS Characteristics	22
3.3.3	QoS Management Functions	22
3.4	QoS Architectures	24
3.4.1	Integrated Services Architecture (IntServ)	25
3.4.1.1	Flow Resource Reservation and Admission Control	26
3.4.1.2	Packet Classifier	27
3.4.1.3	Packet Scheduler	27
3.4.2	Differentiated Services (DiffServ)	27
3.4.3	Observations on IntServ/RSVP and DiffServ	28
4	Quality of Service and CORBA	30
4.1	Introduction	30
4.2	CORBA Messaging	30
4.2.1	Policy Management	30
4.2.2	Messaging PolicyType Objects	32
4.2.3	Note on CORBA Messaging	35
4.3	Real-Time CORBA	35
4.3.1	RTCORBA Extensions	36
4.3.2	Priority Mechanisms	37
4.3.3	Priority Models	38
4.3.4	Threadpools	40
4.3.5	Synchronization Mechanisms	40
4.3.6	Protocol Selection and Configuration	42
4.3.7	Explicit Binding	43
4.3.8	Scheduling Service	44
4.3.9	Observations on Real-Time CORBA	44
5	A Methodology to Manage Performance in C2ISs	46
5.1	Introduction	46
5.2	The Methodology	47
5.2.1	Determining the System's Operational Environment	47
5.2.2	QoS Static Analysis	47
5.2.2.1	Use Case Analysis	49
5.2.2.2	Identifying the High-level MoE	49
5.2.2.3	Attaching Specific MoEs to Use Cases	50
5.2.2.4	Defining QoS Objects	50
5.2.3	QoS Dynamic Analysis	51
5.2.3.1	Identifying Relevant Scenario Walk-throughs	51

5.2.3.2	Instrumenting the System with MoPs	54
5.2.4	QoS Modelling	54
5.2.4.1	Impact on the Business Process	54
5.2.4.2	Impact on the Use Cases	55
5.2.4.3	Impact on Object Interactions	55
5.2.5	QoS Implementation and Deployment	58
5.2.6	Observations on a Performance Methodology	58
6	Conclusions	61
6.1	Performance of Military Information Systems	61
6.2	Scoping the Problem	61
6.3	Future Work	62
6.3.1	Performance Patterns	62
6.3.2	Formal Constraints Representation	62
6.3.3	Variable Weighting of MoMs	63
6.3.4	Closed-loop QoS Specification	63
6.4	Towards Adaptive C2ISs	63
	Bibliography	64
A	Acronyms, Initialisms and Abbreviations	67

List of Figures

1.1	The OODA Loop	3
2.1	MORS MoMs Hierarchy	10
2.2	Hierarchical Relationship of Measures	11
3.1	Mapping the OSI 7-layer Model to Quality Models	21
3.2	Relationships Between QoS Concepts	21
3.3	ISO/IEC 13236 QoS Characteristics Hierarchy	23
3.4	Derived Characteristics and Specializations	24
3.5	Packet vs Message Routing	25
3.6	IntServ Reference Model	26
3.7	DiffServ Reference Model	28
3.8	Combining the IntServ and DiffServ Architectures	29
4.1	Real-Time CORBA Extensions	36
4.2	RTCORBA Priority Mapping	38
4.3	Client-Propagated Priority Model	39
4.4	Server-Declared Priority Model	39
4.5	Threadpools Use Cases	41
4.6	Mutexes	42
4.7	PriorityBand Connections	43
5.1	The Decision-Making Process	48
5.2	Identifying High-level and Specific MoEs of the System	49
5.3	Nominal Mode Scenario	52
5.4	Network Degraded Mode Scenario	53
5.5	Server Down Mode Scenario	53
5.6	Applying Design Patterns	55
5.7	IntServ QoS Modelling	57
5.8	CORBA QoS Modelling	59
5.9	Component Model View	60
5.10	Deployment Model View	60

List of Tables

2.1	Effects of Observer’s Objectives on Measurement Needs	9
2.2	Reliability Criteria of Measures	12
2.3	Validity Criteria of Measures	12
2.4	Measures of Force Effectiveness	13
2.5	Time-based Measures of Effectiveness	13
2.6	Accuracy-based Measures of Effectiveness	14
2.7	C2MOE Handbook MoEs	14
2.8	Hardware and Software Related MoPs	14
2.9	Applications Related MoPs	15
2.10	User Effectiveness Related MoPs	16
2.11	Dimensional Parameters	17
4.1	Standard Messaging Policy Types	32
4.2	Standard RTCORBA Policy Types	36
4.3	RTCORBA Standard System Exceptions	37

Chapter 1

General Overview

1.1 Introduction

Performance and efficiency. Two qualities that we, humans, always strive to achieve, whether at work, at play, or at war. *"At war? What a trivial example!"*, one would say. Well then, why is it that most military information systems and especially Command and Control Information Systems (C2ISs) are built without considering performance and efficiency as part of their design process? Why is it that these important qualities are only sought for afterwards? Why are performance solutions and performance tuning techniques still applied in an ad hoc fashion? Shouldn't information systems holding high-value information and on which lives depend be better designed in terms of performance? Isn't it contradictory to claim a system to be efficient while little is known about the context in which the system evolves or about its underlying communication constraints? C2ISs are intrinsically distributed systems that move information around, offer services to others or require them. They are highly heterogeneous systems as they are implemented on all kinds of hardware platforms and in all kinds of software packages. Some of them may be collocated on the same Local-Area Network (LAN) or may be several hundreds of kilometers apart. The communication mechanisms they use may be impaired by low-bandwidth constraints like in radio-linked networks or may be enjoying the freedom of high-speed gigabyte networks. Users of these systems may be very demanding, may resist the proposed Human-Machine Interfaces (HMIs) or may be very comfortable with them. These are all factors that influence performance of military information systems. Yes, this thesis is a reflection about performance and although it is impossible to address the whole spectrum of performance, a discussion around certain key aspects of performance engineering will be beneficial. This will result in a better understanding of what the concept of performance is, what it involves, and how it should be addressed.

The military realm sets the background of this document. While the problem of performance clearly concerns the whole spectrum of IT industry, this domain is chosen for practical reasons and because it offers its own set of challenges and requirements. Work could be done afterwards to address other domains.

1.2 Performance of Military Information Systems

While a sense of what "performance" is is easy to gain, formulating it formally is harder. This is because performance is not such a simple concept. Performance depends on a lot of factors and it is important to define these concepts first. In the military domain, one must understand what process is supported by information technology. This process, the decision-making process, is known as "Command and Control".

1.2.1 Command and Control

The military realm offers its own set of challenges and requirements on information systems. C2ISs must support the commander in his decision-making process. Before showing how these systems support the process, it is important to understand first what command and control (C2) is. The whole military domain revolves around this concept. The few next C2 definitions from the military literature, far from being complete or exhaustive, constitute a good starting point.

"The C2 process is seen as an instantiation or an example of a dynamic human decision making process that establishes the common intent and transforms that common intent into a co-ordinated action." [1]

Command and Control as defined by the US Joint Chiefs of Staff Publication 1 (JCS Pub 1) and cited in [2]:

"The exercise of authority and direction by a properly designated commander over assigned forces in the accomplishment of his mission. Command and control functions are performed through an arrangement of personnel, equipment, communications, facilities, and procedures which are employed by a commander in planning, directing, coordinating, and controlling forces and operations in the accomplishment of his mission."

Also from the draft Field Manual 71-100-1, Heavy Division Tactics and Techniques, Heavy Division Command and Control and cited in [2]:

"Command and Control is not one word! Command is the art of assigning missions, prioritizing resources, guiding and directing subordinates, and focusing the entire division's energy to accomplish clear objectives. Control is the science of defining limits, computing requirements, allocating resources, prescribing requirements for reports, monitoring performance, identifying and correcting deviations from guidance, and directing subordinate actions to accomplish the commander's intent."

The next definition leads us to the very important concept of situation awareness.

"C2 is the process by which commanders can plan, direct and monitor any operation for which they are responsible. C2 requires that the commander is aware of the tactical situation in order to make a timely decision about the best course of action to be implemented." [1]

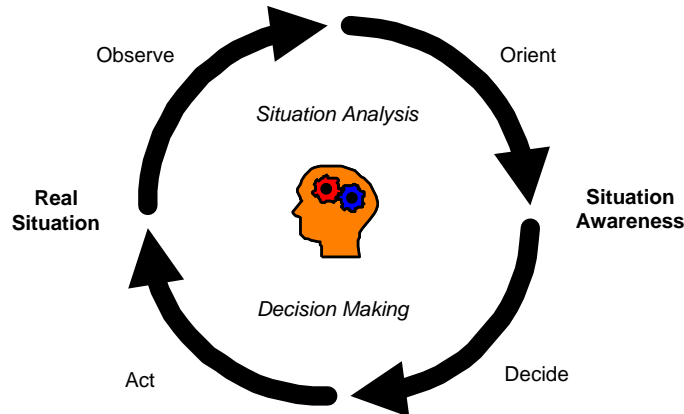


Figure 1.1: The OODA Loop

Thus, efficient C2 requires that the commander has good situation awareness which is defined as:

"... the perception of the elements in the environment, within a volume of time and space, the comprehension of their meaning, and the projection of their status in the near future." [3]

The concept that one must understand here is that an efficient C2IS is one that will provide the commander with good situation awareness. Digging a little bit further into the decision-making process is necessary to understand how C2ISs can support C2 efficiently.

1.2.2 OODA Loop

The Observe-Orient-Decide-Act (OODA) loop represents the decision-making process of a commander in a C2 situation and is illustrated by Figure 1.1. The top half of the loop constitutes the situation analysis part and leads to situation awareness. It is composed of two phases:

- Observe:** Obviously, for the commander to gain situation awareness, he will have to have the current situation presented to him. This is necessary as it will enable him to use resources at his disposal effectively to eventually gain advantage over the opponent forces. In this phase, an efficient C2IS will be one that gives a representation that is perfectly correlated with ground truth. Of course, this is rarely the case as information is often incorrect or out-dated. However, such information is not necessarily useless and may still prove to be of value to the commander.
- Orient:** The commander has to look beyond the simple representation of the current situation and ask himself if there are patterns from which he may infer knowledge. A merchant ship may constitute a meaningless piece of information, unless political, sociological or even historical issues lead the commander to think that this ship might smuggle drugs or refugees. His understanding of the current situation will only be

complete if he reflects on what he sees and what he does not see. As some patterns are well defined and recognized (computerizable) and as an overloaded picture may overwhelm the commander, a C2IS may prove its efficiency by easing the commander's task in this phase.

The lower half of the loop is the one in which the commander will actively assign and deploy resources to influence the likely future of what he perceives as being the real situation. Again, it is composed of two phases:

- **Decide:** This is the part in which the commander will decide to act upon a course of action (COA). The C2IS helps him by suggesting a set of COAs depending on available resources and other parameters. The performance and efficiency of a C2IS at this stage is related to the accuracy of the suggested COAs.
- **Act:** In this final part of the loop, the commander issues orders, monitors the progress of ensuing activities according to the plan and corrects minor deficiencies. He even steps forward in the loop and begins a new OODA cycle. The C2IS is the best tool for disseminating timely information, might it be synchronous or asynchronous.

C2ISs can support the decision-making process at every step of the OODA loop. At a macroscopic level, C2ISs that support the ease and speed of execution of the OODA loop for as many cycles as necessary are considered to be efficient. [1]

"The military community typically states that the dominant requirement to counter the threat and ensure the survivability of the ship is the ability to perform the C2 activities (i.e., the OODA loop) quicker and better than the adversary. Therefore, the speed of execution of the OODA loop and the degree of efficiency of its execution are the keys to success for shipboard tactical operations" [1]

This is the ultimate objective of C2IS performance engineering. It also constitutes the link between the system and the user. In wanting to perform his C2 duties, the commander, whether he is aware of it or not, imposes performance requirements on the C2IS. In other words, a set of performance constraints are imposed on the underlying information system. The challenge is to express these C2 requirements in terms of digestible parameters for the C2IS.

1.3 Scoping the Problem

The general problem of C2IS performance will be divided in four sub-problems. These sub-problems are considered to be fundamental to the general problem as they need to be addressed if any progress is to be made in improving the performance of military information systems and more particularly C2ISs.

1.3.1 The Performance Vocabulary

The first fundamental problem is the lack of a common understanding of the performance problem between the systems designer and the end user. System designers have their own way of viewing and addressing performance problems. However, only the user matters in the end. If his perception of performance is not met, all improvements a systems designer applied are of no relevance. As expressed with the OODA loop, performance is a matter of making the wheel spin faster. Since it is a mental process and the utility of a C2IS lies in the support of this process, the vocabulary of performance has to conform with that of the user. However, at the systems level, performance is expressed in another vocabulary. Therefore, a vocabulary of performance that will address both levels and link them in a certain way is needed.

1.3.2 The "Measuring Performance" Attitude

The second fundamental problem seems to come from a general perception that the performance problem lies in its measurement. It does not. Well, not completely. Performance measurement is only one step. This conception has to evolve in the form of "ensuring" performance. Also, a definition under which one can structure activities and tools in order to achieve performance by design and not by happenstance has to be stated. In other words, there is a need for building blocks to ensure performance on the part of objects in the client-side application, the middleware, the network, and the server-side application.

1.3.3 The "Hidden Behaviour" Impairment

C2ISs are often highly dispersed over network resources. They are also often composed of highly heterogeneous components. The use of middleware to facilitate communication between software components is common. DCOM (COM+), DCE, Java RMI and CORBA are all examples of software architectures that address middleware issues. The idea behind the use of middleware is to expose the functional interface of objects that offer services and advertise it. Client objects can then call these services through the use of stubs and proxy objects. The problem with this approach comes from the very fact that only functional aspects of service objects are exposed. These objects then become opaque to client objects in terms of performance requirements. For example, a client object needing a service would not know how to choose between server objects offering the same service interface, although the performance rendering of one may be superior to the other. This is the "hidden behaviour" impairment.

1.3.4 The Lack of a Performance Methodology

More often than not, performance issues are taken into account in a reactive manner. Software designers are typically focusing on implementing functionalities without bothering too much with performance aspects (the same goes for security issues). This results in systems that are oblivious to user performance requirements. The strategies and tools used to improve performance often only address the system level (e.g. increasing bandwidth, CPU capacity,

etc.) with little impact on the overall performance perceived by the user. These are all symptoms of the fact that there is no performance methodology that integrates the specification of user performance requirements and the software design per se. Software designers must rely on their knowledge to develop systems that target domains with which they may have little or no experience.

1.4 Organization of the Thesis

This document is organized in such a way that it addresses the four sub-problems enumerated earlier on a chapter by chapter basis. Each chapter stands on its own as much as possible so a reader can get to the one that addresses directly the sub-problem that interests him .

Chapter 1 gives a general overview of the "bigger" problem and is an umbrella for the definition of the sub-problems. One can read this chapter to gain a fair understanding of the fundamental problems of performance management for military information systems that are addressed in this text.

Chapter 2 introduces the Measures of Merit (MoMs) as a means to reconcile user-level and system-level performance perspectives. These measures constitute the tool by which performance is expressed. It also creates a link (not an introduction) to the military world, its language and its semantics.

Chapter 3 introduces the concept of Quality of Service (QoS) as a key concept to move from the notion of measuring performance to the notion of ensuring it. In this chapter, a definition of QoS will be constructed and the essential notions that compose it. Fundamental work on QoS will be reviewed.

Chapter 4 gives a brief overview of Messaging and Realtime CORBA that are now part of the CORBA specification. They constitute, to this day, one of the best efforts by a huge consortium of companies, the Object Management Group (OMG), to explicit object behaviour behind its interface and to enrich the middleware layer with mechanisms to propagate QoS information between objects and the network layer.

Chapter 5 is concerned with the construction of a methodology that will help the software designer to apply the tools of performance management. The methodology will sit as much as possible within the confines of the Rational Unified Process (RUP) and be instantiated with the Unified Modelling Language (UML). The real link between user and system performance requirements will be expressed in there. This methodology will serve as a framework for the user and systems designer to speak a language of performance that will be understood by both parties. Tools, architectures and strategies to meet performance requirements will have to be identified there so we will naturally move away from ad hoc solutions towards a sensible and planned approach to performance management.

Chapter 6 closes by making some observations on the exercise of managing performance in a military distributed application. It also considers what future work is to be done to advance in this research field.

Chapter 2

Command and Control Measures of Merit

2.1 Background

Computerized Command and Control Information System (C2IS)s have been in use for more than three decades. Though these systems were recognized to have significant impact in the effectiveness of military forces, it was also recognized that it was nearly impossible to quantify their impact. In [4, p. 1], Bettencourt reports that:

"...the period from mid- to late- 1970s saw development of new automated command and control systems, such as the Tactical Fire Direction System (TAC-FIRE) and the Tactical Operations System (TOS). These systems were being developed with all haste and they held great promise for increasing force effectiveness. ... Unfortunately, the tests and the Cost and Operational Effectiveness Analyses (COEAs) failed to show an increase in effectiveness which could be attributed to these systems. This was due, in part, to the inability to measure the effectiveness of these systems either in the tests or in the combat models used in the COEAs."

Researchers were urged to develop sets of measures and methodologies to assess C2IS performance. Since 1985, there have been several workshops on Measures of Merit sponsored by the US Military Operations Research Society (MORS). An analysis framework for C2IS performance and effectiveness measurement was produced from these efforts [5]. In 1990, the US Training and Doctrine (TRADOC) Analysis Center (TRAC) used this framework to develop the C2 Measures of Effectiveness Handbook [2]. The North Atlantic Treaty Organization (NATO) has also addressed the issue of defining a methodology for C2IS performance assessment. In 1995, the NATO Research Study Group 19 produced the Code of Best Practice (COBP) on the assessment of C2 [6]. This document describes the latest approaches to C2IS performance assessment. It reflects a unification of the ideas presented in earlier workshops and reports.

2.2 Influences on the Choice of Measures

There are several parameters that influence the choice of a useful set of measures. A set of measures is said to be useful if it enables an observer to reach his evaluation goals and needs. Two parameters are of prime importance: The scenario in which the C2IS is deployed and the observer's objectives.

2.2.1 Scenario

C2ISs are used in several different contexts. For example, systems that were designed for military operations are being deployed in civilian contexts. Metrics ordinarily used to measure system's performance in one situation may be irrelevant in another (e.g. for missile tracking, the "maintainability" of the system is far less important than its "timeliness of response").

"Relevant scenarios are required to stimulate each battlefield function, sub-functions, tasks, and organizational components within the context of the environmental aspects. Not all of these elements continuously interrelate and critical tasks must be identified. The scenario prescribes the tempo of operations and induces specific components of a C2 system to react. Partial evaluations may be merged by applying a range of (possibly overlapping) scenarios, with clearly defined hierarchies. Obviously scenarios and the models must reflect common hierarchies and levels of resolution." [7, p. 33]

It is therefore important to set up a scenario that will highlight the properties of the system to be measured. The scenario has to be meaningful in the sense that it must represent the situation in which the C2IS might be used. Once it is defined, the experiment is repeated as often as necessary to finish this part of the evaluation. Then, a scenario that highlights other system properties is run, and so on. This iterative process will eventually lead to a fairly complete knowledge of the system from several viewpoints.

2.2.2 Observer's Objectives

Observers may pursue different objectives in evaluating the performance of a C2IS. For one thing, roles are different from one observer to another, so their needs are distinct. Bettencourt [4, p. 11] is clear on that:

"There are clearly differences in the perspectives of the researchers, analysts, testers and trainers. This is due in part to the difference in disciplines, but it can also be attributed to the different stages of acquisition in which communities interact with the C3IEW system."

Table 2.1, derived from [5, Table 1-2], shows some typical roles and their impact on evaluation needs.

Table 2.1: Effects of Observer's Objectives on Measurement Needs

Observer Role	Analysis Objective	Effects on
Systems, Operational Commands, Requirements Analysts	Conceptual development	Planning and Doctrine System Requirements
System Commands Program Managers	Acquisition	Procurements System Specifications
R&D Scientists Design Engineers System Commands	Design	System Capability Choice of Technology Redundancy
Operational Commands and Commanders Operational Commanders Operational Planners Schools, System Sponsors	Operations	Development of Doctrine and Tactics Measuring Warfighting Capability System Deficiencies

Even for two observers having the same role, different sets of measures may be needed depending on the goals they are pursuing. Typical situations are mentioned in [7, p. 1]:

- For new requirements, the establishment of standards or expectations of performance; the establishment of the bounds of performance of a system and the effects of imposed constraints;
- The comparison and selection of alternative systems that may be very different but are designed to achieve a similar purpose;
- The assessment of the utilization of a system in new or unexpected application domains or missions;
- The identification of potential weaknesses in specific areas of an organization or system (areas of high error potential or high user workload);
- The analysis of the impact of organizational change;
- The determination of the effectiveness of training;
- The determination of the most cost-effective approaches to achieve desired objectives;
- The comparison of a replacement system, or components of the system, against its/their predecessor(s);
- Assistance in requirements generation and validation and the derivation of specific Command and Control requirements from broad statements of objectives; and
- The evaluation of the effectiveness and performance of human decision making in the Command and Control cycle.

The definition of a scenario and the clear statement of the objectives are needed to conduct an evaluation of the system.

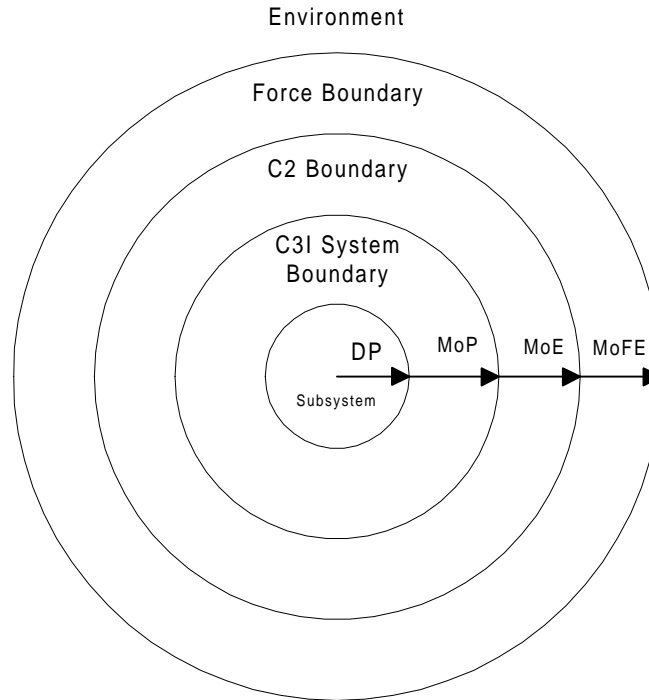


Figure 2.1: MORS MoMs Hierarchy

2.3 Definition of Measures

Since there are several aspects that can be measured in a C2IS, it is nearly impossible to meet every possible evaluation need with a single set of metrics. Since there are also different viewpoints from which to consider C2IS performance, different levels of measures are needed. Indeed, the benefits of having a C2IS can be considered from the commander's point of view to the low-level information system's one. Figure 2.1 shows MORS' four-level hierarchy of measures (see [4]).

These measures are called Measures of Merit (MoMs), and are composed of:

- **Dimensional Parameters (DPs):** These are the physical quantities that are used to represent different characteristics of the system (e.g. pixels, weight, bandwidth, memory, CPU speed, etc). DPs are scenario independent.
- **Measures of Performance (MoPs):** These are measures of the information system's attributes like throughput, mean information retrieval time, mean time to synchronize databases. MoPs are scenario independent.
- **Measures of Effectiveness (MoEs):** Measures that assess the impact of the C2IS on the functional organizational performance within an operational context. MoEs are scenario dependent.
- **Measures of Force Effectiveness (MoFEs):** These measures are related to how well the military force performs its mission. MoFEs are highly dependent on the scenario (e.g. Loss exchange ratios, mission success rate).

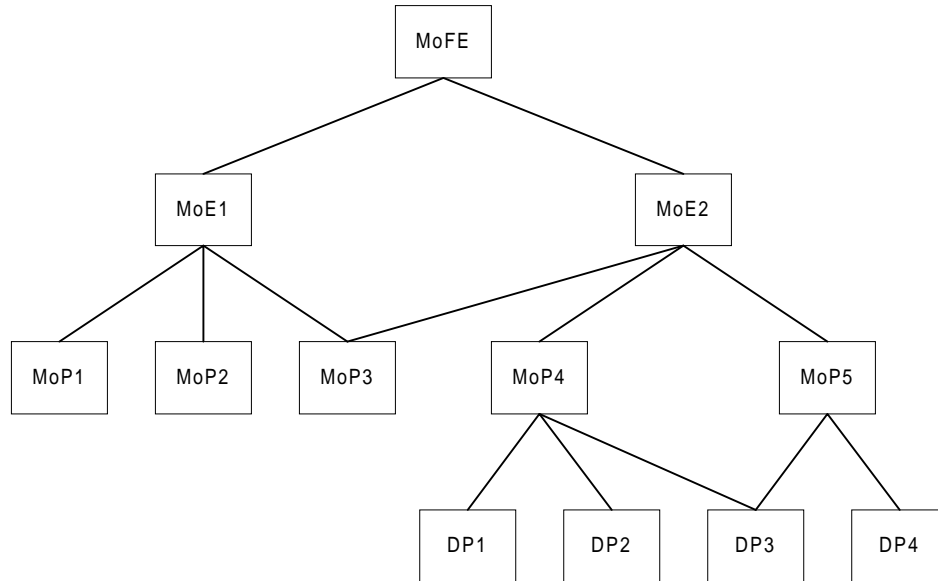


Figure 2.2: Hierarchical Relationship of Measures

The hierarchical nature of MoMs suggests that high-level measures are linked to low-level ones. An example of linkage between MoMs is shown in Figure 2.2 [4]. This mechanism allows the observer to determine subjective measures from objective ones. However, the links between those measures are not always easy to determine. In fact, links that are appropriate in one context may be inappropriate in another. In choosing a set of measures, an observer must be careful not to choose negatively correlated measures since variations will tend to cancel out.

2.4 Characteristics and Properties of Measures

Moving outward from the center of Figure 2.1, the level of abstraction of measures increases. The domains considered also change from the subsystem domain to the military force domain. Measures tend to be more qualitative, more scenario dependent and more expensive (in time and money) to obtain.

From 1985 to 1999, researchers understanding of the inherent characteristics of MoMs evolved. Listed simply as "desired criteria for measures" in [5, Table 6-4] and still in [2, Table 2-1], Pille reinterpreted these criteria (in [7]) and classified them into two distinct property families: reliability and validity criteria (see Table 2.2 and 2.3).

A measure is said to be reliable when multiple measurements of it lead to similar results. It represents precision and repeatability. Ensuring confidence in the value of a metric is time consuming and often difficult. Getting good estimates is sometimes more cost-effective than obtaining reliable measures.

Validity is the degree of focus a measure has on the system's aspect that an observer wants to assess. For example, the "message throughput" MoP is not a good candidate measure to assess database synchronization between two C2ISs. Good synchronization is related to

Table 2.2: Reliability Criteria of Measures

Reliability Criteria	Definition
Discriminatory	Can identify real differences between alternatives
Measurable	Can be computed or estimated
Quantitative	Can be assigned numbers or ranks
Objective	Can be defined or derived, independent of subjective opinion
Sensitive	Can reflect changes in system variables

Table 2.3: Validity Criteria of Measures

Validity Criteria	Definition
Mission oriented	Relates to force/system mission
Realistic	Relates realistically to the C2 system and associated uncertainties
Appropriate	Relates to acceptable standards and analysis objectives
Inclusive	Reflects those standards required by the analysis objectives
Simple	Easily understood by users

message throughput, but not completely characterized by it. It is not a valid criterion. The MoP "Completeness" linked to some low-level DPs might prove better suited.

2.5 Types of Measures

A common approach to C2 assessment is the performance assessment of its constituent parts at the functional level. This level corresponds to the middle layers of Figure 2.1. Therefore, MoPs and MoEs are the MoMs that give the most insight in determining the added value of a C2IS in a military environment. Furthermore, MoFEs are few, often expensive to obtain and can usually be determined only after scenario or operation completion.

Pille [7] sorts MoEs in two categories: Time-based and Accuracy-based measures. The very nature of these categories implies an inverse relationship between them as accuracy is obtained at the expense of time. However, a set of metrics for C2IS performance assessment should comprise measures from the two categories.

Time-based measures focus on:

- Responsiveness to an event
- Time to perform a task
- Frequency analysis of events
- Timeliness of events

Table 2.4: Measures of Force Effectiveness

MoFE
Territory gained/lost
Mission objectives achieved
Mission success rate
Battles won/lost
Enemy losses to friendly losses ratio
Number of targets destroyed
Percent combat effectiveness

Table 2.5: Time-based Measures of Effectiveness

MoE	Example
Time taken to perform a fixed task or sequence of tasks	Planning tasks
Time to perform a variable task	Developing and selecting options or courses of action
Time to recognize or respond to an event	Response to a critical enemy contact
Time to achieve a target state	Tactical objective
Percentage of time on target	Enemy location data up to date
Intelligence latency	Time taken to reflect ground truth
Number of events in queue	Messages pending action
Timeliness of response	Emergency situations

And accuracy-based measures focus on:

- Precision
- Repeatability
- Errors
- Completeness of information

2.6 Examples of Hierarchical Measures

Tables 2.4 to 2.11 from [7] and [2] represent lists of MoFEs, MoEs, MoPs and DPs that have been used in different operational contexts. They are not complete as new measures can always be created or derived from existing ones. An interesting thing to notice is that some measures can be found at two different levels (e.g. Bandwidth). The boundaries between levels are sometimes fuzzy, so that metrics can sometimes be included in a higher or lower layer within the hierarchical model. It is however undesirable to have the same metric at two or more different levels within the same evaluation. Also, the definitions of metrics are not always clear since the very meaning of a metric often depends on the operational context.

Table 2.6: Accuracy-based Measures of Effectiveness

MoE	Example
Accuracy or precision of performance of tasks	Information on maps, databases
Sensitivity of detecting system events	Recognition of events requiring change in plans
Probability of making errors	Errors in fire plan target schedules
Time to recognize existence of error	Necessity for plan alteration
Time to recover from error	Time to redo part of plan
Knowledge of current system status	Comprehension of battle situation
Quality of decision making	Quality of tactical plan

Table 2.7: C2MOE Handbook MoEs

MoE
Proportion fire requests beyond range Number of options remaining Percent action initiated by time ordered Mean dissemination time Proportion friendly elements engaged Percent orders clarification requested Percent planning time forwarded Time from mission to order Time to decision Warning orders to operations orders ratio Changes per order Repetitions per order Required number of commands Number of orders issued

Table 2.8: Hardware and Software Related MoPs

MoP	Example
Availability	Functional capabilities available to users
Survivability	Ability to survive partial destruction of system
Robustness/Endurance	Ability to adapt to environment
Maintainability	Ease of repair or replacement during operation
Computation capacity	Acceptable response times to users
Portability	Ability to operate on different platforms
Mobility	Ability to move with operational units

Table 2.9: Applications Related MoPs

MoP	Example
Interoperability	Communications with other C2 systems
Security	Confidentiality and integrity of data
Confidentiality	Information protected at appropriate level
Integrity	Required for confidence of data
Customizability	Ability to customize parameters to actual activities
Quantity of information	Provide all information required by user
Bandwidth	Ability to support multi-media

2.7 Determining a Set of Metrics

In order to conduct his evaluation, an observer has to know clearly the objectives he wants to achieve. A scenario that will exercise the aspects of the system the observer wants to assess has to be written. Once this is done, metrics must be chosen, linked, and characterized in their range values.

2.7.1 Choosing the Metrics

Knowing who (or what) will benefit from the C2IS evaluation (e.g. analyst, engineer, commander, another system, etc.) gives a good idea of the level of abstraction needed in MoMs. If it is a commander, it is probable that MoFEs will be needed. If the beneficiary is another system, then DPs and MoPs are needed at most. A top-down approach leads to a list of appropriate measures in the context of the evaluation (as an example, see [8]).

2.7.2 Linking Metrics Between Them

Abstract and subjective high-level measures may have to be linked to quantitative and objective low-level ones in order to be determined. In these cases, links are to be defined (see Figure 2.2).

2.7.3 Determining Variation Ranges

As the links between measures are defined, some low level measures may be found to be of no impact on higher level measures they are linked to. Those have to be discarded as they only complicate matters. Negatively correlated measures are of no help either. Finally the range of all metrics should be determined. The next step is to run the scenario and collect data.

Table 2.10: User Effectiveness Related MoPs

Information Quality	
MoP	Example
Selectivity	Ability to provide required information at required amount
Accuracy	Degree information provided is correct
Comprehension	Facilitate understanding of situation
Time Related	
MoP	Example
Response time	Response to requests within established times
Timeliness	Information available at appropriate time
Ease of use	Ease of access to information in a timely manner
Training time	Time to train users
Decision response time	Time available to commanders
New Capability Related Attributes	
MoP	Example
Uncertainty management	Ability to take into account ambiguity and uncertainty
Simulation	Ability to realistically test courses of action before decision
Information age management	Ability to store information for later analyses
Motivation	
MoP	Example
Communication acknowledgement	Positive feedback
Knowledge of consequences	Positive for stimulus but negative for stress
Confidence of information	Not used if untrustworthy
User friendliness	Adaptability to different users
Participation in development	User buy-in
Ineffectiveness	
MoP	Example
Loneliness	Isolation from real world
Interference	Micro-management
Wait and see	Waiting for confirmation instead of deciding
Wargame	C3I reflects real situations

Table 2.11: Dimensional Parameters

DP
Signal to noise ratio
Bandwidth
Resolution
Floating point operations per second
Radar cross-section
Radar resolution
Bit error rate

Chapter 3

Quality of Service

3.1 Introduction

The goal of this chapter is to help the reader to grasp the idea of QoS in its most general sense. It is hoped that the reader will be able to apply, in other contexts, the notions that unfold from this concept. QoS is, and will remain, a fuzzy concept even after reading this chapter. Still, important lessons arise from the work that has been done in this field, one of which being that performance has to be managed and not only measured. The terminology employed in this chapter is consistent with the ISO/IEC QoS framework [9].

3.2 Definition of Quality of Service

In this section, several definitions of QoS taken from the literature will be considered. Some of them are partial, others more complete. Some definitions are near equivalent to others, some stand out as more unique. The goal is to form one encompassing definition of QoS that will serve as an umbrella to the concepts developed in this text. To this end, key terms from each definition will be extracted and collated so a unified definition will stand out. The exercise is not futile since it will allow the pinpointing of the aspects of QoS that will be important to consider when devising a strategy for QoS implementation in military distributed applications.

3.2.1 The Notion of Predictability and Consistency

"In the simplest sense, Quality of Service (QoS) means providing consistent, predictable data delivery service. In other words, satisfying customer application requirements." [10]

The keywords in this definition are "consistent" and "predictable". A service response is said to be predictable when the conditions for its delivery are known over a fixed period of time. These conditions are specified by QoS measures like latency, jitter, throughput, accuracy, etc. Perfect predictability of a service delivery means that the conditions under which the service is rendered are known for an infinite period of time and null predictability

occurs when the conditions are never known in advance. Predictability is further discussed in [11].

Consistency is the correspondence of an expected response to an actual one maintained over a period of time. That is to say, the closer the nature of a response is to the expected one for a long period of time, the higher the consistency is. For example, if a service requester expects to receive an image and gets one, then consistency is said to be high. If a string of ten characters was expected and a fifty page document is actually received instead, then consistency is low. One might think that the strong typing used in object-oriented (OO) programming languages like Java or C++ ensures response consistency. It does to a certain extent, but the nature of certain responses like streaming video can still affect consistency without violating strong typing. Indeed, if a 30 second video stream is expected when a 5 minute video stream is actually received, then consistency is affected.

A distributed application could take advantage of consistency and predictability by adapting its behaviour according to the variations of these properties. Of course, establishing service predictability and consistency is difficult and subject to hot debates, but the key notion here is that distributed applications have to adapt to their environment, much like living beings have to adapt to their environment to survive. Predictability and consistency of service rendering are useful only if the distributed application makes use of them by adapting.

3.2.2 The Notion of Guarantee

"Quality of Service (QoS) is to the ability of a network element (e.g. an application, host or router) to have some level of assurance that its traffic and service requirements can be satisfied." [12]

Ensuring QoS in a system (from a performance perspective) means that certain levels of service delivery have to be guaranteed. From thereon, tools must be built, measures must be taken, strategies must be followed and architectures must be devised to support and maintain these guarantees.

3.2.3 The Notion of Management

"Managing the service response of an Internet network is often referred to as Quality of Service..." [11]

"Quality of Service (QoS) is the term that is being used to loosely organize the collection of activities and technology initiatives that have emerged to improve and control network oriented resource management based on mounting experience with distributed, Internet applications." [13]

"For service providers and network engineers, QoS means engineering and managing network resources to deliver performance levels that satisfy their users' expectations." [14]

"Quality of Service is a network "term of art" for describing technologies that allow service providers to manage network congestion rather than simply to add capacity." [15]

Management is a central concept to providing QoS. [9] defines QoS Management Function (QMF) as the collection of smaller QoS mechanisms (e.g. negotiation, admission control, monitoring, etc.) that helps to meet QoS users' expectations and requirements. The QMF is necessary because there is no single solution to ensure QoS in a distributed application. This comes from the fact that:

- Information systems are dissimilar between one another;
- A distributed application may be used in different scenarios (For example, a C2IS may be used in a search and rescue operation and later in military operations); and
- QoS mechanisms evolve (e.g. new negotiation process, new monitoring tools, etc.).

QMF is necessary in that it helps to achieve predictability and consistency.

3.2.4 The Notion of End-to-end

"To enable QoS requires the cooperation of all network layers from top-to-bottom, as well as every network element from end-to-end. Any QoS assurances are only as good as the weakest link in the "chain" between sender and receiver."

[12]

QoS is achieved in a distributed application only if it has been addressed at every level of the OSI 7-layer model (see [16] for this model). Unfortunately, existing QoS solutions do not address all levels in the stack. IntServ (Integrated Services) and DiffServ (Differentiated Services), the two main QoS architectures, operate solely at the IP level. From a network designer's point of view, end-to-end QoS is achieved when packets are delivered to the application level with respect to a certain QoS context. The fact is that all the tools and mechanisms offered at the IP level within these QoS architectures are only good if the application itself specifies, tunes and shapes its own message traffic. This is to say that similar QoS strategies have to be developed at the application level (between the distributed objects and/or within the middleware) to achieve end-to-end QoS. This exercise could also be pursued further by encompassing business logic. Applied to the military realm, Quality of Information (QoI) for the military user should be ensured. Indeed, the military does not know how to specify low-level QoS requirements, but very well knows his needs, and expresses them in terms of MoEs. Common sense shows us that different levels of QoS should be decoupled and renamed according to the layer they address in the OSI 7-layer stack (see Figure 3.1). Notice that the QoI level stands out of the stack. At this level, business processes requirements have to be specified. In the military domain, it is addressed with MoEs/MoPs specifications, a language that is spoken by the military users.

3.2.5 So, What is QoS Anyway?

The important notions extracted from the few definitions above should now make it possible to forge one encompassing definition of QoS, hopefully more complete:

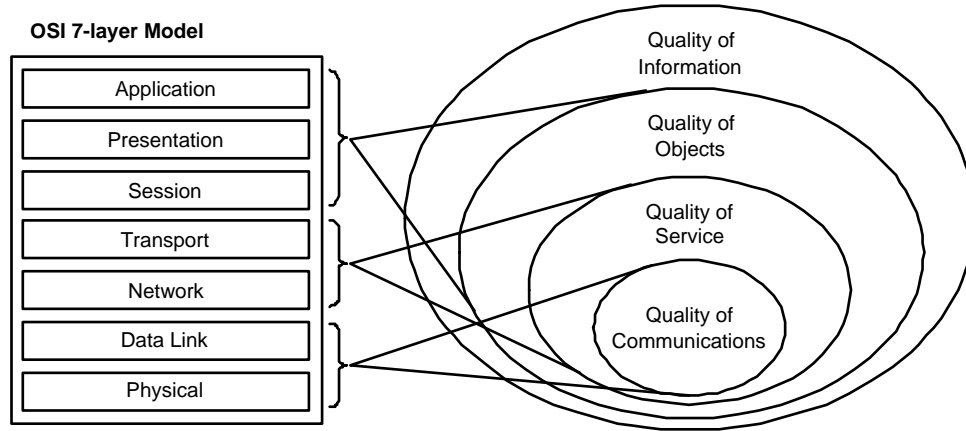


Figure 3.1: Mapping the OSI 7-layer Model to Quality Models

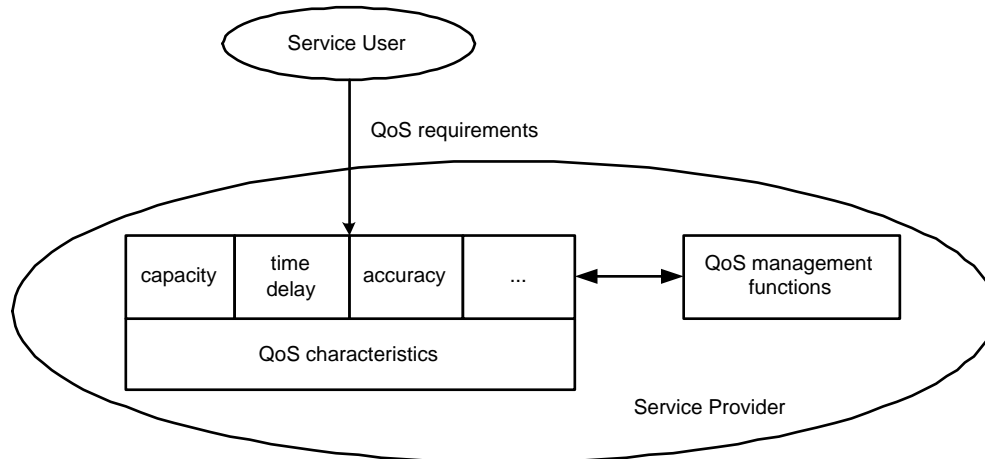


Figure 3.2: Relationships Between QoS Concepts

"Quality of Service is an encompassing term for the collection of activities, management functions and strategies that aim at guaranteeing the end-to-end, predictable and consistent behaviour of network-dependent applications"

3.3 QoS Concepts

Figure 3.2 shows the relationships between QoS concepts, as defined by [9]. It helps to refine the comprehension between several terms related to QoS, namely, QoS requirements, QoS characteristics, QoS parameters, QoS context and QoS management functions.

3.3.1 QoS Requirements, QoS Parameters and QoS Context

The notion of guarantee in the QoS definition supports the fact that QoS management activities are driven by user requirements. In our context, these requirements would be formulated

at the user level (the commander) as sets of MoEs. A correct mapping of these user requirements would lead to the definition of QoS requirements. These requirements are conveyed to the service provider as QoS parameters. QoS parameters may be of different kinds, including (see [9]):

- Reaching a desired level
- Crossing a threshold value
- Issuing a warning or alert to take corrective action
- And so on...

The set of user requirements mapped to QoS parameters forms the QoS context. Figure 2.2 can be seen as a QoS context if dimensional parameters (DPs) in this figure are viewed as QoS parameters. The same rationale developed in Chapter 2 for defining links between MoEs, MoPs and QoS parameters applies in the present context.

3.3.2 QoS Characteristics

QoS characteristics are quantifiable aspects of QoS that are independent of the way they are measured. Timeliness, coherence and reliability are examples of QoS characteristics. A QoS characteristic is quantified using a vector of QoS parameters, as depicted in Figure 3.2. This distinction between parameters and characteristics is necessary because there is no single way to take measures in a system. For example, the throughput is a characteristic that could express the amount of information transmitted over a network channel over a certain period of time. It could also be expressed as the number of messages successfully transmitted over that same channel out of a fixed amount of transmitted messages. The first might be expressed in bytes per second while the second would be in the number of messages per bulk transfer. In the end, both ways express throughput in some fashion.

Generic characteristics (shown in Figure 3.3) have been defined in [9]. The idea behind this is to have them generic enough so that they can be specialized or derived as depicted in Figure 3.4. The process of specializing generic characteristics consists in narrowing and focusing their meaning to better suit a particular context. Derived characteristics are mathematical functions of generic ones. For example, jitter is a function of time delay.

3.3.3 QoS Management Functions

"QoS management refers to all the activities relating to the control and administration of QoS within a system or network." [9]

The ultimate goal of QMFs is to satisfy user requirements in QoS by allocating resources, following determined QoS strategies and using specific QoS mechanisms. The section on QoS architectures will define IntServ/RSVP and DiffServ which are the two main architectures that have been developed for achieving QoS to date.

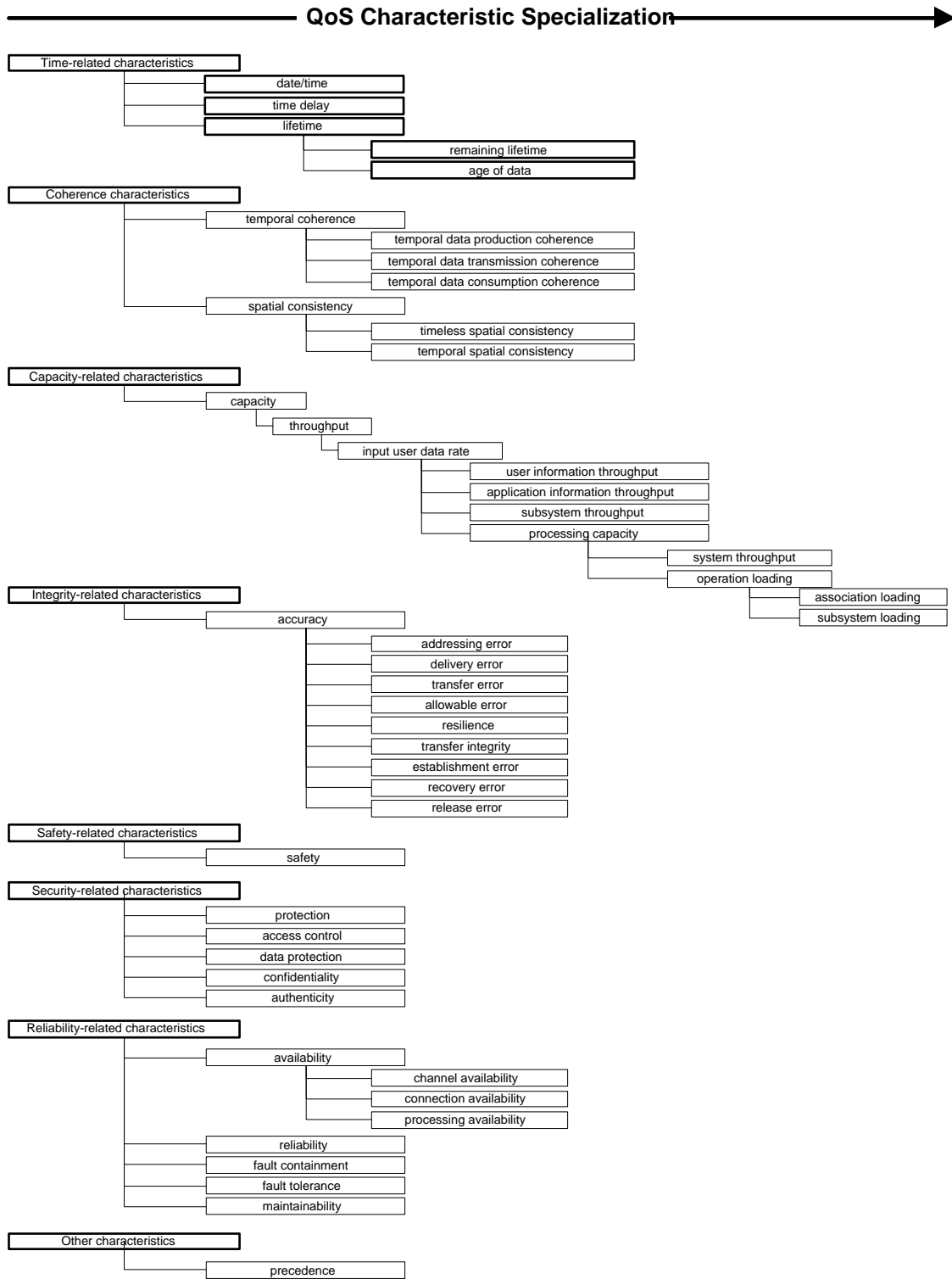


Figure 3.3: ISO/IEC 13236 QoS Characteristics Hierarchy

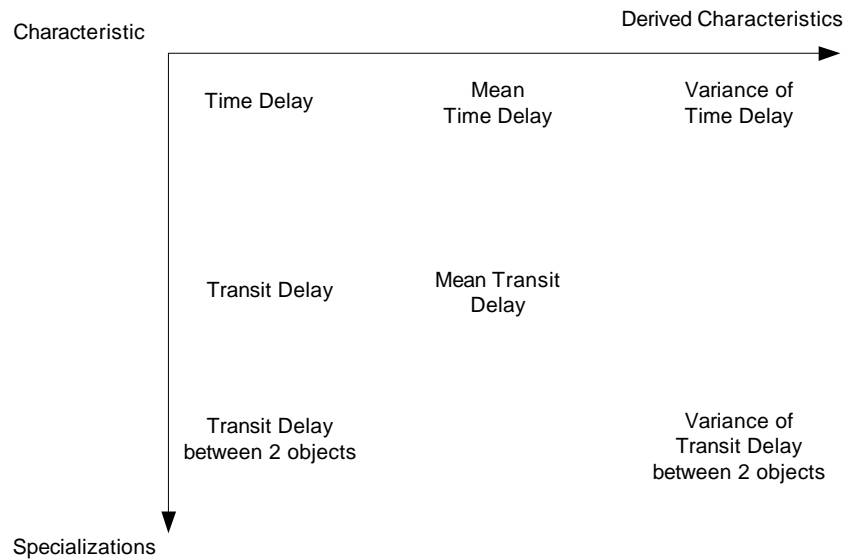


Figure 3.4: Derived Characteristics and Specializations

3.4 QoS Architectures

As mentioned above, QoS solutions, architectures and tools have been developed mainly for the third layer of the OSI 7-layer model. In the TCP/IP protocol suite, this layer corresponds to IP, the Internet protocol. Many QoS tools have been developed, focusing on different aspects of packet routing and delivery, namely, packet classification (using the Type of Service byte in IPv4 or Traffic Class for IPv6), queuing disciplines, packet dropping policies, traffic shaping, admission control, protocol performance tuning techniques and so on. [11] presents these tools extensively. While these tools treat important performance aspects of packet routing, they still have to be organized coherently so that the greater goal of achieving QoS is achieved. To build a house requires a complete set of different tools. To achieve QoS requires a complete set of QoS tools. What is needed is a QoS strategy that will provide the necessary framework and designated strategies that will enable end-to-end QoS. Today, there are two QoS architectures: The Integrated Services (IntServ) and the Differentiated Services (DiffServ) architectures. The reader might argue at this point that the focus of this thesis should stay at the application level, within CORBA and this is true. QoS solutions developed at specific levels of the OSI 7-layer model should not interfere with other levels and should remain independent. For example, an application relies on TCP for reliable communications, and TCP in turn relies on IP to find the destination host. Every layer is concerned with only one aspect of the communication. However, QoS research is more advanced at layers 2 and 3. Also, there are striking similarities between a CORBA Object Request Broker (ORB) and a router, at least at the macroscopic level (see Figure 3.5).

The unit of information that router handles is the datagram. A CORBA ORB handles messages (in the object-oriented sense). Datagrams and messages are of variable length. A router has a routing table that helps it to make forwarding decisions while the ORB relies on its interface and implementation repositories to link client and server objects. Probably the most important similarity between packets and messages is that they are treated on a best-

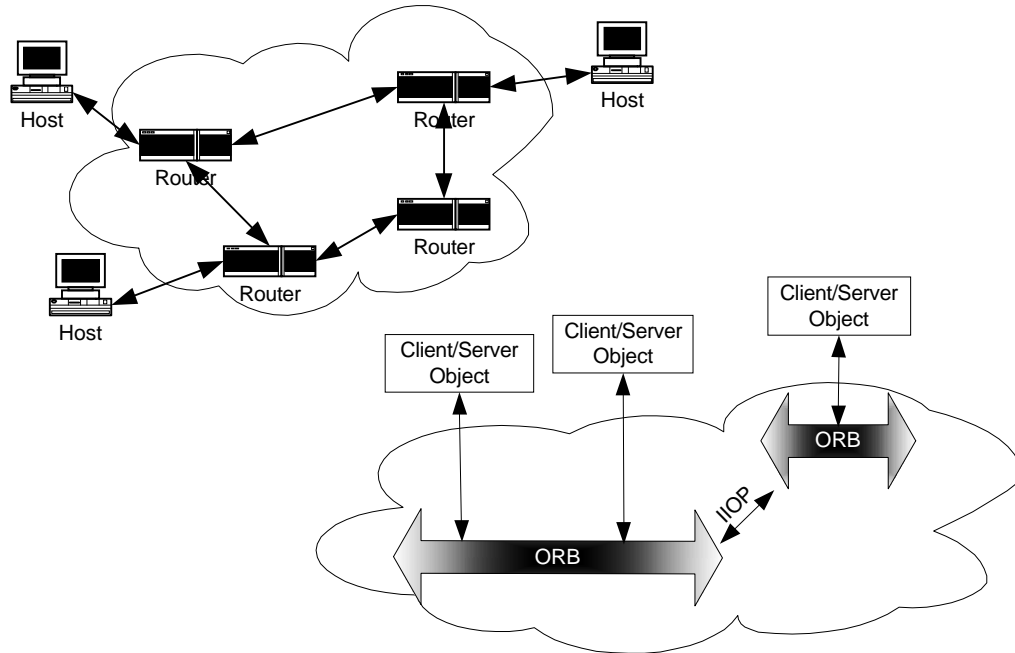


Figure 3.5: Packet vs Message Routing

effort basis. The whole Internet is based on this principle and it is one reason for its spreading. However, it is also a reason for its difficulty in supporting certain classes of services like voice and streaming video. At the application level and especially in the middleware, almost the same approach prevails. It is reasonable to argue that if QoS architectures have to be developed at the packet level, the same exercise has to be conducted at the application level and furthermore within the middleware architecture. Of course, a 1:1 mapping of IntServ or DiffServ is unlikely to happen, but certain concepts may apply.

3.4.1 Integrated Services Architecture (IntServ)

The Integrated Services architecture was developed with the idea of providing a support framework for real-time applications like teleconferencing, IP telephony, distributed simulations and so on. It was designed and wanted as an extension to the actual best-effort traffic delivery model currently in place in the Internet today and not as a replacement. Indeed,

"...real-time applications often do not work well across the internet because of variable queueing delays and congestion losses. The Internet, as originally conceived, offers only a very simple quality of service (QoS)" [17]

It was therefore necessary to develop a service model that would treat real-time traffic preferentially over best-effort traffic. In order to achieve this, assumptions were made:

- Network resources must be managed in order to meet applications requirements.
- The Internet must be kept as the common infrastructure to support both non-real-time and real-time communications.

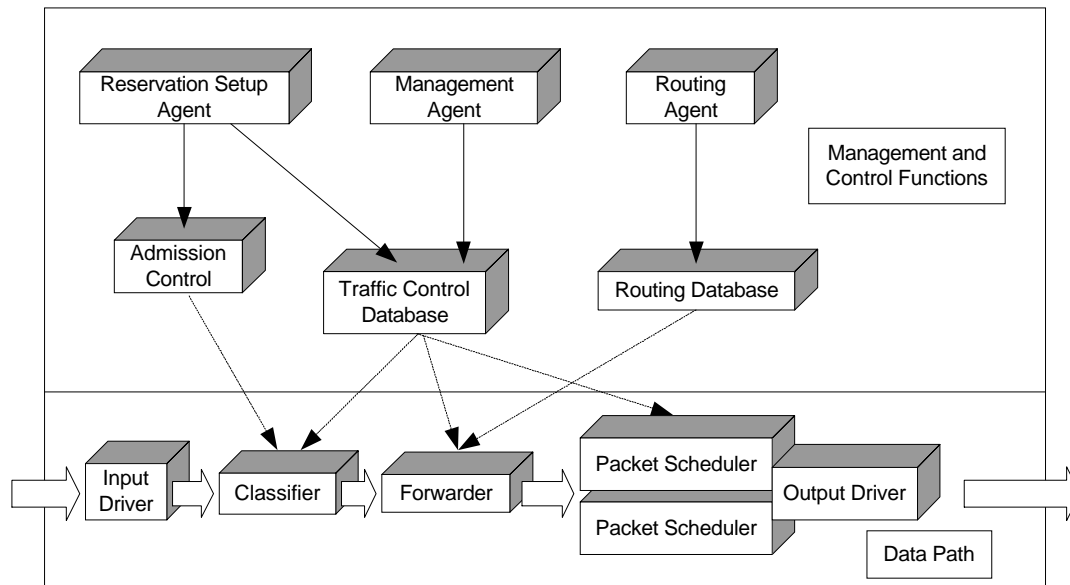


Figure 3.6: IntServ Reference Model

Within the IntServ model, management of resources is ensured by resource reservation and admission control. Being a host-centric model, these mechanisms are embedded into routers as the primary IP traffic forwarders and by applications as the IP traffic end points.

"...Integrated Services provides end-to-end signaling for QoS parameters at layer 3 in the Open Systems Interconnection (OSI) model...each reservation of resource requires a state to be maintained within the network." [11]

A notion central to IntServ is a resource reservation state that will be maintained throughout the whole duration of the service life cycle. A node cannot fall back after it has made a commitment to reserve some of its resources. The idea behind this is to realize predictability of service response. A common analogy to this is a telephone call. Once a number is dialed, a connection is maintained for the whole duration of the conversation until someone hangs up.

Figure 3.6 shows the IntServ reference model. The forwarding path in the lower part of the diagram must be highly optimized to minimize induced delay and jitter. The background code in the upper part influences the forwarding path with routing and admission control decisions and by maintaining resource reservation states for the node. Traffic control is realized in IntServ through two mechanisms: The classifier and the packet scheduler. These mechanisms are in turn influenced by the admission routine and resource reservation controller.

3.4.1.1 Flow Resource Reservation and Admission Control

Within IntServ, an application states its resource requirements by specifying its flow specification, or flowspec (see [18] and also [19]). A flowspec is a characterization of a traffic class

that is understandable by IntServ-aware nodes. Based on it, a router is able to make admission control decisions and resource reservation commitments. These commitments depend on remaining resources, authentication of the flow and other criteria. Once a commitment is made by a node, it is maintained for the duration of the flow. Each node has to make the same commitment for the actual route to be finally set up. This is to say that an appropriate QoS signaling mechanism is needed for the end-to-end commitment to take place. This need led to the development of the Resource ReserVation Protocol (RSVP) (see [20] for a complete explanation of this protocol). From there on, IntServ was often referred to as IntServ/RSVP.

3.4.1.2 Packet Classifier

Although packets may be parts of distinct flows, they may need the same level of treatment. The job of the classifier is to group packets into classes of flows. Each flow in a class will be treated on an equal basis by the packet scheduler. For example, a class may contain all video flows, and another one regroup best-effort traffic flows.

3.4.1.3 Packet Scheduler

Once the traffic flows have been aggregated into classes by the classifier, they are queued. The packet scheduler is responsible to forward packets from the queues to the output driver based on QoS requirements that are imposed by the background code, which is in turn influenced by the commitment that was made by the IntServ-aware node. Most of the time, QoS requirements are expressed as timing constraints. The scheduler is responsible for using the right tools that will help to meet these requirements. For example, First-In, First-Out (FIFO) queuing would be used in underloaded conditions while Weighted-Fair Queuing (WFQ) would be better suited for resource contention resolution (see [11] for more details about queuing algorithms). Packet dropping policy is also a useful congestion control mechanism.

3.4.2 Differentiated Services (DiffServ)

The principle behind DiffServ is to ensure QoS by defining regions of known traffic forwarding behaviour. Within a region, routers, QoS strategies and other QoS mechanisms are known to be able to respond to a certain traffic class. Boundary nodes (known as ingress or egress nodes) are responsible for admission control, traffic shaping, policing and marking. Interior nodes have to be able to forward traffic according to the traffic class of packets, but do not have to apply admission control. Therefore, interior routers are simpler in design than boundary ones (see Figure 3.7). The idea behind DiffServ is to create regions where traffic forwarding becomes deterministic and therefore increase predictability. Within a region, IP traffic is handled only in a certain number of ways, depending on the traffic classes and every class of traffic in a region can be handled by every node.

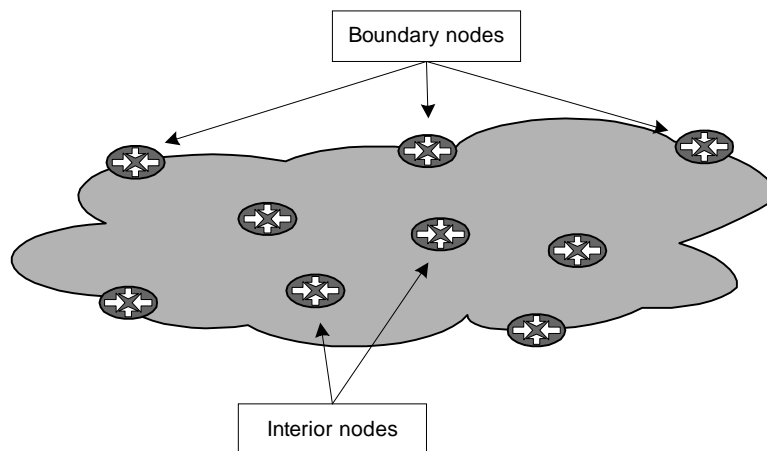


Figure 3.7: DiffServ Reference Model

3.4.3 Observations on IntServ/RSVP and DiffServ

IntServ (coupled with RSVP) and DiffServ, though they aim at achieving predictable packet delivery, present fundamental differences that make them more or less applicable in certain contexts. The first aspect is that of scaling. In an Internet context, DiffServ seems to be the most approachable solution since QoS negotiations occur between egress and ingress routers of concurrent networks. Also, since negotiations only exist at the region boundaries, if regions include large numbers of nodes, DiffServ becomes a cheaper approach than IntServ because interior routers are much simpler than IntServ/RSVP compliant ones.

At small scales, IntServ is a more complete solution than DiffServ. This is mainly because it addresses the end-to-end path more completely than DiffServ. In fact, DiffServ doesn't address it all too well as it is unidirectional and doesn't maintain a resource reservation state as RSVP does.

As it comes out, it seems that neither IntServ nor DiffServ offers a complete solution. IntServ has poor scaling properties, and DiffServ does not do much about managing resources. An internet draft ([21])¹ from the Internet Engineering Task Force (IETF) proposes to merge the two approaches (see Figure 3.8). The development of this model was motivated mainly by the need of end-to-end QoS for video-on-demand, time-critical and mission-critical applications ([11]). The advantage of this approach is that it combines the inherent properties of both IntServ/RSVP and DiffServ, namely scalability and reservation of resources. However, it is important to mention that the DiffServ transit network will have to be able to convey RSVP messages for the combined architecture to work.

The next section discusses QoS in CORBA. The focus will therefore move from layer 3 to layer 7 of the OSI model.

¹Since this document is outdated by IETF standards, its status is unknown and not retrievable from the IETF website <http://www.ietf.org>.

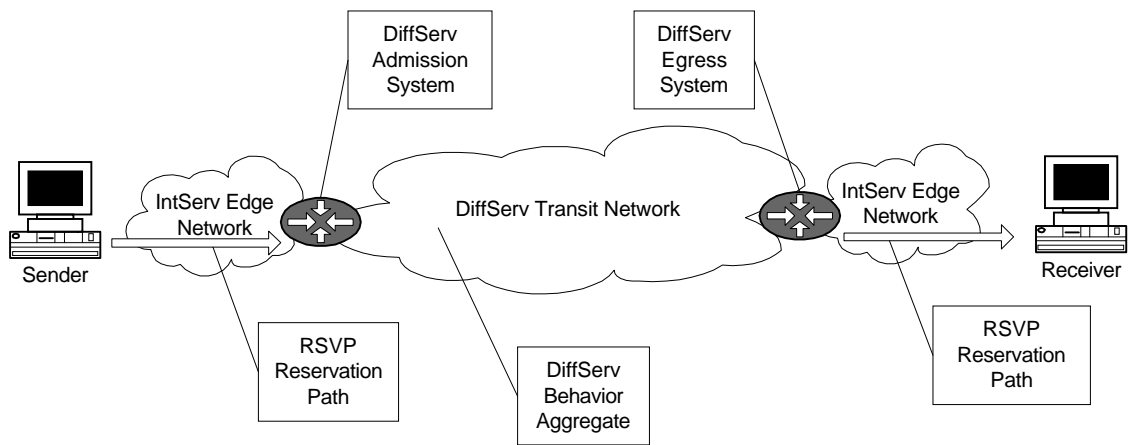


Figure 3.8: Combining the IntServ and DiffServ Architectures

Chapter 4

Quality of Service and CORBA

4.1 Introduction

Until late 1999, there were no QoS mechanisms in the specification of the Common Object Request Broker Architecture (CORBA) [22], no ways to control the ORB behaviour in order to meet stringent needs in performance or timeliness. A short paragraph merely mentioning the term in version 2.2 was abandoned in the revision 2.3. As the need for reliable Distributed Applications (DAs) grew and as CORBA became a de facto standard for Distributed Object Computing (DOC), the pressure was on the Object Management Group (OMG) to give implementation vendors and ultimately software developers the needed framework and tools to ensure QoS in their DAs. It was indeed recognized that some responsibilities regarding the propagation of QoS information between objects composing DAs should fall under the middleware's control. In this regard, the Messaging specification was developed along with Real-Time CORBA (RTCORBA) and minimumCORBA. These were in fact separate white papers and were promised to be integrated in CORBA 3, a major revision. Instead, they were added to CORBA 2.4.1 as new chapters.

This section focuses mainly on Messaging QoS and some aspects of RTCORBA, minimumCORBA being out of the scope of this thesis. It is however important to keep in mind that in the military context, minimumCORBA might be the only affordable choice though. Since this thesis concentrates on how military commanders' choices are reflected in DC2ISs built on CORBA, development of light C2ISs using minimumCORBA is less of our concern.

4.2 CORBA Messaging

CORBA Messaging offers Application Programming Interfaces (APIs) that enable objects to tune the middleware's behaviour regarding rebind modes, synchronization, priority, timeliness, routing, and queueing. This section briefly describes these APIs.

4.2.1 Policy Management

Though the middleware cannot ensure that objects that compose a DA will behave in an appropriate manner and thus ensure end-to-end QoS, it can however commit to propagate

information regarding the QoS offered or required by objects to other objects or even to the middleware. This is made possible through the use of Policy objects associated to the objects composing the DA. A Policy object is further defined by a `PolicyType` object. By defining policies, a developer can carefully tune his application behaviour to better suit certain application classes. The OMG defined 57 standard `PolicyType` objects that are meant to be used in different context APIs, namely, Security, PortableServer, RTCORBA and Messaging. The ones that are specific to Messaging and RTCORBA are listed in tables 4.1 and 4.2.

Policy management concerns the scope within which the `PolicyType` objects apply. Depending on this, policies can be enforced at ORB level, thread level or object level. Policies enforced at object level override those defined at the thread level which in turn override those at the ORB level.

System-level Defaults: ORB implementations come with system defaults for their different behaviours. It is important to understand that these defaults are not specified by CORBA and that if common behaviour across ORBs is sought for, it is then necessary to override them at the appropriate level.

ORB-level Policies: This level concerns the policies applied to objects interacting with an ORB instance. Since an object may *directly*¹ interact with more than one ORB, ORB-level policies may be useful to define minimal behaviour expected within a particular ORB realm, provided that no overriding occurs at the thread or object levels. Policy overrides at this level can be set or retrieved using a `PolicyManager` which in turn is obtained with an `ORB::resolve_initial_references` call, specifying an `ORBPolicyManager` identifier.

Thread-level Policies: These policies apply to the objects of a specific thread unless overrides at the object level exist. Policies at this level can be set or retrieved using a `PolicyCurrent` which in turn is obtained with an `ORB::resolve_initial_references` call, specifying a `PolicyCurrent` identifier.

Object-level Policies: This level is the finest achievable granularity for Policy objects in CORBA. Policies at this level work on a per-object basis. The `CORBA::Object` interface has methods to manipulate them. The *effective* client-side policy of an object is determined by combining those at object level, thread level, ORB level and system-defaults level.

Server-side Policy Management: Server applications can publish the QoS-type information that they support. This can be done along with the creation of the Portable Object Adapter (POA). Server-side policies are exported by the POA with the Object reference they return. By inspecting the `TAG_POLICIES` component embedded in the object reference, client-side objects are able to honor server-side policies, search for other server implementations or cancel their request.

¹This is different from an object using ORBs through IIOP

Table 4.1: Standard Messaging Policy Types

Policy Type	Policy Interface	Tag
REBIND_POLICY_TYPE	Messaging::RebindPolicy	23
SYNC_SCOPE_POLICY_TYPE	Messaging::SyncScopePolicy	24
REQUEST_PRIORITY_POLICY_TYPE	Messaging::RequestPriorityPolicy	25
REPLY_PRIORITY_POLICY_TYPE	Messaging::ReplyPriorityPolicy	26
REQUEST_START_TIME_POLICY_TYPE	Messaging::RequestStartTimePolicy	27
REQUEST_END_TIME_POLICY_TYPE	Messaging::RequestEndTimePolicy	28
REPLY_START_TIME_POLICY_TYPE	Messaging::ReplyStartTimePolicy	29
REPLY_END_TIME_POLICY_TYPE	Messaging::ReplyEndTimePolicy	30
RELATIVE_REQ_TIMEOUT_POLICY_TYPE	Messaging::RelativeRequestTimeoutPolicy	31
RELATIVE_RT_TIMEOUT_POLICY_TYPE	Messaging::RelativeRoundtripTimeoutPolicy	32
ROUTING_POLICY_TYPE	Messaging::RoutingPolicy	33
MAX_HOPS_POLICY_TYPE	Messaging::MaxHopsPolicy	34
QUEUE_ORDER_POLICY_TYPE	Messaging::QueueOrderPolicy	35

4.2.2 Messaging PolicyType Objects

As we have seen above, `PolicyType` objects may be applied at the ORB level, thread level or object level, depending on the need. This section describes the `PolicyTypes` objects that are specific to QoS within the Messaging specification. These objects address several aspects of QoS from a middleware viewpoint, namely, rebind support, synchronization, priority, timeliness, routing and queueing. Messaging QoS `PolicyType` objects are enumerated in Table 4.1.

Knowing how to attach policies to objects in a DA and what scope they are addressing is one thing, but it is also necessary to apply the right policy in the right context. Policy types apply to several categories of DAs such as security, portable server, message routing, CORBA service transactions, Messaging QoS and RTCORBA. In this section, we are concerned with Messaging QoS Policy types (as enumerated in Table 4.1) since they will constitute the support for our experimentation (see Chapter 5). RTCORBA Policy types will be described in section 4.3.

Rebind Support: This is the level of transparency with which an ORB will rebind to an object. The default value is `TRANSPARENT` which means that a remote request will be reconnected silently by the ORB to the object if needed to. `NO_REBIND` allows the ORB to silently reconnect to an object, but rebind on `LOCATION_FORWARD` is not allowed. That may prevent binding to objects that have different QoS policies than the primary target. The last possible setting is `NO_RECONNECT` which forbids the ORB silently rebinding from a request to an object unless it is explicitly requested through a `CORBA::Object::validate_connection` call. Rebind support offers a degree of control to client applications that is crucial to the development of QoS-enabled DAs. It has a great impact on predictability which is the key in achieving these applications.

Synchronization Scope: As executing environments for DAs are often heterogeneous and very dispersed, it is of prime importance for client objects to be able to invoke non-blocking calls to servers. This allows for DAs to gracefully degrade QoS offering instead of presenting abrupt behaviours like hang-ups. CORBA traditionally offered the oneway operation

attribute to realize asynchronous calls. However, the invocation semantics of this call are purely best-effort and failure is not reported to the client. The Synchronization Scope policy refines the concept of asynchronous calls. Instead of fully blocking or blind asynchronous calls, a client can track a call through different levels and thus infer a certain level of assurance that its request is treated. The possible settings for this policy are:

- `SYNC_NONE` - The equivalent of a oneway operation. The ORB returns control to the client before passing the request to the transport protocol.
- `SYNC_WITH_TRANSPORT` - The ORB returns the control to the client after the transport protocol has accepted the request. Depending on the transport protocol used, this policy can give better assurance that the request is taken care of.
- `SYNC_WITH_SERVER` - The client is blocked until a notice from the server comes in, but prior to the execution of the request. Reception of a `NO_EXCEPTION` means that any necessary location-forwarding has already occurred. This type of synchronization is useful in cases of highly reliable target objects.
- `SYNC_WITH_TARGET` - This is the equivalent of a traditional synchronous call which is blocking.

Request and Reply Priority: As the name suggests, requests from clients and replies from targets can be treated according to their respective priorities. The `RequestPriorityPolicy` and `ReplyPriorityPolicy` are set using a `PriorityRange` set from `Priority min` to `Priority max`. The reason for using a range instead of an absolute number is that the client can change it within that range. However, these policies are valid only if the `QueueOrderPolicy` of routers (see paragraph on routers below) along the route has the value of `ORDER_PRIORITY`. A `RequestPriorityPolicy` can be represented in target objects if it is specified when the POA is created.

Request and Reply Timeout: These policies affect the lifetime of requests and replies. They must be used in conjunction with the CORBA Time Service. They may be used to enforce timeliness of requests and replies.

- `RequestStartTimePolicy` - Indicates the time after which a request can be delivered to the target.
- `RequestEndTimePolicy` - Indicates the time after which a request may no longer be sent to the target.
- `ReplyStartTimePolicy` - Indicates the time after which a reply may be sent back to a client.
- `ReplyEndTimePolicy` - Indicates the time after which a reply may no longer be sent back to a client.

- `RelativeRequestTimeoutPolicy` - Indicates the relative amount of time an ORB has to deliver a request, after which it is cancelled.
- `RelativeRoundtripTimeoutPolicy` - Indicates the relative amount of time an ORB has to deliver both the request and reply, after which the former is cancelled.

Routing: Section 3 of CORBA Messaging [22] describes message routing interoperability. As the name suggests, this part of the specification is concerned with offering some degree of control to objects over the way the ORB forwards messages to other ORB agents. Routing influences timeliness of responses and may also impact on predictability of the DA. Therefore, careful planning of this policy is important. `RoutingType` can take three values:

- `ROUTE_NONE` - No routers will be used in this context. This routing type is used within synchronous or deferred synchronous scope.
- `ROUTE_FORWARD` - This route type indicates that a router will be used to deliver the message. It is used within an asynchronous scope.
- `ROUTE_STORE_AND_FORWARD` - Asynchronous Time Independent Invocation (TII) is used. The request will be stored persistently by the router before attempting to forward it.

Routing policies will use these routing types according to what has been set in a `RoutingTypeRange` structure, which has minimum and maximum values of `RoutingType`.

- `RoutingPolicy` - This indicates whether the ORB will ensure message delivery through queuing or use direct delivery instead, depending on the `RoutingTypeRange`.
- `MaxHopsPolicy` - This indicates the maximum allowed number of routers by which messages will be transiting.

Queue Ordering: We have seen that under CORBA messaging, ORBs may be configured to act as if they were routers. Like routers, ORBs will have to face resource contention and treat messages according to certain strategies. Another well known mechanism to resolve this problem is to apply adequate queuing disciplines like , First-In First-out (FIFO) or Weighted Fair Queueing (WFQ) (see [11]). The `Messaging QueueOrderPolicy` offers the possibility to switch between FIFO and priority-based queueing disciplines. WFQ has yet to be considered. It is Important to say that routing has to be enabled, meaning that the routing policy has to be set at least to `ROUTE_FORWARD`. The ordering policies are:

- `ORDER_ANY` - The client does not care about ordering.
- `ORDER_TEMPORAL` - A FIFO policy. This is the default. In a running environment with sufficient resources, this is usually a good strategy.

- `ORDER_PRIORITY` - A priority-based policy. Messages are treated according to their assigned priority policies. This is a very simple way to implement differentiated services. However, the danger lies in the possibility of low-priority queue starvation if high-priority ones stay busy.
- `ORDER_DEADLINE` - Messages are treated according to their time-to-live properties. The shorter the time to live, the higher the priority.

4.2.3 Note on CORBA Messaging

CORBA Messaging is a very important step towards the realization of deterministic behaviour of DAs. The middleware's tendency to hide the objects' locations and their implementation as well as hiding underlying communication mechanisms impacts on a DA's behaviour, predictability and thus QoS offered to users. It is of prime importance for middleware to support QoS information propagation and to act as a facilitator in certain DA modes of operation. Therefore, achieving deterministic behaviour of a DA is the primary goal and Messaging is the proof that the OMG and its community recognize it.

Still in its early age, Messaging has yet to be implemented by ORB vendors (as of February 2001) and the OMG will not officially release CORBA 3 until it has been implemented and tested along with RTCORBA and minimumCORBA. A subset of Messaging is implemented in Borland Visibroker 4.1. More of this will be discussed in Chapter 5.

The next section concentrates on RTCORBA, which is the specification that applies to the problem of timeliness in DAs.

4.3 Real-Time CORBA

We have seen in Chapter 2 that accuracy and timeliness were the two crucial aspects of high-performance systems. Real-time systems belong to the class of applications that are carefully designed to meet stringent timeliness requirements. Medical systems, military command and control systems, manufacturing systems are examples of applications that need to work in time-constrained environments.

CORBA's Interface Definition Language (IDL) provides a way to describe object interfaces while hiding their locations and implementations. By doing so, CORBA also keeps objects from knowing how requests will be handled in a performance perspective. This greatly affects the behaviour predictability of the application and thus its QoS to the user. Recognizing this as a shortcoming of the specification, the OMG worked on key aspects by which an ORB would become a supporting technology to real-time distributed applications. Pioneering work in this domain has finally resulted in Real-Time CORBA as an extension to the original specification. Its goal is:

"...to support developers in meeting Real-Time requirements by facilitating the end-to-end predictability of activities in the system and by providing support for the management of resources." [22]

Table 4.2: Standard RTCORBA Policy Types

Policy Type	Policy Interface	Tag
PRIORITY_MODEL_POLICY_TYPE	RTCORBA::PriorityModelPolicy	40
THREADPOOL_POLICY_TYPE	RTCORBA::ThreadpoolPolicy	41
SERVER_PROTOCOL_POLICY_TYPE	RTCORBA::ServerProtocolPolicy	42
CLIENT_PROTOCOL_POLICY_TYPE	RTCORBA::ClientProtocolPolicy	43
PRIVATE_CONNECTION_POLICY_TYPE	RTCORBA::PrivateConnectionPolicy	44
PRIORITY_BANDED_CONNECTION_POLICY_TYPE	RTCORBA::PriorityBandedConnectionPolicy	45

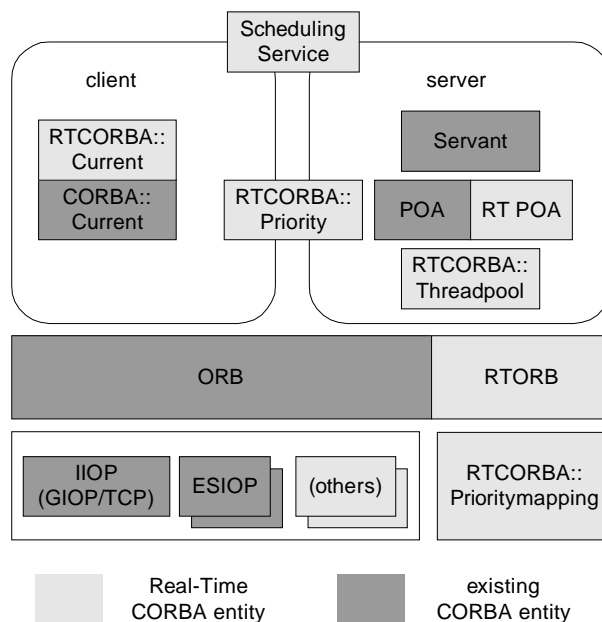


Figure 4.1: Real-Time CORBA Extensions

RTCORBA was fully integrated in CORBA 2.4.1 in October 2000, but was already available as a white paper in late 1999. RTCORBA is tightly coupled with Messaging as it uses its Policy propagation mechanism. It also provides additional Policy objects (see Table 4.2). This section describes RTCORBA special mechanisms.

4.3.1 RTCORBA Extensions

Figure 4.1 shows RTCORBA extensions to the classical CORBA architecture. Keeping in mind that the middleware sits between a distributed application and an operating system while interacting with the communications layer (in most cases TCP/IP through IIOIP), the ORB has to provide hooks, tools and facilities to support end-to-end predictability of overall distributed systems systems.

Real-Time ORB: The **RTCORBA::RTORB** provides extensions to an ORB object. It is not derived from **CORBA::ORB** per se though it is conceptually. An **RTORB** is obtained through an **ORB::resolve_initial_references** with the **RTORB** object id. Also, it is a locality-constrained object in the sense that a reference to such an object may not be used as a pa-

Table 4.3: RTCORBA Standard System Exceptions

System Exception	Minor Code	Description
MARSHAL	2	Attempt to marshal locality-constrained object
DATA_CONVERSION	1	Failure to PriorityMapping object
INITIALIZE	1	Priority range too restricted for ORB
BAD_INV_ORDER	1	Attempt to reassign priority
NO_RESOURCES	1	No connection for request's priority

parameter in an invocation, nor can it be stringified. An RTORB provides an environment in which processor resources, threads and communications can be managed. We will describe these mechanisms in the next few paragraphs. Special system exceptions are described in Table 4.3.

Real-Time POA: In Real-Time CORBA, the POA is narrowed to a `RTPortableServer::POA` type interface. This comes from the need for the POA to:

- Support server-side priority settings; and
- Understand Real-Time Policies.

Derived from the POA, it still supports all POA functionalities at the semantic level. It is obtainable from an `ORB::resolve_initial_references("RootPOA")` call and is locality-constrained.

4.3.2 Priority Mechanisms

Objects of distributed applications exchange messages. By doing so, they initiate flows of information and some of them are of greater importance. It is thus important for the middleware to respect these flow categories and collaborate with the operating system (OS) to support the overall distributed system. By the same token, the ORB will only be as supportive as the OS is. The RTCORBA specification states that the underlying OS should comply with IEEE POSIX 1003.1-1996 Real-Time Extensions in order to work properly. However, RTCORBA is not restricted to such RTOSs and use of others may work as well.

CORBA to Native Priority Mappings: As ORB implementations sit on top of different OSs and thus interact with different native priority mechanisms, RTCORBA ORBs have to present neutral priority schemes to objects and implement their own CORBA-to-native and native-to-CORBA translation mechanisms. Within RTCORBA, Priority values range from 0 (low) to 32767 (high). A vendor implementation will then develop a bidirectional priority mapping scheme depending on the underlying RTOS. Figure 4.2 shows the relationship between 2 ORB endsystems and the CORBA Priority scheme.

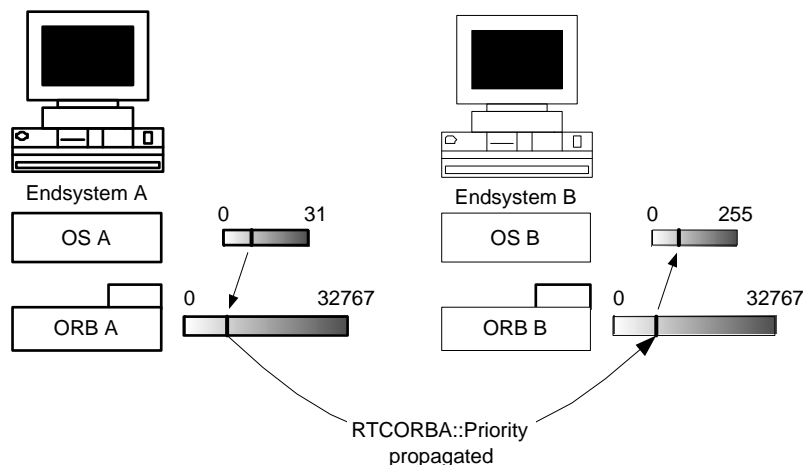


Figure 4.2: RTCORBA Priority Mapping

4.3.3 Priority Models

In distributed applications, the question as to who sets priorities becomes the concern of application developers. In server-centric applications (e.g. a major web server hub), resource management is the key factor to ensure QoS to users. Therefore, priorities should be set at server-side for clients to act according to remaining server resources. However, in other types of applications like C2ISs in which clients are likely to have high-priority requests, (e.g. reporting a missile launch), a DA should rather act on propagated client priorities. RTCORBA supports both server-declared and client-propagated priority models. The model is selected through a `PriorityModelPolicy` object and passed along with the creation of a POA.

Client-Propagated Priority Model: This model is useful when clients in DAs are responsible for specifying their priority requirements. This will be realized through the service context list that is passed along with the GIOP request. This model is enforced when the target object supports the `CLIENT_PROPAGATED` value of the `PriorityModelPolicy` of its associated POA. Furthermore, in the case of a bidirectional call, the reply is treated with the same priority. The same priority also applies to onward invocations (subsequent CORBA calls to other objects by the servant code). Figure 4.3 illustrates the client-propagated priority model.

Server-Declared Priority Model: In this model, client objects execute at a server-declared priority. The priority value is embedded within the Interoperable Object Reference (IOR) and is published to the client as a tagged component, using the propagation mechanism of Messaging. Although a server-declared priority is set at the time of the POA creation, it is possible to override priorities on a per-object basis, which increases flexibility. Figure 4.4 illustrates this priority model.

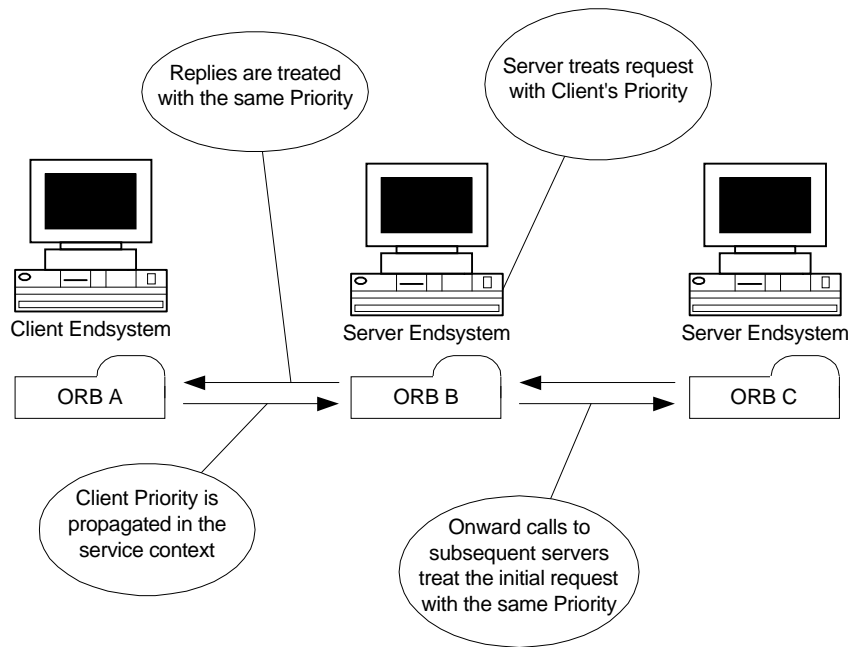


Figure 4.3: Client-Propagated Priority Model

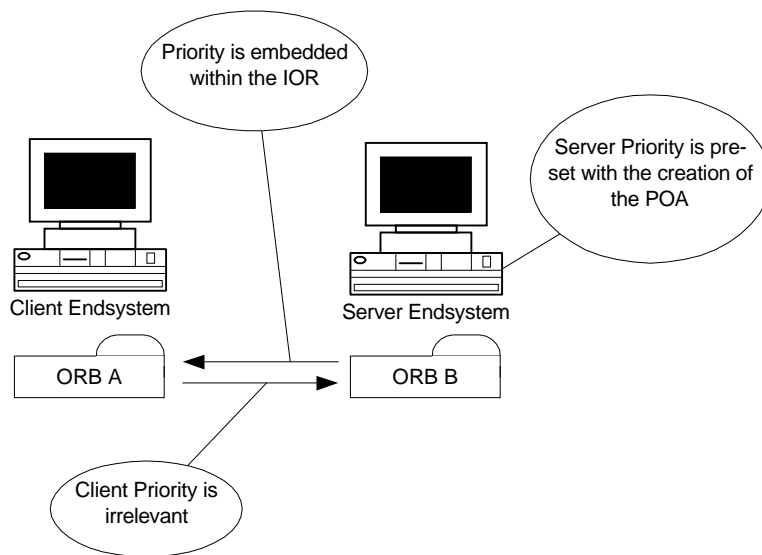


Figure 4.4: Server-Declared Priority Model

Priority Transforms: RTCORBA also provides for user-defined priority mechanisms. Priority Transforms modify the priority of invocations on target objects according to a certain user-defined strategy. Priority Transforms can be realized in two ways. The first one, called an *inbound priority transform* is applied during an invocation upcall, but before the actual servant code is executed. The second one is called *outbound priority transform* and is a transformation that applies when an onward call is invoked. Onward calls occur when target objects relay messages to other servers.

4.3.4 Threadpools

As threads within the server ORB may present similar profiles (priority, CPU need, type of requests, etc.), it makes sense to group them together since it frees unneeded resources. Threadpools is an RTCORBA mechanism that allow a server ORB to regroup several threads together. Moreover, threadpools can be configured to accept buffered requests (to a maximum number of waiting requests) which might give a distinct advantage in realizing QoS-enabled DAs. RTCORBA permits the creation of two types of threadpools: With or without lanes. Both models support the creation of threadpools containing a predetermined number of static threads and a number of dynamic threads created on demand. When both maximum number of static and dynamic threads are reached, further incoming requests are buffered if the boolean parameter `allow_request_buffering` is set to "true". If not, a `NO_RESOURCES` System exception is issued. Threadpools can be associated with multiple POAs.

Threadpools Without Lanes In this model, a default priority is associated with the static threads. Dynamic threads may be given specific priorities.

Threadpools With Lanes This threadpool model acts the same way as the previous one although it is augmented with a sequence of `ThreadpoolLane` objects. A `ThreadpoolLane` acts as a container of both static and dynamic threads. In this case, the priority is set for all by the `lane_priority`. The real utility of threadpools with lanes comes when the `allow_borrowing` boolean parameter is set to "true". From thereon, a lane that exhausts all its threads (both static and dynamic) can borrow threads from lower-priority lanes. The borrowed threads have their own priorities raised to that of the borrowing lane. When the task is finished, the borrowed thread is returned and its priority is lowered to its original value. Figure 4.5 illustrates the different uses of threadpools.

4.3.5 Synchronization Mechanisms

Under RTCORBA, the ORB becomes a multithreaded environment. The ORB's resources may become an object of contention. To resolve this problem, RTCORBA defines a set of locality-constrained operations, contained within the `Mutex` interface. A `Mutex` can be either locked or unlocked. and is unlocked at the time of its creation. `Mutexes` are used at different levels within the ORB as shown in Figure 4.6 and serve as locks to the ORB's resources.

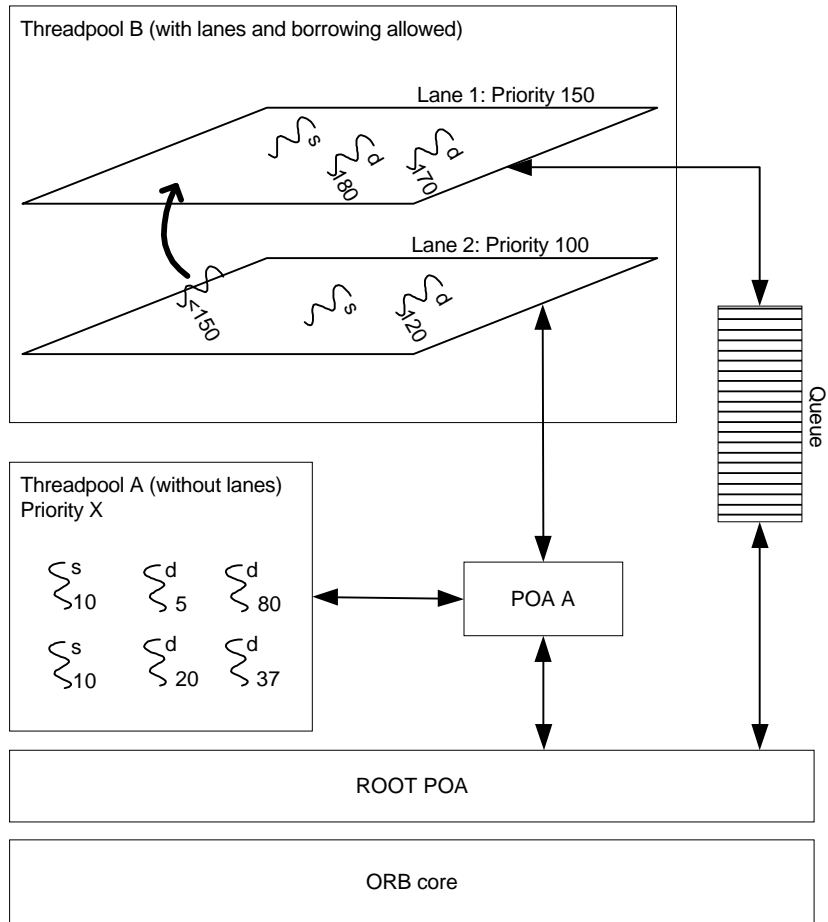


Figure 4.5: Threadpools Use Cases

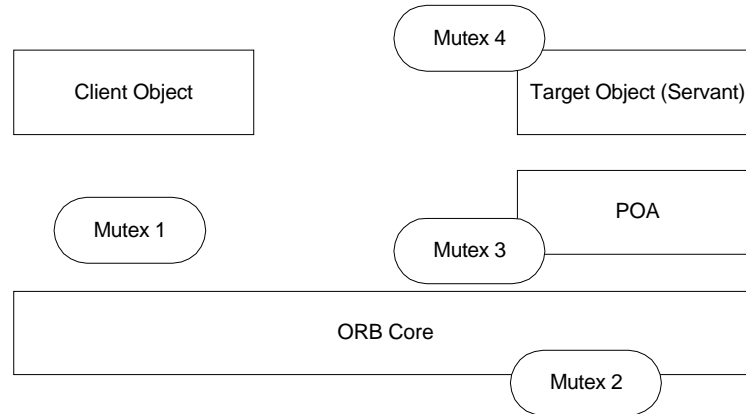


Figure 4.6: Mutexes

Mutexes are also subject to higher-priority thread preempting to avoid priority inversion. Therefore, mutexes inherit the priorities of the threads that invoke them.

On another note, RTCORBA does not define further timing policies than those that are defined in Messaging (section 4.2.2). They were judged to be exhaustive and complete.

4.3.6 Protocol Selection and Configuration

In order to achieve end-to-end QoS in a DA and also to ensure middleware transparency, it is important for the ORB to give the application some way to tune the underlying communications interface. RTCORBA does this by defining two policies: One server-side and one client-side. Each policy, server and client, has two facets as far as protocols are concerned: ORB protocols and transport protocols. A complete protocol configuration under RTCORBA consists of pairs of ORB and transport protocols. For example, a valid pair would be GIOP and TCP, which is in effect IOP (currently the only implemented approach in most ORBs). This approach leaves room for more specialized protocols, but also permits their configuration from an application's viewpoint.

Server-side Protocol Policy: The `ServerProtocolPolicy` is enforced when the POA is created. The protocols listed in the policy are in order of preference, and if one of them is not valid, an `INV_POLICY` exception is raised. Server-side configuration is used for all connections from clients using that particular POA.

Client-side Protocol Properties: The purpose of a `ClientProtocolPolicy` is to signal a target object with a list of preferred protocols (and their particular configuration) at bind time. If none of the protocols listed are available on the client ORB, an `INV_POLICY` exception is raised. If one of the protocols is available but no connection can be established, then a `COMM_FAILURE` exception is raised.

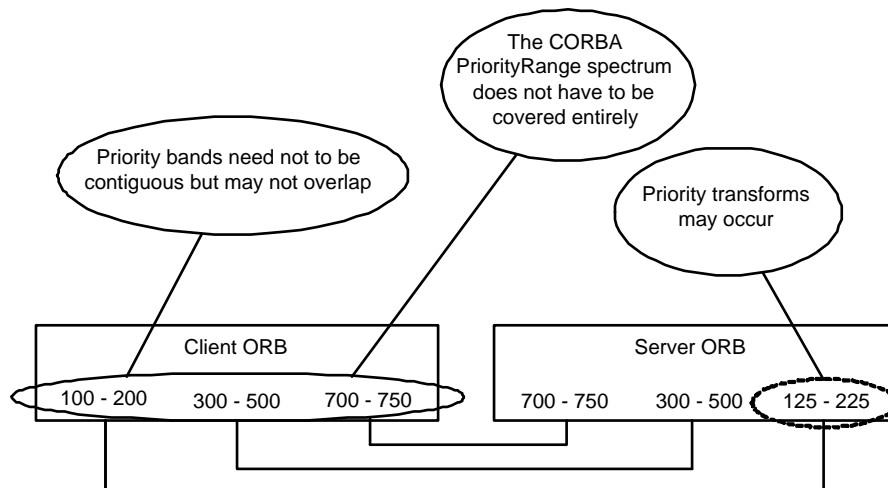


Figure 4.7: PriorityBand Connections

4.3.7 Explicit Binding

CORBA 2.3 and older versions supported only implicit binding on target objects. This approach presents clear advantages like location transparency and deferring communications resource allocations until needed. However, this model influences greatly delay and jitter within a DA and thus may severely impinge upon its predictability. Therefore, an explicit binding mechanism has been introduced in RTCORBA. The `CORBA::Object::validate_connection` forces the ORB to make the connection to the target object and open a communication channel. How that should be implemented is not specified though.

Another shortcoming of earlier versions of CORBA is that there was no control over the communications channel. More often than not, requests in these models are multiplexed, leaving a great risk of priority inversion resulting from the use of a FIFO queue. To avoid this, RTCORBA defines two communication policies that will support effective explicit binding: Priority-banded and private connections.

Priority-banded Connections: Priority-banded connections are communications channels that will tunnel requests with priorities that fall under the ranges of specific `PriorityBand` objects as depicted in Figure 4.7. This ensures that requests of the same priority range will be multiplexed on the same channel. This greatly reduces priority inversion, but does not necessarily address jitter problems effectively depending on queue lengths.

Private Connections: Some threads may not suffer delay or jitter. When this is the case, it is possible within RTCORBA to set up a private connection that will prevent the ORB multiplexing the traffic's thread with other traffic. This is done through the set up of a `PrivateConnectionPolicy` object.

4.3.8 Scheduling Service

Managing the ORB's real-time resources and mechanisms to meet QoS requirements is a difficult task. Also, there might not be a single set of parameter settings that achieves a QoS goal but it is likely that one set will maximize the result while minimizing the resources needed to achieve it. However, knowing what set of parameters is optimal is not necessarily the application developer's specialty (or what he is paid for). Therefore, a meta-mechanism is needed to organize the general purpose real-time tools of RTCORBA. These meta-mechanisms will reflect state-of-the-art knowledge on each aspect of QoS and will organize low-level real-time mechanisms to achieve the optimal response. RTCORBA has defined the Scheduling Service to meet this requirement. ORB implementations need not consider this part of the specification to be RTCORBA compliant. Moreover, it is somewhat imprecise on how ORB vendors should implement it. The important lesson that should arise from it is that applications, RTORBs, and RTOSs' resources and activities have to be coordinated if QoS is sought at the user level.

4.3.9 Observations on Real-Time CORBA

The OMG summarizes very well how performance requirements can be met in a DA (or in any other system for that matter).

"Deterministic behavior of the components of a Real-time system promotes the predictability of the overall system. In order to decide a priori if a Real-time requirement is met, the system must behave predictably. This can only happen if all the parts of the system behave deterministically and also if they "combine" predictably" [22]

A distributed application is dependent on 4 major building blocks:

- The Application's Components;
- The Middleware Layer;
- The Operating System; and
- The Communications Layer.

In a real-time or QoS-enabled context, each of these components is determinant in achieving user-level QoS. The overall application will only be as good as the weakest of these blocks. We have seen in this section that RTCORBA provides the application developer with interfaces that will help in tuning several aspects of the ORB's behaviour, the OS and even the communications layer. RTCORBA will continue to provide implementation transparency but not to the extent of blindfolding the objects composing the DA. Not only does RTCORBA provide additional facilities to QoS-aware applications, but it also promotes further research and refinement in our understanding of the real-time applications class, as it offers a testing platform for new ideas. CORBA played this role already by allowing heterogeneous, non-located objects to interact. Hopefully, it will do the same for real-time DAs.

However, RTCORBA has yet to be implemented by several ORB vendors to really prove its robustness as a specification. It surely cannot be stated as stable yet, were it only for the scheduling service.

Chapter 5

A Methodology to Manage Performance in C2ISs

5.1 Introduction

Previous chapters addressed different problems of today's practices regarding the performance of military information systems. However, keeping these in mind is not enough to design efficient systems. There is still a lack of integration. How do we instrument the system? How do we use the concept of QoS so that it permeates the system? How do we design the system architecture so that client-middleware-server collaboration is promoted and optimized?

This chapter is concerned with the elaboration of a methodology that will enable the construction of military distributed applications that are sensitive to performance rendering (QoS-aware). The goal is not to devise a methodology that should stand on its own, but more in the line of a phase loop that will enrich a more global approach to information systems design like the Rational Unified Process (RUP) or the Reference Model - Open Distributed Process (RM-ODP) (see [23]). The notion of performance management of distributed applications is indeed captured within the concepts of these more encompassing methodologies, so it is logical to refine what has been stated as being part of the whole picture. Embedding this methodology into a recognized one bears two advantages: First, it helps us concentrate on the role of architecting QoS for military applications and nothing else. Second, the methodology developed is given more survivability as it addresses issues that RUP or RM-ODP only address superficially. While the methodological background will be close to the above mentioned, the Unified Modeling Language (UML)¹ [25] will be the means by which QoS concepts will be applied and articulated.

¹In March 2002, the OMG released a Final Adopted Specification on "UML profile for schedulability, performance, and time specification" [24]. This profile uses the concept of "action execution" which is not part of UML 1.4 yet. Concepts of performance modelling in this chapter are aligned but not fully compliant to those of this UML profile for that reason.

5.2 The Methodology

QoS analysis and modelling aims at identifying and applying mechanisms that will make an application sensitive to performance rendering. Again, QoS is:

"... an encompassing term for the collection of activities, management functions and strategies that aim at guaranteeing the end-to-end, predictable and consistent behaviour of network-dependent applications"

Aspects of QoS-enabled models are tuned in such a way that systems implemented according to them exhibit the qualities enumerated in our working definition of QoS.

The high-level steps of the methodology are:

- Determining the system's operational environment;
- QoS Static Analysis;
- QoS Dynamic Analysis;
- QoS Modelling; and
- QoS Implementation and Deployment.

5.2.1 Determining the System's Operational Environment

Software systems, as powerful as they may be, will only work well within the environment in which they were meant to be used. Military information systems, C2ISs, are no exception. Search and rescue (SAR) operations, reconnaissance, target acquisition, and planning are all but a few examples of different operational contexts within which C2ISs evolve. Although one system may address several of these operating contexts, it remains that it will likely underperform for operating contexts other than the ones it was designed for. The first step of the methodology is to define the operating environment of the C2IS. This step will narrow down the possibilities in terms of performance requirements and help in identifying the critical cases that need to be addressed to ensure performance. It will also help in identifying the first few objects needed to get started in analysis and modelling. The objects that will compose the performance toolkit for the application are instantiations of the concepts enumerated in earlier chapters: Instruments to take MoPs, objects to support QoS architectures, delegates to convey QoS information, and so on. This step is the one where a certain vocabulary of performance is defined and where the critical business workflows that need performance output are identified. Figure 5.1 shows a diagram of the decision-making business process² observed by the Canadian Land Forces [26].

5.2.2 QoS Static Analysis

Military actors add their own flavour to performance expectations as they use their own performance language. The QoS static analysis brings into consideration how measures of merit will be tied together and to the system model and attached to the user context.

²This process is nothing more than a specialization of the OODA loop described in Chapter 1.

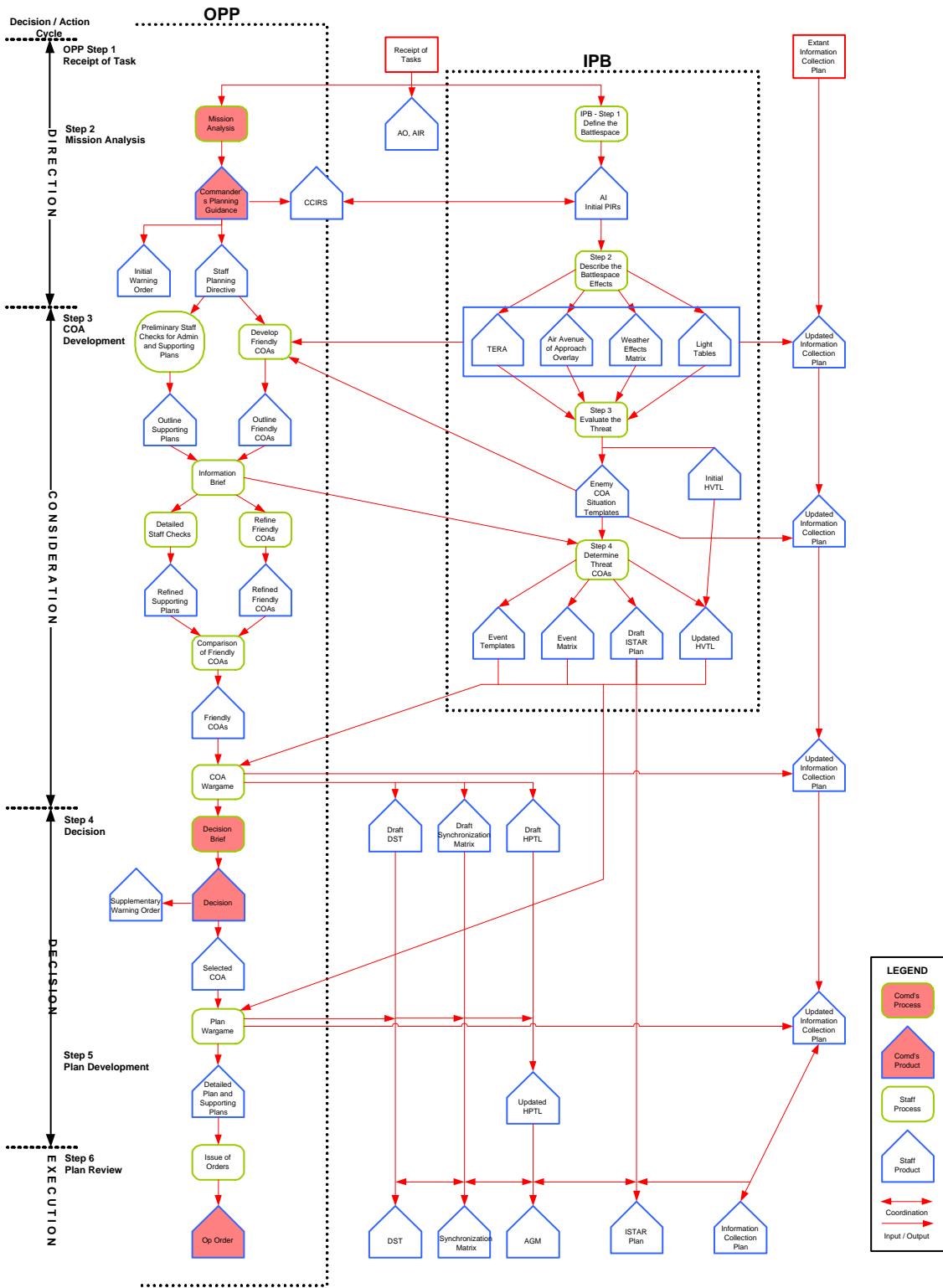


Figure 5.1: The Decision-Making Process

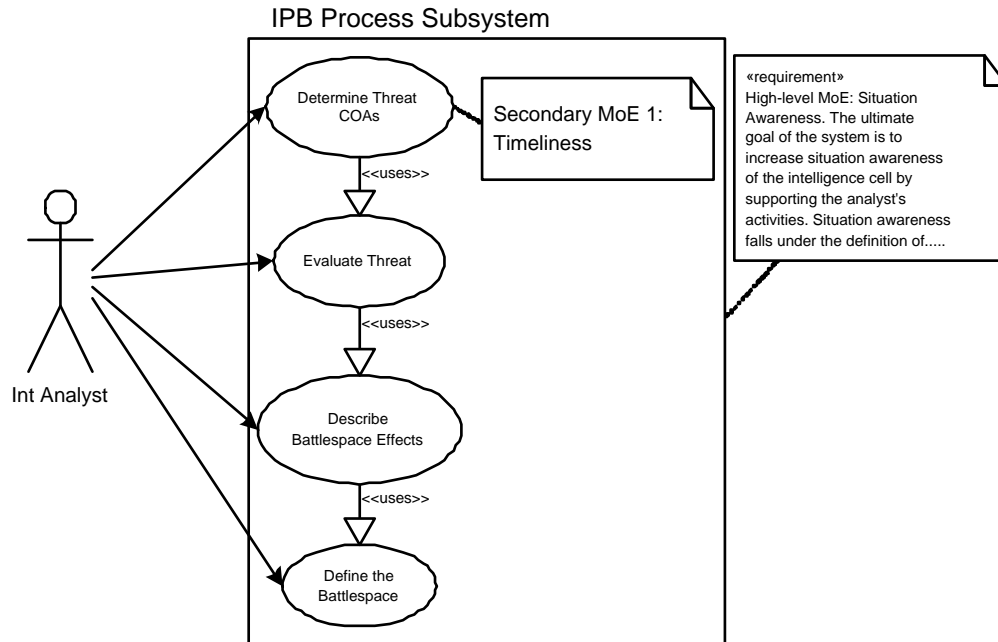


Figure 5.2: Identifying High-level and Specific MoEs of the System

5.2.2.1 Use Case Analysis

Once the business process is known and identified, it is time to see where and how an information system could be of any help. The use case analysis serves this purpose. In UML, use case diagrams are used to picture the system, actors who interact with it and use cases that represent the different utilities of the system towards actors. This constitutes the first entry point for QoS analysis and modelling. Each use case is meant to be realized in such a way that the actor interacting with the system is satisfied in terms of perceived performance, within the context of the appropriate business workflow.

5.2.2.2 Identifying the High-level MoE

The NATO Code of Best Practice for C2 Assessment [6] prescribes an approach where a high-level MoE should be identified, therefore giving a direction to experimentation design. The same is true for systems design. Choosing a high-level MoE metric will focus the analysis and modelling efforts and increase the chances of obtaining a system that will promote good rendering of performance expectations. Figure 5.2 shows a use case diagram representing a system that would come in support of the Intelligence Preparation of the Battle (IPB) process (part of the decision-making cycle process). Attached to the system box is a high-level MoE. This metric is the one that would describe the needed overall performance metric of the system in supporting this part of the process.

5.2.2.3 Attaching Specific MoEs to Use Cases

Since a use case represents a certain utility of the system toward the user/actor, it is then natural to attach a performance requirement tag to each use case. Use cases are the anchors to set user performance expectations, or QoS requirements. Also, these requirements must be expressed in terms of the domain addressed by the system. In the military realm, the user-level QoS requirements are expressed in terms of MoEs (see Chapter 2).

The object of use case diagrams in UML is on the one hand to capture functional requirements and on the other hand to serve as contracts between system developers and users (and other stakeholders). Both parties agree on the functional requirements. Making QoS requirements an explicit part of use case diagrams helps users to gain a fair understanding of what the system will offer in terms of performance and also help developers to really understand the semantics of performance expectations expressed by users. Figure 5.2 shows some MoEs attached to use cases. It is important to know that each of the specific MoEs comes in support to the high-level MoE.

5.2.2.4 Defining QoS Objects

As the definition of the business domain and problem formulation addressed by the eventual system progresses, verbs, nouns and adjectives pop up in the mind of the software engineer (At least they should!). These constitute the first classes of objects that will be needed to design the system. They will be captured in class diagrams. Actors and use cases also help to populate these diagrams. A QoS architect will do the same with a focus on performance objects. Classes of objects pertinent to QoS will fall more or less within certain categories.

QoS Control Objects coordinate the activity of other objects so that performance requirements are met. Often, they are used as synchronization points and sequencers. These objects should follow nicely the "gang of four" (GoF) "mediator" design pattern as objects that orchestrate the global behaviour of other objects. [27].

QoS Boundary Objects are placed at the significant borders of the system like the network-to-middleware boundary, the application-to-middleware boundary and others. They act as watchdogs and trigger events when required. A GoF "façade" design pattern would be the way to represent them as their role is to coordinate performance-related activities in one responsible object. QoS boundary objects may be viewed as specialized control objects but with a focus on a precise part of the system. They can decide on actions to be taken or defer their authority to more global control objects.

Performance Information Carrier (PIC) Objects are objects that convey information related to performance or QoS requirements between two objects. They can be used to relay QoS or performance information between different levels within the distributed application. For example, an application-level object could use a PIC object to state information on its agreed level of service. The PIC object would then be conveyed to other levels in the system. A PIC object is a state object so it should be constructed with appropriate access methods (get/set methods) and be serializable as prescribed by software construction best practices.

QoS Policy Objects act as contracts between entities within the system. They hold information about service level agreements that have taken place between objects or levels of a distributed application.

Measurement Objects can be used to hold information about values of certain metrics within the system. These values quantify dimensional parameters (cf. Chapter 2) and determine MoPs. In turn, these MoPs determine MoEs such that the state of the system in support to the actors is known.

Of course, it is not mandatory to follow closely this taxonomy to perform an adequate QoS static analysis. However, having it in mind may help the software architect apply better judgment to his performance design.

5.2.3 QoS Dynamic Analysis

After a first stab at defining the collection of objects classes that will be needed to realize the use cases that come in support to the business process, the software architect needs to express the relationships between the objects in such a way that performance constraints are taken into account. Sequence, collaboration and state diagrams come out of this exercise.

5.2.3.1 Identifying Relevant Scenario Walk-throughs

Although the system's *raison d'être* is to help realize use cases, there are different environmental conditions under which use case realization will be conducted. Some of these conditions will affect the system's performance. Fluctuations in the network bandwidth, non-availability of components, etc. may impair the system's ability to perform well. As best as possible, the system should have provisions for reacting to such situations. This implies that these situations be captured in the system's design so that mechanisms are triggered whenever a certain scenario affecting performance unfolds. Of course one cannot list exhaustively every possible situation, but nevertheless one can identify the most likely to happen, especially those that occur periodically. The architect captures these significant situations in activity diagrams. Each diagram depicts how objects (coming from the static analysis) collaborate so that performance expectations are met. In different situations, collaboration between objects may be tuned in such a way that use case performance rendering is the least affected. As an example, let's say that critical use cases within a system are played through 2 types of scenario walk-throughs: A nominal mode of operation and an impaired mode of operation. The impaired mode of operation scenario may have to be represented by more than two scenario walk-throughs. This depends on how many situations the architect wants to address. Figures 5.3, 5.4 and 5.5 explicit entity collaborations so that three situations are covered: The first being the normal mode of operation and the others representing two different impaired modes of operation.

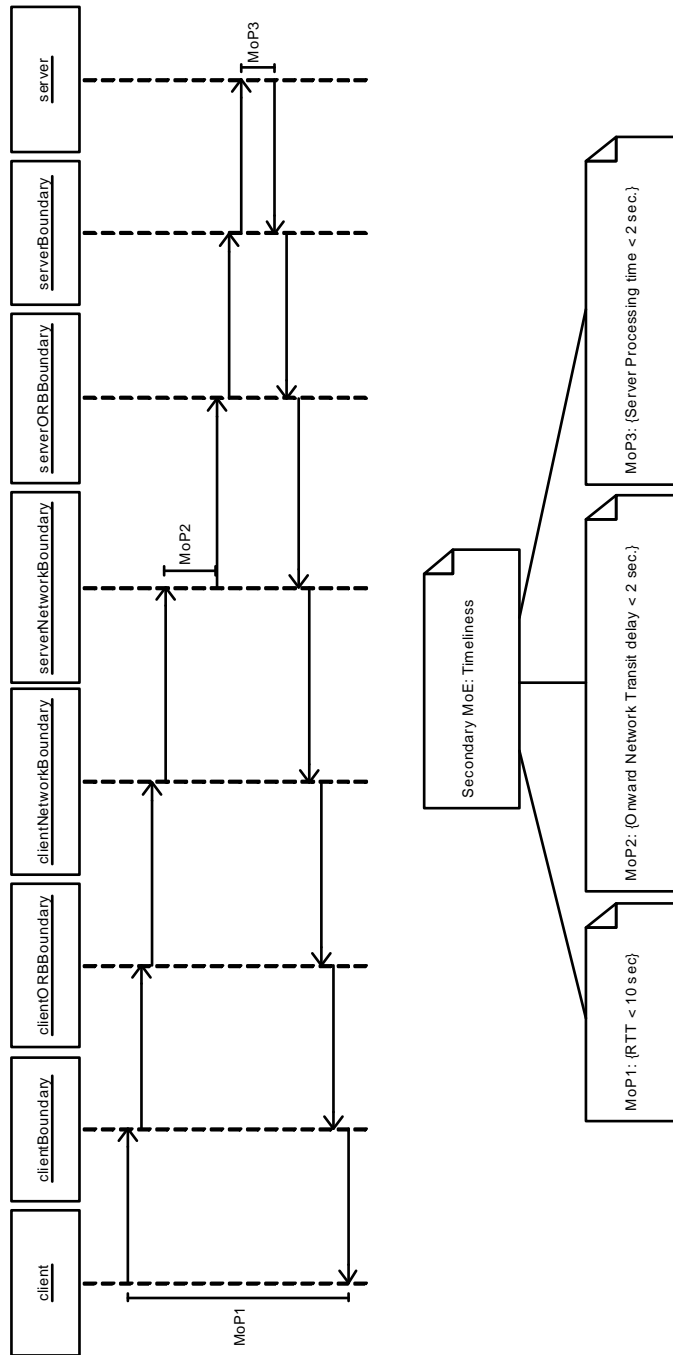


Figure 5.3: Nominal Mode Scenario

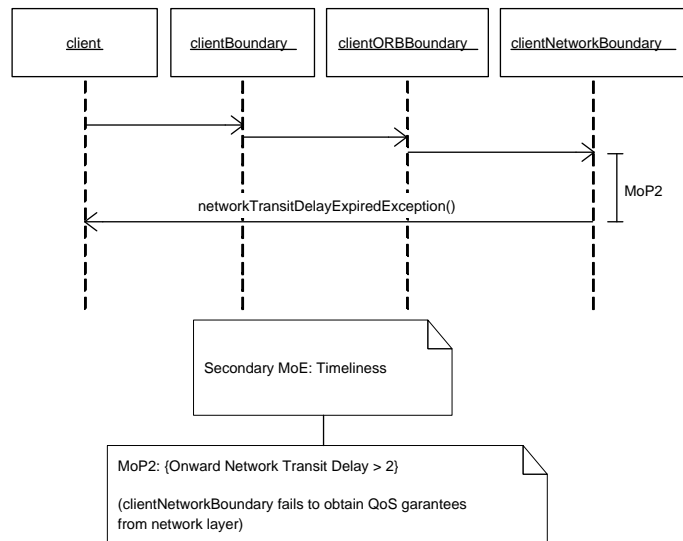


Figure 5.4: Network Degraded Mode Scenario

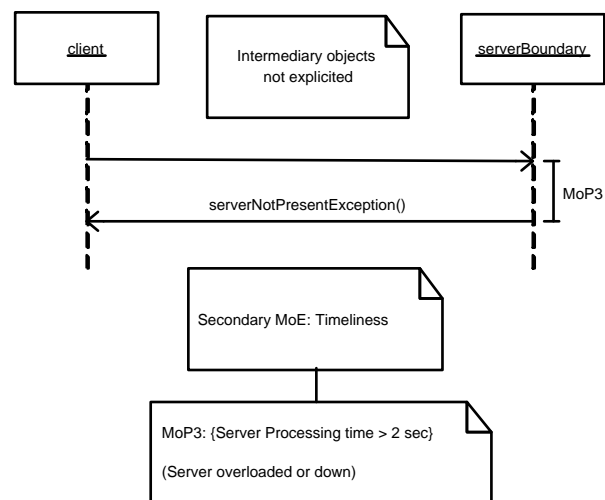


Figure 5.5: Server Down Mode Scenario

5.2.3.2 Instrumenting the System with MoPs

The need for the system to adapt to changes in its environment prompts the need for instruments within the system that will trigger the switching between different modes of operation. From the architect's viewpoint, this means that measurement objects representing MoPs (Chapter 2) have to be placed at strategic points so that an environmental shift triggers the proper behaviour of the application. The architect has to choose the MoPs so that a change of behaviour of the application is consistent with the MoE attached to the use case it realizes. This can be made explicit in sequence diagrams like shown in Figures 5.3, 5.4 and 5.5. In these examples, MoP1, MoP2 and MoP3 are chosen to act as triggers and are considered to be the major contributors to the upper-level MoE which is timeliness. Plainly speaking, when a network failure is detected, the system is to notify the client through a `networkTransitDelayExpiredException()` callback function. In the case where the server is too slow or down, the system is to notify the client through a `serverNotPresentException()` callback function. For each scenario walk-through, the set of metrics with their desired levels constitute as many QoS contexts, as defined in Chapter 3.

5.2.4 QoS Modelling

Static and dynamic analyses of the system give the software architect a sense of how objects within the system should interact so that user goals are achieved. At this point of the software design, the architect should have a fairly detailed blueprint of an instrumented system, that is, the foundation of a QoS-aware system. To proceed further, the architect needs to use what are known to be the best QoS practices and developments. Chapters 3 and 4 introduced in a non-exhaustive manner the building blocks that enable the building of QoS-aware systems. However, these building blocks do not prescribe how they should be used to achieve QoS-aware systems. Often at this stage, software architecture becomes more art than science. That will always be true, but one should try to be a pragmatic artist! QoS modelling starts here, when there's a good understanding of the static aspects and dynamic behaviours of the system, and the necessity to recognize the rightful and clever way to use state-of-the-art QoS concepts arises. The goal of this section is to demonstrate some of the tools and patterns related to QoS through appropriate UML diagrams. As the UML profile schedulability, performance, and time specification ([24]) is being defined so that semantics and best modelling techniques are taken into account, there is a need to demonstrate how QoS concepts described in this thesis can be applied to meet our goal. QoS modelling impacts every level of the design process: The business process, the use cases and the object interactions.

5.2.4.1 Impact on the Business Process

What is generally understood about the impact of IT on the way business is done today, ideally speaking, is that the job is done faster and better. Unfortunately, information systems are often designed against ways of doing business without IT and without considering how it should be done with IT. The military decision-making process (Figure 5.1) is appropriate when the only tools available are paper maps and other basic tools. Should this be the

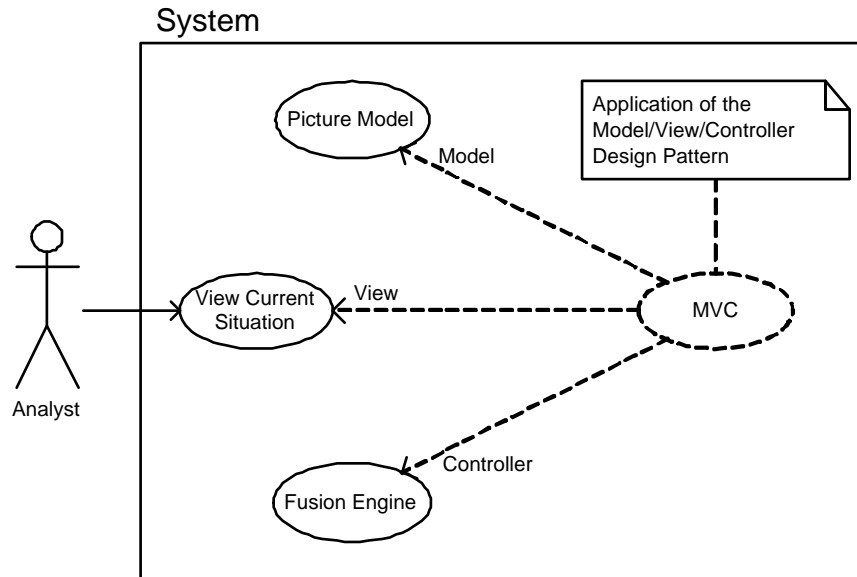


Figure 5.6: Applying Design Patterns

business model upon which a QoS-aware system will be designed? Unlikely! Recognizing this, the Canadian land forces are actually working on an evolution of the decision-making process which is based on a workspace-centric approach. Digging into it is out of the scope of this paper, but suffice it to say that the development of this business model is prompted by the knowledge of current possibilities offered by IT business domain. From a QoS architect perspective, a business process should be looked upon not only for what it is but for what it can become.

5.2.4.2 Impact on the Use Cases

Use cases are units of functionalities that address user needs. The system itself constitutes an integration of these functionalities. One can then easily understand that this integration must be realized in an harmonious way so that the overall goal of the system (expressed by the high-level MoE) is achieved. This integration is expressed with the UML concept of collaboration. Figure 5.6 shows an example in which the realization of three use cases, independent from one another in a first stab at design, should in the end be orchestrated through a Model-View-Controller type of collaboration. In software engineering, recognizing and applying design patterns is a natural way to implement what is known to be the best practice in software design. Usually applied at the object level, using it at the use case level enables the QoS architect to focus on important aspects of performance rendering to the user. It also implies that this particular pattern should be well instrumented so that the use case realization is observed and controlled.

5.2.4.3 Impact on Object Interactions

A first run at describing object interactions in support to use case realization has been done in the dynamic analysis. This run has served the purpose of placing QoS measure instruments.

However, the resulting diagrams do not reflect the reality of what is in the field with QoS tools and strategies. QoS modelling at the object level aims at tying real mechanisms to the theoretical specialized objects (control, boundary, policy, measurement objects and so on) identified in the first run. This is where modelling purists usually have a hard time. The reality of system development is that it will not stay on blueprints or in models. It will be implemented with real technology coming along with its strengths and weaknesses. Choosing a technology to implement a system has an impact on the models and the software architect should reflect this.

In this work, we have considered the two main QoS Architectures IntServ/RSVP and DiffServ. These architectures are such that choosing one over another has a significant impact on the object interactions. As we have seen in Chapter 3, IntServ/RSVP has a traffic-shaping strategy coupled with persistent connections between client and server objects while DiffServ relies on the definition of regions of constant QoS coupled with negotiation schemes between ingress and egress nodes.

The nominal mode scenario walk-through that left us with Figure 5.3 would have to be reshaped to take into consideration the prevalent QoS architecture. Figure 5.7 shows this if IntServ/RSVP were the chosen QoS architecture. The figure has been simplified so that application-to-network interactions are highlighted. Note also that MoPs are kept explicit. A similar exercise would have to be conducted if DiffServ were prevalent or for a combination of the two architectures (Figure 3.8).

The choice of middleware architecture also has an impact on object interactions. CORBA, as a specification, is very explicit on the application-to-middleware interactions, as we have seen in Chapter 4. Other middleware architectures exist (e.g. DCOM, Web Services, etc.) but none are as formal as CORBA for capturing performance aspects. Figure 5.3 has already been adapted from its "pure" form to one that highlights the QoS architecture. The same exercise has to be done to reflect the particularities of the middleware.

The role of the middleware is to negotiate service rendering between client and server objects. The OMG has extended this role through RTCORBA and CORBA Messaging to encompass QoS rendering and negotiation aspects. The `clientORBBoundary` and `serverORBBoundary` objects act as façade objects ([27]) that abstract out the middleware complexity. Zooming into this abstraction let us discover another sequence diagram (Figure 5.8) that specifies the underlying mechanisms of Client-ORB-Server interactions. Network interactions, not being the focus of this diagram, are abstracted out. They might lead to another sequence diagram focused on their aspects. The single most important aspect of this diagram is that relevant MoPs are present, and translated to this context. One can state that appropriate dimensional parameters (ref. Chapter 2) are identified and applied against MoPs, and that would be correct. At this level, MoPs can be measured quantitatively, and have almost reached the lowest necessary level of granularity. For the system to respect QoS requirements through these measures is consistent with what would be perceived at the user level as an efficient system, because this is the consequence of a top-down definition approach.

As we have seen in this section, QoS modelling is the exercise of adding the right level of detail into the first-run models so that actual QoS state-of-the-art principles are adequately represented. This may have a significant impact on the overall design, as some aspects may or may not be implementable. This reality is even more true for the next section on

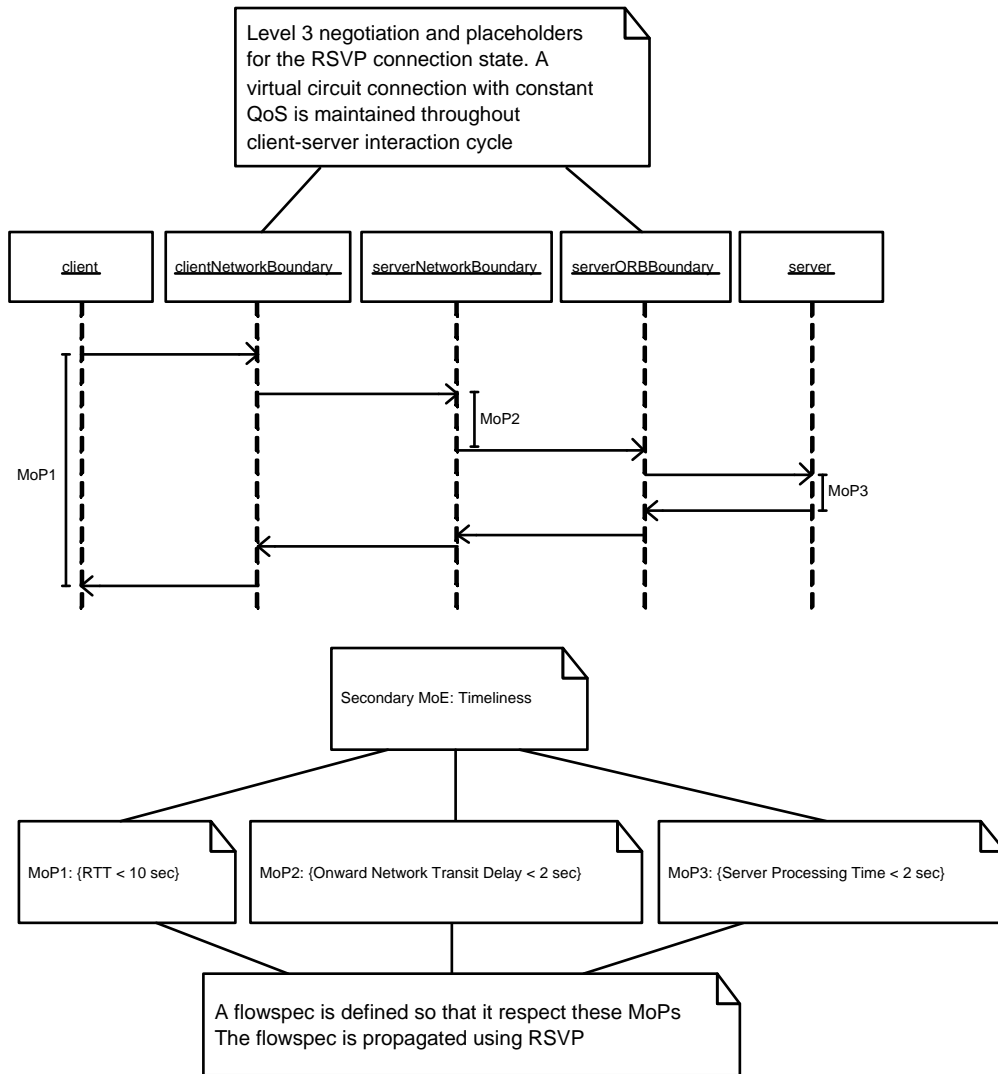


Figure 5.7: IntServ QoS Modelling

implementation and deployment.

5.2.5 QoS Implementation and Deployment

As the exercise of modelling is one of abstraction, there is an inevitable collision with real implementations. The CORBA 3.0 specification has yet to find a true and complete instantiation with ORB vendors. More often than not, ORB vendors implement only parts of the specification, just enough so they can claim to be CORBA compliant. Even when they fully implement the standard, they may take some liberties with the interpretation of certain aspects of the specification. The same holds for QoS architectures. It is most likely that particular implementations will have their own specialized set of tools and architectural components offered to the developer. What burden is therefore imposed on the shoulders of the architect? Most likely the one of knowing the targeted implementation well enough so that object definitions (in their nature and interactions) are aligned with this reality. UML component and deployment diagrams serve this purpose to some extent as depicted in Figure 5.9 and 5.10. Components act as logical containers for the system functionalities and are deployed on certain nodes that can be specified to support them.

5.2.6 Observations on a Performance Methodology

The actual development of QoS tools and strategies is still in its infancy. Implementation of QoS solutions into today's systems cannot be done in ad hoc fashion nor can they be added after the system is built. It is capital to integrate performance aspects of systems at the design stage. Best software development practices prescribe the use of formal models to achieve a shared understanding of performance expectations among software architects, developers, users and managers. UML, as a formal specification for system modelling, is the best and foremost candidate for this task. Through more encompassing methodologies like RUP or RM-ODP, architecting performance seems to become an achievable goal. However, there is a point at which multiplying the number of models (consider all the sequence diagrams we had to build to describe a simple use case realization scenario) makes the aim of modelling collapse on itself. The software architect must carefully keep the balance between the necessity of making system aspects explicit and of keeping simplicity as the prime directive. Indeed, if the model becomes more complicated than the system, why bother? A model is useful when it is simpler than reality, but still offers a good way of representing it.

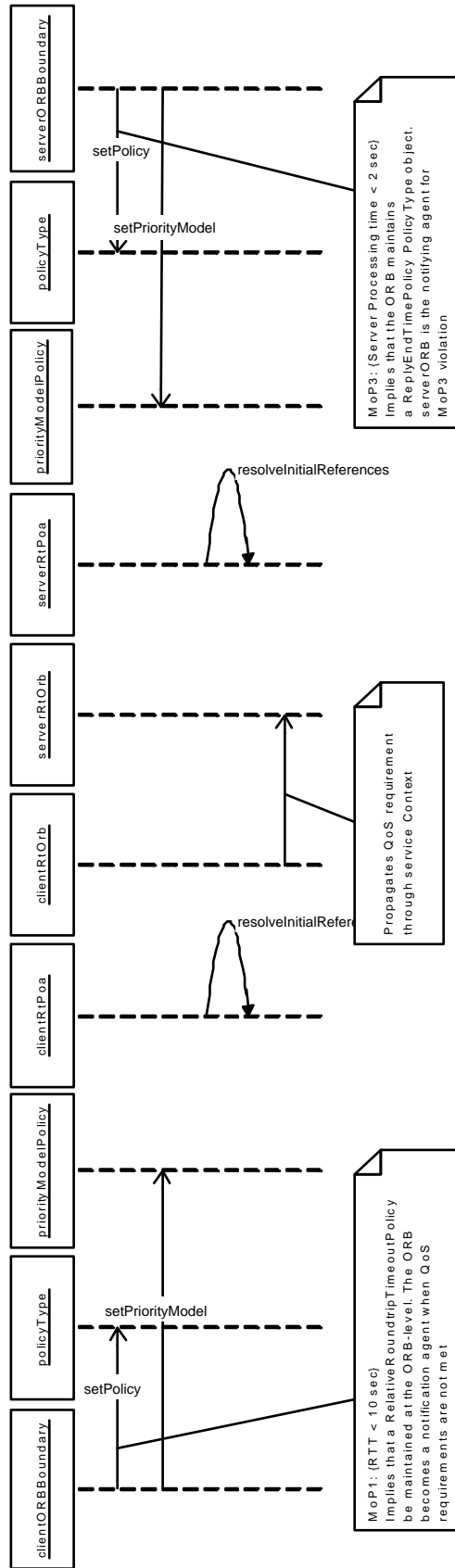


Figure 5.8: CORBA QoS Modelling

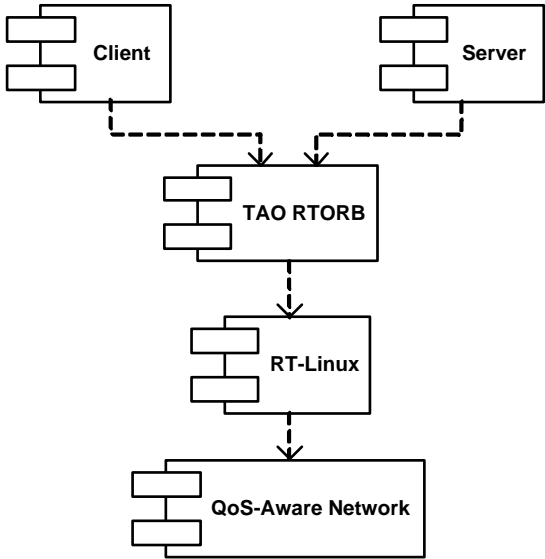


Figure 5.9: Component Model View

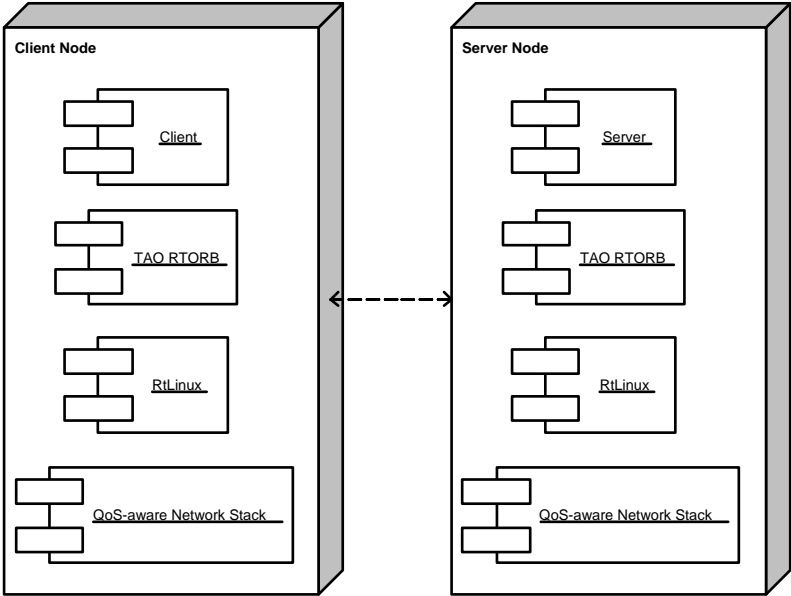


Figure 5.10: Deployment Model View

Chapter 6

Conclusions

This chapter is a wrap-up of the concepts that were stated in this thesis. It also aims at identifying some of the future work that would naturally unfold from this effort.

6.1 Performance of Military Information Systems

The problem of performance of military information systems and more particularly C2ISs is linked to the human decision-making process. An efficient C2IS is one that supports this process. The model chosen to represent this process was the Observe-Orient-Decide-Act (OODA) loop. An information system that would make the OODA loop spin faster would improve the commander's decision making, and therefore be perceived as an efficient system.

This general statement of the performance problem has to be broken down to narrower and more focused problems that can be handled.

6.2 Scoping the Problem

Four problem areas were identified as cornerstones. They can be considered to be stumbling blocks to the realization of distributed applications that adapt to changing conditions.

The Performance Vocabulary as defined in Chapter 2 is a key enabler of efficient systems because it gives a framework to relate different levels of abstraction in metrics. To the user, the most important metrics are situated in the MoE level and above. To the system, MoPs and DPs are most appropriate. Adopting a layered approach to the definition of metrics is a fine way of addressing the semantic disparity of user-level vs system-level metrics.

The "Measuring Performance" Attitude seems to be a way for architects, software developers, project managers and others to comfort themselves with what they think to be an honest consideration of performance aspects of the systems they produce. It is not. Measuring performance is far from guaranteeing system performance. Chapter 3 left us with a definition of what is QoS, as a concept that encompasses the notion of guaranteeing levels of performance and efficiency of a system. Moving from a measuring performance attitude towards a notion of QoS shapes the way we design systems and in the end how adequately

these adapt to their environment.

The "Hidden Behaviour" Impairment occurs when someone uses the very useful concept of interfaces and hides behind it. Components exposing their interfaces offer the world a means to communicate with them, pretty much like an enterprise exposes its services through a web page. Two enterprises offering the same service probably do not perform equally in its rendering. For a customer to decide which is most appropriate in certain circumstances, information has to flow beyond the simple definition of the interface. Chapter 4 shows a formal and standard mechanism by which the middleware becomes an important agent in conveying information between client and server objects.

The Lack of a Performance Methodology is the absence of a formal way of applying the different tools pertinent to QoS and performance rendering. Without it, one is condemned to adopt ad hoc approaches, unable to reproduce favourable results or worse, unable to understand what goes wrong. Chapter 5 proposes a UML-based methodology, enclosed as a logical loop within RUP. Its goal is to help the software architect apply the concepts explained in the earlier chapters. It is a framework within which user needs and system capabilities are linked in a formal manner, within which efforts to increase determinism in distributed applications are promoted.

6.3 Future Work

The reflection started in this work cannot end without any form of proposal for future activities in this domain. Far from being circumscribed, the domain sphere addressed in this work was barely explored. The following ideas constitute a set of research subjects, in no particular order, that would push further the knowledge on the domain.

6.3.1 Performance Patterns

The concept of design patterns introduced in Chapter 5 suggests that specialized patterns of performance aspects of distributed applications would be of great utility to the IT community. Although performance aspects are implicit in design patterns, patterns relevant to distributed applications are of such a low granularity that it is hard to see their applicability in the large scale. The definition of performance patterns (maybe from compounding basic patterns) with the right level of granularity would be of great help.

6.3.2 Formal Constraints Representation

We have seen in Chapter 5 how to instrument the system with MoPs. In the models, these were captured with notes in the standard UML notation. Also, these were expressed in standard Object Constraint Language (OCL), a formalism that aims at capturing constraints within systems. This formalism coupled with UML provides a rich environment for performance modelling. However, using notes to capture constraints indicates a lack of automated tools that could generate code from the different performance constraint expressions. If

UML/OCL is semantically rich enough to specify the system, then there would theoretically be no reason not to have an automatic parser and code generator feeding on the models. Further research is needed to investigate this.

6.3.3 Variable Weighting of MoMs

Hierarchical relationships between MoEs and MoPs are determined in the process of relating system resources to human needs. In different working scenarios, there might be cases where certain MoPs would contribute to MoEs at a greater or lesser degree. This suggests that the weights of the links between MoPs and MoEs vary along with the operating context of the system. This aspect has not been taken into account in this work.

6.3.4 Closed-loop QoS Specification

So far, QoS requirements have been captured once when designing the system according to high-level MoEs. From there on, a forward engineering method of modelling has been adopted. The user has been left out of every decision concerning QoS except in the initial statement of his needs at the higher level. A real-time closed-loop system including inputs from the user would probably promote symbiosis between the two, thus increasing significantly the human-system performance. Also, from the knowledge of the system's internal state, the user could adapt his needs (to the extent possible) to what resources are available. In the car example, the driver will know whether or not he can reach a certain distance with what's actually left in the gas tank.

6.4 Towards Adaptive C2ISs

The military command and control environment is a highly volatile environment of mixed systems and components, each of which can appear or disappear depending on external conditions. Far from desirable, this nonetheless constitutes the reality of the battlefield today. As the military forces have been working on solutions that address this context (TCP/IP was invented on that premise), there is a need to have software applications that can adapt to this kind of environment. Highly survivable and adaptable systems is a requirement. It is hoped that this paper shed some light on the matter.

Bibliography

- [1] Stéphane Paradis, Richard Breton, and Jean Roy. Data fusion in support of dynamic human decision making. Article, Defence Research Establishment Valcartier, 2459, Pie XI North, Val-Bélair, Québec, Canada, G3J 1X5, 1999.
- [2] Louis G. Bornman. Command and control measures of effectiveness handbook (C2MOE HANDBOOK). Technical Document TRAC-TD-0393, TRADOC Analysis Command - Study and Analysis Center, Fort Leavenworth, Kansas, USA, October 1993.
- [3] M. R. Endsley. Toward a theory of situation awareness in dynamic systems. *Human Factors Journal*, 37(1):32–64, March 1995.
- [4] Vernon M. Bettencourt. Command, control, communications, intelligence, electronic warfare measures of effectiveness (C³IEW MOE) workshop. MORS Report MORS FR 9210, Military Operations Research Society, Fort Leavenworth, Kansas, USA, October 1992.
- [5] Ricky Sweet, Morton Metersky, and Michael Sovereign. Command and control evaluation workshop. Workshop Report xxx, Military Operations Research Society, January 1985.
- [6] NATO. Code of best practice (COBP) on the assessment of C2. RTO Technical Report RTO-TR-9 AC/323(SAS)TP/4, NATO Research and Technology Organization, North Atlantic Treaty Organization, BP 25, 7 rue Ancelle, F-92201, Neuilly-sur-Seine Cedex, France, March 1999.
- [7] Valdur Pille. Assessing the impact of information technology on command and control: Measures of merit. DREV Report DREV R-9828, Defence Research Establishment Valcartier, 2459, Pie XI North, Val-Bélair, Qc, Canada, G3J 1X5, September 1999.
- [8] Éloi Bossé and Jean Roy. Measures of performance for the ASCACT multisensor data fusion demonstration program. DREV Report DREV-TM-9618, Defence Research Establishment Valcartier, DREV, 2459, Pie XI North, Val-Bélair, Qc, Canada, G3J 1X5, March 1997.
- [9] Study Group 7. Information technology - quality of service - framework. International Standard ITU-T Recommendation ISO/IEC 13236, ITU - Telecommunication Standardization Sector, Geneva, Switzerland, December 1998.

- [10] Vicki Johnson. The IP QoS FAQ, 1999. <http://www.qosforum.com/docs/faq/>.
- [11] Geoff Huston. *Internet Performance Survival Guide: QoS Strategies for Multiservice Networks*. John Wiley and Sons, Inc, 2000.
- [12] Stardust.com. White paper - the need for qos, July 1999. http://www.qosforum.com/white-papers/Need_for_QoS-v4.pdf.
- [13] Richard E. Schantz. Quality of Service. Kluwer Academic Publishers, 1998. <http://www.dist-systems.bbn.com/papers/1998/QoSArticle/>.
- [14] Arnold W. Bragg. Quality of service: Old idea, new options. *IT Professional: Technology solutions for the enterprise*, 1(5):37–44, September 1999.
- [15] Editor in chief. Qos: New term in the IP lexicon. *IEEE Internet Computing journal*, 4(4):46, July 2000. <http://computer.org/internet/>.
- [16] Andrew S. Tanenbaum. *Computer Networks*. Prentice Hall, second edition, 1988.
- [17] R. Braden, D. Clark, and S. Shenker. Integrated services in the internet architecture: An overview. Request for Comments 1633, Network Working Group, June 1994. <http://www.ietf.org/rfc/rfc1633.txt>.
- [18] C. Partridge. A proposed flow specification. Request for Comments rfc1363, BBN, July 1992. <http://www.ietf.org/rfc/rfc1363.txt>.
- [19] S. Shenker and J. Wroclawski. General characterization parameters for integrated service network elements. Request for Comments 2215, Network Working Group, September 1997. <http://www.ietf.org/rfc/rfc2215.txt>.
- [20] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource ReSerVation protocol. Request for Comments 2205, Network Working Group, September 1997. <http://www.ietf.org/rfc/rfc2205.txt>.
- [21] Y. Bernet, R. Yavatkar, P. Ford, F. Baker, and L. Zhang. A framework for end-to-end QoS combining RSVP/IntServ and differentiated services. Internet draft, Internet Engineering Task Force, March 1998. <http://www.ietf.org/rfc/rfc2460.txt>.
- [22] OMG Team. The common object request broker: Architecture and specification, October 2000. <http://www.omg.org>.
- [23] Janis R. Putman. *Architecting with RM-ODP*. Prentice-Hall, 1st edition, October 2000.
- [24] OMG. Uml profile for schedulability, performance, and time specification. Final adopted specification, Object Management Group, March 2002. <http://www.omg.org/uml>.

- [25] OMG. Omg unified modeling language specification. Formal Specification v 1.4, Object Management Group, September 2001. <http://www.omg.org/uml>.
- [26] Directorate of Army Doctrine. Land force information operations - field manual intelligence. Canadian Doctrine B-GL-357-001/FP-001, National Defence, Canada, January 2001. <http://www.army.dnd.ca/ael>.
- [27] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Addison-Wesley, 1st edition, 1995.

Appendix A

Acronyms, Initialisms and Abbreviations

AGM Attack Guidance Matrix

API Application Programming Interface

C2 Command and Control

C2IS Command and Control Information System

C3IEW Command, Control, Communications, Intelligence, Electronic Warfare

C2MOE Command and Control Measures of Effectiveness

CCIR Commander's Critical Information Requirement

COBP Code Of Best Practice

COA Course Of Action

COEAs Cost and Operational Effectiveness Analyses

COM Component Object Model

CORBA Common Object Request Broker Architecture

CPU Central Processor Unit

DA Distributed Application

DCE Data Communication Equipment

DCOM Distributed Component Object Model

DiffServ Differentiated Services

DOC Distributed Object Computing

DP Dimensional Parameters

DST Decision Support Template

ESIOP Environment-Specific Inter-ORB Protocol

FIFO First-In, First-Out

FM Field Manual

GIOP General Inter-ORB Protocol

GoF Gang of Four

HMI Human-Machine Interface

HPTL High-Payoff Target List

HVTL High-Value Target List

IEC International Electrotechnical Commission

IETF Internet Engineering Task Force

IDL Interface Definition Language

IIOP Internet Inter-ORB Protocol

IntServ Integrated Services

IOR Interoperable Object Reference

IPB Intelligence Preparation of the Battle

ISO International Organization for Standardization

ISTAR Intelligence, Surveillance, Target Acquisition and Reconnaissance

IT Information Technology

LAN Local Area Network

MoE Measure of Effectiveness

MoFE Measure of Force Effectiveness

MoM Measure of Merit

MoP Measure of Performance

MORS Military Operations Research Society

MVC Model-View-Controller

NATO North Atlantic Treaty Organization

OCL Object Constraint Language
OMG Object Management Group
OOA Object-Oriented Application
OODA Observe-Orient-Decide-Act
OOTW Operations Other-Than-War
OPP Operational Planning Process
ORB Object Request Broker
OSI Open System Interconnection
PIC Performance Information Carrier
PIR Primary Information Requirement
POA Portable Object Adapter
QMF QoS Management Function
QoI Quality Of Information
QoS Quality of Service
RMI Remote Method Invocation
RM-ODP Reference Model - Open Distributed Process
RSVP Resource ReserVation Protocol
RTCORBA Real-Time Common Object Request Broker Architecture
RTORB Real-Time Object Request Broker
RTOS Real-Time Operating System
RUP Rational Unified Process
SAR Search And Rescue
SLA Service-Level Agreement
TACFIRE Tactical Fire Direction System
TCP Transport-Control Protocol
TOS Tactical Operations System
TRADOC Training and Doctrine
UML Unified Modeling Language
WFQ Weighted-Fair Queuing