UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

HUMAN MACHINE INTERFACE PERFORMANCE AND INSTRUMENTATION FOR HIGH AVAILABILITY SYSTEMS

Ana Lúcia Alves da Bárbara Caleço

Relatório Publico

MESTRADO EM ENGENHARIA INFORMÁTICA
Sistemas de Informação
2010

UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

HUMAN MACHINE INTERFACE PERFORMANCE AND INSTRUMENTATION FOR HIGH AVAILABILITY SYSTEMS

Ana Lúcia Alves da Bárbara Caleço

PROJECTO

Trabalho orientado pelo Prof. Doutor José Manuel de Sousa de Matos Rufino

e co-orientado por Eng. Manuel António Sousa Dias

MESTRADO EM ENGENHARIA INFORMÁTICA
Sistemas de Informação

2010

Acknowledgements

I acknowledge many people...but in Portuguese because that is something that I am very proud of. Thank you for letting me be Portuguese with so much proud!

Primeiro que tudo obrigada Mãe e Pai por esta oportunidade de vida e todas as outras pequenas coisas que me fizeram chegar aqui. Obrigada Mãe pela força dada em todos os *sprints*. Obrigada Pai por todas as paixões que me passaste, da História à Aviação.

Agradeço a todos os professores que desde a primeira classe me deram notas boas a justificar o meu esforço e empenho e me fizeram chegar aqui; ter a oportunidade de tirar o curso que queria e na faculdade que escolhi. Se não fosse a visita de estudo no 12º Ano tinha perdido o prazer de pertencer à família, de que muito me orgulho de pertencer, de alunos formados na Faculdade de Ciências da Universidade de Lisboa. Um obrigada especial ao Professor Rufino que me acompanhou nesta demanda pelo Graal, que culminou nesta tese, neste momento em que termino o Mestrado em Engenharia Informática.

A todos os colegas que ultrapassaram os dias de aulas e as tardes infindáveis de projectos a meu lado na Licenciatura – temos de voltar ao chinês no Alvaláxia para comer vaca com molho de ostras e bife com batatas fritas.

Aos meus parceiros de mestrado, Nuno e João – aquelas manhãs para trabalhar eram sempre para ler jornais online, gozar com o Veloso e dizer piadas com bastante humor negro; todos os 'Jogos Bonitos' foram bem jogados! Mas em especial, um muito obrigada por tudo o que fizeram num dos piores momentos da minha vida e que também me ajudaram a ultrapassar: PMEs RULAM! E que ninguém vá auditar a nossa AJN09 que é a melhor empresa do mundo!

Sorte é coisa que não existe, mas 5 é um número que gosto bastante. Foi o número que me trouxe aqui, a este mestrado, a este estágio, a esta empresa, a esta oportunidade que sempre quis.

Obrigada à NAV por me acolher tão bem. Por me mostrar o que é o mundo do trabalho a fazer algo que tanto gosto. Um obrigado ainda maior a todo o pessoal da NAV que me recebeu e que viu que ser benfiquista é a melhor coisa do mundo! Grande respeito pelos rivais sportinguistas que me fizeram acreditar em 'Jesus'.

Especial obrigada à equipa ASD por quem tenho grande *Estima* e que puxou por mim e acreditou em mim, de maneira que o meu programa tem *Dias* que até tira *Bica*s!

Manel D., tens aqui alguém para trabalhar e que tem sempre armas para lutar! Grande respeito. Vocês são os maiores! Bica, já sabes, snooker time ao som da Lady GaGa ^_-

A todo o pessoal, que eu e o João mobilizamos para o grande desporto que é o snooker…uma grande tacada ao som da música dos 'chouriços'! Mas cuidado com o Android… ^_-

Claro que não me esqueço dos dois pobres coitados (momento de revirar os olhos) que saíram na rifa que dizia "Agora aturem os estagiários!". Manel e Rui. Obrigada pela disponibilidade da minha FIR, a qual tenho que actualizar para a época 2010/2011. Eu sei que a impressora nunca mais foi a mesma depois de chegarmos, mas agora temos copinhos de água para o AC não secar tanto o ar =þ AHHH! E obrigadinha pelos brindes que nos foram dando, aos quais tínhamos de esperar pelos restos. Gostei muito do meu kit de prevenção da Gripe A ^_-

Um especial abraço ao meu Amigo Secreto natalício pela caneca do SCP…estava mesmo a precisar de um copo para as canetas lol

Agradecimentos ao *Pipol* lá de baixo por todos os bonitos e catitas momentos vividos, entre conversas que não posso citar, piadas a fazer faísca, mas principalmente pela grande disponibilidade para ajudar sempre que foi preciso. *bows*

Um obrigada ao meu chefe de serviço, Vermelhudo, que me fez a melhor chamada para o telemóvel, a marcar a entrevista e, todo o apoio dado desde o princípio.

Também tenho de agradecer ao pessoal do Bar pela boa comida e grande simpatia, às senhoras da limpeza pelos WCs impecáveis, os seguranças pela preocupação com o meu frio no Inverno rigoroso que fez =þ e ao AC do meu gabinete por ser tão bom agora nos dias de 40ºC como foi nos de 0ºC.

João, obrigada por tudo, tudo, tudo. Sempre.

A quem não mencionei, também um obrigada bem grande, como a minha felicidade neste momento.

To Little Ana for what she was

and to Big Ana for what she became..

Abstract

Air Traffic Control (ATC) provides services whose objective is to manage aircrafts to ensure safely orderly and expeditious flows of traffic.

The LISATM system has a number of Controller Working Positions equipped with surveillance display applications, the Operator Display System (ODS). Due to redundancy requirements, there are two equivalent applications developed with different technologies. This project focused on the Air Situation Display System (ASD), an application which provides a surveillance display with user-interaction capability and is developed in Java technology. The application integrates the Radar Fallback keeping the looks and feels from the ODS System, only being developed in a different programming language.

Thus the present work focused on monitoring the performance of high availability Real-Time Human-Machine Interface applications. Recommendations were endorsed for quantifiable performance, being these activities accompanied by a corresponding demonstration of results suitability for the intended context. Modules were defined in order to instrument applications *Human-Machine Interface High Performance and High Availability* (HMI-HPHA) with features necessary and sufficient for the pursuit of Real-Time performance analysis. These modules were developed using JAVA technology using the Java Management eXtensions (JMX), to manage the created agents.

All the monitoring services run in real-time so it can inform the managers of changes of the application state as soon as it happens. This implies that the information provided needs to be not only concise but also adequate to the context.

Keywords: High Availability and High Performance, Real-Time, Mission-Critical, Human-Machine-Interface, Monitoring

Resumo

O Controle de Tráfego Aéreo (ATC) presta serviços, cujo objectivo é direccionar o fluxo de aeronaves para garantir a ordem, rapidez e segurança do tráfego, bem como para dar informações aos pilotos.

O sistema LISATM tem um número de postos de trabalho, para os controladores, equipado com aplicações de vigilância radar (Display System Operator - ODS). Devido a requisitos de redundância, existem duas aplicações equivalentes desenvolvidas com diferentes tecnologias. Este projecto focou-se no Air Display System (ASD). Aplicação que fornece vigilância radar com capacidade de interacção com o utilizador, sendo desenvolvido com tecnologia Java. A aplicação integra o Radar Fallback, mantendo o aspecto e mecânica de utilização igual ao Sistema ODS, com a diferença a residir na linguagem de programação.

O foco do presente projecto incidiu sobre o desempenho de aplicações de Interface Homem-Máquina em Tempo-Real de elevada disponibilidade. Desta forma foram elaboradas propostas de recomendações de desempenho quantificáveis a serem incorporadas no ciclo de desenvolvimento de aplicações HMI-HPHA (*Human-Machine Interface High Performance and High Availability*). Estas foram acompanhadas da correspondente demonstração da adequabilidade ao contexto pretendido. Para isso foram definidos e desenvolvidos módulos de forma a instrumentar a aplicação HMI-HPHA com as funcionalidades necessárias e suficientes para a persecução da análise de desempenho. Estes módulos foram desenvolvidos com recurso à tecnologia JAVA e às ferramentas de automatização dos mesmos, usando Java Management eXtensions (JMX) para gerir os agentes desenvolvidos.

Todos os serviços de monitoria foram executados em Tempo-Real de modo a informar os gestores aquando de alguma mudança de estado da aplicação. Estas informações necessitam não só de ser concisas, mas também adequadas ao contexto.

Palavras-Chave: Elevada Disponibilidade e Elevado Desempenho, Tempo-Real, Missão-Crítica, Interface-Pessoa-Máquina, Monitoria

Resumo Alargado

O Controle de Tráfego Aéreo (ATC) presta serviços, cujo objectivo é direccionar o fluxo de aeronaves para garantir a ordem, rapidez e segurança do tráfego, bem como para dar informações aos pilotos.

O sistema LISATM é o sistema proprietário da NAV Portugal E.P.E. para desempenhar tais funções. Este está equipado com aplicações de vigilância radar Display System Operator (ODS). Devido a requisitos de redundância, existem duas aplicações equivalentes desenvolvidas com diferentes tecnologias. Este projecto focou-se no Air Display System (ASD). Aplicação que fornece vigilância radar com capacidade de interacção com o utilizador, sendo desenvolvido com tecnologia Java. A aplicação integra o Radar Fallback, mantendo o aspecto e mecânica de utilização igual ao Sistema ODS, com a diferença a residir na linguagem de programação.

O foco do presente projecto incidiu sobre o desempenho de aplicações de Interface Homem-Máquina em Tempo-Real de elevada disponibilidade. Desta forma foram elaboradas propostas de recomendações de desempenho quantificáveis a serem incorporadas no ciclo de desenvolvimento de aplicações HMI-HPHA (*Human-Machine Interface High Performance and High Availability*). Estas foram acompanhadas da correspondente demonstração da adequabilidade ao contexto pretendido através. Para isso foram definidos e desenvolvidos módulos de forma a instrumentar a aplicação HMI-HPHA com as funcionalidades necessárias e suficientes para a persecução da análise de desempenho. Estes módulos foram desenvolvidos com recurso à tecnologia JAVA e às ferramentas de automatização dos mesmos, usando Java Management eXtensions (JMX) para gerir os agentes desenvolvidos.

Este passo foi precedido de um estudo intensivo da Framework e da sua arquitectura de forma a determinar as componentes principais no processamento de mensagens e de como estas eram tratadas de forma à sua informação ser apresentada de forma correcta aos controladores.

Juntamente com o estudo da Framework, também foi efectuado o estudo das mensagens de radar recebidas pela aplicação. Estas seguem um formato pré-estabelecido pela EUROCONTROL denominado ASTERIX. Este formato serve de protocolo também, de forma que está dividido por categorias de mensagens de forma a facilitar a sua identificação e troca. Para este projecto foram monitorizadas as mensagens ASTERIX de categoria 30. Estas contêm vários blocos de dados que por sua vez contêm um ou vários registos que agrupam um ou mais itens. São estes registos com os seus itens que darão origem às pistas a serem apresentadas no ecrã ao

controlador com a informação associada numa etiqueta que identifica, nomeadamente a aeronave, a sua altitude e é também apresentado o vector segundo a rota que a aeronave segue.

Para desenvolver a ferramenta de monitoria foi de grande importância manter a modularidade da mesma de forma a ser integrada na Framework de forma a não ser necessário alterar demasiado código nesta. Assim, para desenvolver as classes destinadas a recolher os dados ao longo do processamento de dados de uma mensagem, foi criada uma interface a ser implementada pelas classes das componentes atrás identificadas como sendo as componentes de uma mensagem ASTERIX categoria 30. Depois, uma classe destinada a iniciar o processo de monitoria, com a criação do MBean responsável pela análise dos valores e o seu registo no MBean Server também criado, e também a finalizar o mesmo processo, foi desenvolvida. Esta classe permite identificar uma mensagem mal entre na aplicação (lida de um Link), as várias fases por que passa no processamento até finalmente ser feita a sua representação no ecrã. Esta identificação é feita através de uma marca de tempo e da sua concatenação com um identificador numérico por cada fase que o conteúdo da mensagem passa até ao número de identificação da aeronave. Assim existe uma identificação unívoca das mensagens recebidas que permite assegurar a sua atomicidade, sabendo que a mensagem que entrou no tempo X é a que deu origem à apresentação Y, não havendo trocas de mensagens.

Todos os serviços de monitoria foram executados em Tempo-Real. As informações obtidas necessitam não só de ser concisas, mas também adequadas ao contexto. Para isso foram definidas métricas baseadas no tempo total de processamento, isto é, desde a entrada de uma mensagem até à produção do seu resultado. Obtendo vários valores em vários testes, as médias serviram para determinar, não só que a aplicação opera numa zona de conforto, bem como, muito abaixo do prazo limite que são os cinco segundos de rotação do Radar. Permitiram também definir padrões de resultados que se mostraram dentro do expectável para cada cenário testado. Cenários esses em que se avaliou as operações básicas que a aplicação efectua sobre as mensagens recebidas: criação, actualização e remoção de uma pista. Foram também feitos testes em cenários operacionais, ou seja, situações normais de tráfego aéreo, em que, ao contrário dos anteriores cenários, as mensagens testados não eram de um tipo único por cenário (criação, actualização ou remoção), mas de todos ao mesmo tempo. Valores mais altos observados ocasionalmente requerem estudo mais aprofundado no futuro, sendo que a maioria foram detectados devido a actividades da Java Virtual Machine executar Garbage Collections.

Desta forma o projecto foi completado dentro dos pressupostos iniciais e com sucesso.

Table of Contents

Table of Figures

Table of Tables

Glossary
(Alphabetic order)

ACRONYM	MEANING
ARTAS	ATC Surveillance Tracker and Server
ASD	NAV - Air Situation Display
ASTERIX	All Purpose Structured Eurocontrol Survaillance Information Exchange
ATCO	Air Traffic Controller
ATSS	ATO, TFC and Safety net System
ATD	Actual Time of Departure
ATM	Air Traffic Management
ATC	Air Traffic Controller
ATO	Actual Time Over
CVSM	Conventional Vertical Separation Minimum
CWP	Controller Working Position
DLS	Data Link Server
DSTI	*NAV – Direcção de Sistemas e Tecnologias de Informação*
EDI	Eletronic Information Distribution
ENV	Environment System
ETA	Estimated Time of Arrival
ETD	Estimated Time of Departure
EUROCONTROL	European Organisation for the Safety of Air Navigation
FANS	Future Air Navigation Systems (ICAO)
FDP	Flight Data Processor
FIR	Flight Information Region
FL	Flight Level
FP ou FPL	Flight Plane
FDPS	Flight Data Processing System

GSD	NAV - Ground Situation Display
HMI	Human-Machine-Interface
HPHA	High Performance and High Availability
ICAO	International Civil Aviation Organization
LISATM	NAV - Lisbon Air Traffic Management System
NAV	*Navegação Aérea de Portugal*
ODS	NAV - Operator Display System
OJT	On Job Training
OLDI	On-Line Data Interchange
POACCS	MILITARY FPL information (creation, modification and cancel)
PR	Primary Radar
PSR	Primary Surveillance Radar
RVSM	Reduced Vertical Separation Minimum
SAC/SIC	System Area Code/System Identification Code
SDPS	Surveillance Data Processing System
SIMATM	NAV - ATM Simulator
SIMATM-GS	NAV - SIMATM Game Supervisor
SIMATM-PP	NAV - SIMATM Pseudo Pilot
SIMATM-SE	NAV - SIMATM Scenario Editor
SISINT	*NAV-DSTI: Sistemas – Interface com o utilizador*
SISLOG	*NAV-DSTI: Sistemas – Logística*
SISPRO	*NAV-DSTI: Sistemas – Produção*
SISQUA	*NAV-DSTI: Sistemas – Gestão da Qualidade e Safety*
SITA	Speciallists in Air Transport Communications and IT Solutions *(Sociedade Internacional Telecomunicações Aeronáuticas)*
SSR	Secondary Surveillance Radar
STCA	Short Term Conflict Alert
STS	*NAV - Supervisão Técnica de Sistemas*

TAR	Terminal Approach Radar
TFC	Track Flight Correlation System
TWR	Tower
TMP	Test Management Plan
VFR	Visual Flight Rules
VSM	Vertical Separation Minimum

Table of References

Ref.	Document
[1]	U.S. DEPARTMENT OF TRANSPORTATION, FEDERAL AVIATION ADMINISTRATION, *Air Traffic Control – Chapter 2. General Control* Website: http://www.faa.gov/air_traffic/publications/atpubs/ATC/atc0201.html
[2]	International Civil Aviation Organization (ICAO), Air Navigation Bureau (ANB) Website: http://www.icao.int/icao/en/trivia/peltrgFAQ.htm#23
[3]	Chambost, Germain in : Le filet de sauvegarde resserre ses mailles Website : http ://www.dgac.fr/html/publicat/av_civil/them_1/them_1_34_35.pdf
[4]	EUROCONTROL – SESAR Website: http://www.eurocontrol.int/sesar/public/subsite_homepage/homepage.html
[5]	Raymond, Eric Steven and Landley, Rob W. in: *The Art of Unix Usability Chapter 2. History: A Brief History of User Interfaces*, 2004 Website: http://www.catb.org/~esr/writings/taouu/html/ch02.html
[6]	Can DCS HMI impact operator performance in: *Control Global Community*, Usability Project - OPerations, 2009
[7]	Mintchel, Gary A. in: *Plan Ahead to Build the Perfect HMI System*, ICONICS, November 2001
[8]	Boyko, Brian in: *Fingerpointing, Frustrated Network Engineers, and the Application Performance Blame Game*, Network Performance Daily, 2007
[9]	Real User Monitoring *Website: http://en.wikipedia.org/wiki/Real_user_monitoring*
[10]	Drakos, Nikos and Moore, Ross in: *Network Performance Measurement and Analysis* - Design and Implementation of a WWW Workload Generator for the NS-2 Network Simulator, 2001
[11]	Passive Monitoring

Ref.	Document
	Website: http://en.wikipedia.org/wiki/Passive_monitoring
[12]	Fleury, Marc & Lindfors, Juha in: *Enabling Component Architectures with JMX* – OnJava.Com, 2001 Website: http://onjava.com/pub/a/onjava/2001/02/01/jmx.html
[13]	Goff, Max in: *Java in the management sphere, Part 2 – Java enters the management arena with JMX and Java DMK* Website: http://www.javaworld.com/jw-11-1999/jw-11-management.html
[14]	Java Management Extensions Website: http://en.wikipedia.org/wiki/Java_Management_Extensions
[15]	Pande, Manish & Ganesan, Rajeshwari in: *Gathering Performance Requirements* – Computer Measurement Group, June, 2005
[16]	MVC - XEROX PARC 1978-79 Website: http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html
[17]	Burbeck, Steve in: *Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC)* - ParcPlace Systems, Inc, 1992
[18]	The Linux Information Project - GUI Definition Website: http://www.linfo.org/gui.html
[19]	Piedad, Floyd in: *High Availability: Design, Techniques, and Processes*, Prentice Hall PTR, 2001
[20]	Veríssimo, Paulo and Rodrigues, Luís in: *Distributed Systems for System Architectures*, Kluwer Academic Publishers, 2001
[21]	Wieringa, Roel in: *Design methods for reactive systems: Yourdon, Statemate, and the UML*, Elsevier Publishers, 2003
[22]	Flores-Mendez, Roberto A. in: *Towards a Standardization of Multi-Agent System Frameworks*, XRDS Crossroads, June 1999
[23]	High-Performance HMI - The Seven Step Methodology to High-Performance HMI Website: http://www.pas.com/Solutions/High-Performance-HMI.aspx
[24]	Juvva, Kanaka in: Real-Time Systems Website: http://www.ece.cmu.edu/~koopman/des_s99/real_time/

Ref.	Document
[25]	Singh, Arjun in: *Communication Navigation Surveillance / Air Traffic Management (Cns / Atm) Beyond 2012*, GISdevelopment.net – Technology, 2004
[26]	Jensen's, E. Douglas - Real-Time for the Real World: *My personal manifesto about the widely misunderstood field of real-time systems*, 2001-2010 Website: http://www.real-time.org/realtime.htm
[27]	Sun Microsystems, Inc., "Java Dynamic Management Kit 5.1 Tutorial – June 2004"
[28]	Performance Definition – Business Dictionary Website: http://www.businessdictionary.com/definition/performance.html
[29]	Model–view–controller Website: http://en.wikipedia.org/wiki/Model–view–controller
[30]	Model-View-Controller – OO Books Website: http://ootips.org/mvc-pattern.html
[31]	Model View Controller As An Aggregate Design Pattern Website: http://c2.com/cgi/wiki?ModelViewControllerAsAnAggregateDesignPattern
[32]	Model-View-Controller Pattern - How It Works Website: http://www.enode.com/x/markup/tutorial/mvc.html
[33]	Java Coffee Break: *Inside Java - The Java Virtual Machine*, January 2000 Website:http://www.javacoffeebreak.com/articles/inside_java/insidejava-jan99.html
[34]	Model-View-Controller – MSDN Microsoft patterns & practices Website: http://msdn.microsoft.com/en-us/library/ms978748.aspx
[35]	Deacon, John in: *Model-View-Controller (MVC) Architecture*, Computer Systems Development, Consulting and Training, 1995-2009
[36]	Ramachandran, Vijay in: *Design Patterns for Building Flexible and Maintainable J2EE Applications*, January 2002
[37]	Sullins, Benjamin G. and Whipple, Mark B. in: *JMX in Action*, Manning

Ref.	Document
	Publication, 2002
[38]	JVM Monitoring: JMX or SNMP? Website: http://blogs.sun.com/jmxetc/entry/jmx_vs_snmp
[39]	1 - LISATM_Vision – NAV Presentation
[40]	Navegação Aérea de Portugal – NAV Portugal E.P.E., DSTI – Direcção de Sistemas e Tecnologias de Informação. in: *Air Situation Display* – ASD: Project Vision, Part, Version 0.0, 2003
[41]	Navegação Aérea de Portugal – NAV Portugal E.P.E., DSTI – Direcção de Sistemas e Tecnologias de Informação. in: *Air Situation Display* – ASD: Software Interface Specification, External interface definition; Version: 1.4, 2004
[42]	EUROCONTROL STANDARD DOCUMENT FOR SURVEILLANCE DATA EXCHANGE, Part 1: All Purpose Structured Eurocontrol Surveillance Information Exchange (ASTERIX); Version 1.29, 2002
[43]	Asterix_in_ASD – NAV Presentation
[44]	SNMP Research – Secure Internet and Network Specialists Website: http://www.snmp.com/index.shtml
[45]	ORACLE - Using JConsole to Monitor Applications Website: http://java.sun.com/developer/technicalArticles/J2SE/jconsole.html
[46]	IBM - Using JConsole Website: http://publib.boulder.ibm.com/infocenter/javasdk/v6r0/index.jsp?topic=/com.ibm.java.doc.diagnostics.60/diag/tools/jconsole.html
[47]	Overview of Monitoring and Management Website: http://java.sun.com/j2se/1.5.0/docs/guide/management/overview.html
[48]	United States General Accounting Office – GAO in: PERFORMANCE MEASUREMENT AND EVALUATION - Definitions and Relationships, 1998-2005 Website: http://www.gao.gov/special.pubs/gg98026.pdf

Ref.	Document
[49]	Software performance testing Website: http://en.wikipedia.org/wiki/Software_performance_testing
[50]	Molyneaux, Ian in: The Art of Application Performance Testing - Help for Programmers and Quality Assurance, O'Reilly Media, January 2009
[51]	Gamma, Erich; et al in: Design patterns : elements of reusable object-oriented software; Reading, Mass. : Addison-Wesley, 1995
[52]	Command pattern Website: http://en.wikipedia.org/wiki/Command_pattern

Chapter 1

Introduction

1.1 Motivation

Currently, aviation has become a major mean of transport used by many people. This increase in preference is due mainly to its fastness. The need to make business travel in the shortest possible time for various destinations in aviation is the focal point in the globalization that we are all engaged.

Another point in favour of this mode of transportation is the fact that it still is considered the safest. The odds of something going wrong are very low but when there is an accident, usually it ends up being a tragedy of greater proportions than a car accident, for example.

Thus, in this transport branch the safety is paramount and all decisions are critical. Safety is currently a huge concern in using technology which is updated regularly, either in hardware or at the software level, as well as in simplifying the rules of interaction between people, procedures and equipments [1] [2] [3] [4].

1.2 Goals

The forecast growth in air traffic requires the adoption of new technologies and related procedures enabling the safe and efficient provision of Air Traffic Control (ATC) services to a larger number of aircrafts. The best and most efficient service depends on better tools provided to the controller and it requires applications to be more efficient - hence the monitoring being so important.

As such, the main goal of this project is to develop a Real-Time monitoring framework to obtain performance results from the Air Situation Display (ASD) application itself. To achieve this goal it is necessary to analyse what a Real-Time System is, considering its different flavours, and how an Air Traffic Management (ATM) system can be integrated in this classification, having its concept and context into account. Before that, there is the need to characterize the ATM system as a

Mission-Critical System and how it relates to Real-Time applications. The definition of metrics and ways to interpret the results from the Real-Time monitoring activities in order to improve the application performance is of vital importance to understand the operations and other non-functional attributes of an ATM System [25].

1.3 Host Institution

The NAV Portugal E.P.E. [1] mission is the provision of priority Air Traffic Services in the Flight Information Regions (FIR or RIV) under the Portuguese responsibility - Lisbon and Santa Maria, ensuring compliance with national and international regulations meeting the best safety conditions, enhancing capabilities by focusing on efficiency while meeting environmental requirements. The company operates in the mainland and the autonomous regions of Azores and Madeira.

The Centre for Air Traffic Control of Lisbon and Training Centre are part of the Headquarters located at Lisbon International Aeroport. In the Azores, namely the island of Santa Maria, is located the Centre for Oceanic Control. In addition to these two major centres, the NAV Portugal has other infrastructure with air traffic services operating in the Control Towers of Airports of Lisbon, Porto, Faro, Madeira, Porto Santo, Santa Maria, Ponta Delgada, Horta, Flores and Cascais Aerodrome. For the full realization of its mission of Air Traffic Control the NAV Portugal has a wide range of equipment and installations (radar stations, radio aids and communications) in various parts of the mainland and islands.

The system of air traffic management in Santa Maria Oceanic and phasing in the service of a new generation of air traffic management in Lisbon were decisive steps to maintain the NAV Portugal's among the leading providers of air navigation services [2].

1.4 Work Team

The host team in the company is part of the DSTI - Department of Systems and Information Technology which mission is as follow:

• Conduct technical studies and monitor the operation of the Air Traffic Management Support Systems, in compliance with national and international standards applicable to the sector.

• Prepare development plans, ensuring the inclusion of projects in this area in the Investment Plans and Activities.

• Develop projects related to the Strategic Investment and Development Operations in the ATM, ensuring the delivery of the final product to the user, in compliance with the requirements of quality, time and cost.

• Ensure representation of NAV Portugal in national and international organizations in matters of a technical nature in ATM.

The DSTI is divided in four areas:

• SISINT - Systems User Interface

• SISLOG - Logistics Systems

• SISQUA - Quality Management Systems and Safety

• SISPRO - Production Systems

SISPRO is the host team where this work was developed. This area has the main functions as follow:

• Define the architecture of ATM systems and the various tools to support Air Traffic Management in accordance with operational requirements.

• Undertake projects to develop systems and operating software.

1.5 Documentation Produced

- Preliminary Report;
    - Introduction to the Host Institution and the map of activities to develop
- State of Art - Relating Mission-Critical, Real-Time and Performance concepts;
    - Study and aggregation of concepts of the Real-Time Systems and its flavours and its relation to Application Performance
- KPI – Key Performance Indicators: Concepts of KPIs and *Fuzzy Theory;*
    - Study and presentation of some KPIs and adaptation of the Fuzzy Logic to the ATM System
- Air Situation Display fallback - ASD System Architecture, Components and Data Flows;
    - UML Description of the main components and data flows in the ASD
- ASTERIX Category 30 Messages - Design and implementation approaches to monitor surveillance messages;
    - Proposal of implementations to monitoring this type of ASTERIX messages

- Situation Point Reports:
    - Description of the tasks developed and completed according to the schedule defined and the resources available. Periodicity of one report per month of work.

1.6 Document Organization

This document is organized as follows:

- Chapter 2 – Context and Air Traffic Management System Architecture
- Chapter 3 – State of Art
- Chapter 4 – Air Situation Display Architecture
- Chapter 5 – Technologies Used
- Chapter 6 – Design
- Chapter 7 – Results
- Chapter 8 – Conclusion and Future Work

Chapter 2

CONFIDENTIAL

Chapter 3

Air Traffic Management under a State of Art Perspective

Aviation became one of the most common and used transportation for many people. That came with the idea that's the fastest and most secure and safe way to move around the globe. But that fact demands that all activities incorporated in the area of Air Traffic Management (ATM) are quite decisive and critical. That said we assume that an ATM system must be **Mission-Critical system** [20].

A system like this must give robustness a great deal because a software failure may lead to a catastrophic consequence. This way the Quality of Service (QoS) end-to-end must be assured and determined by the users systems, in this case the HMIs [6][7]. We must assume that a mission-critical system like the ATM system has as principal requirement having all services available with high predictability. And the way to achieve it is making the correct information pass through the system and be delivered at the right place at the right time, validating this way a temporal line.

The concept of **Mission-Critical** is in general connected to the concept of **Real-Time** [20], even if they are orthogonal between each other**.** These systems must assure that an operation to be totally accurate depends on three main factors: logical correctness, time to be executed and the instant in which it's executed. These three factors make the system predictable and deterministic.

Real Time systems are characterized to have time constraints well defined making these systems more reliable than the normal ones, allowing precise data updates even if this only happens a few times per second. Other important characteristic is that the changes of the state of the system are according to the physical time, ie, the instant in which the result is produced – **deadline**. So, the most important point in **Real-Time** is the result produced according to its respective deadline, but the actions must be carried

out in its entirety – predictability is the essential requirement for these kinds of systems and not performance. In this study these two requirements were converged.

Performance become this way a second requirement in the ATM system. **Performance** stands for a completion of tasks before existing standards of accuracy, completeness, cost and speed [28]. But to achieve it, **Performance** must be compared with the settled goals, requirements and expectations, not only for the users but for the application too [8].

3.1 Real Time systems and its goals

Real-time systems are systems that deal with time. But what *time* is this? One of the main problems in real-time systems design is how to relate the several dimensions of time in a distributed system generating many issues:

- Modelling the interaction between the computer and the real world
- Maintaining temporal accuracy of measurements
- Accommodating important load versus finite resources
- Recognizing deadlines and urgency
- Tolerating faults

What is a real-time system? Systems whose progression is specified in terms of *timeliness* requirements dictated by the environment, in other words, the need to synchronize our actions with the environment. The environment has its own pace and thus the system must be adapted to this pace and react according to the evolution of the environment.

There are several classes in real-time:

- Hard real-time systems where timing failures are to be avoided
- Soft real-time systems where occasional timing failures are accepted
- Mission-critical real-time systems where timing failures should be avoided and occasional failures are handled as exceptional events

All systems fail so the fact that measures are taken to secure timeliness specifications in normal operation does not exempt the designer from either designing for the worst-case situation, or endowing the system with the necessary mechanisms to tolerate faults when they happen. This way, **Reliable real-time** means *tolerating or preventing timing faults* according to a pre-defined *fault model* [20].

Real Time systems can be decomposed in three sub-systems: **the controlled object**, **the real time computation system** and **the human operator**. Pictured in Figure 4.

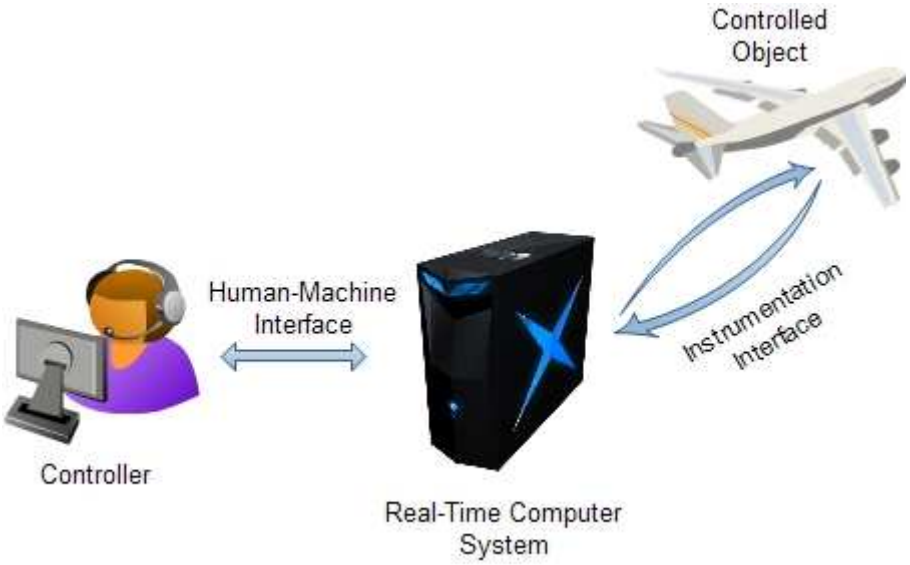

Figure 4: ATM as a Real-Time System

The function of a **Real-Time computer system** is to react to the **controlled object** or **operator** stimulus between time intervals dictated by the application environment [9] [10][24].

This type of computation leads to the concept of **Real Time Computing (RTC)** [20] (the same as **reactive computation** [21]**)** in which a software study is made and it's subject to **Real-Time constraints** [31].

Referring to the goals of a **Real-Time** application exist two main goals: **optimization** and **predictability.** The first one is based in meeting all the deadlines set or minimize the ones which can be failed. This concept is known as **hard deadlines**. The second goal represents what is possible to know about **optimization** (failed deadlines) before the start of a given task which can be in a deterministic or stochastic way. If it is deterministic, the exactly the information about the **optimization** is known before the start of the task, if not, it is stochastic. In this case the concept is known as **soft deadlines**. That happens because deterministic functions always return the same result any time they are called with a specific set of input values and given the same state of the database. Nondeterministic functions may return different results each time they are called with a specific set of input values even if the database state that they access remains the same. So the model is defined as the outcomes are precisely determined through known relationships among states and events, without any room for random variation. In such models, a given input will always produce the same output,

such as in a known chemical reaction. In comparison, stochastic models use ranges of values for variables in the form of probability distributions.

The environment is unpredictable with irregular arrival patterns, for example, an event-triggered message delivered. Thus taking this into account a sporadic pattern assumes that input information arrives irregularly in *bursts*.

It is then to note that sporadic requests must be served within the inter-arrival time without resource disruption. It has long-term regularity and so can be mapped onto a periodic distribution which is adequate for treating sporadic events that are generated by periodic distribution and also called **event showers**. These give us the rate of a periodical distribution where we would have spread the event arrivals throughout the period interval – arrival distributions are mere artefacts to represent the environment behaviour.

It is important to define this if the ASD application has several levels of critical tasks because it must be a **predictable system** as a **Mission-Critical** system must be. In this specific use-case it was considered as a **system of deterministic predictability** because there were no random state transactions, i.e., from a certain starting point the same result would always be produced, respecting the deadline defined following a strict timeline [30]. Then it was considered that every event was causally determined by a chain of prior occurrences unbreakable which brings the idea of the ASD application being **Mission-Critical** (as all the system it's in). But this is somehow *conceptual* because event-triggered systems react to significant events directly and immediately, that is, it adapts well to overload and unexpected events, either from the same or different representatives.

Thus fault tolerance is important in the control systems for mission-critical and safety-critical systems such as the Air Traffic Management. Because of the provisions to support dynamic operation the system allows selective dissemination of information and allocation of resources. It is *susceptible to event showers* because when an external event or burst of events occurs it is transformed into an *event message* of message output which is sent to the interior of the application where one or several computing elements process it, modify the application state and eventually produce outputs, just as the messages arrive.

3.2 Main Factors of a Real Time System

Real-Time systems have three main factors to consider: **deadlines**, **timeouts** and **atomicity**.

The first to consider, **deadlines**, represents the completion of a specific task within a certain time having this way a more useful result than if the completion of the task is beyond a certain time having less utility for the application [36]. This way we have a **timed action** which is the execution of some operation such that its termination event should take place *within* an interval *from* a reference real time instant:

- Deferral Time: the delay introduced before execution is requested, also known as *offset*
- Termination Time: the difference between the termination (end) and request (start) event timestamps in the timeline
- Execution Time: accounts for the duration of the computation in continuous execution (maybe shorter than the termination time)

In ASD, the instant of completion of an action on a desired point of the timeline consist in generating an output, usually graphical information for the controller represented in his/her HMI. This output can have more than one view, i.e., be represented in two or more different ways in the HMI. To measure this elapsed time the evaluation will be the definition of the Termination Time of the action of the application receiving and processing a message.

We can measure this factor assessing how soon the deadline is defined for the processing of that input until it comes out in the right form of the output expected for that information. When this time of processing an input to an output, the **deadline** is exceeded and it means a low utility.

This idea is right but must be distinguished between results produced **before de deadline** and results produced **at the deadline**.

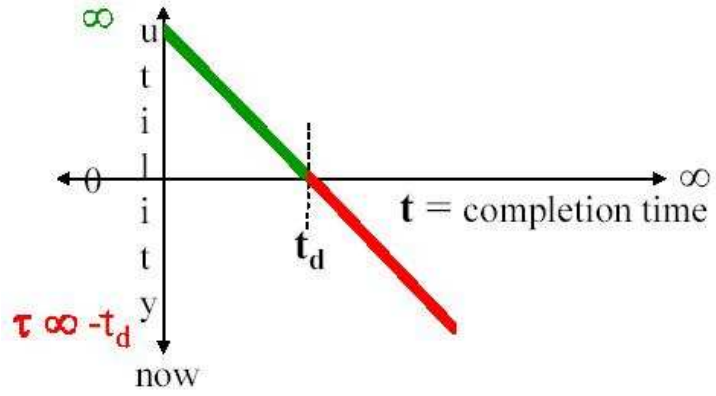

Figure 5: Example of utility being high when near the deadline

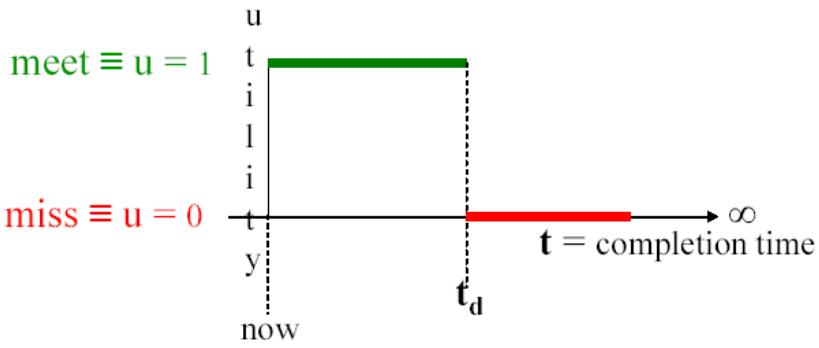

Figure 6: Example of utility being met or not while the deadline is not reached

The differences in Figure 5 and 6, consists in the first one when nearing the **deadline** the utility of some task is higher than before; and after the **deadline** the utility gets less useful every time it gets further passed the **deadline.**

In Figure 6 the **deadline** is met or not. There are only two possible states of usefulness of a task in relation to its **deadline**: the **deadline** is respected or not. This means a binary outcome, 1 or 0, as the deadline is reached or not. It is a **hard deadline** when every output obtained before the **deadline** has utility while every output produced after that **deadline** has no utility at all for the application.

Being the ASD a mission-critical real-time application, there are more of soft deadlines, being the first graphic more accurate to represent what happens in the ASD application.

The second factor identified was **timeout:** specification of the utility of a given task for the application as a function when it is finished. In the ASD application this means that to consider a given task finished, it is necessary to check their atomicity over its deadline. Only then the corresponding utility function can be obtained.

The last factor mentioned was **atomicity** which represents the property that states that if at least one copy of a message was delivered to the addressee then any other recipients of the same will also receive a copy of this message, i.e., the cycle in the system for a particular data stream is completed and is considered finished or not. The **atomicity** should be this way assured by a close monitoring in order to determine if a given task has been completed totally and on time. It can guaranty that through the creation of a tasks list where the critical tasks are identified by order and is defined for each one its **atomicity** and **deadline,** as well as, the complete dataflow since the input until the correspondent output.

3.3 Classification of Real Time systems

There are a variety of ways to classify **Real-Time** systems [26], but the goal of this study only concentrates in classifying it about the application characteristics, more precisely if the application is **Hard Real-Time** or **Soft Real-Time**, considering that it is from a starting point a mission-critical system.

Starting with **Hard Real-Time** the accuracy of an operation after its deadline is considered impractical for the application and may even cause a system failure. This way the **deadlines** established are not negotiable and are determined by the physical objects involved in the entire system which is resumed by the environment the system is in and where the application operates. But the application may not fail itself. If some task in the application is not completed in time, it may lead, none the same, to a catastrophic result. Hard real-time systems are designed in terms of preventing timing failures, however controlled timing failures are allowed in mission critical or soft real-time systems.

These kinds of **deadlines** are then used when is imperative that an event happens within a strict **deadline** or the application will fail entirely if it's not accomplished causing catastrophic consequences. The more demanding hard real-time or mission-critical classes are, more they may face difficulties, like the lack of determinism of the arrival pattern of competing client requests. Hard Time Deadline illustrated in Figure 7.

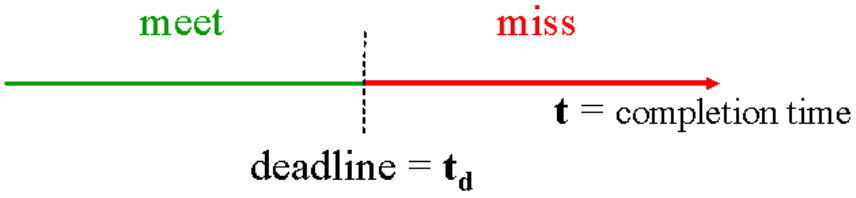

Figure 7: Traditional **Hard Time deadline**

The ASD should be the most deterministic as possible – temporal accuracy – with the **atomicity** of tasks. With that the monitoring of the application and obtaining accurate metrics to evaluate the performance of it is possible.

Unlike **Hard Real-Time** applications, **Soft Real-Time** applications allow some latency which is compensated with some decrease in the Quality of Service. That enables the application to reduce the latency constraints keeping it operating quickly and repeatedly because its functionality depends on fast processing despite having some delays. Soft real-time applications represent nowadays a great part of interactive systems, trying to ensure timeliness specifications in a probabilistic way. Even under

soft real-time perspective the need for availability to fulfil probabilistic deadline assurances exists.

Concerning the ASD application, the decrease in QoS to keep the application operating is not feasible being it a **Mission-Critical** application.

But besides this counterpart this kind decision is at large used with HMIs that allow interaction and execution of operations by its users.

This way the **Real-Time** applications are typically used when there is some concurrency problems and exists the need to maintain a number of interconnected systems up to date with changes in the situation. This fact makes the application less adaptable to changes or updates which have to be well calculated and predicted. Traditional deadline illustrated in Figure 8.

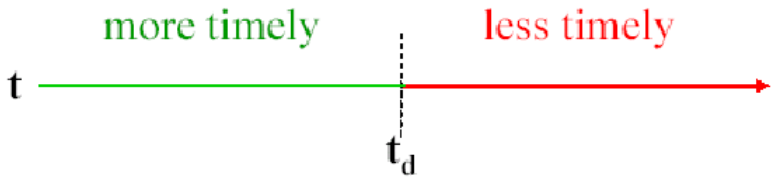

Figure 8: Traditional **deadline** (**Hard deadline** is a special case)

With these characteristics a **Real-Time** application allows artificial **Hard deadlines**. For that a given task is 'more in time' or 'more out of the time' given the prior types of **deadlines** mentioned before.

Now considering **Soft VS Hard** [28]**,** they have differences but usually the applications are a mix of the two, where the notion of **Hard Real-Time** only indicates that the application has some **Hard-time constraints**. Therefore the terms **Hard** and **Soft** can only be seen as designating the *degrees of limitation of time the application has defined*, depending on the severity of the technological limitations of the same.

Characteristic	**Hard real-time**	**Soft real-time**
Response Time	Hard-required	Soft-desired
Peak-load performance	Predictable	Degraded
Control of pace	Environment	Computer
safety	Often critical	Non-critical
Size of data files	Small/medium	Large
Redundancy type	Active	Checkpoint-recovery
Data integrity	Short-term	Long-term
Error detection	Autonomous	User assisted

Table 1: **Hard** and **Soft Real Time** characteristics

Both characteristics from the two types of **Real-Time** can be applied to the ASD application.

The **Response Time** stands for the interval that mediates between the occurrence of an input and the occurrence of the first related output event, can be both **Hard** as can be **Soft** because both of the concepts are seen as degrees of limitation of time depending on the task: which will have a **Hard deadline** and which can run with some latency having a **Soft deadline** then. The *maximum* and the *minimum* response times are relevant variables of real-time systems. The first because it measures the capability of handling the dynamics of controlled processes: slow systems cannot control fast changing processes adequately. The second, because together with the first it measures the variance of the speed of the computer response: quality of control is affected by the latter. The **Peak-load Performance** by the contrary can only be characterized as **Hard Real Time** because the QoS must be predictable and cannot suffer data degradation so the application can keep operating. This would mean that the ATCs would see inaccurate data which would not represent the current information.

Because the environment is such a fundamental constituent in an ATM system the **Control of Pace** can only be consider to be **Hard Real Time**. The environment is what dictates what is happening and give the information inputs for the ASD to operate over.

The fundamental concept in ATM is **Safety**. That said is only logical to consider it a **Hard Real Time** characteristic for the ASD application. It's necessarily critical in a **Mission-Critical** application.

The **Size of Data Files** would not be that large, so it can be considered like having a small/medium size, but despite this, there will be some links in the application transferring a large flow of data needing attention if some of the data are lost or not.

The **Redundancy Type** for instance should be active for when the main application ODS fails there will be a fallback that permit the ATCs keep working - ASD. The **Last Picture** that is displayed to the ATCs when the system fails for some reason does not indicate that exist roll-backs. This way the **Data Integrity** is kept as in the **Soft-Real Time**, ie, assure the data integrity at long-term in the system, for the application to be always up to date with current and correct information which can be accessed at any time by the ATCs as needed, even if it is the last information available to let the controllers work from that point on in a more old manner.

Error Detection in other hand is something that should be accomplished by an autonomous monitoring.

Thus, the ASD application is a **Real Time** application with time constraints well defined but they are more of **Soft** ones. The Radar rotation time is the hard deadline the

application should respect. But some tasks may become more critical and have Hard deadlines, like an alarm, for example, and it will requires immediate attention from the controller, so the monitoring should keep this tasks under great attention because if an alarm message is lost along the application processing it may lead not only to an incident but to an accident – life-critical.

All of these points make real-time application monitoring so important. Because it permit to keep track if something wrong happens in the load flow of the application at real-time and that fact can be notified in real-time too, as to control the situation as soon as possible, keeping rate control (helps balance system load) without glitches: **Safety** properties specify that wrong events never take place – Any delivered message is delivered to all correct participants; and **Liveness** properties specify that good events eventually take place – Any message sent is delivered to at least one participant.

But Liveness is not compatible with the expectations for the ASD application, where the time is used to guarantee synchronization with the environment and the user who produce inputs and receives outputs, like other entities. This is done by defining *timeliness* properties introducing safety conditions.

There is a catch in this situation, that is fully synchronous models do not satisfy the needs of the application because they do not allow timelines specifications. On the other hand, correct operation under fully synchronous models is very difficult to achieve, if possible, in large-scale infrastructures like mission-critical systems, since they present poor baseline timeliness properties.

3.4 Main Factors of a Mission Critical System

Large-scale mission-critical systems like Air Traffic Management resort to dynamic techniques in order to secure deadlines under the uncertain circumstances encountered in the complex environment they normally address. Considering this, three main factors can be defined for a **Mission-Critical** application: **high availability**, **integrity** and **data availability** [26].

The first one – **high availability** – is fundamental in these systems because it is possible to measure the network performance as well as the application itself, so it may serve the users all the time, 24/7, setting the degree of continuity of operation of the system during a given period of time. This is possible if the features, visibility and access to any HMI [19] are assured by the application through a close monitoring using high availability tools.

Integrity it is when secrets are shared between two reliable endpoints with identification mechanisms so the authenticity of a peer can be validated. This way all

the application components must be reliable for keeping the system safety and the QoS as expected in a **Mission-Critical** system. It also means that a specific message has its specific output to the operator monitored and that it is not swapped with other message and its respective output so the real output for each message it's the correct one and not other from an other message.

The third and last main factor is **data availability**, ie, the degree to which a system accurately record and report transactions, as messages or system states.

Being the ASD application **Mission-Critical** the data availability must be assured always because in case of some kind of failure in the system the information displayed by the application to the ATC must be frozen – **Last Picture** – in that moment and only resumed with current data. The data in between does not interest and should not be displayed by the application and is stopped at the input, that is to say, the action of reading messages coming from the links stop until the application recover. For that we must separate data availability from possible failure events so we can determine the existence or not of some data lost.

Based in this main factors and the ASD application it was taken into account the degree of reliability of the application used by the ATM operators – the ATCs – by monitoring the application to ensure that the degree of availability of application data is the highest possible, meeting the expectations and stipulated goals, both in data and temporal mean.

3.5 Real Time and High Performance

Addressing the problem of maintaining the performance of the application within specified limits - it's really important not only achieve all the **deadlines** defined but also in a fast way even if **Real-Time** is not a synonymous of great speed. What we can understand as 'in a fast way' is keeping the right pace, in other words, not letting bottlenecks happen in some part of the path in the data flow until it's represented to the ATC and not delivering the data faster than it can be processed the right way.

Exists some objects devoted to performance/Quality of Service management. These are known as *metric objects* and serve to collect statistical information about system evolution and have methods that compute statistical functions about utilization of resources. The goal of Quality of Service is to ensure that *QoS specification* for a given service remains within the specified parameters for the duration of service provision. One solution to the problem is *monitoring* the application to ensure that the Quality of Service remains within the parameters since the start of application, and detect

deviations early enough. Monitoring may even indicate the systematic failure of a QoS parameter.

Thus the main requirement in a **Real-Time** system is **predictability** and not **performance**, but in this study the objective was converge these two.

But what is meant by application monitoring? It is a paradigm representing the set of activities aiming at knowing the state and evolution of the ASD application during its operation. System devices monitoring is the basis of most tactical management functions. As a matter of fact, tactical management relies on the reactive system principle:

- The loop of monitoring – state acquisition
- Control – state modification

Acquisition is performed through *sensors* which may be implemented either in hardware or in software. The monitoring subsystem must acquire information both by polling or sampling the managed system state. In this matter monitoring incorporates notions on input-output sensing and actuating.

With this work the goal was to extend this basic function to real-time application monitoring. It is the real-time supervision of the state of the application and of the evolution of the same that needs attention, both processing and timely reaction. In consequence a monitoring subsystem was needed to be created that follows these events and extracts state information in real-time: Periodically for state samples and upon their occurrence for events.

This way there are two types of application monitoring that were considered: **proactive monitoring** and **preventive monitoring**. In the first case there is a managing console to periodically monitor the health of the application and to identify problems and possible slowdowns consequences – **latency**. This approach has one problem coming from a constant and automatic surveillance creates an increase in network traffic leading to slowdowns. This may be a problem at a first glance but concerning the ASD application it only represents a residual increase to be considerate as a real problem. Now concerning the second type of monitoring, there is a search for failure points in the application and identification of potential weaknesses before they become real (possible catastrophic consequence) or need some repair. Therefore **preventive monitoring** analyzes the data transmitted in real time before any negative impact on the actual application is effective. Just like the previous type of monitoring this one presents some problems too namely the high costs that requires a prediction in terms of expenditure of resources but as like the previous one, this was considered a residual problem. Thus the solution were converging the two concepts addressed, using a monitoring system that

takes the initiative to verify periodically the application state as well as searching for failing points and identify potential weaknesses.

After considering the types of monitoring, some methods of collecting information to monitor were identified and collected in a list based on the construction of a guide to describe a performance philosophy to follow [23].

1.	Develop guide performance philosophy for the application
2.	Obtain monitoring data that already exists to compare with the guide developed before
3.	Establish specific performance targets to monitor the application in various modes of operation
4.	Perform tasks of analysis to determine the points that lead to performance targets to be achieved or not
5.	Design and develop graphics in view of the guide developed
6.	Constantly evaluate the guide developed for updates as the application is developing or being altered

Table 2: Steps to develop a guide to monitor and evaluate an application performance

So how such a guide can be defined? First of all the goals to achieve with the application must be set, such as response times or updates in real time. Then it is necessary to identify potential problem areas or situations and possible locations of probable congestion – **bottlenecks** - during the normal processing of the application. A more careful inspection can lead to the discovery of a component responsible for a poorer performance of the application. This is an iterative process because when the area in question is improved others may present possible congestion in turn. Therefore this process must be repeatedly systematically until the targets are reached. Finally, as a last step, a methodical process must be followed and have the attention focused on this as well so that after setting the targets the probable changes made to improve enforcement have real impact on its performance. Thus the most frequent tasks will deserve more attention than a task carried out less often even if it takes less processing time than the one that is done more often.

Continuing to monitor systematically brings together a range of information relating to various indicators of performance over the various nodes in the application.

After the congestion detected using this procedure exist the need to solve it and it goes through two possible strategies: change the environment in which the application

is inserted or improve the application itself. The best approach, more realistic and efficient is to improve the application and its components than the environment.

After this main step, determining some metrics is possible to evaluate the application performance. Some are linear while others represent statistical values, but the main goal is to have the most possible global comprehensive idea about the application performance in context [10][15].

i.	Set periods of use as: • Heavy, • Medium, • Light, • Very light.
ii.	Identify the time intervals in which the periods in *i.* occur.
iii.	List the transfers made during the periods in *i.*
iv.	Calculate the average arrival of messages (messages / second) for each period.
v.	Determine if there is another parallel workload to the main one that generates additional load.
vi.	List the criticality of each transaction message.
vii.	Define the: • Current average, • Expected average, • Maximum average for each response time of each transaction.
viii.	Determine the response time limit of the external systems to which the application is connected or dependent of (Future Work).
ix.	Set patterns of arrival of requests and the percentage of arrivals for different workloads.
x.	Determine the operational workload through the browsing patterns in the application made by the operator (Future Work).
xi.	Set response times to expected and current transactions in terms of: • Average, • Maximum, • Values of percentage.

Table 3: List of metrics to evaluate the application performance

This can be resumed into a specific metric:

Delivery Time: interval between the send event of message and the deliver event:

- How steady (constant) is the delivery delay as seen by one participant;
- How tight (simultaneous) is a delivery to multiple participants.

Thus it is possible to identify transactions in the system that response times are not acceptable and a viable solution is to re-examine the design of the application or its code. So a performance problem can be identified for delays or non-response by the system causing some component to reaches a deadline for a given limit stipulated response. This is possible with systematic monitoring in order to discover where the bottlenecks are.

In addition to these solutions is still possible to detect some performance issues with other techniques for improving performance. These may be the optimization of the code, caching strategy, distributed computing, self-tuning, among others. This is already encompassed within the other part that follows the monitoring that is the management and control of the application, but that falls outside the scope of this study.

3.6 Performance Management

There are several ways to monitor and manage application performance and maybe the first idea to come to mind is recording everything in a log. But besides the logs, there are other ways of collecting data for analysis: agents installed on application components, synthetic monitoring which simulates transactions and activity logs, sniffers, real user monitoring, and so on [22]. One of the most known methods of management access is the **Simple Network Management Protocol (SNMP)** [44] but the ASD application and its structure must be taking into account: it was developed using **JAVA.**

Real User Monitoring is a passive monitoring technology thus not affecting the normal system operations, despite recording all interactions by the user with the application [9][11]. This method differs from the synthetic (active) monitoring because it is based on the income and outcome traffic to perform the data analysis and measurement. This method determines if the application users are being served quickly and without errors and can identify the application component that is failing. This allows defining the current Quality of Service and detecting errors and slowdowns that could lead to catastrophic consequences. There is a disadvantage because the problems can only be discovered after they occur.

3.7 Summary

The three main concepts for this study were: **Mission Critical**, **Real Time** and **Performance**. ATM is a critical activity and that incurs the possible risk of loss of lives, and is thus a **Mission-Critical and Critical-Life activity**. To avoid some catastrophe is necessary to ensure the Quality of Service (QoS) through systems that provide services in a predictable way and with a full-time monitoring to assess their **reliability** and **availability of data** in order the QoS to be as high as possible. Three factors contributing to this are the **high system availability** and security on the **integrity** of the system to maintain the aforementioned QoS, and ultimately the **availability of data** in the system so that the ASD operators can have correct data to interact with.

Real-Time represents a temporal specifications counting on three main points: The timing of events related with the execution of real-time communication and computation actions – deadlines; the patterns of arrival of events – sporadic; and the definition of triggering conditions – time lattices. **Real-time** applications are in essence *reactive* or *responsive*. Most part of what they do is related with responding to events produced by the environment and by human users. They must respond according to pre-specified timings. The basic thing about timing is being able to specify action *durations* and event *positions* or *timestamps* in the timeline. Thus real-time tasks main classes can be described as: Aperiodic tasks - impossible to treat deterministically; Periodic tasks - the workhorse of static scheduling design; Sporadic tasks - serve applications that do not have a regular behaviour (request arrival is not periodic). That's why real-time applications can be classified as: Hard real-time where any failure to meet *timeliness* requirements may have a high cost associated. Every task must be always timely in order to avoid costly timing failures - also called time-critical. Not providing the service within the (deadline) interval carries a cost. The criticality of the application is given by the cost of failure versus the benefit of normal operation; Soft real-time applications accept occasional failures to meet timeliness requirements. The application should often enough be timely, ie, it can fail to meet timeliness specifications provided if they do not occur with too high a probability and great deviation or lateness degree. Many interactive systems are non real-time systems, but the truth is that a fair number of them should indeed have been designed as soft real-time systems where user requirements were respected, just like the ASD application; A Mission-critical real-time application where any failure to meet timeliness requirements may have a cost associated and the occasional failure to meet those requirements is considered an exception. It guarantees that timelines requirements are systematically met. However these applications are usually complex and/or the environment behaviour is not totally specified such that timeliness cannot be fully guaranteed. The occasional occurrence of timing faults is tolerated but should be considered exceptional as the service risks being provided near

or after the original recurrently deadline because of timing failures considerable degradation may take place.

Monitoring the application can thus be made through the use of **agents** located in various parts of it that will transmit status information to the **managing agent** in order to evaluate if everything is within the expected parameters, or if there is something that can or must be optimize. But this monitoring is only possible if **metrics** are defined by a **guide performance**. That will permit the study of the results and understand not only their meaning but assess too their usefulness. This process also includes the control and management of application and its components, but falls outside the scope of the current study.

Thus, it is possible to cast a more accurate and reliable picture of the general definition of **performance** and their properties - accuracy, completeness, cost and speed obtaining important data that would justify a systematic and permanent monitoring of an ATM application such like the Radar- Fallback ASD application.

Chapter 4

CONFIDENTIAL

Chapter 5

Technologies

5.1 Agents and Monitoring

The Java Virtual Machine provides a platform-independent way of executing code, by abstracting the differences between operating systems and CPU architectures. Java Runtime Environments are available for a wide variety of hardware and software combinations, making Java a very portable language [33]. A schematic representation is presented in Figure 20.

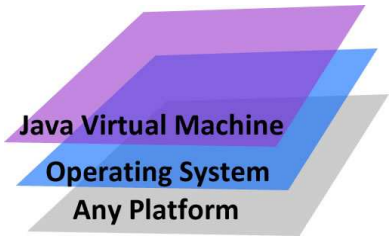

Figure 20: JVM handles translations

Java Management Extensions (JMX) is a specification for monitoring and managing Java applications [12][14][37]. It enables a generic management system to monitor an application; raise notifications when the application requires attention; and change the state of the application to remedy problems. Like other management standards, JMX is a public specification and many vendors of commonly used monitoring products support it. In the **JMX Java** technology the various resources are represented by objects called **MBeans** thus allowing the defined classes to be loaded and instantiated dynamically with application management or monitoring, as well as, breaking the application into components that can be swapped out. The MBeans development is based in the **Java Dynamic Management Kit (DMK)** [13][27] in which given a resource in the **Java** programming language - either an application, a service or an object representing a device - its instrumentation is how its management interface is exposed for management. Thus, the management interface is the set of attributes and operations that are visible to managers that need to interact with that resource. Therefore, instrumenting a resource makes it manageable.

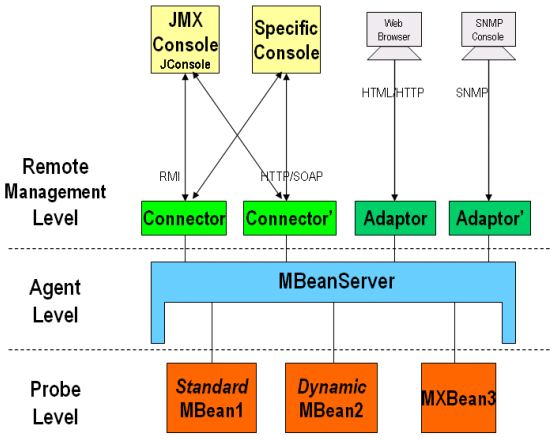

Figure 21: JMX Java Technology levels

In Figure 21 is schematically represented the various levels presented in the **JMX** architecture. The first level, the **probe level**, contains the **MBeans** (probs). This level is also known as **Instrumentation Level**. Then we have the level where exists the **MBean Server,** the **agent level**, serving as a broker between **MBeans** and applications, ie, the bond interface between the other two levels. Finally, the third level, **remote management level**, is where the remote applications can access the **MBean Server** through **connectors** using RMI or HTTP/SOAP protocols or **adapters** using, for instance, HTML/HTTP and SNMP protocols to connect to the management console. In the first case it is possible to establish full remote access and in the second is just an adaptation of a given API to another protocol.

The **JMX** enables a centralized management and monitoring in which **MBeans** act as wrappers for applications, components and other resources and all actions taken on these are allowed through the **MBean Server**. It keeps track of all the existing **MBeans**, exposing interfaces that allow their manipulation. It also dynamically loads **MBeans** (**MBeans** are added or removed when necessary), adds and finds components (nodes in the application) and sends notifications. This ensures scalability for both agents and manager as each application that acts as a server allows customization of classes through an own interface.

JMX agents are this way autonomous and intelligent because they embody in themselves tasks such as **polling** and **filtering** of events considered less important. This allows any reduction in network traffic and mainly makes management applications more robust by making services and devices to be treated as manageable objects.

JMX presents a more appropriate way to monitor and manage the ASD application because some disadvantages existe in using SNMP, even if it was combined with JMX. For instance, the power of expression of SNMP/SMI is limited. It is ideally suited to

describe and monitor numeric gauges and counters (scalar or tabular), but describing complex data is much more awkward. Some things that appear to be trivial in JMX, like for example, returning a thread stack trace, can reach incredible levels of complexity in SNMP. This way describing complex configuration can also rapidly become not feasible because the MIB that makes it possible to manage the JVM through SNMP is closed. The **JVM SNMP Agent** is also closed because it doesn't have a public API. The possibility of evolution for SNMP interfaces is thus limited [38].

5.1.1 Implementation Approaches for the MBeanServer

There are two ways of implementing the monitoring agent that access to resources and receive their notifications - the MBeanServer.

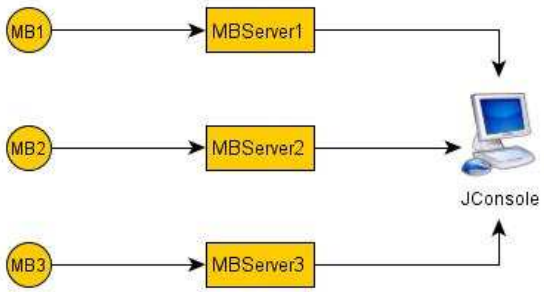

Figure 22 a): Each MBean has its own MBeanServer

In this first approach, pictured in Figure 23 a), for each MBean developed a dedicated server is created and it only manages this MBean and exposes its interface to the monitoring console.

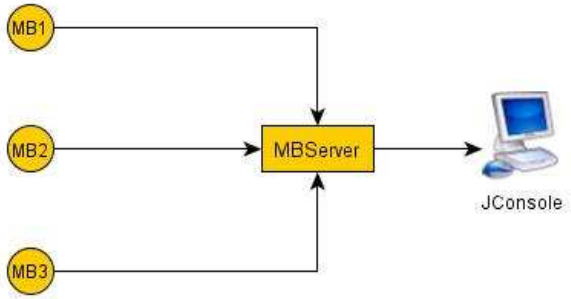

Figure 22 b): All MBeans share a single server

In the latter case, in Figure 22 b), all created MBeans will be registered on a single server. Therefore the server knows all the resources that are being monitored and allows them to be managed in the monitoring console.

Advantages and Disadvantages

Given the hypothesis in Figure 22 a), a distributed solution, in which each MBean represented a particular type of resource or set of pre-defined resources, if its server fails for some reason, it is always possible to continue monitoring all the other MBeans registered in other dedicated servers.

Thus, if there is a fault it does not affect any monitoring of the application but only a particular feature or feature set, continuing the management of others.

In the second case, represented in Figure 22 b) by a centralized approach with a single server to expose the interfaces of all the MBeans and the features that they characterize may seem not only more intuitive but also avoids creating multiple server instances when only one can do the same job.

The problem here is if the server fails, it is impossible to monitor any MBean that exists until the server recover from the failure.

There were three solutions:

- Have a centralized approach with a fallback. In that case when the server fails the other will soon be set up and register again all the MBeans;
- Implement a distributed approach where each MBean has its own server ignoring the overhead that this may cause in the application if it makes a negligible impact for the same;
- Use a composite version: distributed and centralized in which the relatable MBeans are aggregated on the same server and others on another server. It runs multiple servers but each one has its own related MBeans.

5.1.2 Connection with the management console

Implementation Approaches

There is the possibility, given the JMX technology, to use two modes of connection / communication with the management console: **connector** or **adapter**. These two modes are pointed out in Figure 23.

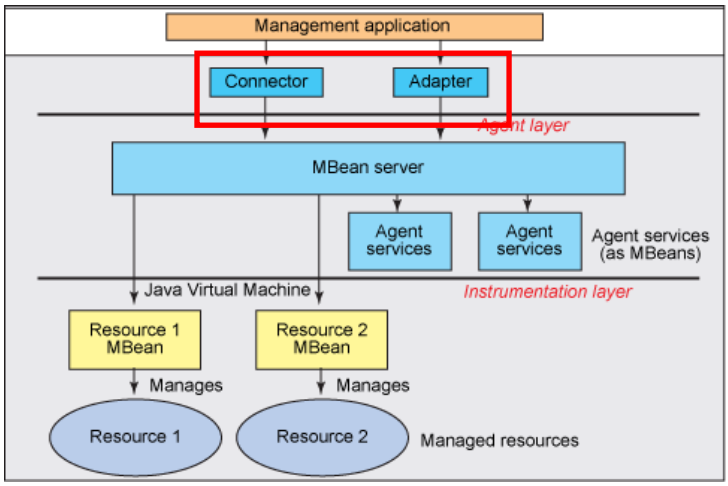

Figure 23: JMX architecture with a Connector and an Adapter

The two modes of connection presented in Figure 25 have some differences and are characterized as following:

Connector:

- Serves to establish the total remote access
- It is itself a management tool easy to use with the MBeanServer
- It supports the common protocols such as HTTP / SOAP and RMI
    - RMI (Remote Method Invocation) is easy to use and it is transparent to the users with objects to be passed as values, ie, the parameters of its methods are all developed in Java

Adapter:

- Adaptation of a certain API to other protocol
- It is more limited than a connector
- It only allows two different interfaces to work together, that is, converts the interface of a particular class to another interface expected by the client
- It is a linear translation
- It does not present an effective management

Advantages and Disadvantages

Because the JMX technology is developed in JAVA, like the ASD application, the best choice is using a connector, like presented in Figure 24, as it is not necessary any kind of translation between the two actors who manage the monitoring: MBeanServer and JConsole (Management Console), as well as the application itself where certain feature are being monitored.

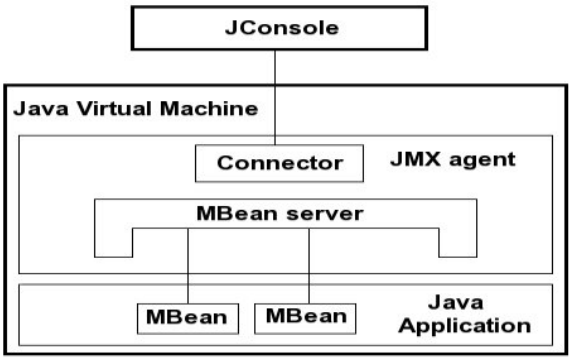

Figure 24: Overall picture of the JMX architecture with Connector

When the agent and server are created locally, and not remotely, and the MBeans are registered locally too, the communication through an RMI connector is automatically established. This is because the RMI (over TCP / IP) is a Java specific protocol, being dependent on language.

This approach also allows remote access but it is necessary to define the RMI connector specifically for remote access to the management console, i.e., it has to be configured in order to establish the connection between the MBean Server and the Management Console. All this is transparent to the user, as stated above, so everything seems to be done locally. Hence the local monitoring is done the same way but automatically without the need to configure the RMI connector.

5.2 Management Console

The Java Monitoring and Management Console (JConsole) tool allows accessing a great variety of information, such as [47]:

- Number of classes loaded and threads running
- Virtual machine uptime, system properties, and JVM input arguments
- Thread state, thread contention statistics, and stack trace of live threads
- Memory consumption
- Garbage collection statistics
- Low memory detection
- On-demand deadlock detection
- Operating system information

It uses the extensive instrumentation of the Java virtual machine to provide information on performance and resource consumption of applications running on the Java platform using Java Management Extension (JMX) technology. [45]

JConsole is a JMX-compliant GUI tool that when is connected to a Java application, it displays information about the application - including its memory usage, the running threads and the loaded classes. This information allows monitoring the behaviour of the application and the JVM, and can be useful in understanding performance problems, memory usage issues, hangs or deadlocks – all the things the API provides as a base. [46]

The JConsole was used for testing and will be used for maintenance of the framework. This way limit values that do not exist yet can be determine, concerning the application performance time in reading and processing the messages, for example. The tests can be performed locally but it will be remotely used in the operational system.

5.3 Summary

The Java virtual machine (JVM) is instrumented for monitoring and management, providing built-in ("out-of-the-box") management capabilities for both remote and local access. The platform instrumentation also provides information on performance, resource consumption, and the JVM and logging settings of applications running on the Java platform. It includes a platform MBean server (management agent) and platform MBeans that a JMX management application can use. Its API provides access to information a diversity of information.

To obtain the information from the resources gathered in the MBean through the MBean Server and displaying these in real-time and in a personalised way to get the most important and relevant information about the application performance exist the JConsole. This GUI tool gives a range of information not only about the state of the application but about the state of the JVM itself like Heap and CPU usage information, as well as, detecting deadlocks which is very important to improve the application performance. The JConsole not only allows a monitoring the application but also manage the same through performing Garbage Collections (GCs) or other operations defined by the console operator.

Chapter 6

CONFIDENTIAL

Chapter 7

CONFIDENTIAL

Chapter 8

Conclusion and Future Work

Having as the main goal of this study to develop a Real-Time monitoring framework to obtain performance results from the Air Situation Display application, this was accomplished.

The determination of ‘what to’ quantify about the application performance was established and then the capacity of ‘how to’ quantify. This was achieved through the study of the framework, its design and implementation. Understanding these factors the monitoring framework design and implementation was done in order to be added to the existing framework without changing it too much and keeping the desired modularity to the monitoring function. To support that, the choice of using the JMX technology was very useful because it allowed an intuitive manner of developing a real-time monitoring conjugating the fact the ASD application was developed in Java and the fact that the JMX is also a Java technology with the necessary elements available to develop a good monitoring framework.

After this, the behaviour of the application was tested in a variety of test scenarios in order to obtain results to compare to what was determined as expected in this kind of application. There were not any other values to compare to, so there was the need to test the application performance for basic actions established a priori as the main actions possible to happen in the application. For that, a specific type of input messages the application reads and processes was chosen as the more significant to serve for testing the monitoring framework. Thus, metrics were defined based in this type of messages that bring information about the Tracks representing the aircrafts and its characteristics to the Air Traffic Controller to manage through the Air Situation Display HMI. These results and consequent metrics calculation were obtained to establish the ASD performance when reading and processing these messages from the input (link) until the output (HMI), that is to say, measuring the performance for the entire path a message follow through the application components. From this overall view of the application performance it is possible to see if the application is performing in the expected time

interval, detecting also if some messages are discarded when some values overpass the defined Deadline of the five seconds of the Radar rotation.

The values obtained in the tests showed that the ASD was working, for the test scenarios, in a very good comfort zone. Some messages may seam to be lost because they are registered as processed but they don't have a time of processing in the end. This happens because they represent the messages that update only the Model and not the View. This means that putting the agent only at the very end of the chain may not be the right answer to the problem and more agents need to be placed along the data flow to get more accurate results in order to not *lose* messages that in fact were well processed. This also means that now it is possible to have values to characterize the limit the application performance can reach in order to keep working well and without problems. And if these limits are surpassed or the average of the values begins to exceed the values established with this study, notifications are generated by the monitoring framework in order to alert for that case.

The ASD application has then proven to be a deterministic application as expected, even if the determinism in this application is located in a determined interval obtained by the monitoring results, in other words, it has a deterministic interval where it operates with good performance with a set of rules and indicators which defines boundaries and tells what to do to be successful in solving problems within these boundaries. About the phase two of tests performed, the results identified indicate that exists a possible problem with the performance of the Garbage Collections made by the application itself besides the ones performed by the JVM. This will care of attention in the future work. With this work done, it is possible to expand the monitoring to other type of messages and to all the ASD Interface Links.

However, being the ATM a large-scale system inserted in an unpredictable environment, defining the application with synchronous model is very difficult to achieve in a mission-critical system like this. This way attending to the obtained values in the tests and to tolerate partial synchrony of the system while securing timeliness proprieties, a better model to characterize the application is a Quasi-Synchronous Model. That is, it can be reliable with bounds on response time can be defined during limited periods of its operation [20].

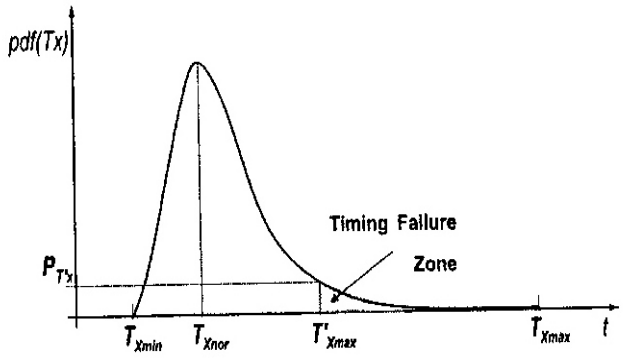

Figure 62: Distribution of termination times

In Figure 62 is represented the probability density function of a message to be delivery and processed. The worst-case termination time corresponds to the Radar Rotation Time that is five seconds, representing the Deadline to be respected. But this value is much higher than the average case, such that it becomes useless. So, the assumption of a shorter, artificial bound increases the expected responsiveness.

Thus the next step in evolving the monitoring framework already implemented goes through the monitoring of another type of ASTERIX messages, this time Category 62 messages. Those messages contain the information needed in the Ground Situation Display (GSD) which is the application used in the Tower by the Tower Air Traffic Controllers. It is the equivalent to the ASD but for the airport traffic control instead of the en-route traffic control.

The GSD is an independent application from the ASD even if they belong to the same applications family tree. This way, there will be the need to analyse the design and implementation of the GSD to understand the different points from the ASD (chain components and the data flow), because in general they are much the same, for once, the flow of data arriving to the GSD are inferior to the flow of data in the ASD. But some adaptation will be needed to be performed to the actual monitoring framework in order to accommodate the monitoring for the GSD.

The GSD has the same characteristics already studied in the ASD, representing this way a Mission-Critical Real-Time application. So the same metrics and results of this kind of application will be expected with the values collected in the monitoring of this application. Of course that the metrics established for the ASD may suffer some adaptation as the rest, but the performance evaluation to the GSD application consists in the same performance with Soft Real-Time deadlines; limit values that will characterize the comfort zone to work and when an alert should be sent to the management console informing of abnormal behaviour and poor performance by the GSD application. Those factors will be determined by the basic actions the application perform since the input of

a message until the output be represented in the HMI, leading to collecting values to identify the limits of a good performance (the deterministic interval), having in this work a good base to follow up and develop even more the framework.