XML compression algorithms benchmark on ARINC 633 data

Author: Anna Gasparín Forné
Director: Eriza Fazli
Supervisor: Xavier Hesselbach Serra

2014 Munich, Germany
# Table of Contents

Table of Contents .................................................................................................................. 1
Resum del Projecte .................................................................................................................. 4
Resumen del Proyecto .............................................................................................................. 5
Abstract ................................................................................................................................ 6
Abbreviations .......................................................................................................................... 9
1 Introduction ......................................................................................................................... 10
  1.1 Context of the project .................................................................................................... 10
  1.2 Goals ............................................................................................................................. 11
  1.3 Structure of the report ................................................................................................... 11
2 Background ......................................................................................................................... 13
  2.1 SESAR .......................................................................................................................... 13
    2.1.1 SESAR goals ........................................................................................................... 13
    2.1.2 Partners & budget .................................................................................................. 15
    2.1.3 Structure of the SESAR project .......................................................................... 16
  2.2 SWIM ............................................................................................................................ 17
    2.2.1 Communication today ......................................................................................... 17
    2.2.2 Concept ................................................................................................................ 18
    2.2.3 Characteristics ...................................................................................................... 18
  2.3 ARINC 633 ................................................................................................................... 21
3 XML .................................................................................................................................. 22
  3.1 Definition ...................................................................................................................... 22
  3.2 Advantages & drawbacks ............................................................................................. 24
  3.3 XML Schema ............................................................................................................... 26
  3.4 Schemas used in ARINC 633 ....................................................................................... 27
4 Compression ....................................................................................................................... 28
  4.1 Compressors classification .......................................................................................... 28
  4.2 Parameters to take into account when choosing a compressor ..................................... 30
  4.3 Chosen compressors ..................................................................................................... 31
5 Compressors ....................................................................................................................... 32
  5.1 Gzip .............................................................................................................................. 32
    5.1.1 Introduction ......................................................................................................... 32
    5.1.2 Deflate algorithm ............................................................................................... 32
  5.2 XMill ............................................................................................................................. 35
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2.1</td>
<td>Introduction</td>
<td>35</td>
</tr>
<tr>
<td>5.2.2</td>
<td>XMl architecture</td>
<td>35</td>
</tr>
<tr>
<td>5.3</td>
<td>XMLPPM</td>
<td>39</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Prediction by Partial Matching</td>
<td>39</td>
</tr>
<tr>
<td>5.3.2</td>
<td>XMLPPM compression</td>
<td>39</td>
</tr>
<tr>
<td>5.4</td>
<td>Fast Infoset</td>
<td>41</td>
</tr>
<tr>
<td>5.4.1</td>
<td>Introduction</td>
<td>41</td>
</tr>
<tr>
<td>5.4.2</td>
<td>XML Information Set</td>
<td>41</td>
</tr>
<tr>
<td>5.4.3</td>
<td>Fast Infoset Documents</td>
<td>42</td>
</tr>
<tr>
<td>5.4.4</td>
<td>Advantages and Properties of the Fast Infoset</td>
<td>43</td>
</tr>
<tr>
<td>5.5</td>
<td>EXI</td>
<td>45</td>
</tr>
<tr>
<td>5.5.1</td>
<td>Introduction</td>
<td>45</td>
</tr>
<tr>
<td>5.5.2</td>
<td>Codification</td>
<td>45</td>
</tr>
<tr>
<td>5.5.3</td>
<td>EXI compression</td>
<td>47</td>
</tr>
<tr>
<td>6</td>
<td>Benchmark</td>
<td>51</td>
</tr>
<tr>
<td>6.1</td>
<td>Evaluated parameters</td>
<td>51</td>
</tr>
<tr>
<td>6.2</td>
<td>Input files</td>
<td>51</td>
</tr>
<tr>
<td>6.2.1</td>
<td>ARINC 633 examples</td>
<td>51</td>
</tr>
<tr>
<td>6.2.2</td>
<td>Flight Plan files</td>
<td>52</td>
</tr>
<tr>
<td>6.2.3</td>
<td>Sample weather files</td>
<td>52</td>
</tr>
<tr>
<td>7</td>
<td>Results</td>
<td>54</td>
</tr>
<tr>
<td>7.1</td>
<td>Compression performance</td>
<td>55</td>
</tr>
<tr>
<td>7.1.1</td>
<td>ARINC 633 examples</td>
<td>55</td>
</tr>
<tr>
<td>7.1.2</td>
<td>Flight Plan files</td>
<td>56</td>
</tr>
<tr>
<td>7.1.3</td>
<td>Weather files</td>
<td>59</td>
</tr>
<tr>
<td>7.2</td>
<td>CPU consumption</td>
<td>61</td>
</tr>
<tr>
<td>7.2.1</td>
<td>ARINC 633 examples</td>
<td>61</td>
</tr>
<tr>
<td>7.2.2</td>
<td>Flight Plan files</td>
<td>62</td>
</tr>
<tr>
<td>8</td>
<td>Conclusions</td>
<td>65</td>
</tr>
<tr>
<td>9</td>
<td>Bibliography</td>
<td>66</td>
</tr>
<tr>
<td>Appendix</td>
<td></td>
<td>68</td>
</tr>
<tr>
<td>A</td>
<td>Benchmark</td>
<td>68</td>
</tr>
<tr>
<td>A.1</td>
<td>Generation of input files</td>
<td>69</td>
</tr>
<tr>
<td>A.2</td>
<td>Tools</td>
<td>70</td>
</tr>
<tr>
<td>A.3</td>
<td>PC characteristics</td>
<td>70</td>
</tr>
</tbody>
</table>
B  Compaction performance ................................................................. 71
B.1  ARINC 633 examples ................................................................. 71
B.2  Flight Plan files ................................................................. 72
B.3  Weather files ................................................................. 73
C  CPU consumption for weather files ........................................ 75
Resum del Projecte

El creixent tràfic aeri a nivell Europeu i les limitacions de l’actual infraestructura aeroportuària fan necessari una modernització de la Gestió del Trànsit Aeri. Aquesta necessitat ha impulsat el naixement de SESAR; el projecte tecnològic que descriu les actuacions necessàries per portar a terme aquesta modernització. Un dels pilars fonamentals de SESAR es basa en la gestió de la informació de tot el sistema en conjunt. El fet que tota la informació hagi de ser compartida per totes les parts implicades en la comunicació fa necessari l’ús d’estàndards. Un dels estàndards proposats és l’ARINC 633 el qual fa referència a la comunicació entre els avions i les seves aerolínies. Tots aquests estàndards utilitzen XML per a l’estructuració de la informació a enviar. XML ha esdevingut un llenguatge àmpliament utilitzat a causa dels molts beneficis que aporta. Tot i això, aquest llenguatge presenta un desavantatge principal: la seva verbositat. Afegint a aquest inconvenient el fet que la informació és enviada a través d’enllaços per satèl·lit, la compressió esdevé un element clau per tal de poder fer la comunicació viable.

Aquest projecte presenta una comparació de diferents tècniques de compressió aplicades sobre fitxers basats en l’estàndard ARINC 633. Per tal de poder dur a terme aquesta comparació s’ha creat un marc de treball que va des de la creació de fitxers mostra fins a la mesura de determinats paràmetres; tot això passant per la compressió, descompressió i validació dels fitxers testejats. Mesurant el radi de compressió i el consum de CPU s’ha pogut determinar com l’ús de compressors específics per a fitxers XML pot representar un benefici respecte a compressors de propòsit general. La comparació també mostra com és pot treure profit de l’ús dels esquemes XML per tal d’aconseguir incrementar notablement el radi de compressió.
Resumen del Proyecto

El creciente tráfico aéreo a nivel europeo y las limitaciones de la actual infraestructura aeroportuaria hacen necesario una modernización de la Gestión del Tráfico Aéreo. Esta necesidad ha impulsado el nacimiento de SESAR, el proyecto tecnológico que describe las actuaciones necesarias para llevar a cabo esta modernización. Uno de los pilares fundamentales de SESAR se basa en la gestión de la información de todo el sistema en su conjunto. El hecho de que toda la información deba ser compartida por todas las partes implicadas en la comunicación hace necesario el uso de estándares. Uno de los estándares propuestos es ARINC 633 el cual hace referencia a la comunicación entre los aviones y sus aerolíneas. Todos estos estándares utilizan XML para la estructuración de la información a enviar. XML se ha convertido en un lenguaje ampliamente utilizado debido a los muchos beneficios que aporta. Sin embargo, este lenguaje presenta una desventaja principal: su verbosidad. Añadiendo a este inconveniente el hecho de que la información es enviada a través de enlaces por satélite, la compresión acontece un elemento clave para poder hacer la comunicación viable.

Este proyecto presenta una comparación de diferentes técnicas de compresión aplicadas sobre archivos basados en el estándar ARINC 633. Para poder llevar a cabo dicha comparación se ha creado un marco de trabajo que va desde la creación de ficheros muestra hasta la medida de determinados parámetros, todo ello pasando por la compresión, descompresión y validación de los ficheros testeados. Midiendo el radio de compresión y el consumo de CPU se ha podido determinar cómo el uso de compresores específicos para archivos XML puede representar un beneficio respecto a compresores de propósito general. La comparación también muestra cómo se puede sacar provecho del uso de los esquemas XML para conseguir incrementar notablemente el radio de compresión.
Abstract

The increasing air traffic at European level and the limitations of the current airport infrastructure make necessary the modernization of the European Air Traffic Management. This need has prompted the creation of SESAR, the technological project that describes the necessary steps to carry out the modernization of the actual European ATM. One of the cornerstones of SESAR is based on the information management of the entire system as a whole. The fact that all information should be shared by all parties involved in the communication makes necessary the use of standards. One of the proposed standards is ARINC 633 which refers to communication between aircraft and its airlines. All these standards use XML for information encoding. XML has been widely adopted due to the many benefits it provides. However, this language has a main drawback: its verbosity. Adding to this problem the fact that the information is sent via satellite links, compression become a key element to make the communication viable.

The current report presents a benchmark of different compression techniques applied to files based on ARINC 633 standard. To carry out this benchmark it has been implemented a complete framework which consists of sample file generation, compression, decompression, validation and performance measurement. The measurement of the compression ratio and CPU consumption shows how the use of specific compressors for XML files increase the compression performance compared with general purpose compressors. The comparison also shows that it is possible to take advantage of the use of schemas to achieve a noticeably increase in the compression ratio.
List of Figures

Figure 1: SESAR’s budget .............................................................................................................. 15
Figure 2: SESAR phases .................................................................................................................. 16
Figure 3: Sharing information today: point-to-point communications ........................................... 17
Figure 4: Sharing information tomorrow: many-to-many exchange of data ......................... 19
Figure 5: Net centric information view point ................................................................................. 19
Figure 6: Communication between airlines and aircrafts .......................................................... 21
Figure 7: XML example .................................................................................................................. 23
Figure 8: Schema example ............................................................................................................. 26
Figure 9: LZ77 compression .......................................................................................................... 33
Figure 10: Huffman Encoding ...................................................................................................... 33
Figure 11: Deflate compression ..................................................................................................... 34
Figure 12: XMIL architecture ....................................................................................................... 36
Figure 13: Fast Infoset ...................................................................................................................... 42
Figure 15: EXI header .................................................................................................................... 46
Figure 16: EXI compression .......................................................................................................... 48
Figure 17: Channeling ..................................................................................................................... 49
Figure 18: Compression performance of ARINC examples ......................................................... 55
Figure 19: Compression performance of small Flight Plan files ................................................ 56
Figure 20: Compression performance of medium size Flight Plan files .................................. 57
Figure 21: Compression performance of big Flight Plan files .................................................... 58
Figure 22: Compression performance of small Weather files ..................................................... 59
Figure 23: Compression performance of medium size Weather files ...................................... 60
Figure 24: Compression performance of big Weather files ......................................................... 60
Figure 25: CPU consumption of ARINC 633 examples ............................................................. 61
Figure 26: CPU consumption of small Flight Plan files .............................................................. 62
Figure 27: CPU consumption of medium size Flight Plan files ................................................ 63
Figure 34: Generation of input files ................................................................. 69
Figure 35: Compression performance of ARINC 633 examples for all combinations .................................................................................................................. 71
Figure 36: Compression performance of small Flight Plan files for all combinations ................................................................. 72
Figure 37: Compression performance of medium Flight Plan files for all combinations ......................................................................................................................................... 72
Figure 38: Compression performance of big Flight Plan files for all combinations .................................................................................................................. 73
Figure 39: Compression performance of small Weather files for all combinations .................................................................................................................. 73
Figure 40: Compression performance of medium Weather files for all combinations .................................................................................................................. 74
Figure 41: Compression performance of big Weather files for all combinations .................................................................................................................. 74
Figure 42: CPU consumption of weather small files ................................................................................................................................. 75
Figure 43: CPU consumption of weather medium size files ................................................................................................................................. 76
Figure 44: CPU consumption of weather big files ................................................................................................................................. 77
Abbreviations

SES – Single European Sky

SESAR – Single European Sky ATM Research

ATM – Air Traffic Management

SWIM – System Wide Information Management

AOC – Aeronautical Operational Control

XML – Extensible Markup Language

SAX – Simple API for XML

PPM – Prediction by Partial Matching

EXI – Efficient XML Interchange

FI – Fast Infoset

SOA – Service Oriented Architecture

PCDATA – Parsed Character Data

AIXM – Aeronautical Information Exchange Model

WXXM – Weather Exchange Model

JAXB – Java Architecture for XML Blinding
1 Introduction

1.1 Context of the project

Contrary to the United States, Europe does not have a single sky, one in which air navigation is managed at European level. Each country is responsible for its own sky, with handovers between controllers and technical systems when crossing borders.

Furthermore, European airspace is among the busiest in the world with over 33,000 flights on busy days and high airport density. This makes air traffic control even more complex.

Today situation is well handled but European air traffic is expected to grow year after year. This growing traffic is the reason why the European Commission proposed the Single European Sky (SES) initiative. This initiative aims to organize airspace into functional blocks according to traffic flows rather than to national borders.

Inside SES, SESAR program is found. The Single European Sky ATM Research project represents the technological part of the SES. SESAR is intended to completely overhaul the European airspace and its Air Traffic Management using advanced technology.

The modernization of the European ATM implies a complete change in the way of managing information. This change is contained in the SWIM project, one of the basic pillars of SESAR. SWIM aims to enable seamless information interchange between all providers and users of ATM information. Its architecture will allow the exchange of data and ATM services across the whole European ATM system creating a common platform for information sharing.

To achieve all interoperable goals of SWIM it is essential that the exchange of ATM data is built on a foundation of open standards. ARINC 633 is likely to be one of these standards. ARINC 633 describes the structure of the AOC messages, those that enable communication between aircrafts and airlines through satellite links.

These entire standards will use XML for structuring data. XML has been widely adopted for message exchanging because of its benefits. However, XML has a main drawback; its verbosity. Adding to this drawback the fact that satellite links are really
slow, compression arises as an essential solution to make ARINC files’ exchanging viable.

1.2 Goals

The aim of this thesis is to find the best compressor for XML files based on the standard ARINC 633.

First, a study of the state-of-the-art of XML compression techniques has been performed. This study allows choosing few compressors in order to compare their performance when compressing ARINC 633 files.

Then a compression benchmarking campaign has been carried out. For this purpose, files based on ARINC 633 standard have been created. Once the samples are created, it is possible to make all necessary measurements by compressing and decompressing files.

Within the benchmark, compression performance as well as CPU consumption has been evaluated for each compressor.

1.3 Structure of the report

This document is structured in 8 chapters.

Chapter 1, the current chapter, provides a brief description of the context of the project. This chapter also explains the goals of the project and how is planned to achieve them.

Chapter 2 presents the context of the project in a more detailed way; explaining the SESAR and SWIM projects and describing the ARINC 633 standard.

Chapter 3 gives a description of the XML language.

Chapter 4 shows how available compressors can be classified. This chapter also exposes which compressors have been chosen and the reason of this choice.

Chapter 5 describes the features of each of the compressors chosen.
Chapter 6 explains the working of the current benchmark, describing which parameters are evaluated and which files are tested.

Chapter 7 exposes all results found through the benchmark.

Chapter 8 presents the conclusions from the work done in this thesis.
2 Background

2.1 SESAR

The European air transport demand has been growing steadily between five and seven percent over recent years. This rate of growth is expected to continue for the foreseeable future. Moreover, the fact that European sky is divided into national regions each one using its own technologies hinders its scalability.

As a response to this problem, the European Commission promotes the creation of the Single European Sky to create a legislative framework for European aviation. It is designed to help European airspace accommodate the increasing air traffic flows, at the same time as cutting costs and improving its performance.

On the technology side, SES is supported by the Single European Sky ATM Research (SESAR) [1] program which will provide advanced technologies and procedures with a view to modernizing and optimizing the future European ATM network.

ATM [2] refers to all processes, procedures and resources which come into play to make sure that aircraft are safely guided in the skies and on the ground, including air traffic control (ATC), aeronautical meteorology, air navigation systems (aids to navigation), Air Space Management (ASM), Air Traffic Services (ATS), and Air Traffic Flow Management (ATFM), or Air Traffic Flow and Capacity Management (ATFCM).

2.1.1 SESAR goals

The actual ATM system needs a modernization. As aviation is a very cautious industry, changes in operations are made carefully and after a lot of testing and consultation. This is the reason why technology developments onboard aircrafts have often outpaced those in the air traffic control centers. Jets produced in the 21st century are flying along routes that were, in some cases, defined by the placement of radar stations in the 1940s and 1950s.

SESAR will provide Europe with a high-performance air traffic control infrastructure which will ensure the safety and fluidity of air transport over the next thirty years, will
make flying more environmentally friendly and will reduce the costs of air traffic management.

All these goals are described in the SESAR ATM Master Plan. The European ATM Master plan is SESAR's roadmap for driving the European ATM modernization program. The plan contains the essential operational and technological changes required from all stakeholders to achieve the performance objectives set by the Single European Sky (SES). The SESAR ATM Master Plan has the following goals for 2020:

- Supplying EU skies with three times more capacity to handle traffic
- Improving safety performance by a factor of 10
- Reducing delays both on the ground and in the air
- Cutting ATM costs by a 50%
- Providing a 10% reduction in the environmental impact per flight

SESAR aims to accomplish these goals by developing the following key features:

- Moving from airspace towards 4D trajectory-based operations
- Using new and innovative technologies and including a high degree of automation
- Dynamic airspace management to enhance coordination between aviation authorities
- Promoting the collaborative network planning
- Moving towards a network centric approach through the System Wide Information Management (SWIM) that represents the technological component of SESAR
2.1.2 Partners & budget

As the technological pillar of the SES initiative, SESAR is the mechanism which seeks to coordinate and concentrate all EU research and development activities in ATM, pooling together wealth experts to develop the new generation of ATM.

SESAR was founded by the European Union and Eurocontrol. However, today this project unites several members [3] representing the whole aviation community. These include airport operators (AENA and SEAC – a consortium made up of BAA Airports Ltd, Flughafen München GmbH, Fraport AG, Schiphol Nederland B.V., Aéroports de Paris and Flughafen Zürich AG) and air navigation service providers (DFS, DSNA, ENAV NATS and NORACON – a consortium consisting of Austro Control (Austria), AVINOR (Norway), EANS (Estonia), Finavia (Finland), IAA (Ireland), ISAVIA (Iceland), LFV (Sweden) and Navair (Denmark)), as well as equipment makers (Frequentis, Honeywell, Indra, NATMIG, the SELEX Consortium and Thales) and aircraft builders (Airbus, Alenia Aeronautica). Several of these members are made up of groups of companies, sometimes with subsidiaries or partners, meaning that a total of 70 organisations participate.

All the research will be carried out during the development phase. The total estimated cost of the phase is € 2.1 billion to be shared equally between the European Community, Eurocontrol and the industry.

Figure 1: SESAR’s budget
2.1.3 Structure of the SESAR project

SESAR is a three phase program. These phases are [4]:

- **Definition Phase** (2004-2008): in this first phase the ATM master plan is delivered. This plan defines the content, the development and deployment plans of the next generation of ATM systems.

- **Development phase** (2008-2013): the purpose of this phase is to carry out through defined and coherent R&D activities in order to develop new equipments, systems or standards that will be deployed in the next phase.

- **Deployment phase** (2014-2020): during the deployment phase it is built the new infrastructure at a wide scale both in Europe and in partner countries.
2.2 SWIM

SWIM [5] represents a complete shift in how the information is managed across the whole European ATM system. This program is designed to make easier sharing ATM system information, such as weather information, flight data, airport operational status, etc. The program arises from the need for network central operations to meet future traffic demands.

SWIM will be “the intranet of the air” where the different stakeholders (aircraft, airports or national space managers, among others) share the ATM information over a common platform.

2.2.1 Communication today

Currently, the ATM environment is characterized by point-to-point communication between the various stakeholders. Within ATM are used many custom communication protocols, each with their own self-contained information systems: on board the aircraft, in the air traffic control centre, etc. Each of these interfaces is custom designed, developed, managed, and maintained individually and locally at a significant cost. Moreover the way ATM information is structured, and the way it is provided and used are specific for most of the ATM systems.

In order to handle the expected increasing capacity demands, economic pressures and increasing attention to environmental impact, the SESAR program is developing a new ATM system for Europe.

![Figure 3: Sharing information today: point-to-point communications](image-url)
2.2.2 Concept

“SWIM consists of standards, infrastructure, and governance that enable the management of Air Traffic Management related information and its exchange between qualified parties via interoperable services” (ICAO Working Paper).

SWIM most important goal is to improve collaborative decision making, providing better quality information to the right people at the right moment. Systems will be able to request and receive information whenever they need it, subscribe for automatic receipt and publish information and services as appropriate. Improving the way of sharing and processing information helps to reduce duplication of tasks and to increase flexibility.

SWIM aims to improve data exchange between various applications in different domains such as flight data management, weather and aeronautical information management. Thus the main challenge in the migration to the SWIM architecture is the heterogeneous nature of ATM in Europe.

2.2.3 Characteristics

In a traditional environment, decisions are often arbitrary and the effects of the decisions are not completely transparent to the partners involved. The more complete the information that is being shared and the easier its accessibility to the community involved, the higher the benefit potential is. Furthermore, increasing common situational awareness by allowing more decision makers to access the same information improves aviation safety.

SWIM enables the concept of net-centric [6] ATM, implementing a Service Oriented Architecture for the ATM network. Net-centricity promotes a many-to-many exchange of data, enabling a multiplicity of users and applications to make use of the same data. The aim of a net-centric system is to make all data visible, available and usable, when needed and where needed, to accelerate and improve the decision making process.
The most common implementation of a SOA is using web services. Web services make functional building-blocks accessible over standard Internet protocols independent of platforms and programming languages. Other systems interact with the web service using SOAP messages, typically transported using HTTP with an XML serialization.

SWIM will provide a common interface framework, reducing the operation and maintenance costs of current interfaces. Most existing ATM systems have been designed with interfaces for specific data exchanges limited by proprietary data formats.
and protocols. With SWIM is intended that new systems interface with each other via SWIM-compliant interfaces, thereby reducing future data interface development costs.

To achieve the interoperable goals of SWIM it is essential that the exchange of ATM data is built on a foundation of open standards. SWIM uses XML for encoding data to exchange. One example is AIXM (Aeronautical Information Exchange Model). AIXM is used to send aeronautical information to others in the form of XML encoded data.
2.3 ARINC 633

The actual data link system ACARS (Aircraft Communications Addressing and Reporting System) used for the transmission of messages between aircraft and ground stations needs to be upgraded to give significant additional capacity to its users. This is why the communications network technology onboard aircraft is evolving to include Ethernet and TCP/IP based networks.

ARINC 633 is the air-ground protocol for ACARS and IP networks used for AOC messages exchanging. Messages are sent through satellite links.

AOC stands for Aeronautical Operational Control and comprises all those messages used for the communication between an aircraft with its airline. These messages have been always user defined because were exchanged between an airline and their own aircraft. So, controlling both ends, there was no need for standardization. But with the SWIM architecture, where all the elements involved in the communication will have access to the all information, a need to standardize appears.

ARINC 633 uses XML language for encoding information like weather, flight plan information, fuel status, etc. This standard provides a schema for each type of data. Schemas specify the structure as well as a set of tags which XML files must to follow in order to be valid. The use of the schemas is like the use of a vocabulary that will be common for all partners; thus this information can be exchanged between all of them.

![Figure 6: Communication between airlines and aircrafts](image-url)
3 XML

3.1 Definition

XML [7,8] is a markup language that defines a set of rules for encoding documents. A markup language is a mechanism to identify structures in a document. In fact, XML is being designed to deliver structured content over the web.

XML stands for Extensible Markup Language where extensible means that the language is a shell, or skeleton that can be extended by anyone who wants to create additional ways to use XML. The meaning of markup is that XML’s primary task is to give definition to text and symbols and language means that XML is a method of presenting information that has accepted rules and formats.

XML specifies neither semantics nor a tag set. This language provides a facility to define tags and structural relationships between them. In fact, XML lets users to make up the tags that they need. These tags must be organized according to certain general principles but they are quite flexible in their meaning. Thereby, since there is no predefined tag set, there can not be any preconceived semantics.

XML documents are structures which are organized in a hierarchical “tree-like” manner. The documents have two basic sections: the structure (skeleton) or metadata of the document and the data itself. The metadata is a tree of nodes where each node is a container for the data.

Below there is an example of one of the XML files used in the benchmark:
<?xml version="1.0" encoding="UTF-8"?>
<DORACK xmlns="http://aeec.aviation-ia.net/633"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://aeec.aviation-ia.net/633 DEICING.xsd">
<!--
*****************************************************************************
**-->
<!--* example dated 01JUL2009 - version 2.0--> 
<!--
*****************************************************************************
**-->

<M633Header versionNumber="2" timestamp="2006-08-13T09:30:47.080"/>

<M633SupplementaryHeader>

<Flight flightOriginDate="2006-08-13"
scheduledTimeOfDeparture="2006-08-13T10:20:00">

<FlightIdentification>

<FlightIdentifier>SAS345</FlightIdentifier>

<FlightNumber airlineIATACode="SA" number="345">

<CommercialFlightNumber>SA345</CommercialFlightNumber>

</FlightNumber>

</FlightIdentification>

<DepartureAirport>

<AirportICAOCode>EKCH</AirportICAOCode>

<AirportIATACode>CPH</AirportIATACode>

</DepartureAirport>

<ArrivalAirport>

<AirportICAOCode>LIRF</AirportICAOCode>

<AirportIATACode>FCO</AirportIATACode>

</ArrivalAirport>

</Flight>

<Aircraft aircraftRegistration="LJT26">

<AircraftModel>

<AircraftICAOType>MD83</AircraftICAOType>

</AircraftModel>

</Aircraft>

</M633SupplementaryHeader>

</DORACK>

Figure 7: XML example
3.2 Advantages & drawbacks

XML is one of the most widely-used formats for sharing structured information today: between programs, between people, between computers and people; both locally and across networks.

XML has been widely adopted as a data representation format because its benefits which are:

- **Universal language for describing structured data**: XML allows describing and organizing information in a way what is easily understandable for both humans and computers therefore the information will be more accessible and reusable. XML files are plain text files what brings simplicity at the time of developing new applications.

- **Interchange of data among applications**: Because XML is nonproprietary and easy to read and write, it is an excellent format for the interchange of data among different applications. XML is not encumbered by copyright, patent, trade secret, or any other sort of intellectual property restrictions. It has been designed to be extremely expressive and very well structured while at the same time being easy for both human beings and computer programs to read and write. Thus, it is an obvious choice for exchange languages as XML can be exchanged between systems that were clearly never designed to do so.

- **Platform independent**: XML is not tied to any programming language, operating system or software vendor. In fact, it is fairly straightforward to produce or consume XML using a variety of programming languages. It makes XML very useful as a means for achieving interoperability between different programming platforms and operating systems.

- **Extensibility**: as it has been explained before, there is any fixed set of tags. Users can create as many tags as they need.

- **Separation of the content from any notion of presentation**: if one application wants to take the information and present it in a certain way there is no notion of how to do so. XML tags describe meaning not presentation. The motto of HTML is: "I know how it looks", whereas the motto of XML is: "I know what it means, and you tell me how it should look." The look and feel of an XML document can
be controlled by XSL style sheets, allowing the look of a document (or of a complete Web site) to be changed without touching the content of the document. Multiple views or presentations of the same content are easily rendered.

On the other hand, XML also has some drawbacks which can hinder the use of this language:

- **Not very good handling large amounts of data**: can become difficult to read if a lot of information is included in one file.

- **Not the best option always**: certain types of data like images are not very good represented using XML.

- **Verbosity**: a considerable part of an XML document consists of its markup. This markup consists of all the tags, begin and end element tags, describing the tree structure. Moreover, the markup tends to be highly repetitive resulting in large document files with great redundancy.
3.3 XML Schema

An XML schema [9] specifies the structure and element types that may appear in an XML file. A schema defines the structure by specifying all the nodes required, order and relations in an XML document. It also specifies the data that the container nodes can have. The data specification is achieved by the definition and use of the following data types: integers, floats, dates, enumerations and user defined data types using regular expressions. For example, a schema can define that a value element has type double or that a number attribute contains a number between 2 and 5. Schemas have also a number of other useful characteristics including namespace awareness and the ability to validate complex structures built up out of many different elements of many types.

There are different languages developed specifically to express XML schemas. The languages more used are Document Type Definition (DTD), XML Schema (with capital S) and RELAX NG.

The mechanism for associating an XML document with a schema varies according to the schema language. The association may be achieved via markup within the XML document itself, or via some external means. However, implicit schemas do not enable the data validation possible by explicitly declaring a DTD or XSD. The use of external schemas may result in smaller XML files and also enables data validation. The least robust schema approach is a DTD, whereas an XSD schema is itself an XML file.

When a file is adhering to the XML standard rules, it is said that the XML is valid. Specific information is needed in order to read or write an XML file for some specific applications.

```xml
<!----Deicing ORder (DOR) messages definition ---->
<!------------------------------------------------------------------>
<xs:element name="DORACK" nillable="false">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="M633Header"/>
      <xs:element ref="M633SupplementaryHeader"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<!------------------------------------------------------------------>

Figure 8: Schema example
3.4 Schemas used in ARINC 633

The ARINC 633 standard provides different schemas attending to the type of information to send. ARINC 633 files have to follow these schemas in order to be valid. ARINC 633 schemas specify a set of tags in order to standardize the way of exchanging information between aircrafts and ground-stations.

The language used to describe these schemas is the XML Schema language. XML Schema language was the first separate schema language for XML to achieve Recommendation status by the W3C. Like all XML schema languages, XSD can be used to express a set of rules to which an XML document must conform in order to be considered 'valid' according to that schema. XML Schemas provide a more complete validations language that includes data typing and range checking.

When compressing ARINC files, as all schemas are available, it is possible to take advantage of them. On one hand, the use of schemas allows to get a better compression ratio. On the other hand, once the compression and decompression processes are performed, it is possible to validate the resulting files in order to compare them with the original files and therefore check that no information is lost.
4 Compression

The language is by nature verbose and hence XML documents are usually larger in size than other specifications containing the same data content. Furthermore, XML also inflates the size of the data due to the repeated tags and structures. The size problem documents hinder the applications of this language, since it substantially increases the costs of storing, processing and exchanging data. This problem is more apparent in bandwidth- and memory-limited settings.

Compression techniques [10] are the important way to overcome the verbosity problem. Moreover, the fact that XML is notoriously verbose has spawned the development of many XML-specific compressors and binary formats.

Compression algorithms can be divided in two basic groups: lossy and lossless. The difference between these two types of compressors resides in the fact that a lossless compressed document can be reverted back to the original document, whereas a lossy compressed document can not be reverted to the same original state. Nevertheless, it can be reverted to a very similar document where the similarity is measured by a concept called distortion.

Lossy transforms are not typically applied to textual or numerical data; although humans can often compensate for missing pixels (frequencies), it is often difficult to guess absent characters (numbers). Thus the compressor that it is needed to compress XML files must be a lossless compressor.

Lossless compression algorithms map blocks of data into sequences of bits. The compressor’s objective is to minimize the number of bits used to represent each block. Unfortunately there is a theoretical limit on this mechanism: entropy.

4.1 Compressors classification

There are many types of lossless compressors. Depending on the feature to take into account, these compressors are classified in different categories. Below two different classifications [11, 12] are presented with their respective examples.
Classification based on their awareness of the structure of the XML documents:

- **General Text Compressors**: this group of compressors is XML-Blind; they treat XML documents as usual plain text documents and hence apply the traditional text compression techniques. This group can be also classified within two classes, arithmetic or dictionary. Some examples are: Gzip, bzip2, PPM.

- **XML Conscious Compressors**: XML documents may have nonlocal redundancy arising from XML’s tree structure which is difficult for text compressors to discover. Moreover, markup tends to be more redundant and compressible than text. So this type of compressors is designed to take advantage of the awareness of the XML document structure in order to achieve better compression ratios over the general text compressors. XML conscious compressors can be further classified according to their dependence on the availability of the schema information of the XML documents and also on they ability of supporting queries:
  
  - **Schema independent compressors**: XMill, XWRT, Xcomp, XMLPPM, SCMPPM, Exalt XML, AXECHOP
  
  - **Schema dependent compressors**: the schema provides valuable information to the compressor by specifying the data type and format of each element in the XML document. In a transmission the schemas does not need to be compressed and transmitted since it already exists in the receiver. Some examples of dependent compressors are: Millau, XAUST, RNGzip, EXI

Classification based on the ability to support queries [13]:

- **Non-Queriable XML Compressors**: this type of compressors does not allow any queries to be processed over the compressed format. The main focus of this group is to achieve the highest compression ratio. By default, general purpose text compressors belong to the non-queriable group of compressors. Other examples are: XMill, XMLPPM, XComp.

- **Queriable XML Compressors**: Although the compression ratio of this group of compressors is usually worse than non-queriable compressors, this type of compressors may be of interest in some applications because they allow queries to be processed over their compressed formats. In fact, the main focus
of these compressors is to avoid full document decompression during query execution as it can be important for many applications which are hosted on resource-limited computing devices. By default, all queriable compressors are XML conscious compressors as well.

- **Homomorphic compressors**: the original structure of the XML document is retained and the compressed format can be accessed and parsed in the same way of the original format: Xgrind, Xpress, QXT

- **Non-homomorphic compressors**: the encoding process of the XML document serrat the structural part from the data part. Therefore, the structure of the compressed format is different from the structure of the original XML document: Xseq, XCQ, TREECHOP XML, XQZip, XBW transform, ISX, Xcpaqs, Xquec

### 4.2 Parameters to take into account when choosing a compressor

For obvious reasons not all the available compressors can be included in the benchmark, therefore some of them have been selected taking into account different parameters.

The first parameter to consider is if the compressor is publicly and freely available either in the form of open source codes or binary versions. At this point, the compressors that still in the selection process among the entire above mentioned are: Gzip, bzip2, PPM, XMll, XMLPPM, SCMPPM, XWRT, Exalt, AXECHOP, Xgrind, XAUST, RNGzip.

The second condition to take into account is the schema’s dependence. ARINC 633 standard provides schemas written in XML Schema language. For this reason the only compressors that can be used in this benchmark are those that use the XML Schema language to compress files. Millau and XAUST require of a DTD to compress files, therefore these compressors are excluded. RNGzip uses a RELAX NG schema, thus this compressor is not included in the benchmark. Finally, the only compressor that can be included is EXI which can take advantage of XML Schemas.

Queriable compressors have been excluded of this benchmark because of the nature of the information exchanged. Information sent trough data links is specific for
each flight. As each aircraft received the information concerned to itself, it is not necessary to perform direct queries on compressed XML formats.

Finally, the compressors have been selected according to the research existent until now, focusing especially on two parameters: compression ratio and CPU consumption.

### 4.3 Chosen compressors

XML is stored in plain text files, so the most obvious approach to XML compression is to use existing text compressors. For this reason the benchmark includes one general-purpose compressor. Among the existent general-purpose compressors, Gzip have been selected because of its popularity. Gzip is widely used due to its good performance.

This benchmark also includes three schema-independent compressors. Two of them have a similar behavior; XMill and XMLPPM. Both of them take advantage of the awareness of the file for separating structure from data. Last schema-independent compressor chosen is Fast Infoset. As can be seen in the next chapter, Fast Infoset is not a compressor itself. Fast Infoset performs a binary encoding to transform XML files into binary XMLs. This is why after encoding files with Fast Infoset, Gzip is applied over binary XMLs to perform the compression.

Concerning to the schema-dependent compressors, just EXI can be useful as it is the only one that uses XML Schema language. EXI, as well as Fast Infoset, encodes files into binary XMLs. However, EXI comes with its own implementation of the deflate algorithm so, once the encoding is done, EXI itself performs the compression.

The next chapter explains how these compressors work and which results can be expected for each of them.
5 Compressors

5.1 Gzip

5.1.1 Introduction

Gzip [14] is the general-purpose compressor chosen for this benchmark. This application was created by Jean-Loup Gailly and Mark Adler as free software. Various implementations of the program have been written. The most commonly known, and the one used in this benchmark is the GNU Project’s implementation.

Gzip is based on the Deflate algorithm, which is a combination of LZ77 and Huffman encoding.

5.1.2 Deflate algorithm

Deflate [15] carries out two steps in order to achieve compression:

- Matching and replacing duplicate strings with pointer (LZ77).
- Encoding symbols with new weighted symbols based on the frequency of use (Huffman coding).

5.1.2.1 LZ77

LZ77 is a lossless data compression algorithm. It finds duplicated strings in the input data. The second occurrence of a string is replaced by a pointer to the previous string, in the form of a pair (distance, length).

To spot matches, the encoder looks at the most recent data, such as the last 2 kB, 4 kB or 32 kB. This method is equivalent to use an explicit dictionary and it is called sliding window. The largest the sliding window is, the more amount of data has the encoder to look for matches.
5.1.2.2 Huffman coding

Huffman coding is an entropy encoding algorithm used for lossless data compression. This algorithm assigns binary codes to symbols reducing the overall number of bits used for encoding. This reduction is achieved by encoding most frequently occurring symbols with less bits and less frequently symbols with longer bit strings.

Any string of letters will be encoded as a string of bits. It is important that smaller codes can not occur as a prefix in the larger codes.
5.1.2.3 How deflate works

First, data enters to the LZ encoder where string matching is performed using hash algorithms, hash tables, and a history buffer. In the case of deflate algorithm, the encoder keep track of the previous 32 kB. The decoder needs to keep this data to interpret the matches the encoder refers to. While distances are limited to 32 kB, lengths are limited to 258 B. When a string does not occur anywhere in the previous 32 kB, it is emitted as a sequence of literal bytes. (In this description, ‘string’ must be taken as an arbitrary sequence of bytes, not only characters).

The output of the LZ encoder is a queue, called the LLD (literals and length/distance) queue, which stores the symbols generated during the LZ encoding stage. The output of the LLD queue is processed by the Huffman Encoder which results in a compressed bit stream. Literals or match lengths are compressed with one Huffman tree, and match distances are compressed with another tree.

![Deflate compression diagram](image)

**Figure 11: Deflate compression**

Gzip compression improves with size; the bigger the file’s size, the better the compression. This is because of the dictionary encoding. When compression starts the dictionary is empty. Thus, if the file is small, the compression will not be very good as the dictionary is almost empty; there are just few words to find matches. However, when the dictionary is full, when looking for matching, it is more probable to find the same string already in the dictionary. Hence for larger files compression performance improves.
5.2 XMill

5.2.1 Introduction

XMill [16] is the first proposed XML-conscious compression architecture in the literature. This compressor takes advantage of the awareness of the file’s structure to perform a pre-compression phase before compressing data with Gzip. There are two specific objectives in the pre-compression phase in order to optimize XML compression:

- To separate the document structural information from data:
  
  o The *structural items* include element tag names, attribute names, and the document skeleton generated in a compact form.
  
  o The *data items* include PCDATA and attribute values.

- To group data items with related semantics in the same container.

  The structure of an XML file is used to be very redundant because of the repeated tags. The aim of the pre-compression phase is to expose this redundancy which can be very difficult to find for general-purpose compressors.

5.2.2 XMill architecture

XMill is formed by different elements that allow separating structure from data and group items into different containers attending to its semantic.

XMill architecture is composed of: a sax parser, a path processor, semantic compressors, containers and a Gzip compressor.

The diagram below illustrates XMill architecture [17]:

![XMill Architecture Diagram]
SAX parser parses the file and sends tokens to the core module. There are three kinds of tokens in an XML document: tags, attributes, and data values.

Path processor

The path processor is the core module of XMill. The mission of this module is to send each token to its pertinent container. With this task, two important issues are achieved: on one hand, structural elements are separated from data items. On the other hand, as the path processor is the one who determines how to map data values to containers, related items are grouped together.

In order to extract structural items from documents, XMill adopts a dictionary encoding approach to store tags and attribute names in a dictionary. Hence tags and attribute names are replaced by dictionary indexes and stored into the structure container. Start tags are dictionary encoded, i.e., assigned an integer value, while end
tags are replaced by the token ‘/’. Data values are replaced with their container number.

**Semantic compressors**

Semantic compressors are applied before storing data into containers. Usually, XML files present specific data types as integers, dates, etc. which is best compressed using specialized semantic compressors.

XMill supports three types of semantic compressors: atomic semantic compressors, combined compressors and user-defined compressors. However, the use of these compressors is optional, therefore when default mode is used, no semantic compressor is applied.

In the current benchmark, XMill works using the default mode. The use of semantics compressors require using complex command-line parameters which also depend on the type of data that each schema contains.

**Containers**

The default XMill usage specifies that each unique element is mapped to one container. This means that all XML paths sharing the same trailing element name are placed in the same container. Thus, the two paths “student/name” and “teacher/name” would have their data values placed in the same container.

Containers whose uncompressed size is smaller than 3 kB are not compressed directly by Gzip. Instead, all such containers are grouped together and compressed in one single Gzip run. This is more efficient, since Gzip yields better results for larger data blocks.

Container expressions can be defined by users. This means that the user can specify if a specific path will be placed in a different container. Following the example before, the two paths “student/name” and “teacher/name” could have their values in different containers if the user indicates it using command-line parameters.

In case the user does not define any expression the default case will be used.
Final compressor

Once all information is placed in its corresponding container, each container (or group of containers, depending on the size) is compressed using Gzip. The use of containers facilitates to the compressor finding the redundancy existing in the document. Gzip uses a sliding window of 32 kB to perform the compression. If structure is separated from data and Gzip is applied just to the structure, it is more likely to find more repeated tags in a window of 32 kB. Moreover, all data corresponding to a specific path is probably very similar, thus compression of data can also be improved.

However, XMill wins over existing techniques only if the data set is large, typically over 20KB, because of the additional bookkeeping overhead and the fact that small data containers are poorly compressed by Gzip.
5.3 XMLPPM

XMLPPM [18] is a combination of the Prediction by Partial Match (PPM) algorithm for text compression and an approach to modeling tree-structured data called Multiplexed Hierarchical Modeling (MHM) where the context given by the path in the structure tree is used to model the text in the sub-tree. Thus, different models are used to code tag names, attribute names, attribute values, textual content, and so on. XMLPPM is based on the intuition that the text under similar parts should follow a similar distribution.

5.3.1 Prediction by Partial Matching

XMLPPM compressor can be considered an adaptation of the general-purpose Prediction by Partial Matching compression scheme (PPM).

A predictive model bases its predictions on a context; a set of previously appeared symbols. For each context, a model provides the probability distribution of the followers. The number of previously appeared symbols taken into account is usually referred to as the order of the model. A model of order 1 therefore takes only one previously appearing symbol into account, while a model of order 0 considers the global probability distribution of the symbols.

In the case of XML compression, this problem is especially interesting since the hierarchical nature of XML provides many different modeling alternatives which can not be applied on flat text. A model could, for example, take advantage of the observation that certain symbols are more likely to appear between or after certain tags and less likely to appear between or after some others. In fact, it is clear intuitively and proved by some researchers that probability of every next symbol is highly dependent on previous symbols.

5.3.2 XMLPPM compression

XMLPPM follows the idea of the XMill of transforming the XML into three components [19]: elements and attributes, text, and document structure, and then to pipe each of these components through existing text compressors. In fact, XMLPPM goes further refining this idea by using different text compressors with different XML components and, in addition, injecting hierarchical element structure symbols into the multiplexed models (these are essentially root to leaf paths to the element).
Multiplexing enables more effective hierarchical structure modeling. A common case for these dependencies is for the enclosing element tag to be strongly correlated with enclosed data. MHM exploits this by injecting the enclosing tag symbol into the element, attribute or string model immediately before an element, attribute or string is encoded. Injecting a symbol means telling the model that it has been seen but not explicitly encoding or decoding it.

XMLPPM begins compression by generating a binary encoding for XML called ESAX (Encoded SAX). This encoding keeps the original XML structure intact (as opposed to scattering elements, attributes, and data values during the compression process). As each token is returned by the SAX parser, it is encoded according to the following scheme: element start tags, end tags, and attribute names are all encoded using a single byte; event indicators (such as the beginning and ending of a sequence of raw character data) are also encoded with one byte; data values are not encoded, but preserved as raw character data. ESAX encoding proceeds in an adaptive fashion, with the encoder sending the name of each newly-encountered symbol to the decoder.

Additionally, XMLPPM applies a technique called multiplexed hierarchical modeling (MHM) to the generated ESAX data. This technique is based on the following two core principles: Different text compression models are multiplexed based on XML structure, thus a particular model is chosen depending on the type of the current ESAX token. Each model shares access to a common arithmetic coder used to perform encoding according to the PPM method. Hierarchical element structure models are "injected" into the multiplexed models. Before a given element, attribute, or data value is encoded, its enclosing tag symbol is sent to the appropriate compression model. This has the effect of restoring any sequential dependencies from the original XML document that have been destroyed by switching between different models.

Another important characteristic of this compressor is that, unlike XMILL, XMLPPM is an online compressor. Online compressors are able to stream the compressed data to the decoder: the decoder is able to begin the process of decompression before the encoder has finished to transmit the compressed data.

The online feature of XML compression tools could be very important for the scenarios where the users are heavily exchanging compressed XML documents over networks. In these scenarios, online decompression processors can effectively decrease the network latency during the transmission process.
5.4 Fast Infoset

5.4.1 Introduction

Fast Infoset [20] is not a compressor itself. Although this technique transforms xml files into more compact documents, a compression must be applied after this transformation in order to get a more compressed document. This is the reason why after applying Fast Infoset, files are compressed using Gzip compressor.

The transformation carried out by Fast Infoset consists of a binary encoding of XML Information Set documents. Its aim is to provide smaller encoding size and faster processing.

The Fast Infoset specification has been implemented as an open source. The project provides a fast, full-featured, and robust implementation in Java.

5.4.2 XML Information Set

XML Information Set (XML Infoset) is a W3C recommendation that defines an abstract data model for XML documents. This data model details the properties of XML trees, representing the information stored in an XML document in terms of information items (abstract representation of the components of an XML).

Each XML document has an information set if it is well-formed and satisfies some namespaces constraints (it does not mean that the document must be valid). Information set is the same as a tree and information item is the same as a node of this tree.

There are information items representing the document, its elements, attributes, processing instructions, comments, characters, notations, namespaces, unparsed entities, unexpanded entity references, and the document type declaration.

XML Infoset defines what should be considered to be significant information in an XML document. For example, it does not distinguish between the two ways of representing an empty element. Thus both forms of representing the empty element airport, `<airport>`</airport> and `<airport/>`, are equivalent for the XML Infoset model.
5.4.3 Fast Infoset Documents

Fast Infoset is an international standard that specifies a representation of an instance of XML Information Set using binary encodings. Hence, finally, to get and infost document from an XML document, it is necessary to represent the XML document in an XML infoset and encode this last representation in a binary way:

First, to achieve the representation of the XML document in a XML information set, it is necessary to parse the XML document in order to identify infoset components, referred to as information items and properties.

After this, the XML infoset is serialized according to the Fast Infoset specification in order to get the Fast Infoset document. The serialization process producing a Fast Infoset document is, conceptually, the result of constructing a Fast Infoset value from an XML infoset and encoding the Fast Infoset value to produce a Fast Infoset document.

The construction of a Fast Infoset value from an XML Infoset or vice versa will result in the use of tables and indexing to compress common string information. This will reduce document size whereas still ensuring that the documents may be serialized and parsed efficiently.

Fast Infoset documents always retain the hierarchical structure described by the corresponding XML Infoset, and depending on which features are selected, can be self-contained or not. Fast Infoset documents that are self-contained can be converted to and from XML without information loss, at least with respect to the information items and properties defined in the XML Information Set.

For the binary encoding, Fast Infoset uses the ASN.1 notation, which is a formal language for abstractly describing messages. In fact, the Fast Infoset recommendation specifies an ASN.1 type whose abstract values represent instances of the XML Information Set. FI also specifies binary encodings for those values, using ASN.1 Encoding Control Notation.
Fast Infoset documents are analogous to XML documents. Each has a physical form and an XML Infoset. Therefore it can be said that both, XML and Fast Infoset are just two different ways of representing the XML Information Set:

![Diagram showing the process of parsing XML and Fast Infoset](image)

**Figure 14: Analogy Fast Infoset – XML**

However, Fast Infoset format has been designed to optimize the axes of compression, serialization and parsing, while retaining the properties of self-description and simplicity. Thus, Fast Infoset documents may be used whenever the size and processing time of XML documents is an issue.

### 5.4.4 Advantages and Properties of the Fast Infoset

There are a number of generic advantages of Fast Infoset documents, and specific properties of the binary format, that contribute to faster parsing, faster serializing, and smaller document sizes when compared with equivalent XML documents.

Such generic advantages are:

- **No end-tags**: the duplication of characters for end-tags is not required.

- **No escaping of character data**: checking each character to see if it needs to be escaped can be time-consuming, and replacing content may result in additional memory usage and copying.

- **Length-prefixing of content**: length-prefixing enables a decoder to allocate
resources accurately and possibly reject content immediately if the length is considered too large.

- **Indexing of repeated strings**: indexing reduces the size of the document by replacing a commonly-used string with an integer. Element and attribute names are examples of such repeated strings.

- **Embedding of binary content**: binary content does not need to be converted to and from the Base64 character representation.

- **Preservation of state for documents with similar vocabularies**: indexing can be preserved and reused for multiple documents that make use of the same information items.

A number of specific, and advantageous, properties of the binary format for Fast Infoset documents are:

- **Huffman-style encoding of [children] information items**: more-common information items that are among the [children] of an element information item or a document information item are encoded using fewer bits than less-common items. For example, an element information item and a chunk of character information items will be encoded in fewer bits than a processing instruction information item.

- **Byte alignment**: alignment on well-defined and known boundaries makes for more efficient implementation of encoders and.

- **Packing of indexes and length prefixes**: the packing of integer values (associated with the indexing of a string or the length of content) makes for smaller sizes of Fast Infoset documents at little expense to the serialization and parsing of such documents.

- **Indefinite length of [children]**: to support streaming (for example, to support SAX-based serializers) the [children] of an element information item or a document information item are encoded in a way that does not require that their number be known in advance.
5.5 EXI

5.5.1 Introduction

EXI stands for Efficient XML Interchange [21]. It is a W3C recommendation for encoding XML files in a binary format. It was developed by the W3C’s Efficient XML Interchange Working Group and is one of the most famous binary XML efforts to encode XML documents in a binary data format.

As Fast Infoset, EXI encodes input files converting them into XML binary files. After this, the compression phase is carried out. EXI specification comes also with its own implementation of the deflate algorithm for compressing files after the encoding phase.

EXI aims to improve performance while it reduces significantly the bandwidth required through a good compression.

EXI is schema informed. This means that EXI is able to take advantage of the schema to improve compactness and performance. However, the use of schema is not mandatory therefore EXI can compress files without schema achieving also a good degree of compression.

5.5.2 Codification

EXI is compatible with XML at the XML Information Set level, rather than at the XML syntax level. This allows it to encapsulate efficient alternative syntax and grammar for XML.

EXI uses its EXI processor to encode input data into EXI streams and/or decode EXI streams in order to get the structured data. Those parts of the EXI processor that allow carrying out these processes are called EXI stream encoder and EXI stream decoder respectively. EXI also uses a grammar-driven approach to achieve very efficient encoding using a direct encoding algorithm and a small set of datatypes representations.

EXI stream encoder uses grammar to map a stream of XML information items into a stream of events. This encoder represents the stream of events using events codes which are similar to Huffman codes but are much simpler to compute and maintain. Events streams have lower entropy and are smaller than input streams. If an additional
compression is desired, these streams are passed to the EXI compression algorithm, which reduces frequently occurring event patterns to reduce streams’ size.

5.5.2.1 EXI streams

Each EXI stream is conformed by an EXI header followed by and EXI body. The EXI header store the options used for encoding the EXI body, while the EXI body carries the content of the document.

EXI bodies consist of a sequence of EXI events representing an EXI document. EXI events permitted at any position in an EXI stream are determined by the EXI grammar.

If the schema is available, the EXI body is interpreted according to the Schema-informed Document Grammar. Otherwise, the EXI body is interpreted according to the Built-in Document Grammar.

The EXI header has the following structure:

<table>
<thead>
<tr>
<th>EXI Cookie</th>
<th>Distinguishing Bits</th>
<th>Presence Bit for EXI Options</th>
<th>EXI Format Version</th>
<th>EXI Options</th>
<th>Padding Bits</th>
</tr>
</thead>
</table>

**Figure 15: EXI header**

- *EXI cookie*: sequence of four bytes to distinguish EXI streams from other data types. Its use is optional yet recommended.

- *Distinguishing bits*: two bit field where the first one contains the value 1 and the second one the value 2. It is used to distinguish EXI streams from text XML documents when EXI cookie is not used. Its use is mandatory.

- *Presence Bit for EXI options*: indicate if EXI options will be used or not.

- *EXI format version*: it is used to identify the version of the EXI format being used.

- *EXI Options*: provides a way to specify the options used for encoding. Its use is optional. In the next chapter EXI options are named and briefly described.
5.5.2.2 EXI Options

Sometimes, when a file is compressed and then uncompressed it is not necessary to recover the exactly original file. Some XML applications do not require the entire XML feature set and would prefer to eliminate the overhead associated with unused features like whitespaces or comments. Such features can be specified using EXI options. Through EXI options it is also possible to specify if a compression must be applied after the encoding.

Bellow, EXI Options available are named and briefly explained.

- **Alignment option**: used to control alignment of event codes and content items. The value is one of bit-packed, byte-alignment or pre-compression. This element must not appear when “compression” element is present.

- **Compression option**: boolean used to indicate if a compression will be performed after the encoding.

- **Strict option**: represented by a boolean in order to use a strict interpretation of the schemas and omit preservation of certain items such as comments, processing instructions and namespaces prefix.

- **Fragment option**: indicates whether the EXI body is an EXI document or an EXI fragment.

- **Preserve option**: set of booleans which can be set independently to enable each fidelity option. Fidelity options are: preserve comments, preserve processing instructions, preserve DTD, preserve prefixes and preserve lexical values.

- **SchemaId option**: allows identifying the schema information used for processing the EXI body.

- **BlockSize option**: specifies the block size used for EXI compression.

5.5.3 EXI compression

Although EXI encoding creates a more compact representation of XML files, sometimes a higher level of compression is required.
EXI compression [22] combines knowledge of XML with a standard compression algorithm (deflate) to achieve higher compression ratios than those achieved by applying compression to the entire stream.

EXI also allows compression when compression option is turned on or when alignment is set to pre-compression. Byte-aligned representations of event codes and content items are more amenable to compression algorithms compared to unaligned representations because most compression algorithms operate on series of bytes to identify redundancies in the octets.

This compression can be compared with the XMill or XMLPPM compression where the compressor takes advantage of the awareness of the XML document structure. EXI, as well as XMill, separates the structural items from the data, and finally uses the deflate algorithm to perform the compression.

EXI compression splits a sequence of EXI events into a number of contiguous blocks of events. All those events that conform a block, are grouped in channels, which will be combined and compressed.
To create different channels, events are grouped according to its nature. The first channel of each block is the structure channel while the remaining channels are value channels. The structure channel defines the overall order and structure of events contained in that block. It contains the event codes related to the structural part of an XML file. On the other hand, value channels contain the data of an XML file. These values are organized into separate channels based on the qname of the associated attribute or element. The diagram below illustrates this process:

![Diagram of XML compression algorithms benchmark on ARINC 633 data](image)

**Figure 17: Channeling**

Once the channels are conformed, these can be combined or not depending on its size. If the value channels of the block contain at most 100 values, the block will contain only 1 compressed stream containing the structure channel followed by all the value channels. The order of the value channels within the compressed stream is defined by the order in which the first value in each channel occurs in the EXI event sequence.

If the value channels of the block contain more than 100 values, the first compressed stream contains only the structure channel. The second compressed
stream contains all value channels that contain at most 100 values. And the remaining compressed streams each contain only one channel, each having more than 100 values. The order of the value channels within the second compressed stream is defined by the order in which the first value in each channel occurs in the EXI event sequence. Similarly, the order of the compressed streams following the second compressed stream in the block is defined by the order in which the first value of the channel inside each compressed stream occurs in the EXI event sequence.

When the value of the compression option is set to true, each compressed stream in a block is stored using the standard deflate Compressed Data Format defined by RFC 1951. Otherwise, when the value of the alignment option is set to pre-compression, each compressed stream in a block is stored directly without performing deflate algorithm on this block.
6 Benchmark

The objective of this benchmark [23] is to compare the performance of the five previous compressors when compressing ARINC 633 files. For this purpose, a complete framework has been implemented which consists of sample file generation, compression, decompression, validation and parameters measurement.

In Annex 10.1 it can be found a more detailed operation of this benchmark.

6.1 Evaluated parameters

The parameters evaluated in this benchmark are compression performance and CPU consumption for compression and decompression processes.

Compression performance refers to the capacity of the compressor to reduce the original file. This parameter is expressed in percentage and it is calculated as follows:

\[
\text{Compression (\%)} = \left[1 - \left(\frac{\text{size of the compressed file}}{\text{size of the original file}}\right)\right] \times 100
\]

CPU consumption refers to the time spent by the CPU for compressing and decompressing files. Each file is compressed and decompressed ten times. Therefore results shown in the next chapter refer to the average of these ten iterations. The reason of doing this average is to achieve more exactly results.

6.2 Input files

Input files used in this benchmark have been grouped in three different groups attending to its nature. Below are listed and explained.

6.2.1 ARINC 633 examples

The recommendation ARINC 633 provides not only the document but also the schemas for the different services. Each schema comes with one or various examples of XML files based on these schemas. As these examples are valid files, they have been analyzed in this benchmark.
6.2.2 Flight Plan files

The examples provided by the ARINC 633 recommendation have a limited size. This is why it is necessary to create more files in order to check the performance of the different compressors when files size increases.

In order to create XML files, the schema chosen among all schemas provided by the standard is the Flight Plan schema. The reason of this choice is that this schema allows the use of more datatypes and therefore is possible to obtain bigger files.

Steps involved in the generation of these files are described in Annex 10.1.2. Files obtained are based on the Flight Plan schema, using only datatypes described in this schema and following the structure also described. All data used to filled the values is random.

Files created have been classified in three groups according to its size:

- **Small files**: files smaller than 30 kB
- **Medium files**: files comprised between 30 kB and 400 kB
- **Big files**: between 400 kB and 1,34 MB

6.2.3 Sample weather files

Weather is not defined yet in the ARINC 633 recommendation. Although there is nothing established at the moment, following the trend, is likely that WXXM will be the standard that ARINC will use to define its own way to send weather information. WXXM is an XML standard to encode weather-related data, based on Geography Markup Language (GML).

In order to have an idea of which compressor would have a better performance when dealing with this type of files, weather files from the German Aerospace Center have been evaluated. These files are weather files based on GML also. However, there is no schema available thus EXI with schema compression can not be tested.

Files are separated in three groups according to its size. This size classification is not the same used for the flight plan size because the smallest file has a size of 18 kB. Weather files have been classified as follows:
o **Small files**: files comprised between 18 kB and 176 kB

o **Medium files**: between 177 kB and 659 kB

o **Big files**: files comprised between 670 kB and 1,7 MB
7 Results

This section presents the most relevant results of the benchmark. For each group of files (of those presented in the previous chapter), results are split in three different graphics according to its size. Each graphic illustrate the results obtained for the five compressors chosen. For EXI two results are shown: EXI with schema (in case the schema is available) and EXI without schema.

EXI and Fast Infoset can work without post-compression. Without post-compression EXI and Fast Infoset just encode files using a binary encoding, thus they are not really performing a compression. The results of this encoding can be found in Annex 10.2. These results are not presented in the current chapter because are not relevant in terms of compression performance.

Section 5.5.2.2 explains all available options when using EXI. The results presented in the current chapter are obtained using EXI with the following options turned on: preserve schema location and preserve prefixes. White spaces are not preserved as they are not necessary for the correct working of any application (they are just useful for making the code more human-readable).

All the graphics presented in this section show the performances of XMill, Gzip, XMLPPM, Fast Infoset plus Gzip, EXI applying deflate and EXI with schema and deflate (expressed on the chart legend as EXI(Sch)+D).

It is also important to note that the file size is expressed in bits.
7.1 Compression performance

7.1.1 ARINC 633 examples

![Compression performance of ARINC examples](image)

The graphic above shows that the best compression is achieved when using EXI, with or without schema. Gzip, XMLPPM and Fast Infoset plus Gzip have a similar performance while XMill presents the worst results.

EXI can achieve very good results as its compression involves many steps that allow a high reduction of files: binary encoding, separation of structure from data into channels, re-grouping channels, channels’ Gzip compression and the use of different options.

For very small files, EXI provides a compression of more than 30% compared with Gzip. By using schema, this percentage increases up to 50% approximately. The graphic also shows that the bigger is the size, the little is the difference between performances. This is because the redundancy existing in the files increase.

The use of schema benefits compression performance especially when dealing with very small files. This is due to the deflate dictionary. As have been explained in chapter 5, deflate algorithm is used in all compressors tested. This algorithm uses a dynamic dictionary of 32 kB to replace words already appeared in the text and thus compress files. When the compression starts, the dictionary is empty. However, this is not the case of EXI with schema. EXI takes advantage of the schema to initialize the dictionary. This is the reason why such good results can be achieved.
Results achieved by Gzip, XMLPPM and Fast Infoset plus Gzip are very similar. It means that for files with these sizes the use of Fast Infoset before Gzip’s compression does not provide any improvement to the compression performance.

XMILL is the one with the worst compression performance. This is due to its architecture. As the section 5.2 explains, XMILL transforms XML files separating structure from data. Then, Gzip compression is applied to the different containers. This transformation facilitates the compressor to find the redundancy existing in the XML structure. But, if the file is small, it does not make sense to apply this transformation because the dictionary (sliding window) is not full yet. What is more, by grouping data in different containers, we are adding some overhead in order to indicate how the data is distributed. This overhead has also to be compressed; therefore there is more input data to compress.

### 7.1.2 Flight Plan files

#### 7.1.2.1 Small files

![Compression performance of small Flight Plan files](image)

**Figure 19: Compression performance of small Flight Plan files**

These files follow the trend of ARINC 633 examples. However the difference of percentage of compression achieved among the different compressors is not so big. This just indicates that these files have less redundancy than ARINC examples.
For ARINC files, there is a gap between sizes 5.580 and 26.853 bits. With flight plan files generation, this gap is filled and compression performance can be tested for sizes compressed between these two numbers. By doing this, it can be seen more clearly how compressor’s performance evolve within this range.

XMill’s performance improves achieving the same compression as Fast Infoset plus Gzip. The graphic illustrates that, for files not very small, it is worth the application of Fast Infoset encoding before compressing with Gzip.

It is also noted that the difference of compression performance between EXI with schema and the others compressors decreases substantially.

### 7.1.2.2 Medium sized files

![Figure 20: Compression performance of medium size Flight Plan files](image)

The compression percentage of EXI, EXI with schema and XMill becomes more similar as the file size increases. About XMLPPM, it maintains a good compression ratio; however its performance does not increase so much as XMill compressor does.

With the increase of file size, also increases the compression’s difference between Fast Infoset plus Gzip and Gzip alone. It means that, when files are big enough, is recommended to use Fast Infoset before applying Gzip compression.

Using EXI or XMill is possible to get a gain of 15% with respect to Gzip.
7.1.2.3 Big files

Figure 21: Compression performance of big Flight Plan files

ARINC 633 specifies a set of tags for each feature. In this case, for flight plan, there are also defined a set of tags in order to organize the information that will be send. Most of these tags can be repeated again and again depending on what information is needed to send. This means that the bigger is the file, the greater the probability that the same tags are repeated. Hence the percentages of compression tend to be constant with bigger files.

Following the trend of the previous graphic, this one illustrates how XMIII, EXI and EXI with schema, finally, present the same percentage of compression.
7.1.3 Weather files

7.1.3.1 Small files

![Image: Graph showing compression performance]

**Figure 22: Compression performance of small Weather files**

These files follow the same trend as all previous files of the same size: EXI is the one that performs better, followed by XMLPPM. XMill is the one with the worst performance at the beginning, but its performance improves with the file size. Gzip and Fast Infoset plus Gzip have the same performance for smaller files; however for bigger files is better to apply Fast Infoset before compressing files with Gzip as it improve the result.

The biggest difference between this graphic and the previous graphics is the compression range. For weather files, percentage of compression is comprised between 90% and 98% what means a really good compression.

This graphic shows performance drops for some files. These drops appear when files contain many coordinates. Coordinates consist of a lot of floating-point numbers and are difficult to compress for all algorithms. Nevertheless, Gzip can handle better compression of coordinates than EXI.
7.1.3.2 Medium sized files

![Diagram of compression performance for medium size files]

**Figure 23: Compression performance of medium size Weather files**

This graphic shows how, when file size increase, all compressors excluding Gzip tend to have a similar performance. Moreover, Gzip presents an unstable behavior.

7.1.3.3 Big files

![Diagram of compression performance for big files]

**Figure 24: Compression performance of big Weather files**

The performance of XMill, XMLPPM and EXI tends to be constant for bigger files. On the other hand, Gzip and Fast Infoset plus Gzip present a more unstable performance.
### 7.2 CPU consumption

This chapter presents the results for ARINC 633 examples and flight plan files. Results for weather files are not presented because are very similar to flight plan results hence they do not provide any extra information. However, results of the CPU consumption for weather files can be found in Annex 10.3.

#### 7.2.1 ARINC 633 examples

![Compression Time Diagram](image1)

![Decompression Time Diagram](image2)

**Figure 25: CPU consumption of ARINC 633 examples**

EXI, with or without schema, is able to carry out compression and decompression processes with just some milliseconds. As the graphic illustrates, it does not matter if the schema is used or not, the time needed to perform these processes is really similar. XMill and XMLPPPM have a very similar performance for both, compression and decompression processes carrying out these processes with just 10 milliseconds. Gzip
and Fast Infoset are the compressors which spend more time for compressing and decompressing however they are still very low times.

7.2.2 Flight Plan files

7.2.2.1 Small files

![Compression Time Graph](image)

![Decompression Time Graph](image)

**Figure 266: CPU consumption of small Flight Plan files**

These graphics follow the same trend as the ARINC files graphics. Nevertheless, in this case, it is clearer how XMLPPM increases the time spent in compression and decompression processes when the files size increases. The other compressors show a more stable performance.
7.2.2.2 Medium sized files

Figure 27: CPU consumption of medium size Flight Plan files

For the compression process, time increases with the size, however this increase is higher for XMLElement and EXI.

The second graphic shows that while time spent for XMLPPM for decompressing increases considerably, the other compressors presents a more stable performance. Because of its algorithm (Prediction by Partial Matching), this compressor needs much more time than the others to compress files. This is due to the fact of having to perform predictions in a big file.
7.2.2.3 Big files

For biggest files, XMLPPM is the compressor that shows the worst performance as it needs 120 ms (two times more than Gzip) for compressing and 160 ms (times more than Gzip) for decompressing. However, EXI even does a little bit worst when compressing. Despite these results, it is important to keep in mind that this magnitude is expressed in milliseconds.
8 Conclusions

Within this thesis it has been studied the performance of different compression techniques over ARINC 633 files creating a framework which allows the compression and decompression of these files in order to get the required parameters. After analyzing the obtained results it can be concluded the following items.

For small files the best compression performance achieved for the files studied in this thesis is when using EXI. Especially, for very small files, up to 1 kB, EXI presents a really good performance for both cases, with or without schema. EXI without schema achieves a compression performance of more than 30% compared with Gzip. When the schema is used, this percentage increases up to 50%. For files of this size, the other compressors studied do not provide any improvement in comparison to Gzip. What is worse, XMill even presents a less compression performance than Gzip. It is therefore suggested the use of EXI for files smaller than 1 kB.

When size increases compression performances change. EXI with schema still provides the best compression ratio but the difference with the others ratios decreases, especially for files with a size of more than 20 kB. XMLPPM and EXI without schema present a very similar performance, with a difference of just a 5% compared with EXI using schema. XMill also improves its performance whereas Gzip become the one with the worst compression.

For those files bigger than 40 kB (0,3 Mbits), EXI with schema, EXI without schema and XMill present almost the same performance. Thus the use of the schema does not provide any benefit. XMLPPM also presents a good performance (1% less than EXI) while Gzip is the one with the worst compression; 5% less than EXI. Hence, for files bigger than 40 kB it is worth the use of some structure-aware compressor.

About the CPU consumption and computation latency, for the compression of ARINC 633 files, depending on the safety criticality level of the application, it is likely not a restricting factor. All measures are in the range of milliseconds. On satellite links, the transmission and link access latency is in the order of seconds, thus are more constraining factors. Therefore, even in the worst case, 160 ms for XMLPPM decompression, it does not represent a reason for not using this compressor. The safety criticality assessment is, however, outside of the scope of this thesis.
9 Bibliography


[8] XML Tutorial, http://www.w3schools.com/xml,


Appendix

A Benchmark

In order to carry out the compression benchmarking campaign a code has been implemented. The language used to write the code is Java.

The code implements a loop of ten iterations where input files are compressed and decompressed. By performing these processes is possible to get three parameters: compression ratio, compression time and decompression time. Compression and decompression are carried out so many times as the loop is performed. In each iteration, a new CSV file is created. This file contains the parameters calculated. The diagram below illustrates this operation:

![Diagram of benchmark operation]

**Figure 28: Benchmark operation**
After all this process decompressed files are validated and all CSV files are treated in order to get the graphics.

### A.1 Generation of input files

As ARINC 633 just provides some XML examples, it has been necessary to create more XML files based on this standard. The process followed for creating these files works as follows:

![Diagram showing the process of generation of input files]

- First, Java objects are created from the XML schema using JAXB. JAXB stands for Java Architecture for XML Binding. This tool provides two main features: the ability to marshal Java objects into XML and the inverse, to unmarshal XML back into Java objects. Therefore, with this tool, objects that represent the tags and attributes of the XSD are obtained.

- Once all objects are obtained, the structure that will follow XML files is programmed. Structured has been developed to be the more random possible. This means that if an element is not mandatory to appear in the XML, this element have been written inside an “if” which depends on a random Boolean.
Then, tags and attributes values must be assigned. For this purpose, random variables have been used: random strings, random floats, random integers... The type of the variable to use is specified in the ARINC standard.

Finally, Java objects are marshaled into XML files.

A.2 Tools

For Gzip, XMll and XMLPPM compressors, their executable file have been used to perform compression and decompression processes. However, there is no executable file for Fast Infoset and EXI. A code has been writing for each of them using their libraries. EXI library had to be modified in order to allow compression of big files.

Other libraries used are:

- XMLUnit, version 1.5
- CommonsIO, version 2.4

A.3 PC characteristics

The PC used for carry out this benchmark is equipped with an Intel Core i5-2400 processor. This processor has the following specifications:

- # of cores: 4
- Clock speed: 3.1 GHz
- Cache size: 6 MB

Although this is a 64 bits processor, the OS used is Fedora and just works with 32 bits.
B Compaction performance

This section presents the same results as the section 7.1 but including the results for EXI using different options. These options are expressed in the legend as:

- EXI PW: EXI preserving whitespaces (in this case as deflate is not used EXI just encode files, there is no post-compression).
- EXI+D PW: EXI applying deflate compression and preserving whitespaces.
- EXI(Sch): EXI with schema (encodes files taking advantage of the schema).
- EXI(Sch) PW: EXI with schema and preserving whitespaces.
- EXI(Sch)+D: EXI with schema and deflate compression.
- EXI(Sch)+D PW: EXI with schema and deflate compression preserving whitespaces.

B.1 ARINC 633 examples

![Diagram of compression performance](image)

**Figure 30: Compression performance of ARINC 633 examples for all combinations**
B.2 Flight Plan files

Small files

![Small file compression graph](image1)

**Figure 31:** Compression performance of small Flight Plan files for all combinations

Medium size files

![Medium size file compression graph](image2)

**Figure 32:** Compression performance of medium Flight Plan files for all combinations
Big files

![Compression performance of big Flight Plan files for all combinations](image)

Figure 33: Compression performance of big Flight Plan files for all combinations

B.3 Weather files

Small files

![Compression performance of small Weather files for all combinations](image)

Figure 34: Compression performance of small Weather files for all combinations
Medium files

Figure 35: Compression performance of medium Weather files for all combinations

Big files

Figure 36: Compression performance of big Weather files for all combinations
C CPU consumption for weather files

Small files

Figure 37: CPU consumption of weather small files
Medium files

![Compression Time Graph](image)

![Decompression Time Graph](image)

Figure 38: CPU consumption of weather medium size files
Big files

Figure 39: CPU consumption of weather big files