Interactive Realtime Multimedia Applications on Service Oriented Infrastructures

ICT FP7-214777

WP3 – IRMOS Platform Specification

D3.1.4 Final version of IRMOS Overall Architecture

IRMOS_WP3_D3_1_4_NTUA_v1_00

Scheduled Delivery: 30.10.2010
Actual Delivery: 18.01.2011
Version 1.0

<table>
<thead>
<tr>
<th>Dissemination Level</th>
<th>PU</th>
<th>PP</th>
<th>RE</th>
<th>CO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Public</td>
<td>Restricted to other programme participants (including the Commission)</td>
<td>Restricted to a group specified by the consortium (including the Commission)</td>
<td>Confidential, only for members of the consortium (including the Commission)</td>
</tr>
</tbody>
</table>
Responsibility Partner: ICCS/NTUA

Revision history:

<table>
<thead>
<tr>
<th>Date</th>
<th>Editor</th>
<th>Status</th>
<th>Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>20.09.2010</td>
<td>Andreas Menychtas</td>
<td>Draft</td>
<td>0.01</td>
<td>Initial Release with ToC</td>
</tr>
<tr>
<td>24.09.2010</td>
<td>Andreas Menychtas</td>
<td>Draft</td>
<td>0.02</td>
<td>Changes on the structure</td>
</tr>
<tr>
<td>12.10.2010</td>
<td>Andreas Menychtas</td>
<td>Draft</td>
<td>0.03</td>
<td>Initial contributions in sections WF management, Monitoring and Evaluation Contributions in sections Data Management, SLA Management</td>
</tr>
<tr>
<td>27.10.2010</td>
<td>Andreas Menychtas</td>
<td>Draft</td>
<td>0.04</td>
<td>Contributions in sections Data Management, SLA Management</td>
</tr>
<tr>
<td>27.10.2010</td>
<td>Andreas Menychtas</td>
<td>Draft</td>
<td>0.05</td>
<td>Contributions in sections ISONI sections</td>
</tr>
<tr>
<td>27.10.2010</td>
<td>Andreas Menychtas</td>
<td>Draft</td>
<td>0.06</td>
<td>Minor changes in structure</td>
</tr>
<tr>
<td>03.11.2010</td>
<td>Andreas Menychtas</td>
<td>Draft</td>
<td>0.07</td>
<td>Contributions in Platform Interactions Section</td>
</tr>
<tr>
<td>09.11.2010</td>
<td>Andreas Menychtas</td>
<td>Draft</td>
<td>0.08</td>
<td>New section: Control Loops</td>
</tr>
<tr>
<td>10.11.2010</td>
<td>Andreas Menychtas</td>
<td>Draft</td>
<td>0.09</td>
<td>Contributions in Platform Interactions Section</td>
</tr>
<tr>
<td>06.12.2010</td>
<td>Andreas Menychtas</td>
<td>Draft</td>
<td>0.10</td>
<td>Updated References</td>
</tr>
<tr>
<td>07.12.2010</td>
<td>Andreas Menychtas</td>
<td>Draft</td>
<td>0.11</td>
<td>Contributions in service engineering and security sections</td>
</tr>
<tr>
<td>18.01.2011</td>
<td>Andreas Menychtas</td>
<td>Final</td>
<td>1.0</td>
<td>Minor corrections in several sections</td>
</tr>
<tr>
<td>18.01.2011</td>
<td>Andreas Menychtas</td>
<td>Final</td>
<td>0.12</td>
<td>Corrections following the internal review</td>
</tr>
</tbody>
</table>

© ICCS/NTUA and other partners of the IRMOS consortium 2008-2011
Authors
Andreas Menychtas, Spyridon Gogouvitis, Kleopatra Konstanteli, Georgios Kousiouris, Dimosthenis Kyriazis, (ICCS/NTUA); Eduardo Oliveros (TID); Ganesan Umanesan, Malcolm Muggeridge, Sai B Narasimhamurthy (XY); Karsten Oberle, Thomas Voith (ALUD); Michael Boniface, Bassem M. Nasser (IT-Inn); Soeren Berger (USTUTT)

Internal Reviewers
Stephen Phillips (IT-Inn); Juergen Jaehnert (USTUTT); Consolidating Reviewer: Eduardo Oliveros (TID)

Copyright
This report is © by NTUA and other members of the IRMOS Consortium 2008-2011. Its duplication is allowed only in the integral form for anyone's personal use and for the purposes of research or education.

Acknowledgements
The research leading to these results has received funding from the EC Seventh Framework Programme FP7/2007-2011 under grant agreement n° 214777.

More information
The most recent version of this document and all other public deliverables of IRMOS can be found at http://www.irmosproject.eu.
## Glossary of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>*aaS</td>
<td>Everything as a Service</td>
</tr>
<tr>
<td>AAA</td>
<td>Authentication, Authorization and Accounting</td>
</tr>
<tr>
<td>AC</td>
<td>Application Component</td>
</tr>
<tr>
<td>ACC</td>
<td>Application Client Component</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>ASC</td>
<td>Application Service Component</td>
</tr>
<tr>
<td>ASCD</td>
<td>Application Service Component Description</td>
</tr>
<tr>
<td>A-SLA</td>
<td>Application SLA</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>CRUD</td>
<td>Create, Retrieve, Update and Delete</td>
</tr>
<tr>
<td>DoW</td>
<td>Description of Work</td>
</tr>
<tr>
<td>EASC</td>
<td>External Application Service Component</td>
</tr>
<tr>
<td>EE</td>
<td>Execution Environment</td>
</tr>
<tr>
<td>FP</td>
<td>Framework Programme</td>
</tr>
<tr>
<td>FS</td>
<td>Framework Services</td>
</tr>
<tr>
<td>FSC</td>
<td>Framework Services Controller</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphic User Interface</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hyper Text Transfer Protocol</td>
</tr>
<tr>
<td>HTTPS</td>
<td>Hyper Text Transfer Protocol Secure</td>
</tr>
<tr>
<td>laaS</td>
<td>Infrastructure As A Service</td>
</tr>
<tr>
<td>ICT</td>
<td>Information Communication Technology</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>ISONI</td>
<td>Intelligent Service Oriented Network Infrastructure</td>
</tr>
<tr>
<td>IXB</td>
<td>ISONI Exchange Box</td>
</tr>
<tr>
<td>KPI</td>
<td>Key Performance Indicator</td>
</tr>
<tr>
<td>MAP</td>
<td>Modelling, Analysis and Planning</td>
</tr>
<tr>
<td>MDS</td>
<td>Monitoring and Discovery System</td>
</tr>
<tr>
<td>OMG</td>
<td>Object Management Group</td>
</tr>
<tr>
<td>OSD</td>
<td>Object Storage Device</td>
</tr>
<tr>
<td>OSS</td>
<td>Object Storage Servers</td>
</tr>
<tr>
<td>OWL</td>
<td>Ontology Web Language</td>
</tr>
<tr>
<td>PAAS</td>
<td>Platform As A Service</td>
</tr>
<tr>
<td>PDF</td>
<td>Probability Density Function</td>
</tr>
<tr>
<td>PH</td>
<td>Physical Host</td>
</tr>
<tr>
<td>QA</td>
<td>Quality Assurance</td>
</tr>
<tr>
<td>QoE</td>
<td>Quality of Experience</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>RAM</td>
<td>Random Access Memory</td>
</tr>
<tr>
<td>RMn</td>
<td>Resource Manager Node</td>
</tr>
<tr>
<td>SAAS</td>
<td>Software As A Service</td>
</tr>
<tr>
<td>Acronym</td>
<td>Definition</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
</tr>
<tr>
<td>SAML</td>
<td>Security Assertion Markup Language</td>
</tr>
<tr>
<td>SC</td>
<td>Service Component</td>
</tr>
<tr>
<td>SLA</td>
<td>Service Level Agreement</td>
</tr>
<tr>
<td>SOA</td>
<td>Service Oriented Architecture</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>SOI</td>
<td>Service Oriented Infrastructure</td>
</tr>
<tr>
<td>SP</td>
<td>Service Provider</td>
</tr>
<tr>
<td>SPI</td>
<td>Service, Platform, Infrastructure</td>
</tr>
<tr>
<td>SQT</td>
<td>Storage QoS Target</td>
</tr>
<tr>
<td>T-SLA</td>
<td>Technical SLA</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modelling Language</td>
</tr>
<tr>
<td>VAR</td>
<td>Virtual and Augmented Reality</td>
</tr>
<tr>
<td>VM</td>
<td>Virtual Machine</td>
</tr>
<tr>
<td>VMU</td>
<td>Virtual Machine Unit</td>
</tr>
<tr>
<td>VPN</td>
<td>Virtual Private Network</td>
</tr>
<tr>
<td>VSN</td>
<td>Virtual Service Network</td>
</tr>
<tr>
<td>WP</td>
<td>Workpackage</td>
</tr>
<tr>
<td>WS</td>
<td>Web Service</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Service Description Language</td>
</tr>
<tr>
<td>WSRF</td>
<td>Web Services Resource Framework</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>
# Table of Contents

1. Executive Summary ........................................................................................................... 10
2. Introduction .......................................................................................................................... 11
   2.1. Objectives .................................................................................................................... 11
   2.2. Changes in the D3.1.x series ....................................................................................... 12
3. Real-Time & Clouds ............................................................................................................. 13
   3.1. Real-time Interactive Media Applications .................................................................... 14
   3.2. Real-time QoS Provision in Clouds ............................................................................. 15
4. Platform Analysis ................................................................................................................. 18
   4.1. Requirements Traceability ......................................................................................... 18
   4.2. IRMOS Platform Phases ............................................................................................. 19
   4.3. Control Loops .............................................................................................................. 20
5. Capabilities ............................................................................................................................ 22
   5.1. Software-as-a-Service (SaaS) ...................................................................................... 22
       5.1.1. Application Wrappers ......................................................................................... 22
   5.2. Platform-as-a-Service (PaaS) ...................................................................................... 23
       5.2.1. Service Engineering ............................................................................................. 23
       5.2.2. Mapping & Benchmarking ................................................................................... 26
       5.2.3. Discovery ............................................................................................................. 29
       5.2.4. SLA Management ............................................................................................... 34
       5.2.5. Workflow Management ...................................................................................... 36
       5.2.6. Monitoring and Evaluation ................................................................................... 37
       5.2.7. Framework Services Security ............................................................................ 39
   5.3. Infrastructure-as-a-Service (IaaS) ............................................................................... 41
       5.3.1. Execution Environment Management .................................................................. 41
       5.3.2. Network Management ........................................................................................ 43
       5.3.3. Data Management ............................................................................................... 43
       5.3.4. ISONI Services Security ..................................................................................... 45
6. IRMOS Cloud Subsystems & Interfaces ............................................................................. 47
   6.1. IRMOS Platform Architecture Overview .................................................................... 47
   6.2. IRMOS Framework Services ....................................................................................... 49
       6.2.1. Overview .............................................................................................................. 49
       6.2.2. Service Engineering ............................................................................................. 51
       6.2.3. Service Management ............................................................................................ 58
   6.3. Intelligent Service Oriented Network Infrastructure (ISONI) ..................................... 62
       6.3.1. Introduction and relation to WP6 and WP7 .......................................................... 62
       6.3.2. General description and major building blocks ..................................................... 62
   6.4. Platform Interactions ..................................................................................................... 68
       6.4.1. Offline .................................................................................................................. 69
       6.4.2. SLA Negotiation .................................................................................................. 70
       6.4.3. Estimation Correction ........................................................................................... 74
       6.4.4. Monitoring and Evaluation ................................................................................... 76
       6.4.5. Service Resilience ............................................................................................... 77
       6.4.6. QoS Guaranteed Storage ..................................................................................... 78
7. Conclusions ......................................................................................................................... 82
8. References ............................................................................................................................. 83
Annex A.  User Requirements Traceability Matrix ................................................................. 84
Annex B.  Security Requirements Traceability Matrix ............................................................. 96
List of Figures

Figure 1: Mapping IRMOS to the Cloud .......................................................... 13
Figure 2: Example interactive application from film postproduction .................. 14
Figure 3: IRMOS Platform Phases .................................................................... 19
Figure 4: IRMOS Control Loops ...................................................................... 21
Figure 5: FS-ASC-Interface ........................................................................... 23
Figure 6: Mapping, Abstraction and Complexity ............................................. 24
Figure 7: Price list focus public IP addresses .................................................. 30
Figure 8: Price list focus EE resources ............................................................ 31
Figure 9: Price list focus virtual link resources .............................................. 32
Figure 10: Price list focus long term storage resources ................................. 33
Figure 11: SLAs Types between IRMOS Actors ............................................. 34
Figure 12: The monitoring and evaluation functionality of IRMOS ................. 38
Figure 13: IRMOS Security Model .................................................................. 40
Figure 14: Execution Environment .................................................................. 41
Figure 15: ISONI Services Interactions ............................................................ 46
Figure 16: ISONI VSN VL isolation .................................................................. 46
Figure 17: IRMOS Platform Overview ............................................................. 47
Figure 18: PaaS Architecture ......................................................................... 50
Figure 19: Service Engineering ....................................................................... 52
Figure 20: Performance Estimation .................................................................. 54
Figure 21: Performance estimation sequence diagram .................................... 55
Figure 22: Mapping Service Overview and Structure ..................................... 57
Figure 23: Service Management ..................................................................... 58
Figure 24: VSN Description .......................................................................... 64
Figure 25: Deployment process in the IRMOS architecture ............................. 65
Figure 26: ISONI Internal Functional Blocks .................................................. 66
Figure 27: IRMOS Offline Process .................................................................. 70
Figure 28: SLA Negotiation Process ............................................................... 71
Figure 29: SLA renegotiation triggered by the client ....................................... 72
Figure 30: SLA renegotiation triggered by application policies ....................... 73
Figure 31: SLA renegotiation triggered by A-SLA violation ............................. 74
Figure 32: Performance Feedback Architecture ............................................ 75
Figure 33: Sequence diagram for interactions of the Monitoring Service ....... 76
Figure 34: Redundancy modes of IRMOS ...................................................... 77
Figure 35: Storage interactions with ISONI .................................................... 80
List of Tables

Table 1: WP3 Deliverables List ................................................................. 12
Table 2: User Requirements Traceability Matrix ......................................... 84
Table 3: Security Requirements Traceability Matrix .................................... 96
1. Executive Summary

The purpose of this report (D3.1.4) is to describe the overall architecture of IRMOS platform which has been designed and implemented in the frame of IRMOS EU project. The main objective of IRMOS project is to build a Service Oriented Infrastructure (SOI) to support the deployment and operation of interactive applications with real-time requirements. The work described in the report was carried out in WP3 -IRMOS Platform Specification- for the analysis, design and specification of the architecture of IRMOS platform. This is the final report of WP3 that describes in detail, the overall functionality of IRMOS platform, the capabilities of each layer following the SPI cloud model, the individual components and their interactions. D3.1.4 (as well as all documents in D3.1.x series) is a self-describing document. The report explains the functionality of all building blocks and covers the real-time, virtualization, business and security aspects of the project with respect to the relation of IRMOS SOI with the cloud and *aaS concepts for agility, scalability and reliability.
2. Introduction

This document is the D3.1.4 “Final version of IRMOS Overall Architecture” deliverable of the EU ICT-2007-214777 IRMOS project. It presents the final IRMOS overall architecture (fourth report in D3.1.x series), a Service Oriented Architecture capable to support interactive applications with real-time requirements. The document is structured in four sections.

Chapter 3 describes the cloud computing concepts and its relation to the IRMOS platform design. The analysis presents the association of IRMOS platform layers to cloud classes of service-based systems and highlights the IRMOS innovations for real-time interactivity. Following the main objective of the project to “facilitate real-time interactivity in SOIs”, the required real-time attributes are explained along with the key features of the platform that are required to provide the real-time functionality.

Chapter 4 presents the WP3 work on the platform analysis which includes the traceability of the user requirements and how these are addressed in the platform design. In addition, the application phases in IRMOS system as well as the methodology that was followed for the control of the application, the platform and the resources.

In chapter 5 the final set of IRMOS platform capabilities is detailed based on the recommendations from the platform evaluation process that took place in the frame of WP8 in order to address the updated user and technical requirements.

Finally, chapter 6 presents an updated description of the IRMOS overall architecture and the main interactions between the subsystems (Framework Services and ISONI). For the analysis and design of the subsystems, WP3 worked closely with the component developers of the technical WPs, providing valuable feedback to them and also with WP8 producing guidelines for the evaluation and testing of the platform.

2.1. Objectives

The main objective of this document is to present the overall architecture and functionality of the IRMOS platform. The document defines the platform capabilities considering the requirements outlined in deliverable D2.1.1 “Initial version of Requirements Analysis Report” [2]. The overall architecture, the main building blocks, the individual components and their interfaces are described in detail in order to provide a complete view of the project objectives and innovations. The functional and non-functional capabilities are identified and analyzed, focusing specifically on the interactive real-time attributes with respect to business and commercial orientations as a cloud solution offering scalability, reliability and low operational cost.

The intended audience for this document are application and technology developers building service-oriented and cloud-based systems, as well as the European Commission services.
2.2. Changes in the D3.1.x series

The IRMOS architecture is expected to evolve throughout the lifetime of the project due to new and changed requirements resulting from prototyping and evaluation. To ensure that the architecture is accurately described and reflects current design decisions and implementation IRMOS will produce four architecture deliverables (Table 1). Each document will be self-describing and self-containing providing more details on the architecture and its capabilities as the project advances.

<table>
<thead>
<tr>
<th>Deliverable ID</th>
<th>Deliverable Name</th>
<th>Project Month</th>
</tr>
</thead>
<tbody>
<tr>
<td>D3.1.1</td>
<td>Preliminary version of IRMOS Overall Architecture</td>
<td>M7</td>
</tr>
<tr>
<td>D3.1.2</td>
<td>IRMOS Overall Architecture</td>
<td>M12</td>
</tr>
<tr>
<td>D3.1.3</td>
<td>Updated version of IRMOS Overall Architecture</td>
<td>M24</td>
</tr>
<tr>
<td>D3.1.4</td>
<td>Final version of IRMOS Overall Architecture</td>
<td>M33</td>
</tr>
</tbody>
</table>

This report (D3.1.4) as the final in D3.1.x series covers all critical aspects of the platform. First of all, the Cloud concept of the platform are analysed and described. Additionally, the results of the requirements traceability work are presented along with the final set of platform capabilities. Finally, the functionality and interfaces of the IRMOS subsystems are explained in depth and their interactions are clarified.
3. Real-Time & Clouds

Cloud computing offers the potential to dramatically reduce the cost of software services through the commoditization of IT assets and on-demand usage patterns. However, cloud computing is such a generalized paradigm it is impossible to consider ‘the cloud’ as a single set of Quality of Service (QoS) issues. To some extent, issues with cloud computing are necessarily related to the application characteristics and purpose(s). However, the complexity of determining resource provision policies for applications in such complex environments introduces significant inefficiencies and has driven the emergence of new classes of service-based systems called Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS), and Infrastructure-as-a-Service (IaaS). Today, these three main classes in the cloud services stack are generally agreed upon (SPI cloud model):

- **Infrastructure as a service (IaaS):** the provision of ‘raw’ machines (servers, storage, networking and other devices) on which the service consumers install their own software (usually as virtual machine images).
- **Platform as a service (PaaS):** the provision of a development platform and environment providing services and storage, hosted in the cloud.
- **Software as a service (SaaS):** the provision of a pre-defined application as a service over the Internet or distributed environment.

![Figure 1: Mapping IRMOS to the Cloud](image)

Clouds incorporate many architectural paradigms from distributed computing such as service-oriented infrastructures, Grid and virtualisation. Although IRMOS was initiated prior to the general acceptance of the cloud stack the architecture concepts identified in the project are essentially the same and map directly onto the current cloud classification (Figure 1). Following the cloud concept, IRMOS architecture supports different operators for each layer i.e. the execution and virtualization environment is offered as a service by a value chain entity to the framework services provider.
3.1. Real-time Interactive Media Applications

A real-time system is a computing system where the absolute correctness depends not only on the correctness of the output, but also on the time that this is produced. Traditionally, ‘real time’ refers to hard real-time systems, where even a single violation of the desired timing behaviour is not acceptable, for example because it leads to total failure, possibly causing loss of human lives. However, there is also a wide range of applications that also have stringent timing and performance needs, but for which some deviations in Quality of Service are acceptable, provided these are well understood and carefully managed. These are soft real-time applications and include a broad class of interactive and collaborative tools and environments, including concurrent design and visualization in the engineering sector, media production in the creative industries, and multi-user virtual environments in education and gaming. In particular, we focus on interactive soft real-time applications where one or more users interact with the application and with each other.

Soft real-time applications are traditionally developed without any real-time methodology or run-time support from the infrastructure on which they run. The result is that either expensive and dedicated hardware has to be purchased to ensure good interactivity levels and performance, or that general-purpose resources are used as a compromise (e.g. commodity operating systems and Internet networking) with no way to guarantee or control the behaviour of the application as a result. For such applications IRMOS needs to support techniques and services for modelling, predicting, provisioning and monitoring resource and QoS requirements commitments across software, platform and infrastructure layers, and applying such techniques in a general way so they can be exploited in different application domains.

An example from the postproduction scenario is shown in Figure 2 based on the Digital Film Technologies Bones Digital Dailies production system [1]. In the scenario, collaborative and distributed colour correction is performed as part of film postproduction. A postproduction house is contracted to perform colour correction to some film shots that will be selected by a film director during his review of the digital...
dailies of a film currently under production. The number of shots needing colour correction cannot be determined in advance as this depends on decisions made by the director. The director estimates that colour correction will be applied to approximately 30 +/- 10 minutes of footage. The colour correction and review activities occur concurrently and consists of: (1) the colour effects specialist downloads the digitised video from the Bones Service provider who provides storage, processing and networking resources procured from the Cloud, (2) the colourist applies initial correction to the video, by streaming it from the service provider to the post-house so they can determine the correction settings needed, and (3) the colourist and director interactively review the corrections that are applied through real-time stream processing of the video using applications installed at the service provider.

The application consists of storage for video and control metadata, a colour correction station, a variable number of image processing units depending upon the data rates and a load/balancer broadcaster responsible for delivery of the stream to the views. In the scenario shown in Figure 2, we use 4 image processing units at 24 frames per second. Each image processing unit queries the image processing control for the current set of colour correction parameters and caches these locally in order to be able to operate as quickly as possible on any frame. The load balancer and broadcaster component is the heart of the system with a clock ticking at the specified frames per second and a local buffer of processed frames. The load balancer instructs each of the four image processing units to process frame one by one. An image processing unit will retrieve an uncompressed frame from the video storage at 2500 Mbps, apply the colour correction according to its local parameter cache, downsize the original 12.8 MB frame to the size required for streaming (approximately 0.07 MB) and apply JPEG compression. The broadcaster part has a separate connection to each connected viewer through which it pushes the “current frame” according to the clock, in this case four streams at 13 Mbps. If the connection to the viewer is too slow then frames will be skipped. The component also has a control channel with each of the viewer components through which it can receive start, stop and seek instructions. During the review process the colourist may be asked to adjust the colour correction for a particular section of video. The colourist will work on this scene using the colour correction workstation in the same way as the initial phase and publish the correction the parameters to the image processing control which notifies the image processing units of the update. The image processing units then request the updated parameters and update their local caches accordingly.

### 3.2. Real-time QoS Provision in Clouds

The primary objective of IRMOS project is to develop cloud solutions that support real-time QoS guarantees for online interactive multimedia applications. The architecture considers the full service lifecycle of service-based systems deployed on cloud resources including service engineering, Service Level Agreement design, provisioning and monitoring. QoS parameters at application, platform and infrastructure levels are given specific attention as the basis for provisioning policies in the context of temporal constraints. The architecture supporting real-time interaction between distributed set of people and resources requires the following key features:
• **Real-Time QoS Specification**: specification language and associated toolkit for the specification of IRMOS applications and application service components considering both structure and real-time QoS.

• **Event Prediction**: QoS oriented service engineering methodology and models for predicting QoS requirements contingent on application and resourcing events considering temporal profiles of application service components deployed on virtualised infrastructures.

• **Dynamic SLA Negotiation**: SLA negotiation and management services supporting the dynamic negotiation of Application-SLAs considering customer requirements, and dynamic discovery of resource providers (Technical-SLAs) through automated processes of the platform and re-negotiation.

• **On-Demand Resource Provisioning**: provisioning services for application service components on virtualised infrastructures through combination of intra and inter the virtual service networks and execution environment management applying high availability techniques such as redundancy and live migration.

• **QoS Event Monitoring**: monitoring services for measuring Quality of Service at both application and infrastructure levels targeting trigger events for runtime adaptability of resource provisioning estimation and decision making.

Service-oriented principles are an important part of the IRMOS architecture throughout all layers of the cloud. IRMOS adopts a service oriented approach to allow services to interact dynamically and continuously, spanning between different domains, and ranging from the application level and down to the level of network resources management and the execution environment. The challenge is to carefully design and synchronize this rich set of services so as to efficiently operate, manage and reconfigure all the resources under real-time conditions, providing to the end users and to the associated applications the appropriate and required Quality of Service. All QoS terms must be dynamically negotiated and represented in SLAs between various actors in the value chain considering both application and resource level QoS guarantees. All platform and infrastructure capabilities are offered as on-demand services, although the architecture of the media applications vary from traditional n-tier enterprise applications to service-oriented workflows. The most important aspect in the architecture is that services and their orchestration are built and developed in a way so as to preserve the real-time attributes throughout the whole infrastructure including the resources, the virtual execution environments as well as networks to the applications and to the end user.

A major challenge for SaaS providers wanting to exploit the benefits of cloud computing is to manage QoS commitments to customers throughout the lifecycle of a service. The PaaS offers SaaS providers services and tools for estimating resource needs in advance of execution, negotiating QoS with service providers, provisioning virtualised resources and when assessing the probable technical and economic outcomes of provisioning policies and management actions if either the application or resources do not perform as expected or need to be adjusted. The approach considers analysis and decision support within temporal and business constraints to determine which actions are triggered offline (i.e. pre-execution) or online (i.e. during execution). Because faults are inevitably going to occur, strong fault detection and recovery mechanisms are implemented. This
can have a great impact on the real-time capabilities of the platform, since intelligent fault recovery mechanisms allows timing constraints to still be met in case of a failure. The performance of the monitoring and control loops between cloud layers is as essential factor in ensuring that QoS guarantees are maintained.

At the IaaS layer, real-time functionality is supported by the intelligent networking and the Execution Environment (storage boxes and execution nodes). IRMOS virtualizes several types of resources (network, storage, computational). The IaaS layer manages the resources (e.g. Linux Kernel, Network Routers) to the set of services that are deployed with the application in the virtual environment. The Execution Environment considers multitasking, threads with priorities and an appropriate number of interrupt levels to achieve QoS objectives.

An essential element of cloud computing is the ability to deliver on-demand services with minimal manual configuration. All subsystems can be self-managed and reconfigured in order to achieve management efficiencies, to react to QoS failures (such as an SLA violation or Network link failure) in a timely way and avoid the escalation of such problems.

Cloud utilisation involves several processes that span in different cloud layers and stakeholders. For example, IRMOS supports application developers in engineering their applications for the cloud following where possible standard specifications and methodologies, while other processes must support application provisioning and execution through the virtualised execution environment and networking infrastructure. Therefore, IRMOS platform does not only provide a set of services but also cross layer workflows that consider the control channels and information exchanges which are required to support real-time management of interactive applications throughout the full lifecycle (see section 4.2).
4. Platform Analysis

WP3 worked on the direction of extending and fine-graining the architecture in order to cover all user, technical and security requirements taking into account the progress in the technical work packages and the feedback from the development teams about the design and the technical limitations. The first step in the new iteration of the architecture specification was an analysis of the requirements and their relation with particular building blocks and components of the platform. The results of this analysis are summarized in the Requirements Traceability document which is included in the annex of this report. This document provides an overview on the user requirements and the project strategy to address them from the application and the end-users to the components implementation and the management of the resources. Additionally a detailed analysis of the IRMOS phases as well as description of the “control loops” is included so as to clarify the processes taking place and the components that are involved.

4.1. Requirements Traceability

During the first year of the project, the user requirements were exploited to produce the initial architectural design and the capabilities of the IRMOS SOI. In addition, during the second iteration of the platform specification this input was used to make a Requirements Traceability document (available in Annex A and Annex B) between the user requirements collected in WP2 and the building blocks and components of the platform that fulfil it. The document included matrixes describing the IRMOS, and more specifically, each WP strategy in order to address each requirement. Finally, guidelines for testing of the requirements have been created which will be used by WP8 to produce the final test cases and validate the IRMOS platform in the frame of task T8.2 (Testing and Evaluation).

The scope of the Requirements Traceability document is to provide a complete view on all IRMOS project requirements regarding their adoption in the architecture design and the implementation of the various components. It should be noted that the “requirements traceability” was a suggestion by the EC reviewers in the first projects review.

In the tables of the document (Table 2, Table 3) each user and security requirement is associated with the corresponding building blocks and components of the platform that fulfil it. Additionally the IRMOS, and more specifically, each WP strategy, in order to address each requirement, is described. Following the analysis of the requirement, WP3 also proposed guidelines for testing of the requirements. The user requirements (Table 2) will be used by WP8 to produce the final test cases in D8.2.3 and validate the IRMOS Platform in the frame of task T8.2 as long as the final prototype is affected. The prototype is running on an isolated testbed therefore security requirements are not covered by T8.2.

The tables reflect the work done in the project to:

© ICCS/NTUA and other partners of the IRMOS consortium 2008-2011
Gather and analyse the user and security requirements,
Specify the platform architecture,
Define the functional and non-functional platform capabilities,
Provide guidelines to the technical WPs for the implementation of the various components and finally
Establish methodologies for testing and validating the platform.

It should noted that 62 of the 72 initially identified requirements are addressed in the project while the rest 10 were considered either as application specific or as irrelevant with the R&D scope of IRMOS for which of-the-shelf solutions are applied. Part of the traceability of requirements throughout the platform and the project WPs was the conception of guidelines for testing and validation which are considered as useful input for task T8.2 to produce the final test cases.

4.2. IRMOS Platform Phases

To achieve the real-time functionality and the required QoS level for the applications, IRMOS platform operation is separated in phases, the offline, on one hand, where the application and application component are prepared (development, modelling, etc) and the online, on the other hand, where the resources are negotiated and reserved and the application initialized and operational.

Offline Phase
- Application Components Development
- Application Modeling
- Benchmarking
- Performance Estimation

Online Phase
- SLA Management
  - Discovery & Negotiation
  - Reservation
  - SLA Renegotiation
- Service Resilience
  - Redundancy
  - Live Migration
- Real-time Operation
  - Real-time aware Execution Environment
  - Intelligent Networking
  - QoS aware Storage
- Workflow Management
  - Components Initialization
  - Application Control
- Monitoring & Evaluation
  - Application & Resources Monitoring
  - Events Evaluation

Figure 3: IRMOS Platform Phases

Offline - Design-time service engineering
This phase includes the processes for developing or the adaptation of the application components to the IRMOS environment and the creation of descriptors and documents for the application operation such as models, mapping rules, initialization scripts, SLA templates and workflow descriptions.

Online - Negotiation and Execution
Initially the customer submits to the platform the high level requirements for the execution of the application which are mapped to low level parameters from the
respective services. Later the platform initiates the SLA negotiation for both the application and technical SLA with ISONI providers. As soon as the SLAs are signed, ISONI reserves the resources (computational, storage and network) for use within the requested time interval. When the execution of the application starts, the Framework Services are responsible for orchestrating and monitoring, up to completion, the workflow execution. At any time during the execution, in case of exceptions and/or SLA violations, the renegotiation of SLAs can take place in order to guarantee the QoS provision of the application and the application services given that the infrastructure layer is able to provide the additional resources required.

### 4.3. Control Loops

In order to provide QoS guarantees for interactive real-time multimedia applications, IRMOS platform provides a set of services and cross layer workflows that consider the control channels and information exchanges which are required to support real-time management throughout the full lifecycle. All subsystems are self-managed and reconfigured in order to achieve management efficiencies, and to react on QoS failures (such as an SLA violation or network link failure) in a timely way. To achieve this, IRMOS introduces three control loops at technical level and provide the necessary functionality in order to maintain QoS metrics across the architectural levels. The **IRMOS Control Loops** are the following and are depicted in Figure 4:

1. **Application Control:** It deals with the relationship between users and applications required to guarantee the application QoS. This control loop is managed by the application itself and the application developer in response to either user events or platform events. It is implemented with the use of models, workflows and tools that produce artifacts capturing the applications’ behavior and estimating resource needs in advance of execution. During runtime it refers to application monitoring that may for example trigger events or require for changes in the provided resources.

2. **Environment Control:** It deals with the relationship between applications and virtual resources in order to guarantee the platform QoS, as agreed in the SLAs. This control loop is managed by the platform services in response to application and virtualisation events. It is implemented by the framework services (set of tools) that support and manage the applications at run-time (e.g. actions triggered if either the application or resources do not perform as expected or need to be adjusted).

3. **Virtualization Control:** It deals with the relationship between virtual and physical resources in order to guarantee the infrastructure QoS. This control loop is managed within ISONI in response to platform or physical events. It is implemented by intelligent networking mechanisms as well as by the Execution Environment for computing and storage services.
The actual implementation of the control loops refers to tools and services used on different levels in order to monitor the applications’ execution, communicate possible events and take corrective actions if needed. We identified five (5) main processes / channels implementing the control loops, which are analyzed later in the document:

- Service Engineering
- Negotiation
- Reservation
- Monitoring and Evaluation
- Re-negotiation
5. Capabilities

The following capabilities for the IRMOS platform emerged from the application requirements and use cases as they are described in the previous sections and also from the general functionality that a real-time aware cloud platform must provide. An initial set of capabilities has been identified in D3.1.1 [5] and D3.1.2 [6] based on the early user requirements and use cases. The first update on the capabilities was included in D3.1.3 [7] where the capabilities are described in more detail and further features are presented following the updated requirements coming from WP2 and WP4. In this section we have finalized the platform capabilities based on the feedback acquired from the integration and testing processes of the project. The capabilities are categorized following a cloud motivated classification in:

- Software-as-a-Service (SaaS)
- Platform-as-a-Service (PaaS)
- Infrastructure-as-a-Service (IaaS)

As explained in section 3.2, real-time attributes span almost the whole IRMOS platform and have been taken into consideration in the definition of all the following capabilities.

5.1. Software-as-a-Service (SaaS)

5.1.1. Application Wrappers

Software is deployed within IRMOS by a process of wrapping the various application components with standard interfaces that provide configuration, modelling and control interactions with the framework services. IRMOS does not mandate specific application architectures as long as they can be connected to the necessary interfaces. For example, this could include a service-oriented application or traditional enterprise application. The role of Application Wrappers in the architecture is on only to act as a bridge between the application and the Framework Services or the infrastructure in general which are agnostic to the application internal functionality but also to provide error handling in application level and handle the states of stateful ASC (Application Service Component).

Each ASC implements a run-time interface that is accessed by the framework services (FS) for configuring, controlling and monitoring the ASC. To facilitate implementation a generic wrapper is provided which is responsible for the communication to the FS. The wrapper calls three different executables (most likely: scripts) which act as bridges to the ASC core. Through the wrapper's interfaces, IRMOS system is able to communicate with the application components for acquiring valuable monitoring data and controlling the operation of the components during runtime. The control communication includes the workflow defined procedures as well as various corrective actions triggered by the end user or the performance monitoring system of IRMOS.
The ASC developer must provide these three scripts and make the ASC core work with them. While this is rather trivial for ‘configure’ and ‘control’, the ASC has to provide monitoring information during run-time. For example the ASC writes current fps-values to a log-file and in turn, the ‘monitor’-script parses this log-file and translates this output into a given format which is passed back to the wrapper which itself passes it back to the FS.

5.2. Platform-as-a-Service (PaaS)

5.2.1. Service Engineering

The IRMOS platform provides service engineering tools to support the modelling analysis and planning of resources required by applications and to help actors in the value chain support in managing these uncertainties. Hence the service engineering environment provides capabilities to:

- construct conceptual models of activities and workflow models
- estimate performance of conceptual models in terms of QoS criteria
- estimate and optimise resource provisioning needs necessary to enact a workflow according to SLA guarantees
- design workflows and services that have measurable and predictable behaviour
- estimate service performance and responsiveness within normative workflows
- monitor application behaviour and QoS at the ISONI level
- refine high level application and workflow performance requirements into ISONI QoS parameters

The service engineering environment helps application developers understand the best way to take their applications and break them down into a set of services underpinned by ISONI in a way that is efficient and results in behaviour that is predictable. Aspects of performance modelling for real time systems have been considered by OMG in the UML profile standards [18] and [15]. Existing approaches and tools have been enhanced for the specification of IRMOS application requirements and used to develop structural models based on UML profile standards. The structural models will then form the basis of behavioural models and resultant performance data necessary to predict resource demands and develop appropriate provisioning strategies.
D5.1.1 [9] describes how IRMOS will advance the state of the art with respect to reactive real time application models. D5.1.1 contains details of the underpinning theoretical foundations for these models and how they apply to the IRMOS use cases. The service engineering environment provides techniques based on this foundation for linking performance requirements between the different abstract performance characteristics shown in Figure 6.

Applications with real-time attributes (e.g. the need for guaranteed completion by a particular deadline, or the need to achieve a certain level of interactivity for a concurrent user community) require careful planning when selecting service providers so that neither under-provisioning (likely failure of the application to execute) nor massive over-provisioning (unnecessarily high costs) occur. This planning, especially when there is uncertainty involved, is not currently well supported in conventional tools. From a real-time application modelling perspective, the important aspects of planning in these new value chains, and hence the need for modelling, is the separation of the resource-level (storage, processing, networking) parameters from the application-level parameters (e.g. video effects processing). At the application level, the parameter space can be large and complex, even for simple applications. For example, as shown in Figure 2, colour correction as a component application in film postproduction is characterised...
by a large number of parameters, including the characteristics of the film to be corrected, the corrections to be applied, and the use of colour correction within a workflow. The SaaS provider uses mappings to abstract this application-level complexity and characterise the application execution requirements in terms of its need for storage, processing, memory and networking. In effect, the SaaS provider is abstracting application complexity when one or more IaaS providers deliver the necessary resources. Likewise at the hardware resource level the space is equally large and complex due to the details of architectures and configurations, e.g. buses and caching, memory, networking, processors, storage.

The approach to modelling in IRMOS is threefold:

- At the workflow level, i.e. the business process within which one or more applications are used (e.g. in film production a workflow might be dailies review, colour correction, special effects rendering and audio correction), standard workflow languages are used, e.g. YAWL, BPML or BPEL. For high level descriptions, especially where human comprehension and authoring is required, YAWL and BPMN are suitable. BPEL is more appropriate for detailed workflows involving service to service interactions where the target platform is well defined.

- Services, their interfaces and QoS specifications are modelled using UML, in particular the UML Profile for Modelling and Analysis of Real-time and Embedded Systems (MARTE) [15] and UML For Modelling Quality Of Service And Fault Tolerance Characteristics And Mechanisms [18]. MARTE provides concepts and mechanisms explicitly for modelling and analysis of real-time and embedded systems, which makes it ideal for IRMOS.

- Service performance is predicted using a hybrid approach that combines statistical models from benchmark data with stochastic discrete event simulation. A service performance model is developed for each application and used to determine levels of reliability in respect to key performance indicators of the application. E.g 90% reliability that a response time will be less than 50mSecs or 85% reliability that less than 10 frames will be dropped over 1 hour.

The service engineering environment is dependent on data inputs from many of the other IRMOS subsystems (Workflow management, SLA management, performance monitoring, parameter mapping). A summary of these dependencies is given below:

- input from Workflow management
  - interdependencies between activities
  - timing constraints on start/stop/duration of activities
  - attributes of timing constraint, e.g. hard/soft/priority

- input from SLA management
  - QoS performance and response requirements for each activity class in a workflow
  - policy for timing and performance constraints and dynamic modification, e.g. in the eLearning scenario, the application servers required based on the number of end users
  - policy for workflow optimisation with regard to QoS parameters
5.2.2. Mapping & Benchmarking

In the context of business oriented SOIs the relation between the customer and service provider is defined through SLAs. To ensure SLAs can be negotiated on-demand it is necessary for the service provider to have (prior to the negotiation process) an estimation of the resources required for an application and be aware of the QoS parameters that have critical influence on applications performance. Thus a mapping mechanism is needed in order to translate these high level application QoS requirements (like fps, resolution of the video, application end time etc) into the low level resource parameters that are required in order to meet the end user constraints.

The first step in such a process is to have enough data samples that are produced by benchmarking or historical data of the applications for a range of expected inputs and different hardware configurations. Applications in IRMOS are composed of a number of fundamental ASCs. Each application developer can combine these ASCs into an application/workflow according to his needs or the goal he wants to achieve. If we benchmark on workflow level, then we must iterate the process for each case of workflow design and the results will be applicable only in this particular case. If we benchmark on ASC level then we will be able to extract conclusions about every possible combination of these components, thus making the design much more generic. Mapping and benchmarking is therefore performed on an Application Service Component (ASC)
level and adaptation to the workflow level is conducted by the PES (Performance Estimation Service).

In general, the role of the Mapping Service is to provide an ANN-based rule/model, that depicts the relationships between the ASC characteristics/inputs, the different hardware configurations and the resulting QoS levels. In this process, this service should be able to exploit the results of the benchmarking phase and produce the rules that will be accessible to the PES for further processing.

5.2.2.1. Benchmarking Phase

During the integration of an ASC to the IRMOS platform, the developer benchmarks its performance under different resource configurations. The ASC is executed as a standalone component while in a normal execution it is part of a workflow, taking input for example from another ASC. In the benchmarking mode inputs are simulated so that the standalone execution can be feasible. The selection of resources in this phase is performed randomly across the solution space and the effect of this selection will be depicted in the QoS metric specified by each ASC. During this phase the developer provides a range of values for the input variables or characteristics used by the ASC. These values represent different use cases of the ASC and ideally the cases where the behaviour of the ASC alters. The output is the QoS characteristics of the ASC (self-monitorable characteristics like response time in a request-response application, fps for the output video produced from a multimedia application etc). The performance data is provided as input for the Mapping service in order to create a model for the estimation of the ASC performance. This service is then used by the PES during workflow performance analysis. One characteristic that is of great importance is the study of the ASC’s behaviour with regard to the scheduling parameters. For example, assigning to an ASC 50% of a CPU is not adequate. We should also have information regarding this ASC’s response with regard to the granularity of this allocation (e.g. 50 msec over 100 msec, 0.5 seconds every 1 second etc). This is ASC specific and cannot be known by the platform in advance. That is why it is inserted as a benchmarked parameter.

The T-SLA for the entire benchmarking process may be one for all the benchmark test runs because it is not known what kind of resources are needed in order to meet the A-SLA QoS constraints. This means that an elementary benchmarking workflow may be constructed by the ASC developer and different parallel and independent branches of this workflow may be assigned different values, in order to obtain easily and swiftly the results. This also simplifies the interventions needed in the standard services and process sequences of the regular phases of IRMOS.

The monitoring of the resources used or the performance parameters defined is crucial for the benchmarking process in order to obtain the resource data necessary for the modelling phase. For example, if a specific parameter required by an application to measure the QoS (such as the time to process a frame) is not provided by the ISONI monitoring component, then the application developer/expert should provide means of measuring this, since he/she is the only one that has internal knowledge of when this happens.
The benchmarking process is not very different from the normal execution of a job. The main difference is that the resources needed, are not be determined based on the models but in a random fashion, evenly distributed in the sample space. Also due to the fact that benchmarking will be conducted on an ASC level, it is the job of the ASC developer to provide the necessary input set in order for this component to be executed as a standalone. This is due to the fact that some applications need specific input files, whose characteristics cannot be determined in an automated way by the Framework Services (like the resolution of a video file) Then as in the normal execution, the rest of the process remains the same (VSN creation etc.).

Due to the fact that benchmarking is considered as a phase and not a service, the main capability is derived from the entire platform functionality. This comprises the use of the normal tools for any regular application, with a few adjustments that are needed for this phase (similar to the aforementioned random selection and feedback mechanisms) plus the abilities that are needed from the ASCs (such as self-monitoring mechanisms). These additions require mainly an elementary benchmarking workflow for each ASC, with all the required components inside, like simulators etc. Therefore the ASC can be executed as a standalone application using a standard A-SLA, comprising of multiple parallel and independent branches, each with specific high level ASC inputs, in order to take the necessary measurements.

5.2.2.2. Mapping

The first step is to obtain the benchmark data and create the model. This data can be obtained either by ISONI if the feature wanted is indeed within ISONI’s monitoring capabilities or from the ASC itself, which must have an internal way of measuring this specific feature, e.g. to mark the start and end time of a frame’s process in order to know exactly how much time this process takes.

The mapping service must be in a position to receive a data set from the Monitoring DB inside the Framework Services regarding the behaviour of the ASC in relation to the input data parameters used during the execution of the specific ASC on a specific hardware platform. It then creates the ANN model of the ASC, connecting these high level input parameters and the hardware used with the resulting QoS levels expressed in a specific ASC metric. The model, stored it in a repository, can be retrieved in the future and based on the current input data, and is used for the run time estimation of the QoS parameters required. The mapping process should be as generic as possible. Also the mapping service should be accessible by the PES in order to provide the translation from high level to low level parameters. This can be achieved either by requesting the Mapping Service directly or by providing access to the repository where the models are stored, in case there is the need for the model and not just for a single estimation value. The necessary scripts for the second case have been created, based on the needed functionality and interfaces.

The basic constraint for the Mapping service is that the models that will translate the high level parameters to low level ones will be created based on the ASCD. In the ASCD, the developer must determine which high level parameters should be translated and provide the according ranges or allowed values that these parameters may acquire. If a
high level parameter is not included in the ASCD then is not possible to be incorporated in the model and therefore in the A-SLA. Each parameter that is intended to be chosen by the client in the A-SLA should be included in the ASCD, so that based on his preferences a suitable estimation may be performed and according resources may be reserved.

During the online negotiation phase, the mapping service must be able to receive the high level values for a particular run and produce the low level values to be used during SLA negotiation. Due to the fact that the PES needs to have several requests towards the ANN rules in its attempt to optimize the workflow, it was decided that these ANNs will be stored in the ASC repository so that they can be retrieved by the PES and can be executed locally. This was planned for performance reasons due to the fact that possibly thousands of queries may be performed towards the rules and their response through a service interface would be too time consuming. However, service interfaces have also been implemented for the Mapping Service.

5.2.3. Discovery

Discovery is the process that enables the localization of either specific application service or ISONI resources that nearly matches the user operational and business criteria (e.g. the user requests Intel computing nodes and a Digital FX unit in the same domain). In a large scale, with heterogeneous network of providers, the number of available resources increase rapidly and therefore it is hard to locate the desired service, so it is important to develop mechanisms which will be able to do the matchmaking efficiently and in time. Discovery Service may involve several components that span between different layers of the platform and interact each other and with other platform services.

The Advertisement and Registration of all type of resources so as to be discovered includes specific criteria about their type and QoS characteristics (e.g. CPU types, RAM, storage, QoS constrains) as well as the pricing information. As a result, the matchmaking mechanism that will be implemented by the Discovery components is based on the information provided by the ISONI infrastructure through the Advertisement system. The ISONI capabilities [14] that are available to the Discovery Service are only related with the availability of each ISONI Provider in terms of resource type offered. This does not include of course the amount of free resources (CPU type, RAM, storage etc) but the supported QoS type, in the sense that some resources will be used in conjunction with certain QoS constraints. In addition, the IRMOS approach for the service and resources descriptions is an XML based syntax expression. Also, pricing info is advertised by ISONI and registered to the Discovery Service.

Pricing information supplements the ISONI capability information that provides the basis for the discovery service to select an ISONI domain. The price information is provided as an XML document. The XML schema is shown in Figure 7 to Figure 10.
The price list provides pricing information about resource used for deployed application services. It addresses usage of public IP addresses, Execution Environment (EE), Virtual Link (VL) and Long Term Storage (LTS) resources analyzed in section 6.3. The prices are specified for a specific valid time period in general as well as time period for dedicated bundles defined by ‘Valid_From’ and ‘Valid_Until’ allowing the specification of scalable and bundled resources.
A bundled price offer, especially for EE, specifies the package price for a combination of EE resources (computational, storage and network). In contrast the scaling part allows price definition of the individual EE resources. Bundles and scaling price specifications may exist in parallel.

For EE resources the prices depends on:
- Range_Type_enum: Between_upper_and_lower, More_than_upper, Up_To_Upper allowing to specify the type of range
- CPU_Upper and CPU_Lower specifies the range limits (depends on Range_Type)
- RAM can be specified just for dedicated values
- Volatile_Storage price definition depends on Range_Type (upto, perUnit)
Figure 9: Price list focus virtual link resources

For VL resources the prices depends on:

- **VL_type** specifying best-effort (BE), best-effort-with-granted-bandwidth (BEP), real-time-streaming (high delay), real-time-interactive (low delay). The delay budget for low and high delay is given in the ISONI capability information. Therefore the provider can specify via 'VL_type'.
- **Max_Bandwidth** specifies the maximum bandwidth, which is limited by ISONI during execution phase.
- **Granted_Bandwidth**, which specifies the granted bandwidth during execution phase. (Note: The throughput experienced by the application may be less)
Figure 10: Price list focus long term storage resources

The pricing for Long Term Storage (LTS) are decomposed in prices for the LTS pool and the LTS access. The LTS pool represents the permanent storage capability itself negotiated via a separate SLA. Usually this occurs independently of any deployed application accessing the LTS pool. The pricing of any Service Component accessing the LTS pool is reflected by ‘LTS_access’ in the pricing list.

For LTS pool resources the prices depends on:
- Range_Type (upTo, perUnit)
- LTS_Pool_Capacity: Maximum size of LTS to be used
- Resiliency: for protected or unprotected storage content
- Storage class: reflecting the strongest QoS capability needed for accessing the LTS pool based on the profile of the application that uses the service (see 5.3.3.2).

For each SCs accessing the LTS pool the prices are defined depending on:
- I/O bandwidth: required I/O bandwidth accessing the LTS pool
5.2.4. SLA Management

The SLAs have a key role in the service oriented business environments. Based on the user requirements, use cases and the value chain analysis in WP2, we concluded that more than one SLA types are required in IRMOS and furthermore the SLA related processes should be automated and dynamic.

There are two different types of SLAs automatically managed by IRMOS. As it was explained in the value chain description [12], there are different IRMOS actors placed at different levels within IRMOS: we have clients (consumers or customers), IRMOS Application Providers, IRMOS Providers and ISONI Providers. One player in the value chain uses the service of one or multiple providers and adds value to the product that is offered to its clients. Each player offers a different service, and therefore a different SLA is required to fix the requirements, guarantees and compensations of the service. The parameters typically presented in a SLA are not the same for the customer than for a resource provider (ISONI Providers). This is the main reason why there should exist different one-to-one agreements between the actors here identified (see following picture).

![Diagram of SLAs Types between IRMOS Actors](image)

- **The Application SLA**: This is an agreement established between the Client and the Application IRMOS Provider (WP4). Both actors represent the consumer/customer of the applications that are going to be deployed over the IRMOS platform and the Application Provider itself. This SLA contains the high level QoS parameters of the application defined by the user.
• **The Technical SLA**: This agreement is going to be negotiated between the **ISONI Provider** (WP6&7) and the **IRMOS Provider** (WP5). Therefore, this agreement contains low level QoS parameters associated with the infrastructure. The translation of the high level parameters coming from the Application SLA to the low level parameters is performed during the Mapping process, described in section 5.2.2.2.

There is another SLA established between Application Provider and the IRMOS Provider, but this relation is a long-term relationship that is previous to the automated SLA negotiation process and that does not have to be negotiated every time a new customer uses an application over the IRMOS platform. Therefore, during the SLA negotiation phase this SLA is considered as fixed.

Most existing SLA Management procedures consider the negotiation to be a process that takes place before the execution phase. Once the negotiation has been produced the service is monitored against the corresponding SLA established and in case there is a violation (or potential violation), then several actions (reflected in the SLA in most cases) are performed. However, sometimes the causes and origin of the violations could be somehow solved by establishing again a process of negotiation. This is called SLA renegotiation or negotiation at execution time. This is one of the innovative aspects of SLA Management in IRMOS, this is also known within IRMOS as dynamic SLA. In that sense, the SLAs can be updated during runtime to reserve additional resources following a user request or a corrective decision from the platform in order to maintain the requested QoS. A prerequisite in cases of renegotiation is always the availability of the additional resources in the infrastructure layer; otherwise the renegotiation of the SLA will be rejected from the ISONI provider. Another important aspect in the SLA Management within IRMOS is the nature of the negotiation process: in IRMOS, we consider the SLA negotiation process to be automatic, this means that the part of the IRMOS architecture involved in the process has to some extent the capacity to decide without human intervention about the SLA negotiation, this process is based on policies defined by the actors involved in the negotiation.

There are three reasons in IRMOS that can trigger a SLA re-negotiation process:

1. **The user** would like to change some parameters of the application that affect the quality of the service. For instance, change the frames-per-second provided by the streaming application will require to increase the resources provided by the infrastructure to support the new frame speed.

2. **The platform** after the initial negotiation with the ISONI provider of the infrastructure resources (T-SLA), detects (through monitoring) that the estimation of resources was too low to support the expected quality of the application. The platform can initiate the re-negotiation of the Technical-SLA with the ISONI provider, without affecting the Application-SLA already in place, to solve the application degradation situation.

3. **The application** could be configured with a scalability rule to allow the re-negotiation of the resources when the requirements of the application increase. For instance, the application could have a rule to automatically launch a re-
negotiation process each time the number of user increments in 15, so to have enough resources for the new users connecting to the application.

5.2.5. Workflow Management

Workflow management is another essential aspect of IRMOS, which provides the required flexibility and dynamicity for the interactive applications deployed and executed on the platform. A prerequisite for the efficient workflow management is a workflow description. IRMOS workflow modelling and definition tools are used to generate a workflow description which is an inextricable part of the A-SLA between the customer and the platform. This workflow description will undergo various transformations during the negotiation and reservation phases until the final concrete workflow is generated. The concrete workflow will be used by the workflow management services of the platform during the execution of the application.

One of the innovations in IRMOS is the workflow management implementation. Because of its real-time orientation, the workflow management has to react very fast on the workflow processing and the various control actions in order to maintain the QoS level and the smooth operation of the application services. To this direction the Workflow Enactor Service is deployed within the VSN to have direct access and control on the ACSs. This service is responsible only for components of the particular VSN and other instances are deployed for the other applications. All Workflow Enactor Services (instances) are controlled centrally by the Workflow Manager Service. The Workflow Enactor Service is responsible for configuring the ASCs prior to their execution. This is done as soon as the ASCs are ready to receive the configuration information that is needed, in order for them to be ready for execution at the start of the execution phase.

Moreover, the Workflow Enactor Service manages the workflow during the execution phase of an application. This service is deployed within the corresponding ISONI and provisions are made during the set up phase for it to be included inside a VSN. The Workflow Enactor starts and stops services according to the workflow description document that is created by the workflow modelling and definition tools. The service is also able to receive input from the ASCs through the IRMOS Monitoring and Evaluation mechanism. This mechanism, based on the overall status of the resources and the application, triggers various corrective actions which the Workflow Enactor Service executes. For example, in the case that an application component runs in redundancy mode and a failure occurs in the primary virtual machine, once the secondary virtual machine is deployed, the Workflow Enactor Service reconfigures the respective workflow parameters, initializes the ASC(s) and starts them so that the application is able to continue its normal operation.

Part of workflow management can be considered the Fault Detection and Recovery capability of IRMOS, which affects though all the layers in order for the platform to be robust. Therefore Fault Detection and Recovery can be divided into two different levels, namely Framework Services level and ISONI level. It uses health supervision for virtual machines, virtual links and storage objects to detect outages at the infrastructure. Depending on the type of outage ISONI is able to restore the previous application without any further notice for the Framework Services or the application. The
Framework services will mainly be responsible for errors that occur on the software level. With this in mind the Framework Services will provide the following capabilities:

- If a software error occurs, the end user gets a notification and has the option to restart the execution of the task on the same resource or a different resource through renegotiation. This is provided by IRMOS as it is the simplest and fastest way to recover from minor errors.
- The system will provide the user with the ability to define corrective actions in certain cases of failure. These actions can only be part of the current VSN setup and agreed in the SLA. This can make the system much more tailored to the needs of the users.

All parts of the workflow enactment engine rely heavily on the **Monitoring and Evaluation Services** (described in the next section) in order for it to make decisions during execution or in case of a software fault reported by an ASC. Workflow enactment also relies on **SLA Management** in order for configuration parameters to be passed to the ASCs. The service also communicates with the **Portal Service**, receiving commands to start or stop the workflow execution.

### 5.2.6. Monitoring and Evaluation

While IRMOS's basic concept is real-time application execution within a fault tolerant system, the monitoring capabilities rise as topics of great importance. The overall scope of the Monitoring and Evaluation Service is to gather information from various components and levels, process them and forward them respectively to other components of the IRMOS platform as the basis for both design time and runtime decision making. Monitoring will provide runtime information for QoS violations at both application (Framework Services) and infrastructure levels (ISONI). The capabilities of the Monitoring Service are extended into aggregating and storing the information to the Historical Database (for offline usage) for updating the performance models in cases of inaccurate estimations.

The monitoring functionality of the IRMOS platform will be provided by the Monitoring mechanism. The following component diagram depicts the architecture as well the component interactions and interfaces for the monitoring system.
Monitoring and evaluation processes take place during the Execution phase and the Benchmarking phase of an IRMOS application. During the execution phase, the Monitoring service gathers information both about low-level performance parameters coming from the ISONI infrastructure through its ISONI Gateway as well as information about high-level performance parameters coming from the Application Service Components that are being executed. The monitoring information are aggregated and relayed to the evaluation mechanism of the platform. The Evaluation is responsible for receiving and evaluating numerous events that can occur during runtime, including events propagated by the ISONI layer and application events, as well as constantly checking the monitored parameters against the requirements present in the SLA. If any violation is detected, proper events are generated and propagated to other IRMOS services to trigger the necessary corrective actions. During the benchmarking phase, monitoring information is propagated to the Mapping Service for calibrating the models (mapping application parameters to resource needs).

The Monitoring and Evaluation Services are built upon open standards such as WSRF and WS-N specifications which define the mechanisms used to describe information sources, access information via both queries and subscriptions, and manage information lifetimes. By using this XML based specification for the service interaction we can succeed in the manipulation of any kind of monitoring data and notifications through a generic and standardized mechanism.
5.2.7. Framework Services Security

The IRMOS security requirements and implementations are documented in WP2. The project recognises that cloud security is an active and broad research topic covering many business, regulatory and technical issues. IRMOS's objectives focus on research related to Quality of Service in clouds rather than security topics and therefore IRMOS does not try to advance the state-of-the-art in cloud security solutions. IRMOS does however recognise that appropriate security is important for the exploitation of multi-stakeholder distributed systems.

IRMOS adopts off-the-shelf solutions at different levels of the infrastructure stack. These solutions can be deployed and configured to meet the security requirements of each of the stakeholders within the IRMOS value chain. However, we acknowledge that significant research and development work will be necessary before cloud security becomes as flexible as current solutions for other cloud requirements such as scalability and elasticity.

What is interesting about cloud and cloud security is that it is not possible to consider 'the cloud' as a single set of business models with a single set of security, privacy and trust issues. To some extent, issues with cloud computing are necessarily related to the application purpose and the application-driven approach used in the IRMOS requirements analysis is the only effective method of deriving a system that can meet operational security requirements.

The IRMOS trust model is related to the IRMOS value chain as described in deliverable D2.1.3 Final version of Requirements Analysis Report. The value chain describes the business stakeholders expected to contribute services for the delivery of real-time interactive applications. The stakeholders map directly onto the cloud architectural model:

- ISONI Provider: an infrastructure-as-a-service provider offering virtualised compute, storage and networking on demand
- IRMOS Provider: a platform-as-a-service provider is a market maker, offering tools and services to support the engineering and deployment of applications targeting virtualised infrastructure
- Application Provider: a software-as-a-service provider offering cloud deployable applications
- Customer: a budget holder responsible for procurement and use of software services
- Consumer: a user of software services with delegated authority from a customer

The IRMOS security model is shown in Figure 13. IRMOS does not provide a single security solution but applies security solutions that are appropriate to the communication protocols they are trying to protect. What is applicable for the framework services may not be applicable for ISONI or application security. Figure 13 highlights the important security domains in relation to stakeholders in the value chain and the systems they are operating. There are four domains
- **SaaS**: Application Security: interactions between customers and consumers of using applications (Pink)
- **PaaS**: Platform Management Security: interactions between customers, application providers, IRMOS provider and ISONI provider for managing applications and provisioning infrastructure (Yellow)
- **IaaS**: Infrastructure Management Security: interactions between IRMOS provider, hardware and networks for managing infrastructure (Green)
- **IaaS**: VSN Security: interactions between infrastructure at execution time (Red)

Each of the security domains are discussed in more detail in D2.4.1 [4]. For Framework Service security, the IRMOS Provider is responsible for brokering interactions between Customer’s, Application Providers and ISONI Providers. Some interactions are human-system others are system-system. Both the customer and the application provider access the framework services using portals that are developed for each purpose. The IRMOS portal is targeted at the customer and allows them to negotiation and monitor A-SLAs with an IRMOS provider. The Performance Management portal is targeted at the application provider and allows them to configure applications, models and management policies, and monitor the performance of applications and A-SLAs. Both portals are based on Apache Tomcat and offer username/password authentication with role-based access control.

The PaaS services offered by the IRMOS provider are based standard web service frameworks. Two different frameworks are used Globus [19] and GRIA [20]. Globus provides the foundation for service management and the ISONI gateway. GRIA provides service-oriented versions of the service engineering tools. Both GRIA and Globus provide implementations of web service security standards such as WS-Security so transport and message level interoperation is no problem. However, each provides a different set of solutions for token issuance and authorisation. IRMOS places no
restriction on how authorisation is implemented between framework services. We can envisage that situations that use SAML [21] protocols, WS-Trust/WS-Federation or even Globus grid-map files if there were few dynamics in service relationships.

5.3. Infrastructure-as-a-Service (IaaS)

5.3.1. Execution Environment Management

Essentially, the Execution Environment offered by WP6 is a virtual machine with added capabilities (Figure 14). These added capabilities are what make our Execution Environment unique, innovative, advanced and perfectly suitable for the goals of the IRMOS project.

![Execution Environment Diagram](image_url)

Figure 14: Execution Environment

One of the fundamental capabilities of an EE is its isolation capabilities. The EE prohibits services from interaction with other services that are not explicitly allowed by policy. Services are also isolated from the failure or “misbehaving” (like a program bug that makes the software run in an infinite loop requesting all CPU power) of other services.

Real-time is a distinctive capability of the EE and the IRMOS platform will be able to detect and exploit this real-time capability. To achieve this innovative feature, we are developing a global resource management and allocation policy using appropriate scheduling methodologies.

The EE will have redundancy and fault tolerance capabilities that, combined with the migration process, will assure that technical SLAs offered by ISONI Providers
Committing EE resources are guaranteed even in case of failures. Particularly challenging is ensuring that real-time applications suffer as little as possible, if at all, during a fault.

The fault tolerance functionality (further detailed in section 6.4.5) can be achieved by providing physical backup capacities for the single physical instances. Another capability of the EE is to calculate when to start the booting process so that the service becomes ready just in time. Booting too late would violate the SLA but booting too early would allocate resources during a non necessary time interval.

The EE has the capability of providing appropriate mechanisms and interfaces for resource metering, monitoring and reporting of the software running in it.

The project, and in particular the WP6 –Execution Environment-, is progressing to fulfil these capabilities. Technical discussions lead to identifying and detailing the capabilities the Execution Environment must satisfy. Many of them are in relation of integrating the Execution Environment with the IRMOS modules developed by other workpackages and, specifically, integration within the ISONI platform developed jointly by WP6 and WP7 (Intelligent Networking).

The main identified capabilities are in the ISONI context and shared by the WP7. Those are:

- Aggregation of data and several data detail levels, for the sake, among others, of scalability. i.e. there is no sense of exposing all the details of the execution environment resources (e.g. RAM available in all the computers) to layers that just want to know whether a specific request can be fulfilled (e.g. needing 4Gb RAM)
- Publication of available resources without disclosing confidential operational information e.g. the Execution Environment will announce that it has Intel and AMD CPUs but not how many of each type are available and used
- It is foreseen that the ISONI interface towards the Framework Services (WP5) will use the SOAP protocol thus the Execution Environment and ISONI must implement this protocol for this interface
- The execution environment must provide infrastructure events and monitoring data, as detailed in section 5.2.6
- Seamless integration of data storage into a VSN without the requirement for virtualisation of that storage as a reduced capability replica of the original (this allows for rapid integration of new technologies when they come on-stream by not requiring custom IRMOS changes of standard protocols)
- Multi-level protection of data both from security and resiliency perspectives
- Soft Real-time scheduling capability: whenever a provider will want to share the same physical resources among multiple Service Components concurrently running, the Execution Environment must be capable of guaranteeing a predictable timing behaviour of the individual activities, independently of one another.
5.3.2. Network Management

Network management will provide a failsafe network overlay by connecting components through an intelligent network infrastructure, enabling automated SLA negotiations and monitoring & reporting to enable delivery of QoS assurances as required by real-time interactive applications.

Specifically the following capabilities will be provided by the Network Management:

- One of the fundamental capabilities of ISONI is to offer an infrastructure for automated provisioning of multi-component services (described by VSNs) across different locations under consideration of “edge-to-edge” service guarantees. One of the key components required to fulfill this ambitious task is to provide QoS guarantees in the transport network, which connects the service defining components in potentially different locations.
- The important concepts with respect to network virtualization in the context of IRMOS are segmentation, isolation, and encapsulation. Segmentation allows several different services to share a physical link with given QoS properties. Means for isolation and encapsulation are needed to mimic a dedicated virtual network for every individual service and to suppress any unwanted crosstalk between services sharing the same physical link.
- Network resource management focuses on ensuring the efficient and reliable utilization of available network resources. This includes topics like ensuring Quality of Services, ensuring reliable connectivity, reducing costs and so on. In an infrastructure oriented to support Real-Time applications and services, it is a crucial aspect the global path supervision and link control tasks, for checking both the status of the network and links.
- When focusing on soft real-time systems, the research efforts shift completely on the problem of finding appropriate trade-offs between an efficient and optimum resource usage and the guarantees provided to the real-time activities. In this context, QoS optimization and adaptive QoS control for soft real-time tasks are particularly relevant research areas.

In order to support redundancy and live migration as performed on Execution Environment layer to meet real world threats like: components failure, connection failures, etc, the intelligent network layer will support those EE counteractions on network layer.

5.3.3. Data Management

It should be noted that the Data Storage implementation method employed has been substantially changed since the previous versions of architectural reports. Particularly in the storage infrastructure which now employs a ‘Lustre’ file system with dedicated Storage network within an IRMOS Node. The primary goal of storage within IRMOS is to provide QoS assured access (QoS guarantees) to data storage, which is provided by the Lustre solution. The Lustre solution provides scalable performance and capacity to match the requirements of any of the demonstrations required. The framework and provision of QoS guarantees by the storage solution is described later in section 6.4.6.
Storage uses its own network to access the corresponding hosts as it is typically the case in large providers. As described in D7.2.1 [13] the combination of different types of network usage patterns can decrease the throughput exponentially. Having a separate network for this type of special storage enables the provisioning of high data rates without disturbing other application data flows such as video or audio.

“Storage” and “data” are not necessarily the same in this approach. These two concepts are considered separate. Data is considered as an amount of virtual space where your data is kept. Storage defines how this data is accessed and by which virtual machines. Certain data “lives” in a specific storage device. Although this data might be moved around in order to help resource allocation typically this will be the other way around due to the size of this data (e.g. TB) and will have to be treated as a constraint in the resource selection mechanism.

The Physical reference to the data is kept inside ISONI to obey to the virtualization premise. WP5 defines a certain data space in the T-SLA which can be referenced in the same T-SLA or later on in future negotiations in order to use this data inside a VSN indicating at that point in which way (QoS) this data is to be accessed by which ASC (This is described later as the ‘Reservation’ and ‘Commit’ processes). In order to support the domain and node concept of the ISONI, this reference – from T-SLA storage definition to physical storage – will be kept at two levels: one level keeps track of in which node certain data is currently stored and the other level keeps track of the absolute path to the data in the specific storage device. The former is kept in the SLA-manager and the latter is kept in the resource manager node. Security requirements will be considered in the second iteration of the project.

Storage exists on each node. It is connected to the Physical Hosts (PHs) via a storage network using Lustre protocol. To an application (SC) the Storage network and its protocol is invisible. Lustre network file systems are run in each PH below the virtualization layer. Directories in the Lustre network are mapped through the virtualization layer and appear as storage devices to the virtual machines. Files within the Directories are managed as Objects inside the storage system and are distributed across multiple Object storage elements as required to provide necessary Performance and resiliency. This is highly scalable and flexible allowing dynamic expansion and redistribution of data to meet the needs of different applications.

5.3.3.1. Persistent storage

Reservation: Using this approach, persistent storage capacity can be realised by Framework Services requesting a storage pool to be provisioned using the T-SLA as described above. The resource manager domain (which must keep a reference to the node that the data is being Reserved on) pre-reserves the storage resource via resource manager node (which keeps reference of the physical data location) this negotiates the storage Reservation with the storage manager to provide the storage capacity with the targeted performance capability on the required node.
At **Commit** time, this “data” pool, or portions of it, will be allocated to ASCs through a T-SLA negotiation between the Resource Manager Node (RMn) and the Storage Manager which returns a Lustre file system reference to allow the RMn to mount the file-system and access data on the physical host. Directories in the Lustre network are mapped through the virtualization layer as described above, the data appears as a storage device to the SC with the T-SLA QoS characteristics negotiated between the RMn and Storage manager.

### 5.3.3.2. Framework Services Modelling

As storage performance is greatly affected by the access profile of an application, in order to assist Framework services in understanding how storage is characterized the Pre-Reservation process will provide data to allow a mapping between four or five high-level storage access I/O profiles (such as “database”, “video playback”) and low level storage QoS parameters. Example access patterns could be: Database server, Video database, Video playback, Mail server. These profiles can be learned by means of benchmarking or other sources. A choice must be made when ‘Reserving’ the storage which access profile is required (as well as Capacity and Lifetime of that storage provision).

### 5.3.4. ISONI Services Security

ISONI has to deal with certain security aspects. First the services provided through the ISONI Gateway as indicated in Figure 15. Any interaction with the ISONI system requires authentication. The ISONI Provider is responsible for setting up the authentication system and configuring the services to use it. The ISONI Gateway offers a single access point interface, which implies an authentication as a first step and subsequent authorization for the use of each ISONI service. The Registrar is the core of the ISONI AAA (Authentication Authorization Accounting and will provide the required AAA information for any actions on the ISONI Gateway, whether locally or remotely.

The **ISONI Gateway** has been put in place to let ISONI **enforce authentication and authorization for FS interactions**. Three components communicate through the ISONI Gateway with IRMOS Framework Services. The **ISONI Info System** uses the ISONI Gateway to securely advertise the ISONI’s VSN ontology at the IRMOS Providers’ (i.e. the specification of allowed directives and quality limits in the particular ISONI Domain). The **ISONI SLA Manager** and the **Repository Manger** are registered with the ISONI Gateway to become the denoted service instance for T-SLA Negotiation. Henceforward, the protocol for T-SLA negotiation between the IRMOS T-SLA Manager and the ISONI SLA Manager is secured by the ISONI Gateway. Analogously, each **Deployment Manager instance** of a VSN deployment registers its endpoint with the ISONI Gateway to deliver the ISONI low-level monitoring reports to the IRMOS Monitoring FS.
Second, a deployed VSN without any external connectivity specified is always isolated, since concurrent deployment of independent, customer-specific application service networks in IRMOS demands for mutual network address space isolation to prevent crosstalk. The VMUs just can interact over the network on the in the VSN specified Virtual Links (VLs). Same applies to the specified external connection until the border of an ISONI domain. Firewall function on the transition point at the domain border is a first security stage protecting the any deployed VSNs. These issues are depicted in Figure 16. ISONI is not application aware, which enforces the applications inside a VMU to implement security mechanisms for traffic passing the transition point.

Third the isolated network paths between ISONI nodes as indicated in Figure 16 are secured in case they transferring insecure networks. Authentication as well as encryption can be applied in such cases. This is also relevant for resource collaboration of different ISONI domains as described in [11].
6.  IRMOS Cloud Subsystems & Interfaces

6.1. IRMOS Platform Architecture Overview

The objective of IRMOS is to research, design and develop a real-time aware Service Oriented Infrastructure for interactive multimedia applications. During the last year of the project, WP3 was able to exploit the output of WP2 and WP8 on requirements analysis and evaluation, and extend the architecture description to include work done in the technical WPs (WP5, WP6 & WP7) reflecting the complete set of IRMOS functionalities. The final version of platform specification details the capabilities all individual IRMOS subsystems and clarifying how these subsystems are interacting to achieve real-time operation and interactivity for both the platform and the applications. The bird’s eye view of the IRMOS platform architecture is shown in Figure 17.

![IRMOS Platform Architecture Diagram]

This figure presents the two main building blocks of the IRMOS platform: the Framework Services and ISONI which map respectively to the PaaS and IaaS layers of the cloud model. In IRMOS, we followed an innovative approach on how these blocks will interact, and their relation is considerably different of the conventional SOA or Grid platforms because of the real-time and virtualization capabilities of the designed platform. Initially, the Framework Services provide service engineering tools for the
Application Developer and provisioning services for the IRMOS Provider responsible for offering applications. End users are able to access IRMOS platform in two ways:

- A Customer negotiates A-SLAs and monitors the application lifecycle through the Framework Services.
- A Consumer accesses an application deployed in a VSN through Client Component(s) without intervention of any Framework Services.

Each VSN is instantiated by ISONI based both on the particular technical requirements defined by the Application Developer at design time and the specific QoS customisation defined by the Customer at runtime. These requirements are relayed to ISONI during the SLA negotiation through the Framework Services and upon a successful negotiation, the virtual environment for the deployment and execution of the application defined and initialized. It should be noted, that ISONI infrastructure cannot be accessed directly by end users (Customers or Consumers) and their requests are analyzed and relayed to ISONI by the framework services. However, end users are allowed to access the applications services running in the virtual environment through specific gateways predefined in the VSN description.

The Framework Services communicate continuously and in various ways with ISONI providers. Each ISONI advertises its capabilities to Framework Services so as to be discovered later and, in a second step, negotiate the Technical SLA for an application. Additionally ISONI provides monitoring data and notifications in cases of failures for each VSN to Framework Services that are used for both runtime (control) and design time (development and modelling). The fact that real-time functionality is required on some components of the Framework Services layer as well as direct control of the ASCs demands that instances of these components will be deployed and run in the VSN where the real-time QoS and efficient interactivity between the components is guaranteed. As presented in Figure 17, the Framework Services interact with the instances running in VSNs for controlling and monitoring the application during the execution phase. The Framework Services, their subsystems and the interfaces are described in the next section (6.2) of this document.

The Execution Environment and the Intelligent Networking subsystems are architecturally close and are expected to communicate continuously during all the IRMOS processes. These subsystems are wrapped in the ISONI infrastructure. The main objective of ISONI is to virtualize resources, provisioning of application services and resource monitoring without any knowledge for the application itself. The EE subsystem, considered as an enhanced virtualization platform, includes the storage systems that are implemented so as to address the QoS and especially the real-time requirements of the application service components implementing various high availability solutions such as redundancy and live migration. The network resources, provided as a VPN like approach, are classified and advertised to the Framework Services in QoS classes. ISONI has been initially described in D3.1.1 [5] and in the ISONI Whitepaper [14] while in section 6.3 of this document we present more details for the ISONI itself and how it interacts with the other subsystems of the IRMOS platform.
6.2. IRMOS Framework Services

6.2.1. Overview

Cloud computing is such a generalized paradigm it is impossible to consider ‘the cloud’ as a single set of business models with a single set of Quality of Service issues. To some extent, issues with cloud computing are necessarily related to the application characteristics and purpose(s). However, it is possible to classify cloud modalities, and identify common stakeholders and concerns in each classification. Today, there are three main classes in the cloud services stack which are generally agreed upon:

- **Infrastructure as a service (IaaS):** the provision of ‘raw’ machines (servers, storage, networking and other devices) on which the service consumers install their own software (usually as virtual machine images).
- **Platform as a service (PaaS):** the provision of a development platform and environment providing services and storage, hosted in the cloud.
- **Software as a service (SaaS):** the provision of a pre-defined application as a service over the Internet or distributed environment.

A major challenge for SaaS providers wanting to exploit the benefits of cloud computing is to manage QoS commitments to customers throughout the lifecycle of a service. The complexity of this problem has driven the emergence a new PaaS offers that aim to abstract this complexity through targeted tools and services. PaaS aims to be a developer’s friend – the “platform” is a development platform in the large. The idea is simple, even if the execution is complex: multiple applications share a single development platform and common services, including authentication, authorization, and billing. PaaS developers build web applications without installing any tools on their computer and deploy those applications without needing need to know or care about the complexity of buying and managing the underlying hardware and software layers. A PaaS is built on IaaS and uses both a multi-tenanted development tools and deployment architecture.

A PaaS architecture supporting real time interaction between distributed set of people and resources requires the following key features:

- **Real-Time QoS Specification:** specification language and associated toolkit for the specification of IRMOS applications and application service components considering both structure and real-time QoS.
- **Event Prediction:** QoS oriented service engineering methodology and models for predicting QoS requirements contingent on application and resourcing events considering temporal profiles of application service components deployed on virtualised infrastructures.
- **Dynamic SLA Negotiation:** SLA negotiation and management services supporting the dynamic negotiation (re-negotiation) of Application-SLAs considering customer requirements and dynamic discovery of resource providers (Technical-SLAs).
- **On-Demand Resource Provisioning:** provisioning services for application service components on virtualised infrastructures through combination of workflow and service-based management wrappers enhanced to support temporal profiles.
- QoS Event Monitoring: monitoring services for measuring Quality of Service at both application and technical levels targeting trigger events for runtime adaptability of resource provisioning estimation and decision making.

The IRMOS Framework Services (FS) is the layer between IRMOS applications and virtualized resources offered by ISONI. This layer corresponds to the PaaS layer in the Cloud computing terminology. The PaaS architecture is shown in Figure 18 highlighting the core FS components which either run independently (PaaS layer) or in the VSN of each application (IaaS layer). The architecture consists of two main elements Service Engineering and Service Management which are described in more detailed in the subsequent sections.

The IRMOS FS layer aims to provision and manage the execution of real-time services on request of the Application Layer inside the ISONI Environment, while conforming to the real-time constraints as determined in their A-SLA. Apart from the execution of the services that are offered to customers, Framework Services support service engineering, fully automated SLA negotiation and re-negotiation, mapping high level performance parameters to low level, discovery and reservation of the ISONI resources needed for the execution of an application. In the execution phase of the application, FS monitor continuously and manage the application components and the resources either directly through the application wrappers based on predefined application specific policies or...
relaying the management requests to ISONI layer based on operational policies of the platform.

For the communication with the user of the platform the *IRMOS Portal* component has been implemented, which provides the necessary interface to enable the final user of the application in order to request the A-SLAs templates, invoke the negotiation process as well as the reservation of ISONI resources. In addition, its functionality includes starting and stopping of an application execution. IRMOS Portal provides all necessary interfaces and functionality in order the WP4 Client to initiate the A-SLA negotiation, resource reservation process and trigger the workflow execution. Through the GUI, the user can easily complete the appropriate parameters during the negotiation and the execution phases. On the other hand, the IRMOS Portal, implements the operations that the IRMOS Portal GUI invokes and thus provides access to the rest WP5 Framework Services such as for the renegotiation.

### 6.2.2. Service Engineering

#### 6.2.2.1. Overview

Quality of Service oriented Service Engineering (QoSSE) supports two important features for real time systems: Real-Time Specification for Applications and Event Prediction. The activities in service engineering are shown in Figure 18. As previously stated, Real-Time Specification is related in how an application is represented to the PaaS architecture and Event Prediction is associated with the procedures and mechanisms needed to model an application and determine the appropriate infrastructure requirements.

The QoSSE interact with application’s developers and with the application itself through a Development Interface. It would be very difficult to monotonically describe applications; instead applications are broken into application components (AC) developed by an application component developer. These components are referred as Application Service Component (ASC). In order to use ASC, these need to be described and registered (at the Application QoS/QoE repository in Figure 2). QoE (Quality of Experience) allows a service provider to make observations that may differ from the QoS guarantees owing to factors outside of the service provider’s control. The consumer may use QoE measurements to validate the QoS reported to it by the service provider, but must recognise that any discrepancy may be due to factors outside of the terms of the SLA (e.g. local network latency). The application service component description (ASCD) comprises the definition of the input and output interfaces of an ASC as well as the required computing and network resources (which may be depending on the input and output formats actually used as well as timing constraints, e.g. ‘real-time’). ASCD are based in QoS specifications modelled using UML.
The QoSSE interfaces with Service Management through a Performance Estimation Interface that is used to calculate resource provisioning policies from customer requirements and application constrains (i.e. image resolution, required video streaming parameters, maximum completion time, etc.) in the form of SLA. To calculate the appropriate resources such as processing, memory and network required by the SLA requested, the QoSSE uses a variety of performance evaluation mechanisms such as parameter mapping and statistical modelling tools.

The purpose of the parameter mapping process is to produce mapping rules that can be used to translate the high-level parameters described in the ASCD to QoS requirements. To produce this mapping, benchmarking techniques and Artificial Neural Networks (ANNs) are used. The purpose of benchmarking is to gather a set of data that will come from the test executions of an ASC with different high level parameters on different platforms, characterized by a benchmark index. This index is used to train an ANN that generates mapping rules and provides algebraic functions that allow low level resource parameters (such as processing - in whatever metric chosen, CPU cycles or other - storage or network traffic produced) to be calculated from high level ASC terms (such as fps, resolution, etc) without actual knowledge of the internal source code of the ASC. These functions provide basic knowledge about the behaviour of the ASC and can then be used by subsequent models to calculate completion time probabilities.

The resource requirements generated by parameter mapping do not consider the user’s interaction with the application. It would be very difficult to accurately create a model representing the internal application behaviour. However, we can use the knowledge about externally observed behaviour of the application, the behaviour of the execution environment and the pattern of user interactions to build a statistical high level performance model incorporating the behavioural aspects found in real time and interactive systems. This is done by using Finite State Machine modelling techniques, specifically we use discrete time stochastic finite state automata and the PRISM model checking tool [17]. PRISM accepts specifications in probabilistic temporal logic that allows us to express probabilistic properties answering questions such as: “The
probability to finish task X in Y time”, “the probability of a system interruption after Z minutes”, etc. The result of these properties is further use in the calculation of the ASC requirements.

6.2.2.2. Architecture

An architecture for QoSSE has been defined and is shown in Figure 20. The architecture has four interfaces each targeted at different actors.

- The Business Assessment interface provides strategic decision support and allows an Application Provider to define and assess how well they are doing in accordance with their business goals. The assessment of business performance metrics such as profitability, customer satisfaction are derived from operational service level key performance indicators.
- The Development service interfaces with the Application development tools. The created Application models are pulled from the Application Description (AD) and Application Service Component (ASC) repository to be used by the performance estimation engine.
- The PerformanceEstimation interface is used by SLA negotiation and evaluation services to get resource specification decisions. The PES service is called whenever a client triggers A-SLA negotiation process. The estimation result feeds into the T-SLA to be negotiated with the ISONI provider.
- The PerformanceFeedback interface reports actual application and resource behaviour that are used to check whether the required QoS is actually supplied by the ISONI provider. This data may be used to validate the accuracy of models used by the engine by comparing the predictions with actual measurements.
Note that the scope of IRMOS focuses on the technical issues associated with guaranteeing QoS for real-time interactive applications. WP5 recognises that all such guarantees need to be understood in the context of business objectives and governance. However to limit the scope of modelling to that consummate with the resources available, WP5 does not focus on the higher business assessment interface and uses a
simple cost minimisation function to drive the overall optimisation. The following sequence diagram shows the interactions between the PES and the different services and components.

![Performance estimation sequence diagram](image)

Figure 21: Performance estimation sequence diagram
The request document supplied by the SLA negotiator as an input for this method includes multiple sections:

1. Application Specification: a specification of the requested application e.g. name, id as well as components and topology information. This will feed into the performance estimation as well as the VSND generation.

2. ASP Identification: identifiers of the ASP offering the application (e.g. DN, name, id). This allows the PES to choose which data/objectives to consider for this Customer request.

3. Customer identification: identifiers of the client asking for the A-SLA. This will allow the PES service to pull any history information needed for understanding the customer behaviour.

4. Interactions (seen as customer obligations): specification of the interactions with the application that this customer is allowed to do (e.g. can stop/pause the application for 2 times/day for duration of 3 mins). This allows the optimisation engine to estimate the application completion time.

5. Workload features (seen as customer obligations): these are the characteristics of the workload that the customer is allowed to submit under this A-SLA. E.g. average video length 15 mins, frameRate 30 fps)

6. Customer Requirements: this includes the customer requirements. Mainly this will include completion time and cost limits.

7. Modelling requirements: this specifies the modelling time and modelling accuracy needed. The more time the optimisation engine is allowed, the more accurate the resource specifications are. The SLA negotiator is offered the possibility of specifying the accuracy via this input.

8. ISONI Providers Info: this contains information about the ISONI providers and their pricing terms. The optimisation engine will then take this information into consideration while looking for the optimal cheapest resources.

6.2.2.3. Service Engineering Tools

The Service Engineering Tools subsystem is responsible for the adaptation of multimedia applications to the IRMOS Platform and “translation” of application and end user requirements to a specific set of technical parameters. For these purposes, several state of the art techniques have been applied in the areas of modelling and performance estimation. The Service Engineering Tools communicate (both offline - during the application adaptation- and during the runtime) with the Applications Layer and the SLA and Workflow Management subsystems.

Papyrus UML Design Tool

This tool is intended to be the main interface between the application developer and the IRMOS platform. Through this, the application developer will be able to construct a workflow from the individual available ASCs or even provide his own ASC along with all necessary parts, such as executables, A-SLA templates, ASCDs etc. In these ASCDs, the parameters of the ASC will be included (inputs, outputs etc.) as well as the parameters regarding configuration, workload features, QoS metrics.
Mapping Service
This service is responsible for mapping ASC high-level workload characteristics / parameters and low-level hardware configuration parameters with the QoS metrics used by the specific ASC. In more detail, the Mapping Service will create the models/rules for each individual ASC from the values provided from the benchmarking process and the historical data. Through the creation, training and optimization of Artificial Neural Networks (ANNs), the function that describes the relationship between workload features that are part of the A-SLA and resources needed for each ASC in order to meet the desired QoS will be provided, plus a PDF giving the probability of the error of the estimation. This information is used by the PES in order to optimize the workflow and conclude on the number of resources needed.

Figure 22: Mapping Service Overview and Structure

Application Performance Models
Applications offered by an application provider can optionally provide a performance model that is used by the performance estimation service to estimate resources necessary to meet SLA commitments. A set of performance models have been developed for IRMOS applications that are configured and deployed within the Framework services

Service Performance Data Management
This component is responsible for storing performance configuration data, performance estimations and actual performance data collected from application and infrastructure monitoring. The component is implemented based on the Hibernate object-relationship
data management system and is used during performance estimation and performance feedback use cases.

**Service Performance Management Portal**
This portal provides an application provider with a user interface to configure application performance models, to view estimates generated by these performance models and to compare estimated with actual performance data. The portal is implemented using the Google Web Toolkit.

### 6.2.3. Service Management

Following the outcomes of the QoSSE tools, IRMOS service management proposes the so-called “Online Process” (as depicted in Figure 4) that includes Application Concretion (during which the application template is populated with concrete QoS parameter values), Discovery & Negotiation (referring to IaaS providers), Reservation (the IaaS resources are reserved from the PaaS provider), Service Instantiation (referring to the setup of the Virtual Service Network of the IaaS and the Virtual Machine Units that contain the ASCs), SC Configuration (during which the instantiated ASCs are configured according to the application’s configuration), Execution & Monitoring (referring to the execution of the ASCs and monitoring of them as discussed previously in this paper) and Cleanup (where the ASCs are stopped and the Virtual Service Network of the IaaS is torn down). Figure 23 depicts the main components involved in the Service Management, which are described below.

![Figure 23: Service Management](image)

The service management components are responsible for on-demand SLA negotiation, resource reservation, service instantiation, execution and monitoring of Quality of Service for application provisioned at an IaaS provider. In IRMOS the IaaS is provided by ISONI (which is described in more detail in [11][12][14]). The Service Management
components are shown in Figure 18: (i) SLA Management, (ii) Orchestrator, (iii) Event Monitoring and Provisioning Rules and (iv) Application Wrapper. Each component is now described in more detail in the following sections.

### 6.2.3.1. SLA Management

The SLA Management subsystem carries out all the SLA related processes of IRMOS. It is responsible for the dynamic and automated negotiation of SLA in both the application and ISONI layer which are considered of major importance for the successful and reliable execution of an application. It also implements discovery mechanisms for discovering the ISONI Providers that will negotiate with the Technical SLAs and the Performance Estimation Service in order to translate the high level application requirements into low level parameters that can be understood by the ISONI.

**Information Service & Index Service**

The Information Service is used by ISONI Providers to advertise their capabilities towards the Framework Services. Therefore, the Framework Services are unaware of the existence of an ISONI unless there is an advertisement about it. The capabilities of each ISONI Provider are stored into the Index service that is thereafter used in the discovery process. These advertisements will contain information about the types of resources and capabilities offered by the corresponding ISONI plus pricing information regarding the infrastructures provided by each ISONI provider.

**Discovery**

This service is responsible for finding registered candidate ISONIs, which meet the low level QoS constraints defined in the Technical SLA. It is designed so as to apply rules based on the advertised capabilities of the ISONIs and the QoS types that are supported.

**Performance Estimation Service**

This service is the service-oriented version of the application modelling and services planning tool of Tasks 5.1 (Real time application modelling) and 5.2 (Planning service networks). PES is responsible for collecting the abstract workflow description from the design tool, optimize it and transform the high level application parameters to high level ASC parameters. To this end, it enquires the results of the Mapping Service for the ASC models in order to come to a decision about the low level resources needed to meet the QoS. This part is also responsible for creating the VSN descriptions that will later on be used by the SLA Management section during the T-SLA negotiations.

**SLA Negotiator**

This service is a central component in the negotiator process. This component orchestrates all the components involved in the SLA (re)negotiation process. The SLA negotiation always starts from the Portal. The main interactions of the SLA Negotiator are with:

- A-SLA Manager to manipulate and store A-SLAs and T-SLAs.
- Discovery to obtain the list of ISONI providers and their capabilities and price lists.
- PES to perform the estimation of the infrastructure resources.
- Workflow to communicate the configuration parameters to use during the execution of the application.

On the other side, the re-negotiation of A-SLA is always triggered by the Evaluator. During the re-negotiation process the PES is also involved to refine their estimations (sometimes based on new high level parameters of the applications that affect the QoS of the application but also to refine bad initial estimations that could have caused SLA violations).

The main interface of the A-SLA Negotiator with ISONI is through the T-SLA Manager. The T-SLA is established through this relation.

**SLA Manager**

This service queries, publishes, creates and updates SLA and SLA templates into the A-SLA & T-SLA repositories. The main interface is the SLA Negotiator, but it is also used by the IRMOS Portal to retrieve the A-SLA Templates at the beginning of the negotiation with the customer.

**A-SLA Template Repository**

In this repository the A-SLA templates are stored. These are created directly by the Application Provider and are used by the customer that fills in them specific values before the A-SLA negotiations. A-SLA Templates are automatically generated using the ASC and Application models generated by the developers.

**A-SLA & T-SLA Repositories**

These repositories store the Application and Technical SLAs as well as the Application SLA Templates generated by the Service Engineering Tools. These repositories should be accessed by all legal parties that the SLAs are signed by.

6.2.3.2. **Workflow Management**

The Workflow Management subsystem is responsible for the execution and control of the application on the Framework Services layer. At this point it should be noted that instances of real-time critical Framework Services such as the Workflow Enactor (as well as the Monitoring Service) are part of each VSN and are dedicated to supporting the execution of this specific VSN, i.e. each VSN has each own dedicated instance of the Workflow Enactor that reside inside the ISONI domain and are controlled by a higher-level service that resides in the IRMOS Provider domain.

**Workflow Manager**

This service is in charge of the orchestration of service components that comprise the application workflows on the reserved ISONI resources. The Workflow Manager Service follows a hierarchical structure that includes a central manager and multiple instances of it named Workflow Enactors. The central manager will reside in the IRMOS Provider domain and will be responsible for controlling the workflow executions on ISONI level whereas its instances will be located inside each VSN and will be responsible for orchestrating the workflow on VSN level. The Workflow Enactor instances will be able configure as well as to start and stop the execution of the ASCs deployed in the VSN.


6.2.3.3. Monitoring and Evaluation

The Monitoring and Evaluation subsystem of IRMOS is responsible for gathering information for the resources usage, the status and performance of the application components in order to assure the system health and the necessary QoS levels for the application. This is achieved by aggregating the various monitoring data and evaluating them against the agreed SLA parameters based on predefined policies. The policies define the methods that the data are evaluated as well as the corrective actions that are triggered for recovering from any abnormal situations. The Monitoring and Evaluation subsystems include the following components:

**Monitoring Framework Service (MFS)** is part of the platform's framework. It exposes appropriate interfaces for starting/stopping the operation and has access to the collected data from both the infrastructure level and the application level. It is the major component of the mechanism while it orchestrates the monitoring of all applications towards the VSN and has access to the aggregated information through the Monitoring Central Index. Also, it can serve different and concurrent applications that are being deployed in separated VSN.

**Monitoring Service Instance (MSI)** is located within a VSN and is specific to every application workflow deployment. It exposes an interface towards the MFS in order to initiate the application monitoring. During this action, the latter provides the configuration parameters for the application workflow monitoring, such as the private IP addresses of the VSN, the time granularity of each ASC monitoring script etc.

**ASC Monitoring** script is deployed in the same VMU with each application component in order to collect data from the application execution and publish them into the local Monitoring Index. The implementation of this component is part of the application adaptation to IRMOS infrastructure and is performed by the Application Developer. The ASC Monitoring script is automatically executed with a time interval that is defined by the MSI during the registration process. The outcome of this execution is a set of parameters (name, value, unit etc) formatted in plain XML and fed into the local Monitoring Index.

**Monitoring Index Service** serves the role of a local repository of the monitored parameters for each application component. It is also deployed in the same VMU with an application component in order for the respective ASC Monitoring script to provide the necessary high level information. While it keeps the application monitoring parameters, it publishes them at the same time to the Central Monitoring Index, outside the VSN. The values within the Monitoring Index Service are refreshed every time the ASC Monitoring script provides a new set of values.

**Monitoring Central Index** is the global repository of the platform. The high level monitored parameters of all application components as well as the low level parameters deriving from the infrastructure are published here. The values of the parameters listed in this repository are refreshed with a different time interval based on the report...
granularity of each individual application component.

**Evaluator Framework Service** lies on the platform level and is a part of the IRMOS Framework services. The role of the component is to compare the monitoring information with the respective A-SLA in order to trigger some actions if necessary (e.g. re-negotiation of an A-SLA) as well as receive notifications from ISONI in case of a hardware failure or a T-SLA violation. As a result, the Evaluator is responsible for detecting any A-SLA violation during execution and also receives the notification when a T-SLA violation is detected by ISONI. In general, the operation of this component is based on the reaction to various *Events* and by applying specific *Rules*, defined by the Application Developer, relevant *Actions* will be realized.

**Binaries Repository**

In this repository the executables of the ASCs that contain all necessary parts (libraries etc.) are stored. This repository is accessed by ISONI during the VSN initialization phase so as to get the ASCs and deploy them on the virtual machines.

## 6.3. Intelligent Service Oriented Network Infrastructure (ISONI)

### 6.3.1. Introduction and relation to WP6 and WP7

The ISONI (Intelligent Service Oriented Network Infrastructure) is an infrastructure, consisting of a network of resources (hardware like CPU, storage,..., and software) managed and controlled by an ISONI middleware, that allows resource sharing among multiple services. The general idea is to provide a SOI (Service Oriented Infrastructure) for SOA components and services. A service is usually composed out of several smaller and simpler services, in the following called Service Components (SC). ISONI is agnostic to services, thus the decomposition of services into SC is not its responsibility, and this is done by the Framework Services layer, in IRMOS developed by WP5. ISONI will provide these SCs the best resources (Execution Environments and network links).

The work related to the Execution Environment and its related resources like CPU and Storage will be carried out in WP6 while all issues regarding the Intelligent Networking layer will be handled by WP7, (see chapter 6.3.2.2 for more details).

### 6.3.2. General description and major building blocks

The basic purpose of the ISONI is to considerably reduce the complexity for service providers and developers to roll out new network based services as it takes care of the automatic deployment of the services on best fitting resources distributed in a network. The solution strives to reduce global costs introducing a resource provider in the value chain that optimizes costs by means of virtualization techniques, whereby tailored resources can be provided for the deployment of services. Additionally, the ISONI will provide means to isolate different deployed services from each other in order to prevent
unwanted crosstalk between them. Future expansion stages shall allow necessary resource re-arrangements and (re-)allocations for running services that adapt better to changing resource availability. To live up to that purpose the ISONI has to carry out several tasks.

The first major task of the ISONI is to completely separate the management of all kind of hardware resources distributed in a network from that of deployed services and their associated service components. By that, the actual status and distribution of resources are hidden from the service developer’s view. The infrastructure provides him with fully virtualized resources, including the network resource. That enables a service developer to deal with a complicated network of resources in a simplified way at a level of high abstraction. This full virtualization of a network of distributed resources is the essential prerequisite to get the freedom of resource and service management we need to serve the purpose of the ISONI as described above.

The second major task of the ISONI middleware is to deploy and instantiate the service developers’ service on the ISONI. The ISONI will be able to accomplish this task automatically and autonomously, which is the main goal of the ISONI development. For that the ISONI needs an abstract description of all the requirements of the service on the execution environment including the description of the interconnections and their individual QoS demands. The level of abstraction of this description should be as high as possible to ease its creation, while still allowing for automatic matchmaking. In particular, the creation of the description shall not require special knowledge about the network infrastructure. This description has to be delivered by the service developer in form of a Virtual Service Network (VSN) description, which has been presented in more detail in [5].

The third task of the ISONI will be the monitoring of running services concerning resource usage. Those monitoring data will be available to the service developer via the IRMOS Portal, e.g., web service interfaces. It is part of the support functions the ISONI will offer in the future during service lifetime.

6.3.2.1. The VSN Concept (Virtual Service Network)

ISONI exposes its virtualized infrastructure in the form of VSNs (Virtual Service Network). In IRMOS the Framework Services will state their infrastructure requirements using a VSN description. It is the role of the Framework Services—which is application aware—to decompose its Services and Applications into Services Components (SCs) to build the VSN description to request resources to ISONI which is application unaware.

All the descriptions and parameters are merged in a so-called Virtual Service Network (VSN) description (this process is detailed in [14]). The VSN can be seen as a graph whose vertexes are the SCs and whose edges are the Virtual Links, see Figure 24. The VSN description is transferred to the ISONI with the request to instantiate the service. Then, the ISONI has to automatically and autonomously map the highly abstracted resource request in form of the VSN description onto the network of real resources, to deploy the components in tailored execution environments on suitable resources, and to
interlink them while observing QoS requirements. This instantiated VSN builds an independent layer 3 overlay network, i.e., there is no limitation on the L3 protocol stack used by the SCs.

![VSN Diagram](image)

**Figure 24: VSN Description**

Framework Services will provide a VSN description to ISONI and ISONI will map this VSN into concrete resources (computational units and network links). Figure 25 shows an example of this concept. This process involves systems developed by different IRMOS workpackages and, thus, it’s a major process in the overall IRMOS architecture.
Deployment of a VSN requires resources on the infrastructure. Therefore the deployment process as depicted in Figure 25 is embedded in the SLA negotiation as shown in chapter 5.2.4. The complete deployment process starts with the creation of an application beforehand by developers, which will be delivered by application provider to the IRMOS platform. The IRMOS FS adds some framework functions for application control and monitoring and maps it on low level resource requirements expressed in a specification of VSN Description, which is delivered to ISONI during SLA negotiation process. ISONI generates virtual machine units (VMU) using the VSND information and deploys the VMUs on adequate ISONI Physical Hosts (PH). The general principles are described in the ISONI white-paper [14].

6.3.2.2. ISONI Management

The ISONI management architecture is composed of functional blocks, where each takes on a different task for the management of the ISONI resources and deployed VSNs.
Figure 26 shows the functional building blocks of the ISONI management. The interfaces to the IRMOS Framework Services and Tools are indicated on top. The functional blocks Resource Manager, Storage Manager and Path Manager would be deployed in a two-level management architecture based on the composite structure, i.e. Domain level and Node level. The resource responsibility lies with the Node level, i.e. the Node control and resource reservations are maintained by the middleware functional blocks running at Node level, whereas the Domain-level instances coordinate the ISONI Nodes. As described in chapter 3.1 of the ISONI Whitepaper [14], the two-level structure has been developed for scalability reasons.

**ISONI Info System**

The ISONI info system provides information about general supported capabilities of the ISONI domain, OWL templates for VSND and Price information. The ISONI capabilities contain information such as:

- Types of resources offered: Examples of resource types are CPU types (Intel, AMD, a concrete architecture or even generic), RAM, storage and so forth. In addition to hardware resources also software resources such as supported OSs are exposed.

- Supported QoS types: Some of the resources will be used in conjunction with certain QoS constrains. Information of what QoS constrains are supported for what resources will be included.

The reported ISONI capabilities assist the IRMOS SLA management in the ISONI Provider selection process. The ISONI capability information does not contain any report of the current available resources.
ISONI SLA Manager
The SLA Manager is responsible to negotiate and re-negotiate technical SLA with the IRMOS Framework Services’ technical SLA Management. If the technical SLA has been checked successfully, the SLA Manager launches the preparation and deployment by instantiating a VSN specific Deployment Manager instance, if not already existing. The SLA Manager also creates a topic for monitoring information with the ISONI Monitoring Service, to which the Deployment Manager will send monitoring information. The SLA Manager subscribes itself for notifications for the created topic as well, in order to be able to peruse the monitoring reports and compute if SLA violation events are occurring. SLA violation events are reported to the Framework Services via the notification interface as well as other information about passed and ongoing outages and recovered VMUs, which need to be reconfigured by IRMOS Framework Services.

Deployment Manager (per VSN deployment)
The Deployment Manager has the overall control to trigger resource selection and reservation, Execution Environment preparation and finally, to schedule deployment. Towards the IRMOS Framework Services, it provides execution monitoring and control.

Upon instantiation, the Deployment Manager triggers resource selection based on aggregated information on the Domain level. With the pre-selected ISONI Nodes at hand, it requests resource reservation while permuting through preferred deployment options on the selected Nodes. The control provided to the IRMOS Framework Services would cover e.g. pause/continuation and abortion of the VSN execution and monitoring controls.

Resource Manager
The Resource Manager occurs on both, domain and node level of the two-levelled ISONI management. On Domain level, the Resource Manager collects aggregated resource availability information. When resource selection is requested, it matches technical SLA requirements of the VSN description against the collected availability information and proposes ISONI Nodes, which are able to satisfy the requirements. On Node level, the Resource Manager manages the actual resources of Physical Hosts in the ISONI Node. Upon reservation request, it schedules resources reservation and deployment of the VSN sub-part that it has been assigned.

EE/VMU factory
The EE/VMU factory tailors the Execution Environment for the VSN, i.e. the factory creates the ISONI Service Components following the specification of the VSN Description and ISONI SC descriptions. The ISONI Service Components are then provided to the Nodes, ready to be deployed.

Path Manager
Like the Resource Manager, the Path Manager occurs on both, the domain and the node level of the ISONI. On domain level, the Path Manager matches network resource requirements of the VSN description against aggregated network availability information and network paths, which are able to satisfy the required network
resources. On node level, the Path Manager reserves the requested network resources according to the VSN requirements.

**Inter-Domain Manager**

The Inter-Domain Manager is responsible to manage the additional tasks, which are needed to enable resource collaboration directly between ISONI Domains. In case of resource shortage or due to the need of special capabilities, an ISONI Domain may contemplate resource collaboration with another ISONI Domain. The Inter-Domain Manager is further described in [11].

**ISONI Registrar**

Any communication over the ISONI Portal and Inter-Domain Manager needs to be authenticated and authorized for any action. Therefore the Registrar keeps the information to identify external actors an ISONI system. And it keeps the related credentials like password, one-time tokens, and certificates. In respect to authorization it keeps the different levels of privileges and/or restrictions. The scenario regarding Inter-Domain Managers is described in [11].

**Storage Manager**

Like the Resource Manager, the Storage Manager exists at both domain and node levels. The domain level Storage Manager keeps track of available storage resources on all ISONI nodes and determines which ISONI node can accommodate a given storage request. The node level Storage Manager is responsible for maintaining the Lustre based global file system and managing all available storage resources on that node. It also manages the storage QoS and interacts with the node level Resource Manager during negotiation for storage resources.

**Object Storage Servers**

The data is stored on Object Storage Servers (OSS); these are connected to the Physical Hosts in the node and also a node-level Storage Manager by a dedicated storage network. Each OSS has a Storage QoS Target (SQT) which manages the storage QoS assignments it has been given from the Storage Manager.

**Repository Manager**

The Repository Manager is an interface to Framework services that allows the uploading of binary data for use when creating a VMU. The role of this interface is to manage the references to this data. The Repository Manager is also used by the VMU Factory to access this data with a clear time prediction when a VMU is being created and instantiated. The Repository Manager provides this prediction capability by allocating a guaranteed bandwidth based on the negotiated SLA for the duration of VMU image download.

### 6.4. Platform Interactions

In order to achieve the real-time functionality for the applications deployed and executed in the platform the interactions between the various subsystems, especially between the cross-layer ones, are carefully designed and developed. To highlight the
main interactions we analyse in the following sections the key processes taking place in the platform, the Offline in which the application components and models are developed, the SLA Negotiation and Renegotiation, and the Monitoring and Evaluation. Additional information about these processes can be found in the respective reports of the technical workpackages.

### 6.4.1. Offline

During the offline process the application components are adapted in the IRMOS environment and the required documents that describe each application components and the application as a whole are produced. The steps that are required in order to prepare an application are the following:

- **Step 1 - Application Components Creation:** The application has to be split into application components (ACs) at least separating the user interface part (ACC) from the computing part(s) (ASC(s)). How the application is separated depends on the specific application logic and what should be achieved. While it may be convenient for one application to be split into a large number of modules as the module specific tasks have a high degree of reusability (think of the topic of componentized development and service-oriented applications), it might be different for another application that has specific subtasks which have to be componentized and put on, e.g. highly demanding computational resources. The conclusion is that “componentizing” your application is the way to unlock the power of the IRMOS platform by making use of scalable QoS-guaranteed resources in an “on-demand”-fashion.

- **Step 2 - Components Interfaces Creation:** Components must be equipped with interfaces to the IRMOS framework:
  - ASCs need a run-time interface (wrapper, see 5.1.1) to be configured, controlled and monitored by the framework. The purpose of this interface is to act as a mediator between the AC and the IRMOS framework services, represented on the application component’s node by an IRMOS framework service that gathers the aforementioned information. However the amount of information might vary according to the information made available by the AC.
  - ACC need to be started with specific parameters required for accessing the application service running on IRMOS. The configuration information provides the necessary details to allow the ACC to access the various components (ASCs) on the IRMOS platform that need to have direct connections to an ACC.

- **Step 3 - Components Packaging:** ASCs must be packaged for deployment.

- **Step 4 - Components Description:** ASCs must be described, especially resource-wise (see 5.2.1).

- **Step 5 - Application Description:** The application (as a whole) must be described, which includes, models with information for mapping the high level QoS requirements to low level resource parameters, SLA templates and abstract workflows presenting the inter-application processes.
6.4.2. SLA Negotiation

The SLA Negotiation is considered as one of the most important processes in the platform and a key one for the operation of an application on the required QoS level. In order the SLA negotiation to take place, the publication of ISONI capabilities from the ISONI providers to the Framework Services is required. Based on this information the Framework Services are able to select the appropriate ISONI providers for the particular application and trigger and SLA negotiation with them. Additionally the Framework Services rely on VSN descriptions (VSND) as specification for the resources needed from ISONI.

Figure 28 depicts the entire SLA process covering A-SLA and T-SLA negotiation according to Figure 11. The T-SLA part (Figure 28: flow 11, 12, 17, 18) among IRMOS provider and ISONI provider is described in detail in D7.2.1.
6.4.2.1. **SLA Renegotiation**

The SLA re-negotiation process takes place when there is an A-SLA and T-SLA already in place, during the execution of the application, and it could be caused by:

1. A client asking for more resources.
2. The application scalability policies defined in the A-SLA.
3. The detection of an A-SLA violation.

In case of T-SLA violation, the infrastructure is not able to provide the required resources for the correct application execution. In this situation the only possibility is to stop the application, pay the penalties and let the customer negotiate the execution of a new application from the beginning.
In the next sections the three SLA re-negotiation scenarios are described in more detail.

**SLA renegotiation triggered by the client**

The client, during the execution of the application, could be interested in changing the parameters of the application that affect the QoS, for instance to accept more users or change the frame-per-seconds speed of the application. The following Figure 29 shows the main interactions among the value chain players.

![Figure 29: SLA renegotiation triggered by the client](image)

The customer first retrieves the active Application SLA to update the terms with new values (e.g., new image resolution), this new request is sent to the IRMOS provider that estimates the infrastructure resource required to support the new application’s QoS. The new T-SLA is negotiated with the ISONI Provider that can accept the T-SLA terms or reject it (for instance, in case of lack of resources). In any case the response is communicated to the Customer.

**SLA renegotiation triggered by application policies**

The customer can define certain scalability policies in the A-SLA that allows the application to automatically scale in terms of infrastructure resources when the context of the application changes. The Key Performance Indicator (KPI) used to trigger this process could be, for instance, number of user connected or an incremental threshold on certain monitoring values. When this threshold is reached the platform starts a re-negotiation to allow the new increment in the application configuration (for instance, to allow 15 more connected users to the application). Figure 30 depicts the process.
Inside the IRMOS Platform the Evaluator detects the need to scale the application based on the rule defined by the customer in the A-SLA and the monitoring information received from the application (1). This starts a new SLA renegotiation with new A-SLA terms. The new infrastructure requirements are estimated inside the Framework Services by the Performance Estimation Service (2). These new values are then renegotiated with the infrastructure provider (ISONI Provider) under a new T-SLA (3). At the end, the customer is informed about the re-negotiation result (4-6). If the negotiation of the new T-SLA with the ISONI Provider succeeds, the application is reconfigured with the new parameters (for instance new limit for maximum number of users connected) (7).

SLA re-negotiation triggered by A-SLA violation

The application continuously sends monitoring information relevant to the application to the IRMOS Framework Services. The Evaluator, based on the information received, can detect possible A-SLA violations, under this situation it tries to re-estimate the infrastructure resources to avoid the problem and negotiate these new VSND with the ISONI Provider. The result of this negotiation is forwarded to the customer. In case the A-SLA cannot be kept, the application is terminated and the IRMOS Provider should face the penalties for this situation.
6.4.3. Estimation Correction

Estimation correction processes are required to ensure that the accuracy of performance models is tracked and mitigating actions taken when accuracy drops to unacceptable levels. The responsibilities for estimation correction are the responsibility of the service engineering tools based on failure events identified by the evaluator component. Estimation correction is especially challenging for real-time interactive applications and must consider:

- how long the infrastructure has to react to an application failure (actual or predicted) before a SLA breach will occur; and
- how long it takes to understand and react to the monitoring information

The required reaction time to (possible) failures is the important design constraint especially that IRMOS considers real-time applications where failure to react in appropriate times will result in drops in Quality of Service. The challenge is that the time necessary to monitor performance, identify problems, assess possible mitigating actions and consequences will take much longer than the time available. For example, if this process requires an update to an application performance model used to generate a resource provisioning estimate this time could take days in addition to the time necessary to re-run the model which in itself could take many minutes. Therefore changes to application performance models are considered an out of bound human assisted process. Future developments may investigate learning algorithms but this is not going to be implemented in the lifetime of the IRMOS project.

The estimation correction implementation makes the following assumptions:

- The reaction time for performance estimation is greater than the maximum reaction time necessary to preserve QoS in an SLA given the temporal constraints of the IRMOS applications. This holds even if we try to predict a failure by setting a threshold notification from the evaluator significantly below the actual QoS constraint defined in the SLA.
The assessment of performance occurs within the Service Engineering tools at two levels, using predefined automated renegotiation rules and through the service performance feedback manager. Fast (short term) reactions to underperformance are dealt with using renegotiation rules and longer term fixes the use the service performance feedback manager to assess the problems and recommend updates to application performance models.

In the IRMOS implementation, the evaluator notifies the SLA manager of underperformance events. The SLA manager then notifies the PES of the problem and makes a request for an alternative resourcing policy. This new policy attempts to provide a short term reaction to the failure using the predefined renegotiation rules. The renegotiation rules themselves are defined when the original estimate is made and are based on resourcing solutions with an increased reliability. Obviously there comes a point when increasing resourcing makes no sense as the cost is prohibitive so renegotiation rules always ensure that solutions keep the service provider within the profitable range whenever possible.

For longer term corrections to performance models an application provider uses the service performance management portal to understand possible corrective actions. Through this interface the application provider can view reports on SLA breaches, actual QoS and estimated QoS. Specific failure events are accompanied with recommended actions (e.g. get report that assesses QoS data for specific SLA between a time period) which could result in a conclusion to analyses performance model V1.x as it is producing a resourcing policy for frames dropped with accuracy of < 50%.

The main components contributing to estimation correction are related to the Performance Feedback Service as shown in Figure 32.

Figure 32: Performance Feedback Architecture
6.4.4. Monitoring and Evaluation

A successful completion of the SLA Negotiation results on contracts accepted from all stakeholders involved and obligations to all sides. On the application level, the application bundles should be made available to the platform so as for them to be instantiated when required. On the infrastructure level, the necessary resources for the deployment and execution of the application should be reserved for the time period that was agreed in the SLAs while the services that are responsible for the monitoring and control of the application execution should be in place to support the execution process. The IRMOS Monitoring and Evaluation Services are in charge for collection of monitoring information and notifications from other components (ISONI and Application) and based on this information assess the execution status for the applications and the resource usage. In cases of failures or abnormal conditions, respective corrective actions are triggered in real-time by evaluator in order to maintain the QoS level and the smooth operation of the applications.

The Monitoring Service instance inside the VSN communicates with the application components to gather information for the execution and performance status of each component. This information is relayed to the main Monitoring Service in the FS which are aggregated with the monitoring data from ISONI regarding resource usage. The complete monitoring dataset is compared with the agreed parameters in the application and technical SLAs and in cases where violations are detected or likely to happen, notification events are produced and forwarded to Evaluator. Similar notifications regarding the status of the ISONI layer, such as the restart of a virtual machine, are also communicated to the Evaluator. The Evaluator has the key role in this process to evaluate the importance of the event and based on predefined policies to trigger appropriate actions in order to recover from this situation. These actions are forwarded to the respective components which will execute these actions, e.g. the SLA Manager for renegotiation, the Workflow Enactor for restarting a components etc. The following figure summarizes the monitoring and evaluation process of IRMOS.

![Figure 33: Sequence diagram for interactions of the Monitoring Service](image-url)
6.4.5. Service Resilience

Application Providers (customers of the IRMOS Platform) can specify redundancy levels for certain (or all) Application Service Components. Therefore, the IRMOS platform will establish a set of mechanisms to replicate ASCs in case of failure, minimizing the effect of outages and quickly restoring the normal execution of the application.

6.4.5.1. Redundancy

IRMOS provides several redundancy modes for applications (Figure 34):

- The high availability (HA) mode maintains two copies running and computing simultaneously, if one suffers a failure the other one will take over immediately. This needs to be modelled in the VSND, so it cannot be done implicitly by ISONI. Then in addition a balancing unit also needs to be part of the deployed VSN.
- In the implicit redundancy (IR) mode resources are reserved in case spare VMU image needs to be booted in case the active one has crashed.
- The last mode is the NR (no redundancy), where not machine is replicated nor resources are reserved for the eventual case of an outage.

![Figure 34: Redundancy modes of IRMOS](image)

The HA mode is not transparent for the application. This means it has to be modelled in the VSND and the application have explicitly support redundant resources. This does not exclude that for an active SC also a standby resources needs to be foreseen as in passive redundant case.

If passive redundancy is specified in the VSND, ISONI will keep resources available so that in case of a VMU outage a spare VMU image is booted on the spare PH. If this case happens, the IRMOS FS will be notified. Once the newly booted VMU starts its normal execution, the workflow enactor inside FS will configure and start the redundant ASC to recover the application. In this case the application not has to be aware of the redundancy mode. From the point of view of the application, this new ASC will replace the old one, keeping the same network and storage configuration. From the application perspective the failing ASC stops working for certain period of time and then it is
recovered (the fact that it is in fact executed in another VM is transparent to the application).

6.4.5.2. Live Migration

Live migration VMUs within ISONI nodes to another PH will happen in case a PH has to be emptied for maintenance actions or due to changed computing resource requirements in case of renegotiation. Live migration VMUs among ISONI nodes in case of predictable catastrophes like floodwaters, hurricanes, thunderstorms or simply management activities is a real gain for operators. The forecast in these cases would be evaluated by human beings that are authorized for emptying an ISONI node. In most cases there is enough time (days or at least hours) to move VMUs, before such a looming disaster impacts a node. This decision can hardly be automated and it is not required to do so. The emptying of a node results in relocation of VLs, live migration of VMUs and maybe movement of permanent storage data. Live migration within a node may be done in case of rearrangements for saving energy, caused by a renegotiation for meeting the new resource demands or in case of PH maintenance emptying the PH before be switched-off. In the final prototype it is planned to demonstrate the node internal live migration.

6.4.6. QoS Guaranteed Storage

We first briefly summarise the three stages involved in the development and deployment of applications in IRMOS from the perspective of storage before we look into the platform interactions within IRMOS storage architecture for providing QoS guaranteed storage.

In the Application Component Development phase, (where the application is “componentised” into multiple ASCs with a corresponding ASCD), the binaries corresponding to these ASCs are stored in an IRMOS repository, which is a part of the IRMOS storage architecture as mentioned earlier.

In the Application Service Design stage, the ASCD’s and their interactions are described through an Application Description (AD) template. These Application Descriptions have the information needed by Framework Services to generate Virtual Service Network Descriptions (VSNDs) as needed by ISONI. The A-SLA templates are then provided through the Framework Services, corresponding to each of these ADs. These A-SLA templates contain references to any pre-existing storage or information that the ASCs need for newly allocated storage for a VSN.

In the Application Use phase, the service is used by the IRMOS customer after agreeing to an SLA with the IRMOS service provider. At this stage, the application templates are filled with storage specific SLA requirements as needed by the IRMOS customer. The SLA negotiation then happens where an A-SLA is created on behalf of the customer. This gets translated to the T-SLA, which contains the storage parameters as needed by ISONI on behalf of the IRMOS customer. Any new storage references not part of the A-SLA template is obtained during this negotiation. After successful negotiation, the ASCs from
the repository are deployed. The running ASCs now continue to use the storage resources adhering to the negotiated QoS parameters.

The following are the negotiated QoS parameters for storage:

1. **Capacity**: This indicates the total “amount” of storage. This parameter is typically negotiated for storage pools. The unit for capacity is bytes.
2. **Lifetime**: This indicates the total duration during which a storage pool will be available, or will be available to any VMU. Lifetime has the same units as time.
3. **Resiliency**: This indicates the redundancy/reliability levels that need to be provisioned on a storage pool. Resiliency is a numerical value depicting a level of reliability.
4. **Bandwidth**: This is the total throughput in MB/s that should be available to a VMU connection accessing a storage pool.
5. **Jitter**: This is the variation in bandwidth that can be tolerated by the applications on VMUs. This has the unit of bandwidth per unit time.
6. **Latency**: This is the total delay (due to I/Os to/from storage) that can be tolerated by applications. This has the units of time.

The IRMOS storage architecture from the perspective of platform interactions is described in Figure 35. We mainly show interactions between the storage modules and the involved ISONI elements.

The Storage Manager and the Repository Manager interface with the rest of ISONI, both at the node and the domain level. The QoS Manager depicted here is internal to the Storage Manager. The QoS Target is also internal to the Storage Manager and works with the QoS Manager. The Lustre file system is the back-end high speed GbE/infiniband interconnect based architecture through which all the data is used/managed by the Service Components as if they were local storage elements directly connected to them. The Lustre file system consists of client modules and server modules. The storage hardware, which is RAID protected disk drive (commodity Serial ATA drives) units attached to the Lustre server module, are connected to this high speed network to be accessed by Lustre clients.
The process of using the storage modules start with the Deployment Manager receiving the VSN description that corresponds to a T-SLA between Framework Services and ISONI. The storage specific requirements such as capacity, bandwidth and latency for Service Components are included in the T-SLA through metadata descriptions corresponding to each Service Component. The Deployment Manager communicates with the Resource Manager Domain to select the best ISONI nodes that can accommodate the SC's. Fine grain division of SCs within an ISONI node is done by the Resource Manager Node. The SCs are deployed to the selected physical hosts through Virtual Machine Units, which are tailored Operating Systems that can accommodate the SCs. This forms the execution environment which directly interacts with the Lustre module, adhering to storage specific requirements in the T-SLA. The Resource Manager at the node level, interfacing with the Storage Manager makes sure that the storage specific T-SLA requirements are reasonably met. The Repository Manager is used to upload ASC binaries from the Framework Services that will later be used in the creation of VMU images (By the VMU factory). The Storage QoS manager and the Storage QoS Target which is part of the Storage Manager implements the actual QoS mechanism for storage (User Data and VMU Images) as required by the rest of ISONI and in turn the Framework Services.
The main functions of the QoS Manager is to accept or reject SLAs (on behalf of the storage subsystem) for storage pools, VMU connections for storage pools or client connections to the repository, based on the input parameters discussed earlier. This happens during the resource negotiation phase of VSN deployment. To accomplish this:

a. The QoS Manager gets SLA request and makes final decisions on behalf of the Storage Manager, before the Storage Manager sends out the result of the negotiation.

b. The QoS Manager interacts with the Repository Manager Node to get the SLA request for a client connection and then sends back a decision.

c. The QoS Manager interacts with storage devices through QoS Target modules.

d. The QoS Manager interacts with a system profiler, which is used offline to benchmark the storage subsystem

The QoS Manager defines a new parameter called “System Capability”. The System Capability method helps to make decisions to accept and reject connections based on the bandwidth/latency/jitter requested. The QoS Manager also provides a bandwidth throttling mechanism though QoS Targets connected to the storage pools and hence guarantee bandwidth for VMU’s or client connections to the repository without being affected by rogue applications.
7. Conclusions

This report presented the results of the work carried out in the scope of WP3 "IRMOS Platform Specification". The report highlights the relation of IRMOS platform with the cloud technology concepts and, in addition to this, how the real-time interactivity is achieved from the architectural design perspective. Furthermore the results of the requirements traceability analysis that took place following a suggestion from the project reviewers are presented. Based on the feedback from the technical WPs and the integration WP, the final analysis iteration on the architecture design took place, which led to the final set of platform capabilities and their association with particular layers of the cloud computing concept for providing reliability, scalability and low operational cost. Additionally the IRMOS components in Framework Services and ISONI were described in more detail as well as their interactions in the main processes of the platform which includes the functionality that is offered by the final IRMOS prototype.
8. References

[8] IRMOS Project D4.1.1 Definition of the three scenarios and their real-time requirements, TSG (GVG) and other partners, January 2009.
[9] IRMOS Project D5.1.1 Models of Real time Applications on Service Oriented Infrastructures, IT Innovation and other partners, January 2009.
Annex A. **User Requirements Traceability Matrix**

The following table summarises the user requirements, the IRMOS strategy, the related components and guidelines for testing for each one of them.

The cells of the matrix have been coloured to indicate:

- White - the requirements that are addressed (by particular components),
- Grey - the requirements that are **not** addressed but are not related with the core functionality of the platform (e.g. security or application specific requirements).

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Severity</th>
<th>IRMOS Strategy</th>
<th>Guidelines for testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>DM-1</td>
<td>File Transfer and FTP Server</td>
<td>High</td>
<td>WP5</td>
<td>DM-1-WP5-TC1: Check if the respective templates (A-SLA, T-SLA, VSND) support the local file system specification. DM-1-WP5-TC2: Check the successful creation of the A-SLA, T-SLA and VSND for one of the application scenarios.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DM-1-WP6-TC1: Check the correct allocation of Storage resources. DM-1-WP6-TC2: Check that the transference takes place. DM-1-WP6-TC3: If the application requires that some content is available in the storage previous to the execution of the application, it is needed to test the upload of data to the booked storage (in concrete for the postproduction scenario).</td>
</tr>
<tr>
<td>WP6</td>
<td>The Repository Manager will manage the storage interface for VMU Images and other data, WP5 will upload data via a defined interface.</td>
<td></td>
<td></td>
<td>Repository Manager</td>
</tr>
<tr>
<td>WP7</td>
<td>Network links will be provided between the</td>
<td></td>
<td></td>
<td>Path Manager</td>
</tr>
</tbody>
</table>
### ID | Name | Severity | IRMOS Strategy | Guidelines for testing | Components |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>WP Strategy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DM-2</td>
<td>Data Storage and Disk Space in a File System</td>
<td>High</td>
<td>WP5</td>
<td>The Storage subsystem and the EE (depending on QoS requirements and permanent / non permanent) will provide a certain amount of disk space according to what was requested by the user. Also an ASC can be deployed in a VMU with local storage.</td>
<td>DM2-WP6-TC1: The VMU has enough local storage for an ASCD with requirements of local file system, specified in the VSN. DM2-WP5-TC2: VSND that include requirements for local storage (based on the requirements of the ASCD).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WP6</td>
<td>Execution Environment, Storage Manager</td>
<td></td>
</tr>
<tr>
<td>DM-3</td>
<td>Secure Data Exchange</td>
<td>Medium</td>
<td>In principle IRMOS does not implement secure data exchange among EASC/ACC and ASC inside IRMOS. The application itself must implement the secure mechanisms.</td>
<td>DM3-WP6-TC1: Try to access concrete data from a different initiator.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WP6</td>
<td>Storage unit</td>
<td></td>
</tr>
<tr>
<td>DM-4</td>
<td>Scanning/Ingest service</td>
<td>High</td>
<td>This requirement is a special case of DM-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DM-5</td>
<td>High storage size capability</td>
<td>Low</td>
<td>WP6</td>
<td>Automatically identify when a special storage unit is needed or when regular storage can be used. This also applies to the size of the storage box and the maximum allowed capability.</td>
<td>DM5-WP6-TC1: Check storage size on instantiation. DM5-WP6-TC2: Review the limitations of the storage box in terms of size capability.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Resource Manager, Storage Manager, EE</td>
<td></td>
</tr>
<tr>
<td>DM-6</td>
<td>RAID System</td>
<td>Medium</td>
<td>WP6</td>
<td>In order to provide certain resilience RAID systems can be provided, although this is opaque to the user, who only request the level of redundancy needed</td>
<td>DM6-WP6-TC1: Succeed on the request of storage device with higher resiliency / redundancy.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Resource Manager, Storage Unit</td>
<td></td>
</tr>
<tr>
<td>DM-7</td>
<td>Data/Information</td>
<td>Medium</td>
<td>WP5</td>
<td>The FS will not implement any additional mechanism</td>
<td>DM7-WP6-TC1: Ensure that two</td>
</tr>
</tbody>
</table>

© ICCS/NTUA and other partners of the IRMOS consortium 2008-2011
<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Severity</th>
<th>IRMOS Strategy</th>
<th>Guidelines for testing</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>WP</td>
<td>Strategy</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>to ensure the integrity of the information in transit. The VSND specifies which storage boxes are accessible with QoS levels to be enforced</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WP6</td>
<td>No integrity is provided at EE level at this point. However, the storage box should implement the mechanism to ensure that different ASCs of different VMUs do not have access to the same information.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WP7</td>
<td>Networking uses the usual integrity mechanisms of IP layer. Integrity on higher layer is covered by the application and correlated OS.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>EE-1</td>
<td>Real-time deployment</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WP6</td>
<td>EEs need to be running at the time they were requested to. Not too late but also not too early to not waste resources.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WP7</td>
<td>Network connections will be deployed while the EE is being deployed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>EE-2</td>
<td>Control Panel and Administration Tools</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>EE2-WP5-TC1:</strong> Test the portal UI for all control and monitoring events. <strong>EE2-EE2-WP5-TC2:</strong> Compare the actual log files with the data exposed to the user via the portal to check if the events are properly propagated.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>EE-3</td>
<td>HTTP Server</td>
</tr>
<tr>
<td>ID</td>
<td>Name</td>
<td>Severity</td>
<td>IRMOS Strategy</td>
<td>Strategy</td>
</tr>
<tr>
<td>-----</td>
<td>--------------------------</td>
<td>----------</td>
<td>---------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>to be previously uploaded.</td>
</tr>
<tr>
<td>EE-4</td>
<td>SOAP Server</td>
<td>High</td>
<td></td>
<td>WP6</td>
</tr>
<tr>
<td>EE-5</td>
<td>Java EE Environments</td>
<td>High</td>
<td></td>
<td>WP6</td>
</tr>
<tr>
<td>EE-6</td>
<td>Interpreted languages</td>
<td>Low</td>
<td></td>
<td>WP6</td>
</tr>
<tr>
<td>EE-7</td>
<td>Database Server</td>
<td>High</td>
<td></td>
<td>WP6</td>
</tr>
<tr>
<td>ID</td>
<td>Name</td>
<td>Severity</td>
<td>IRMOS Strategy</td>
<td>Guidelines for testing</td>
</tr>
<tr>
<td>------</td>
<td>------------------------------------------------</td>
<td>----------</td>
<td>----------------</td>
<td>----------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>EE-8</td>
<td>Control of the Application</td>
<td>High</td>
<td>Same with EE-2</td>
<td></td>
</tr>
<tr>
<td>EE-9</td>
<td>Dynamic Context</td>
<td>High</td>
<td>WP5</td>
<td>FS will map the dynamic context application requirements to specific resource parameters and produce the respective VSN request to ISONI. Performance Estimation Service, Mapping Service EE-9-WP5-TC1: Check the smooth operation of an application with dynamic context requirements.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WP6</td>
<td>ISONI is unaware of the high level applications requirements such as the dynamic context, thus given that the VSN has the correct parameters these applications are treated the same way with any other application.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WP7</td>
<td></td>
</tr>
<tr>
<td>EE-10</td>
<td>Provide separation of services running over the same infrastructure</td>
<td>High</td>
<td>WP6</td>
<td>An isolated environment is provided to each service component in which it cannot interfere with other services. Running VMU EE-10-WP6-TC1: Try to access services in the same host from another service running in a VMU.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WP7</td>
<td>Services are connected through virtualized network connections and only allowed to connect to a restricted number of other services defined in the VSN. Running VMU EE10-WP7-TC1: Try to access services over the network from a service running in a VMU.</td>
</tr>
<tr>
<td>EE-11</td>
<td>Minimize and load balance resource usage</td>
<td>High</td>
<td>WP6</td>
<td>In order to keep resources used optimized automatic migration can occur inside a node. Renegotiation may also cause a migration inside a node for meeting the new resource requirements. Resource Manager, Migration Manager EE-11-WP6-TC1: Depends on allocation algorithm, can only be checked a-posteriori when it happens. EE-11-WP6-TC2: Enforce a situation for a running VMU, which makes a migration to another PH necessary to meet the new requirements. Check that in the end re-negotiation is successfully by occurred node internal migration</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WP7</td>
<td>VMU reallocation implies transparent network reconfiguration. Path Manager EE11-WP7-TC1: Check that network connection is alive after a migration.</td>
</tr>
<tr>
<td>EE-12</td>
<td>QoS in Execution Environment and storage</td>
<td>High</td>
<td>WP5</td>
<td>The FS produce a VSN with the requested QoS requirements and track the actual delivery of the requirements at runtime Performance Estimation Service, SLA Negotiator, EE12-WP6-TC1: Check that the application works in the required real-time conditions for execution and storage</td>
</tr>
</tbody>
</table>

© ICCS/NTUA and other partners of the IRMOS consortium 2008-2011
<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Severity</th>
<th>IRMOS Strategy</th>
<th>Components</th>
<th>Guidelines for testing</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>WP</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WP6</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Storage QoS can be defined in the VSN as well as QoS in the execution environment is defined by the real-time characteristics. These parameters are communicated to WP6-7 by means of the VSND, which also defines, minimum storage, RAM, and so forth.</td>
<td>EE (scheduler), Storage unit.</td>
<td><strong>EE12-WP6-TC2</strong>: Check that the corresponding BW, delay, etc. is delivered.</td>
</tr>
<tr>
<td>NET-1</td>
<td>Network provision reservation</td>
<td>High</td>
<td>WP6</td>
<td>Resource Manager</td>
<td><strong>NET1-WP6-TC1</strong>: Check whether the VMUs are placed considering propagation delay.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The collocation of VMUs depends also on delay parameters contained in the VSND.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WP7</td>
<td>Path Manager, IXBs</td>
<td><strong>NET1-WP7-TC1</strong>: Check that the corresponding BW, delay, etc. is delivered.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ISONI ensures accessibility as defined in the VLD of a VSND (part of T-SLA).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NET-2</td>
<td>Various users connected to the same application</td>
<td>High</td>
<td>WP5</td>
<td>SLA Manager, Workflow Enactor</td>
<td><strong>NET2-WP5-TC1</strong>: Check if the correct public IP addresses are assigned for a public accessible ASC.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IRMOS FS assigns public IP addresses to related ASC.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WP7</td>
<td>Path Manager, IXBs</td>
<td><strong>NET2-WP7-TC1</strong>: Check if the accessibility for a deployed VSN with public IP addresses is given from the Internet.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If configured in T-SLA (VSND), ISONI allows access from outside (e.g. Internet) to dedicated SCs by assigning a public IP address. The amount of users is not restricted, but the bandwidth of all users is limited. The range of available public IP addresses has to be known by IRMOS FS.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NET-3</td>
<td>Possibility of defining a Computer Cluster Network</td>
<td>High</td>
<td>WP5</td>
<td>SLA negotiator, Performance Estimation Service</td>
<td><strong>NET3-WP5-TC1</strong>: Check if the resources allocation is valid.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IRMOS FS reflects the Computer Cluster Network in a bounded set of computing resources specified in the T-SLA (VSND).</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WP6, WP7</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Treated as any other VSN deployment</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NET-4</td>
<td>Network Resource Monitoring and</td>
<td>Medium</td>
<td>WP5</td>
<td>Monitoring</td>
<td><strong>NET4-WP5-TC1</strong>: Check if received QoS is consistent with what was requested. <strong>NET4-WP5-TC2</strong>: Check if QoS is</td>
</tr>
</tbody>
</table>
## ID | Name | Severity | IRMOS Strategy | Components | Guidelines for testing
---|---|---|---|---|---
| | Reporting | | WP6 | ISONI will provide monitoring information for a deployed and running application. This information will consist of measurements done in relation to EE, storage and networking. | ISONI GW Deployment Manager, Path Manager | sufficient for application QoS
| | | | WP7 | | |
| | NET-5 | High | WP6 | ISONI can re-negotiate EE parameter, if sufficient resources are available. | ISONI GW, SLA-Manager, Resource Manager | NET5-WP6-TC1: If sufficient EE resources are available, the EE resources can be re-negotiated.
| | Adding a partner to an initiated session | | WP7 | ISONI can re-negotiate bandwidth, if necessary. The topology and the QoS requirements of a VLs have to remain the same. | ISONI GW, SLA-Manager, Path Manager | NET5-WP7-TC1: If sufficient bandwidth is available, the bandwidth of VL can be re-negotiated.
| | NET-6/15 | High | WP5 | Overall methodology, tools and services for management of RT SOI on virtualised infrastructures | FS | NET6-WP5-TC1: This is the final system test.
| | Provide edge-to-edge QoS for soft real-time services | | WP7 | The QoS network parameter bandwidth, jitter, delay can be provided with a very high probability over time (soft-QoS). | IXB | NET6-WP7-TC2: Check whether the monitored data of VL are within the specified required range.
| | NET-7 | High | WP5 | PES will estimate network performance characteristics of which jitter is a parameter. The estimated requirements are based on models whose accuracy will be updated following runtime monitoring | PES, evaluator, monitor | NET7-WP5-TC1: Test if the models for estimating jitter are sufficient to deliver A-SLA Quality of Service.
| | Maximum jitter specification | | WP7 | Maximum allowed jitter can be specified within the VSND of a T-SLA. | VSN Template | NET7-WP7-TC1: Check whether the max. Jitter can be specified per VL.
| | NET-8 | High | WP5 | PES will estimate network performance characteristics of which delay is a parameter. The estimated requirements are based on models whose accuracy will be updated following runtime monitoring | PES, evaluator, monitor | NET8-WP5-TC1: Test if the models for estimating delay are sufficient to deliver A-SLA Quality of Service.
| | Maximum delay specification | | WP7 | Maximum allowed delay can be specified within the VSND of a T-SLA. | VSN Template | NET8-WP7-TC1: Check whether the max. Delay can be specified per VL.
| | NET-9 | High | WP5 | PES will estimate network performance characteristics of which latency is a parameter. The | PES, evaluator, monitor | NET9-WP5-TC1: Test if the models for estimating latency are sufficient to
<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Severity</th>
<th>IRMOS Strategy</th>
<th>Components</th>
<th>Guidelines for testing</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>specification</strong></td>
<td></td>
<td><strong>estimated requirements are based on models whose accuracy will be updated following runtime monitoring</strong></td>
<td></td>
<td>deliver A-SLA Quality of Service.</td>
</tr>
<tr>
<td></td>
<td><strong>WP7</strong></td>
<td></td>
<td><strong>Maximum allowed latency can be specified within the VSN of a T-SLA.</strong></td>
<td><strong>VSN Template</strong></td>
<td><strong>NET9-WP7-TC1</strong>: Check whether the max. Latency can be specified per VL.</td>
</tr>
<tr>
<td>NET-10</td>
<td>Minimum throughput specification</td>
<td><strong>High</strong></td>
<td><strong>WP5</strong> PEs will estimate network performance characteristics of which throughput is a parameter. The estimated requirements are based on models whose accuracy will be updated following runtime monitoring</td>
<td><strong>PES, evaluator, monitor</strong></td>
<td><strong>NET10-WP5-TC1</strong>: Test if the models for estimating throughput are sufficient to deliver A-SLA Quality of Service.</td>
</tr>
<tr>
<td></td>
<td><strong>WP6</strong></td>
<td></td>
<td><strong>Throughput advertised will be provided during storage provisioning to meet application needs, SLA rejection will result if not possible.</strong></td>
<td><strong>Storage Manager</strong></td>
<td><strong>NET10-WP6-TC1</strong>: Check whether the minimum bandwidth can be specified for long term storage.</td>
</tr>
<tr>
<td></td>
<td><strong>WP7</strong></td>
<td></td>
<td><strong>Throughput is reached on ISONI by specifying a minimum granted bandwidth for VLs of a VSN.</strong></td>
<td></td>
<td><strong>NET10-WP7-TC1</strong>: Check whether the minimum bandwidth can be specified per VL.</td>
</tr>
<tr>
<td>NET-11</td>
<td>DNS Server</td>
<td><strong>High</strong></td>
<td>Not required for as a core IRMOS Platform Service. If an application requires it, it can be deployed as any other application service.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NET-12</td>
<td>Collaborative Work</td>
<td><strong>Medium</strong></td>
<td>Application Specific.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NET-13</td>
<td>Virtual Reality and Real World in the same scenario</td>
<td><strong>High</strong></td>
<td>Application Specific.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| NET-14 | Other Real-Time related Requirements     | **High** | **WP6** QoS Parameters for storage dependent on application can include:  
- Bandwidth  
- Capacity  
- Latency  
- Jitter  
- Resiliency | **Storage Manager** | **NET14-WP6-TC1**: Check performance achieved.  
**NET14-WP6-TC2**: Check that T-SLA is rejected if additional demand cannot be met. |
<p>|     | <strong>WP7</strong>                                   |          | <strong>Interactivity, Audio/Video Quality, Short Response</strong>                           |                              | Covered by NET7-10, nothing in                                                         |</p>
<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Severity</th>
<th>IRMOS Strategy</th>
<th>Components</th>
<th>Guidelines for testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>NET-16</td>
<td>Provide a virtualized private connectivity among service components across networks</td>
<td>High</td>
<td>WP7 Providing Intra-Node, Inter-node connectivity for SC of a VSN.</td>
<td>IXB</td>
<td>NET16-WP6-TC1: Check SC connectivity of VSN deployed on same PH, different PH and/or different ISONI Nodes.</td>
</tr>
<tr>
<td>NET-17</td>
<td>Provide a fail-safe network infrastructure</td>
<td>High</td>
<td>WP7 ISONI provides redundancy implicitly by redundant transport network interfaces, explicitly by multiple network paths. The switch-over may be seamless or actively triggered due to outage/degradation detection. In extraordinary cases the VMUs may be migrated to another node.</td>
<td>IXB</td>
<td>Not implemented in demonstrator</td>
</tr>
<tr>
<td>SEC-1</td>
<td>Provide Secure user Authentication to access the IRMOS platform</td>
<td>High</td>
<td>WP5 FS are responsible to analyse the security requirements of the users and produce a respective VSND. Which users will be able to access the VSN through EASC is defined by the application. WP6 WP7 The EE is password protected and only accessible if the network allows it. Therefore this is restricted to application level. No relevance to ISONI.</td>
<td>IRMOS Portal</td>
<td></td>
</tr>
<tr>
<td>SEC-2</td>
<td>Access Denied to the IRMOS platform</td>
<td>High</td>
<td>WP5 IRMOS Portal will handle the user authentication to IRMOS Platform. WP6 WP7 If a user introduces wrong credentials this event is logged in the syslog of the machine.</td>
<td>IRMOS Portal</td>
<td></td>
</tr>
<tr>
<td>SEC-3</td>
<td>Authorization mechanism</td>
<td>High</td>
<td>WP5 IRMOS Portal will handle the user authentication to IRMOS Platform. Which users will be able to access IRMOS Portal</td>
<td>IRMOS Portal</td>
<td></td>
</tr>
</tbody>
</table>
## IRMOS Strategy

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Severity</th>
<th>WP</th>
<th>Strategy</th>
<th>Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEC-4</td>
<td>Logout from the IRMOS platform</td>
<td>High</td>
<td>WP5</td>
<td>IRMOS Portal will handle the user authentication to IRMOS Platform. Which users will be able to access the VSN through EASC is defined by the application.</td>
<td>IRMOS Portal</td>
</tr>
<tr>
<td>SEC-5</td>
<td>Customizable User Interface</td>
<td>Low</td>
<td>WP5</td>
<td>Users will be able to access the platform through IRMOS Portal. API will be available to create custom user interface.</td>
<td>IRMOS Portal</td>
</tr>
<tr>
<td>SEC-6</td>
<td>Data/Information Confidentiality</td>
<td>Medium</td>
<td>WP6</td>
<td>The system does not allow other users to access this data. In addition to that, the user is in principle entitled to provide its credentials and encrypt his data to hide it from the infrastructure (ISONI). The only exception on this is the long term storage where credentials are required.</td>
<td>EE, IRMOS Portal</td>
</tr>
<tr>
<td>SEC-7</td>
<td>Secure video and audio exchange</td>
<td>Medium</td>
<td></td>
<td>Does not differ to any other deployment. This is restricted to application level.</td>
<td></td>
</tr>
<tr>
<td>SLA-1</td>
<td>SLA Management service</td>
<td>High</td>
<td>WP5</td>
<td>In FS layer the required mechanism will be included to store the SLAs and their templates and also to CRUD them in both the offline and online phases. During the execution of an application, the A-SLAs and T-SLAs will be continuously checked to detect violations and also trigger renegotiation when required.</td>
<td>A-SLA Manager T-SLA Manager and respective repositories to store both the templates and the SLAs.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WP6-WP7</td>
<td>ISONI will be responsible for the T-SLA management of the T-SLAs from the provider point of view in order to provide the agreed QoS levels.</td>
<td>ISONI SLA Manager</td>
</tr>
<tr>
<td>SLA-2/3</td>
<td>Provide tools to perform SLA Negotiation (between Framework Services and ISONI)</td>
<td>High</td>
<td>WP5</td>
<td>Considering that in IRMOS we follow a two layer SLA approach, the framework services layer will be responsible for the A-SLA negotiation and will also trigger the negotiation of T-SLAs. An important prerequisite on this is the mapping of the high level requirements to low level parameters and the creation of VSND to perform the T-SLA request.</td>
<td>SLA Negotiator</td>
</tr>
</tbody>
</table>

**Guidelines for testing**

- **SEC6-WP6-TC1**: The long term storage is first accessible after specifying the credentials for its access.
- **SLA1-WP5-TC1**: Test the CRUD functionality of FS for A-SLAs.
- **SLA1-WP5-TC2**: Test the CRUD functionality of FS for T-SLAs.
- **SLA1-WP6-TC1**: Test the CRUD functionality of ISONI for T-SLAs.
## IRMOS Strategy

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Severity</th>
<th>WP</th>
<th>Strategy</th>
<th>Components</th>
<th>Guidelines for testing</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>WP6-WP7</td>
<td>ISONI will support the automated negotiation of T-SLAs and also renegotiation when more resources are required given that the VSN layout is not changed and there are resources available.</td>
<td>ISONI SLA Manager</td>
<td>Same with SLA2-WP5-TC2 and SLA2-WP5-TC3.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WP5</td>
<td>FS aggregates monitoring information from both the application layer and ISONI and based on these is able to detect SLA violations.</td>
<td>Monitoring, T-SLA Management, A-SLA Management</td>
<td>SLA4-WP5-TC1: Test SLA violation detection mechanism.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WP6</td>
<td>The Execution environment and storage unit produces monitoring information</td>
<td>Monitoring</td>
<td>SLA4-WP6-TC1: Monitoring information is received</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WP7</td>
<td>ISONI will provide monitoring information for a deployed and running application. This information will consist of measurements done in relation to EE, storage and networking.</td>
<td>Monitoring</td>
<td>SLA4-WP7-TC1: Monitoring for networking is provided</td>
</tr>
</tbody>
</table>

### SLA-4

**Provide tools for monitoring and reporting**

**High**

**WP5**

**Guidelines for testing**

<table>
<thead>
<tr>
<th>WP</th>
<th>Strategy</th>
<th>Components</th>
<th>Guidelines for testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>WP6-WP7</td>
<td>ISONI will support the automated negotiation of T-SLAs and also renegotiation when more resources are required given that the VSN layout is not changed and there are resources available.</td>
<td>ISONI SLA Manager</td>
<td>Same with SLA2-WP5-TC2 and SLA2-WP5-TC3.</td>
</tr>
<tr>
<td>WP5</td>
<td>FS aggregates monitoring information from both the application layer and ISONI and based on these is able to detect SLA violations.</td>
<td>Monitoring, T-SLA Management, A-SLA Management</td>
<td>SLA4-WP5-TC1: Test SLA violation detection mechanism.</td>
</tr>
<tr>
<td>WP6</td>
<td>The Execution environment and storage unit produces monitoring information</td>
<td>Monitoring</td>
<td>SLA4-WP6-TC1: Monitoring information is received</td>
</tr>
<tr>
<td>WP7</td>
<td>ISONI will provide monitoring information for a deployed and running application. This information will consist of measurements done in relation to EE, storage and networking.</td>
<td>Monitoring</td>
<td>SLA4-WP7-TC1: Monitoring for networking is provided</td>
</tr>
</tbody>
</table>

## VS-1

**Visualization of multimedia content**

**High**

This is mainly an application requirement. However, IRMOS will provide the mechanisms / tools for the underlying layers.

<table>
<thead>
<tr>
<th>WP</th>
<th>Strategy</th>
<th>Components</th>
<th>Guidelines for testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>WP5</td>
<td>Modelling of the visualization functionality is required to map it to particular requirements and parameters, e.g. streaming, permanent storage.</td>
<td>Performance Estimation Service</td>
<td>VS1-WP5-TC1: Check the smooth operation of a multimedia application when the appropriate VSN is deployed.</td>
</tr>
<tr>
<td>WP6</td>
<td>ISONI will support the permanent storage of multimedia content.</td>
<td>Storage node</td>
<td>VS1-WP6-TC1: Implement test with 2 VSN’s one following another and ensure data is accessible.</td>
</tr>
</tbody>
</table>

## VS-2

**Multi-Video Conference service**

**High**

This is an application requirement.

<table>
<thead>
<tr>
<th>WP</th>
<th>Strategy</th>
<th>Components</th>
<th>Guidelines for testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>WP5</td>
<td>Modelling of the visualization functionality is required to map it to particular requirements and parameters, e.g. streaming, permanent storage.</td>
<td>Performance Estimation Service</td>
<td>VS1-WP5-TC1: Check the smooth operation of a multimedia application when the appropriate VSN is deployed.</td>
</tr>
</tbody>
</table>

## VS-3

**Video quality**

**High**

FS will provide an API for video streaming. Furthermore mapping of the high level video quality requirements to low level resource parameters is necessary in order to reserve and initialize accurately the required resources.

<table>
<thead>
<tr>
<th>WP</th>
<th>Strategy</th>
<th>Components</th>
<th>Guidelines for testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>WP5</td>
<td>Performance Estimation Service, Streaming API</td>
<td>VS3-WP5-TC1: Test the correct mapping of high level video quality requirements to low level resource parameters.</td>
<td>VS3-WP5-TC1: Test the correct mapping of high level video quality requirements to low level resource parameters.</td>
</tr>
</tbody>
</table>
## D3.1.4 Final version of IRMOS Overall Architecture

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Severity</th>
<th>IRMOS Strategy</th>
<th>Guidelines for testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>VS-4</td>
<td>Multi-Video Conference</td>
<td>High</td>
<td>IRMOS will handle this as any other application.</td>
<td></td>
</tr>
</tbody>
</table>
Annex B. Security Requirements Traceability Matrix

The following table summarises the security requirements, the IRMOS strategy, the related components and guidelines for testing for each one of them.

The cells of the matrix have been coloured to indicate:
- White - the requirements that are addressed (by particular components),
- Grey - the requirements that are not addressed but are not related with the core functionality of the platform (e.g. security or application specific requirements).

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Severity</th>
<th>IRMOS Strategy</th>
<th>Guidelines for testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFR-SEC1</td>
<td>Confidentiality</td>
<td>High</td>
<td>WP5</td>
<td>IRMOS Portal</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WP5 implements use existing technologies to provide confidentiality to the information: login/password, and https, ftps, etc for data transfer.</td>
<td></td>
</tr>
</tbody>
</table>

|     |            |          | WP6            | EE, storage unit        |
|     |            |          | VMUs are independent and opaque running units. These might be attached to a storage unit, this connection needs to be secured. | | NFR-SEC1-WP6-TC1: Check if a target (storage unit) can be accessed from a different initiator (EE) that it is supposed to. |
### ID | Name | Severity | IRMOS Strategy | Components | Guidelines for testing
--- | --- | --- | --- | --- | ---
<p>| | | | WP | Strategy | |
| | | | WP7 | Data being exchanged between VSN components cannot be eavesdropped by other VSNs. | IXBs | |
| NFR-SEC2 | Integrity | High | Same with DM-7. | | |
| NFR-SEC3 | The system should be able to identify its users for validating later security decisions | High | WP5 | The IRMOS Portal as entry point for IRMOS should provide an authentication mechanism for the clients. | IRMOS Portal | NFR-SEC3-WP5-TC1: Only authenticated users can access the IRMOS portal (for instance, with a login/password) |
| | | | WP6, WP7 | The user of ISONI is the IRMOS FS. Each actions are authenticated by ISONI GW | ISONI GW | NFR-SEC3-WP6-TC1: The user (IRMOS FS) has to authenticate each action via ISONI GW. |
| NFR-SEC4 | Prevent any inappropriate access to services or information via IRMOS | High | WP5 | The IRMOS Portal should ensure that the information is only accessible by the right person. | IRMOS Portal | NFR-SEC4-WP5-TC1: Users already logged into the system are the only who can access his/her own information. |
| | | | WP7 | The VSN isolation concept avoids inappropriate access to a running VSN. SCs which have to be accessible form outside needs to specify a public IP address. | IXB | NFR-SEC3-WP7-TC1: Test, that just SCs with public assigned IP addresses are accessible from the Internet. NFR-SEC3-WP7-TC2: Test whether SCs without a public IP address can reach the Internet, they must not! |
| NFR-SEC5 | Security outside | Medium | Security mechanisms are limited to IRMOS and to directly connected clients with IRMOS. | | |
| | | | WP6, WP7 | The IRMOS FSs, such as the discovery service, deal with several ISONIs meaning different realms, which requires authentication on ISONI. | ISONI GW | NFR-SEC5-WP6-TC1: IRMOS FS has to be authenticated for doing any action via ISONI GW. |
| FR-SEC1 | IRMOS should allow the enforcement of different authorisation privileges for accessing the Framework | | WP5 | IRMOS Portal will be responsible for AAA functionality in FS level. | IRMOS Portal |</p>
<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Severity</th>
<th>IRMOS Strategy</th>
<th>Guidelines for testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>FR-SEC2</td>
<td>IRMOS should be able to control access on Virtual Service Network (VSN) level where each user is associated with a list of reserved VSN resources</td>
<td>WP5</td>
<td>SLA Manager, IRMOS Portal</td>
<td>FR-SEC2-WP5-TC1: The customer should be able to specify EASC (e.g. permanent storage location) within the SLA. A secure connection is made when communicating with this EASC. FR-SEC2-WP5-TC2: Information about the VSN deployment access is supplied only to the appropriate customer.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WP6 WP7</td>
<td>ISONI GW</td>
<td></td>
</tr>
<tr>
<td>FR-SEC3</td>
<td>IRMOS should enable the application user to authenticate for accessing the Framework Services</td>
<td>WP5</td>
<td>IRMOS Portal</td>
<td>FR-SEC2-WP6-TC1: ISONI will prevent any access to the VSN that is not indicated within the VSND.</td>
</tr>
<tr>
<td>FR-SEC4</td>
<td>IRMOS should allow mutual authentication for secure communications amongst the Framework Services</td>
<td>WP5</td>
<td>GT4</td>
<td></td>
</tr>
<tr>
<td>FR-SEC5</td>
<td>IRMOS should allow authentication of Framework</td>
<td>WP6</td>
<td>Repository Manager, Deployment Manager,</td>
<td></td>
</tr>
</tbody>
</table>
### D3.1.4 Final version of IRMOS Overall Architecture

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Severity</th>
<th>IRMOS Strategy</th>
<th>Guidelines for testing</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Services against ISONI</td>
<td></td>
<td>WP6</td>
<td>Resource Manager Node</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WP7</td>
<td>ISONI GW</td>
</tr>
<tr>
<td>FR-SEC7</td>
<td>IRMOS should be able to offer confidentiality for the data in transit over unsecured networks</td>
<td>WP5</td>
<td>Not supported in FS level thus it should be addressed by the application itself.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>WP7</td>
<td>In general ISONI is based on dealing with trusted network. In case ISONI needs to go via unsecured network, the data will be encrypted. Public IP traffic is not encrypted by ISONI, therefore the application itself has to ensure the security here,</td>
<td>IXB</td>
</tr>
<tr>
<td>FR-SEC6</td>
<td>IRMOS should allow applications to have their own security mechanisms</td>
<td></td>
<td>This is possible in application level.</td>
<td></td>
</tr>
<tr>
<td>FR-SEC8</td>
<td>IRMOS should provide mechanisms to preserve integrity of data</td>
<td></td>
<td>Same with DM-7.</td>
<td></td>
</tr>
<tr>
<td>FR-SEC9</td>
<td>IRMOS should isolate SC to limit mutual impact on the infrastructure and the VSN</td>
<td>WP5</td>
<td>FS are only responsible for mapping the user requirements to VSN requests. The isolation of the SC is performed in ISONI layer.</td>
<td>EE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WP6</td>
<td>Between different VSN the isolation is given. Inside a VSN the application itself is responsible to avoid mutual impact. The Scheduler for VMUs ensures that SC cannot consume more CPU power than requested.</td>
<td>Path Manager, IXB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WP7</td>
<td>Between different VSN the isolation is given. Inside a VSN the application itself is responsible to avoid mutual impact. The IXB ensures that VL traffic cannot</td>
<td></td>
</tr>
</tbody>
</table>

© ICCS/NTUA and other partners of the IRMOS consortium 2008-2011
<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Severity</th>
<th>IRMOS Strategy</th>
<th>Guidelines for testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>FR-SEC10</td>
<td>IRMOS should isolate SC data to prevent access from other SC</td>
<td>WP6</td>
<td>Two service components can only communicate if the VSN were designed to make them communicate. Also volatile storage is just accessible by related VMU itself. Credentials are needed to access the long term storage.</td>
<td>Same with SEC6-WP6-TC1.</td>
</tr>
<tr>
<td>FR-SEC11</td>
<td>Security management offered by IRMOS should include controlling ISONI administrative interface</td>
<td>WP6, WP7</td>
<td>The user of ISONI is the IRMOS FS. Each action is authenticated by ISONI GW.</td>
<td>ISONI GW</td>
</tr>
<tr>
<td>FR-SEC12</td>
<td>IRMOS should allow enforcing access control on the content stored at ISONI storage nodes</td>
<td>WP5</td>
<td>The storage access requests are forwarded to ISONI as part of the VSND.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>WP6</td>
<td>Access control is provided by the long term storage node itself.</td>
<td>Storage Manager</td>
</tr>
<tr>
<td>FR-SEC13</td>
<td>IRMOS should be configurable to allow SC within a VSN to communicate with components in another</td>
<td>WP5</td>
<td>FS will analyze the requirements and the application configuration options and produce respective VSNDs.</td>
<td>Not implemented</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WP7</td>
<td>The connectivity among VSN has to be defined within the VSND. ISONI just connects the VSNs as specified.</td>
<td></td>
</tr>
<tr>
<td>FR-SEC14</td>
<td>IRMOS should allow enforcement of security</td>
<td>WP7</td>
<td>In case of multiple domains security gateways has to ensure the confidentiality and integrity between the domains.</td>
<td>Not implemented</td>
</tr>
<tr>
<td>ID</td>
<td>Name</td>
<td>Severity</td>
<td>IRMOS Strategy</td>
<td>Guidelines for testing</td>
</tr>
<tr>
<td>--------</td>
<td>----------------------------------------------------------------------</td>
<td>----------</td>
<td>----------------</td>
<td>----------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>measures (confidentiality and integrity) on communication spanning multiple ISONI domains</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FR-SEC15</td>
<td>IRMOS should be configurable to enable different security mechanisms by each adopter</td>
<td></td>
<td>WP5</td>
<td>This should be addressed by the application itself.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WP6</td>
<td>A user can provide his own security mechanisms on top of the ones provided by IRMOS at EE level.</td>
</tr>
<tr>
<td>FR-SEC16</td>
<td>IRMOS security infrastructure should permit accountability between partners</td>
<td></td>
<td>WP6 WP7</td>
<td>As the environment is created on the spot and allocated to a VSN, it is easy to audit who has done what and take actions upon it. The FS is authenticated and actions inside the VSN are logged however this second is not directly related to security.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Not implemented</td>
</tr>
<tr>
<td>FR-SEC17</td>
<td>IRMOS should offer simplified access to framework services within inter-organisational environment</td>
<td></td>
<td>WP5</td>
<td>Access within inter-organisational environment could be offered by extending the IRMOS Portal API.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WP6 WP7</td>
<td>IRMOS Portal</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Not implemented</td>
</tr>
<tr>
<td>FR-SEC18</td>
<td>IRMOS should support the use of legally binding Service</td>
<td></td>
<td>WP6 WP7</td>
<td>ISONI supports legally binding between IRMOS FS and ISONI reflected by T-SLA</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SLA Manager</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>FR-SC18-WP6-TC1: Check whether the T-SLA negotiation between IRMOS FS and ISONI is working.</td>
</tr>
<tr>
<td>ID</td>
<td>Name</td>
<td>Severity</td>
<td>IRMOS Strategy</td>
<td>Guidelines for testing</td>
</tr>
<tr>
<td>------</td>
<td>-------------------------------------------</td>
<td>----------</td>
<td>----------------</td>
<td>------------------------</td>
</tr>
<tr>
<td></td>
<td>Level Agreements (SLA) between partners</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>