Interactive Realtime Multimedia Applications on Service Oriented Infrastructures

ICT FP7-214777

WP 6/7

ISONI Whitepaper

IRMOS_WP6_7_ISONI_White_Paper_ALUD_USTUTT_v2_0.doc

Scheduled Delivery: 30 September 2008
Actual Delivery: 05 September 2008
Updated: 01 July 2009
Version 2.0

Project co-funded by the European Commission within the 7th Framework Programme		
Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission)	
RE	Restricted to a group specified by the consortium (including the Commission)	
CO	Confidential, only for members of the consortium (including the Commission)	

Responsible Partner: ALUD/USTUTT

Revision history:

Date	Editor	Status	Version	Changes
30.05.2008	Thomas Voith	Draft	0.1	Initial draft
03.07.2008	Markus Kessler	Draft	0.2	Updated Draft with major revisions
04.07.2008	Karsten Oberle	Draft	0.3	Minor update of all chapters.
14.07.2008	Dominik Lamp	Draft	0.4	Consolidation of comments; restructiorning + general overhaul
15.07.2008	Karsten Oberle	Final draft	0.5	Overall final editing.
29.08.2008	Dominik Lamp, Marcus Kessler, Karsten Oberle	For approval	0.6	Incorporated comments from internal quality review.
05.09.2008	Karsten Oberle	Final Version	1.0	Approved Version.
01.07.2009	Karsten Oberle	Final Version	2.0	Improved readabiblity.

Authors

ALUD: Thomas Voith, Marcus Kessler, Karsten Oberle
USTUTT: Dominik Lamp, Antonio Cuevas, Patrick Mandic, Andreas Reifert

Internal Reviewers

Matthew Addis (IT-Innovation), Dimosthenis Kyriazis (NTUA), Tim Courtney (XY, Consolidator)

Copyright

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

Acronym	Definition
ATCA	Advanced Telecommunication Architecture
API	Application Programming Interface
DSP	Digital Signal Processor
EE	Execution Environment
GT4	Globus Toolkit 4
ISONI	Intelligent Service Oriented Network Infrastructure
IXB	ISONI eXchange Box
JRE	Java Runtime Environment
MDS	Monitoring and Discovery System
PH	Physical Host
SC	Service Component
SDP	Service Delivery Platform
SIP	Session Initiation Protocol
SLA	Service Level Agreement
SOA	Service Oriented Architecture
SOI	Service Oriented Infrastructures
URI	Uniform Resource Identifier
VM	Virtual Machine
VSN	Virtual Service Network
VPN	Virtual Private Network
WAN	Wide Area Network
WS	Web Service
WSDL	Web Service Description Language

Table of Contents

List of Figures

1. Executive Summary

The deployment of multi-component services in a network of distributed resources is a complex task. To guarantee sufficient overall performance of the service, i.e. to guarantee the required Quality of Experience, individual components must be deployed to appropriate resource locations selected to meet their resource and communications footprint. Real-time (conversational) services, e.g. video or voice, impose additional quality of service (QoS) constraints on the inter component linkage, mainly in terms of bandwidth, jitter, and delay.

From a resource provider's point of view, a major challenge is to offer service providers sufficient degrees of freedom regarding the selection of resources while enforcing certain restrictions to prevent unwanted crosstalk between services deployed by different service providers. ISONI is a framework for automated provisioning of multi-component services in a fully virtualized resource environment in compliance with QoS guarantees.

.

2. Introduction

Businesses can present themselves to a huge customer base through the Internet, thus enabling even small and medium sized companies to address a large audience. While proof of concept implementations are relatively simple to provide, deep know-how in application development and service deployment on distributed hardware resources is required to create services that scale well to a growing customer base. This process is further complicated by the need to respect possible real-time requirements of services too.

Service delivery platforms (SDP) that follow the service oriented architecture (SOA) approach support the development of scalable services software. However, typical current frameworks, e.g. Grid solutions, do not take the resource infrastructure necessary for the execution of the service into consideration. In most cases, those frameworks focus on providing huge and extremely divisible applications with hardware resources distributed over several provider domains and manage only computing related resources like CPU and RAM. They take resources like network connectivity for granted and do not holistically consider Quality of Service (QoS) or other real-time aspects (e.g. jitter & delay) of the message and data exchange between possibly thousands of components.

The ISONI (Intelligent Service Oriented Network Infrastructure) is an infrastructure, consisting of a network of resources (including CPU, storage, networking 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). For SCs orchestrated into a complete service, a virtual machine (VM) will be provisioned, although it will still be possible to place several SCs in one virtual machine. ISONI is agnostic to services, it focuses on providing the best resources for these SCs to execute, whatever these SCs are. Links between SCs will be provided as needed too.

3. Intelligent Service Oriented Network Infrastructure

The basic purpose of the ISONI is to reduce the complexity for service providers/developers to roll out new network based services. It does this by undertaking the automatic deployment of the services on best fitting resources distributed in a network. The solution strives to reduce global costs by 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. 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 hardware resources distributed in a network from that of deployed services and their associated service components. Thus the actual status and distribution of resources are hidden from the service developer's view. The infrastructure provides him with fully virtualized resources, including network resource. This 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 execution environment requirements of the service, 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 provisioning. In particular, the creation of the description must 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, this will be presented in more detail in chapter 3.2.

The third task of the ISONI will be the monitoring of running services and their resource usage. This monitoring data will be available to the service developer, e.g. via web service interfaces.

The principal service deployment in ISONI is illustrated in Figure 1. Logical Components are mapped on Physical Hosts (mapping shown by dotted thin lines) and provided with virtual interlinks (coloured thick lines) that are isolated from each other, reflecting the VSN definitions. The upper part of this figure shows the abstract view of a service, in fact of two examples of a service, in form of VSN's (each example highlighted in yellow and green colour). Each VSN is composed of multiple SC's with virtual links between, each described by virtual link descriptions. The lower part of Figure 1 shows the mapping of the highly abstracted resource requests (two VSN's) onto the network of real Execution Environment resources (Virtual Machine Units) and real links through the network(s)

between those VMU's. The corresponding functionality of the ISONI eXchange Box (IXB) as shown in Figure 1 is introduced in chapter 5.3.1.

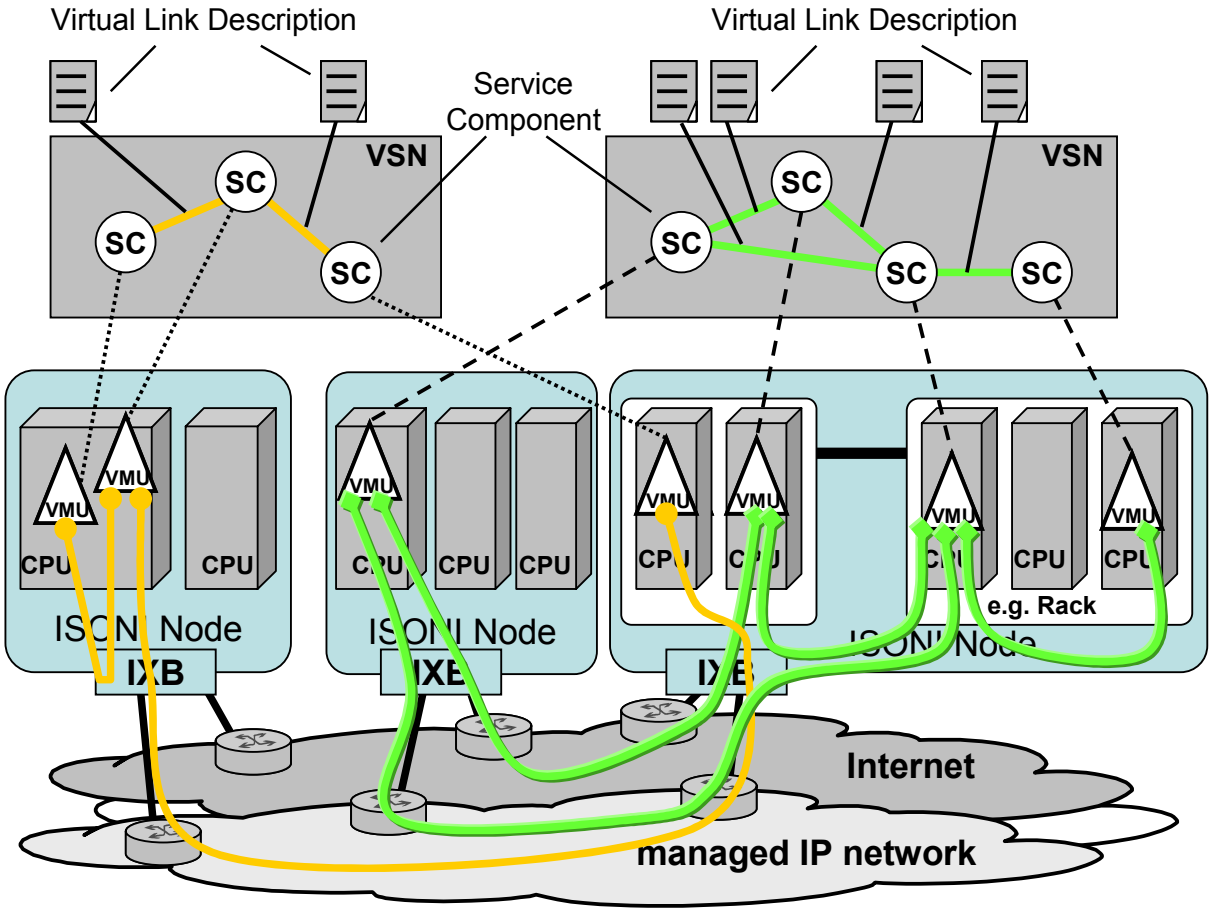

Figure 1 Deployment of services on the ISONI

3.1. Layering of the ISONI

The ISONI manages resources, Service Components, and their respective interconnections. For proper resource assignment an accurate view of the current usage of all these resources, components, and connections is necessary. For scalability reasons, the management is hierarchically organized and decentralized wherever possible. Figure 2 shows the two hierarchical levels of ISONI, domain and node level, that exhibit fundamental architectural differences.

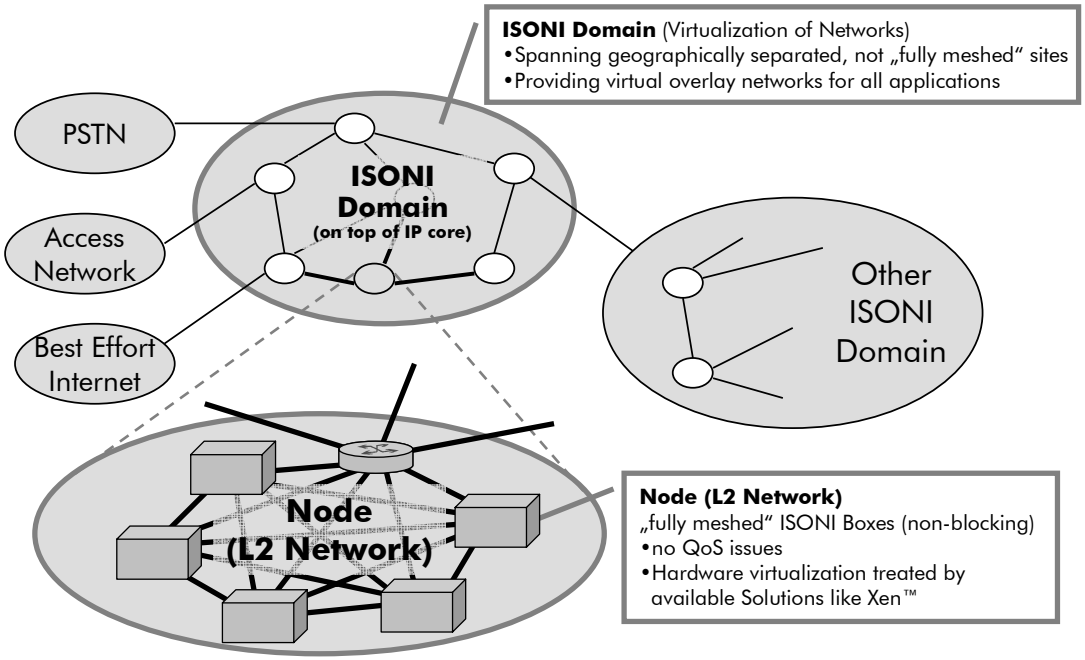

Figure 2 ISONI layers

Node level

In the ISONI context, a node consists of a number of general purpose racks equipped with several "blades". Resources provided by these "blades" include computing platforms, DSP-farms with hard-coded functionalities, storage, etc.

As the ISONI concept is not limited to any particular rack architecture, but can be used with any kind of hardware, individual computational units are called *Physical Hosts (PH)*. Additionally, boxes comprising special purpose hardware, i.e. special hardware installed for specific purposes, may be part of a node. All boxes, blades, and racks are interconnected by a high performance network. Due to this (massive) overprovisioning, which is quite inexpensive to achieve in a small physical area, it is assumed that QoS is no issue on node level.

Domain level

The ISONI Domain consists of several ISONI nodes that are governed by identical policies and are managed by one ISONI provider. Usually, these nodes are geographically distributed in a wide area network. The nodes are not fully meshed and QoS provisioning is an important issue here. One ISONI domain may span several network provider domains, i.e. one ISONI provider might have his physical hosts hosted by multiple network and PH providers.

Access to the general internet is only possible through specialised Gateway Service Components that are deployed to the Virtual Service Network, these are introduced in the next section.

3.2. Virtual Service Networks (VSNs)

In order to deploy a service, the service developer has to attach an abstract description of the service's (run-time) requirements when transferring his service to the ISONI. To generate the description they:

- design their service by selecting and parameterizing appropriate Service Components that are either already present within the ISONI or which are transferred to it. Each of those SCs is annotated by its programmer with a document we call *Metadata* (see Figure 3 for illustration) which is a description of all the requirements of its execution environment. This includes the description of the interfaces or sockets and their respective properties, provided by the SC.
- define the interconnections of those SCs in form of a Virtual Link Description (see Figure 3 for illustration) resembling the actual communication structure of the SCs in the VSN.

All the descriptions and parameters are merged in a so-called Virtual Service Network (VSN) description. The VSN can be seen as a graph whose vertices are the SCs and whose edges are the Virtual Links (see Figure 3). The VSN description is transferred to the ISONI with the request to instantiate the service. The ISONI then:

- Automatically and autonomously maps the highly abstracted resource request in form of the VSN description onto the network of real resources
- Deploys the components in tailored execution environments on suitable resources
- 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. The overlay network and the underlying addressing scheme are described in section 5.3.

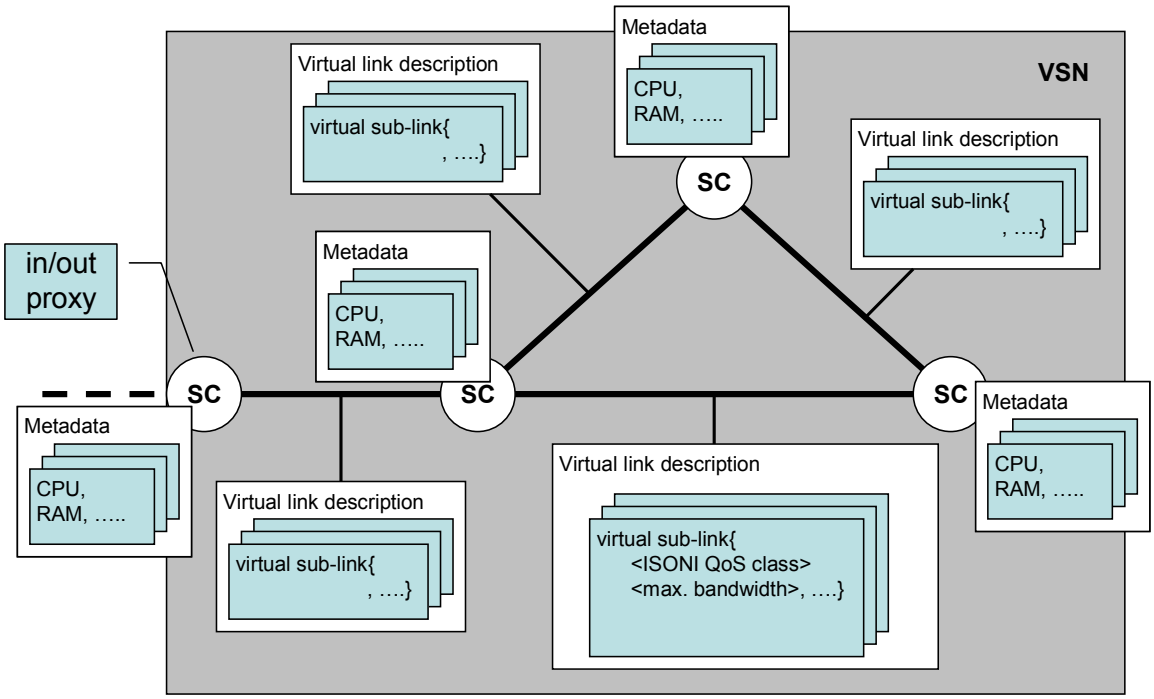

Figure 3 VSN description

3.3. Scheduling of Service Components

As the VSN description contains the complete information that is necessary to determine whether or not the ISONI domain is able to host a service at the desired QoS level, it can be seen as a *technical SLA request.*

Once this SLA request is received by the ISONI, it checks whether the required resources are available during the desired time interval. QoS management takes place on several layers in the ISONI. It is considered both for the VMUs, i.e. on Execution Environment (EE) layer, and also on the network level.

Furthermore, ISONI performs reservation of these resources so that, during the requested service runtime, the availability of the requested resources can be guaranteed and so that the customer does not have to deal with non-availability.

In the current model, the computation resources are expected to be the limiting factor, i.e. it is quite unlikely that no links are available to interconnect the PHs chosen to host the VSN's SCs. Thus, the computational resources are discovered first. ISONI then generates a set of possible mappings to the physical infrastructure. These mappings are then checked against the available network resources and the overall costs are calculated.

Based on the outcome of this ISONI internal process, ISONI responds with an SLA response towards the ISONI customer.

4. Execution Environment (EE)

Service Components (SC) run in Execution Environments (EE) provided by ISONI. An Execution Environment contains one or more virtual machines with added capabilities, these are called Virtual Machine Units (VMU). These added capabilities are the distinguishing factors that make the ISONI Execution Environment unique and innovative. Several Service Components and Execution Environments can be run on the same physical host.

Among the most distinctive capabilities of the EE are:

- isolation. The ISONI has a deny-all default policy. Therefore any legitimate network traffic in the VSN must be specified a-priori. Services components are also isolated from the failure or "misbehaviour" of other services, i.e., both foreign VSNs as well as individual SCs in the same context. Isolation is also performed against the Internet and other networks. Both inbound and outbound connections must be explicitly allowed,
- real-time. The ISONI platform addresses the challenge to provide soft real-time capabilities even in virtualized environments,
- redundancy and fault tolerance capabilities that, combined with the migration process, will assure that the confirmed conditions engaged with the service components running on top of our execution environment are kept even in case of failures,
- mechanisms and interfaces for resource metering, monitoring and reporting of the services components.

4.1. Automatic configuration and provisioning of Virtual Machines

The Metadata of a SC as introduced in section 3 contains, among others, information about type and amount of required hardware resources (CPU, RAM, file system, storage), as well as a description of the necessary runtime environment. The latter includes specifications about operating system, libraries and their versions, other virtual runtime environments, etc. The Metadata is specific to a SC instance in a VSN. Hence a SC can be scaled to the estimated load in a VSN.
Based on this information, the ISONI is able to determine the actual quantitative and qualitative resource requirements of a SC. This is in turn used to tailor a VMU to the needs of the SC, and to equip it afterwards with the corresponding executable component. This process is detailed in section 6.1.

4.2. Legacy applications

ISONI nodes are an addendum to existing networks. A lot of older systems are already deployed in those networks too. In many environments, e.g. telecom environments, it is

desirable to introduce new systems on a step by step basis, i.e. run a mix of new and existing systems.
Therefore the ISONI platform will be designed to allow the inclusion of legacy components in Virtual Service Networks.
In general at least two ways are possible to integrate legacy systems and their resources:

- Proxying. In some cases, e.g. IMS (IP Multimedia Subsystem, specified by www.3gpp.org) platforms, it is not feasible to migrate the existing service. Such services can be accessed through proxies allowing a controlled information exchange between VSNs and remote services. However, the ISONI cannot provide QoS beyond the proxy; if an SLA is required, it has to be negotiated conventionally.
- Integration. As the ISONI is able to provide nearly any kind of host environment as a VMU, many services can be "ISONIfied" by migrating them into a VMU, thus creating a (legacy) SC. If this SC's resource requirements are known, a QoS level can be guaranteed by the ISONI.

4.3. Native Service Components (SC)

Service Components (SC) are only fully manageable by the ISONI if four mandatory descriptive documents are provided. The collection of these four documents is the Metadata that is associated to the SC. Before a deployment can be initiated, the software itself has to be transferred to the ISONI in a format that is suitable for automatic deployment. The metadata documents associated with SC's are:

- An SC General Description (Which service does the specific service component offer?),
- An SC Interface Description (How is the service accessed?),
- An SC Parameterization (What are the limits/constraints of the component?),
- An SC Execution Environment Description (What special execution environments and resources, in detail computational resources, storage, operating system, and runtime environment are required?),
- And, optionally, runtime environment specific software.

The Service Component General Description contains a unique identifier for the SC in the form of a URI or name, a human readable description, vendor information, and a version tag. The Service Component Interface Description is equivalent to a WSDL [1] PortType definition. However, it is not limited to messages and protocols following the Web Services framework, but can also describe interfaces of non-WS protocols such as SIP [2], Diameter [3], etc.
A service developer does not have, and does not need, any knowledge of the availability and distribution of physical resources within the ISONI. He only specifies the resource types and quantities that are required for a concrete instance of a SC. It is the task of the ISONI to discover the required resources and to reserve and to bind them to the SC.
For more complex SCs that may be provided by the owner of an ISONI domain, mapping functions from high level requirements, e.g., number of simultaneous users, to low level requirements, e.g., amount of RAM, may be included in the SC Metadata. In this case, only these parameters need to be provided in the VSN description and the calculation of the actual resource requirements is performed by the ISONI.

The functionality characterized by a SC can be implemented/realized in different ways:

- SDP developer SW (3rd-Party SCs) running within a runtime-environment prepared VMU. Examples of such prepared runtime environments
  - An OS (e.g. Linux)
    plus an installed Java Runtime Environment (JRE) in case Java code is delivered.
  - An OS with specified installed packages like modules, libraries, services (e.g. http-server, Tomcat, …)
    plus Executable (e.g. C, Perl …)
- Special HW/SW, which exposes an interface, which can be used at runtime from other SCs
- Special HW/SW, which need to be adapted by an additional implementation (e.g. within a VMU) so that it becomes useable for other SCs.

An automatic provisioning and deployment process is provided by ISONI as described in chapter 6.

4.4. QoS

QoS guarantees at network level need to be complemented by real-time execution guarantees at the program execution level. In order for the system represented by a VSN to comply with real-time constrains as a whole, both network and execution resources,need to support QoS.

EE-QoS in ISONI will be supported at different levels:

- Processing QoS: Guarantees in terms of CPU speed will be given, which will be fed to the system by having the program designer fill out the SC metadata based on test bench execution results and the VSN designer tune this information according to the input data and use of the service to be deployed. A feedback on the employed resources will be given to further tune the required resources.
- Memory access: Processing guarantees are directly dependent of memory access times and therefore QoS constrains will also have to be supported.
- Storage access: As well as in the previous point, storage access QoS is crucial for the real-time execution of programs. This is even truer when working in media processing scenarios where the amount of data to be handled is very large.

In order to provide a complete, coherent and intuitive QoS-enabled execution environment, different classes of QoS bundles will be defined to support different types of applications. This bundle will take into account QoS basic constrains such as access times, data access bandwidth, CPU time, an so forth.

5. Networking concept

5.1. Network resources

The ISONI makes use of VPN like transport resources between ISONI nodes with predefined QoS properties (provided by Network Provider(s)), augments them with their own shaping and scheduling mechanisms, and assigns bandwidth of individual links to VSNs as per the QoS requirements provided in the VSN description. The ISONI manages an own set of QoS classes that are appropriately mapped to the QoS classes and types of the individual links. The ISONI is able to act in this way because it owns dedicated transport resources in differently managed IP networks.

5.1.1. Network

The networking layer is the ISONI part responsible for granting QoS on an ISONI level. It has to trigger path reservation (taking QoS demands into account) between VMUs. The critical part of path discovery and reservation is performed at the ISONI domain level. The network at the domain level is WAN-like. Thus, efficient methods of path building on domain level are essential for providing QoS by ISONI whereas local networks at node level should pose no problems.

5.1.2. QoS

ISONI provides link resources that are classified in several ISONI QoS classes. The classifications are ISONI internal and their definition will not be made public. The intention of the classification is to optimize the usage of the link resources owned by the ISONI. Within the VSN description a service creator describes the required properties of the links by giving parameter sets comprising bandwidth, delay, jitter, max loss rate etc. The ISONI will map those parameters into an ISONI internal QoS class that provides the required links with at least the QoS properties as requested by the VSN description. That process is exemplified in Figure 4 using only two parameters delay and jitter. If the parameter values can not be mapped on to ISONI QoS classes it could imply that the affected SCs have to be collocated on one physical host or even in one VMU. For example in Figure 4 there is a class of service, class 3, that guarantees 35ms jitter and 400ms delay, therefore a request for 32ms jitter and 380ms delay must be mapped to class 2 to ensure its needs are met. In this example system, any request for less than 10ms jitter or less than 60ms delay cannot be supported by any of the available classes and is tagged as “not supported” which means affectedSCs have to be collocated on one physical host or even in one VMU.

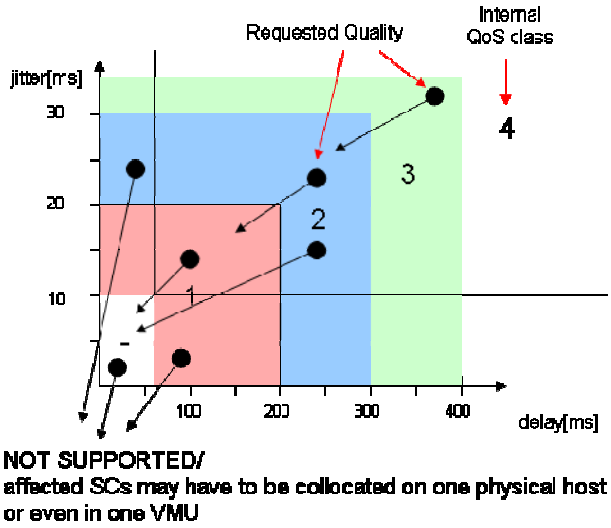

Figure 4 Mapping link parameter values on ISONI internal QoS classes

There are two main reasons for hiding the definition of QoS classes and to force service creators to describe the links in their VSN descriptions in terms of parameter values.
Firstly, experience shows that there will be no common definitions of QoS classes for all possible ISONI providers. Describing link properties in the VSN description in terms of QoS classes would then force service creators to generate different VSN descriptions for different ISONI providers.
Secondly, ISONI practice will probably lead to the necessity to redefine the QoS classes in order to improve resource usage within an ISONI domain. Relying on parameter values means that a VSN description needs not to be updated to mirror the new QoS class definitions as would otherwise be the case.
The definition and number of QoS classes may change in the future, but that does not change the principle in which those QoS classes are implemented by the ISONI.
Four example ISONI QoS classes for framework services and applications to start with may be the following:

- Best effort data (be-data) for data traffic without any QoS guarantee.
- Best effort data with minimum guaranteed bandwidth (be-prio) for data traffic, which needs minimum guaranteed connectivity, e.g. SIP signalling traffic among IMS applications
- Real-time voice (rt-voice) and
- Real-time video (rt-video) dealing with loss-tolerant media traffic. Rt-voice is treated by ISONI with higher priority than rt-video due to its lower bandwidth requirement and its smoother traffic characteristic.

Loss-intolerant real-time traffic is not regarded for the time being, but the ISONI QoS concept can be extended by adding additional ISONI QoS classes.
The ISONI provides link resources by adapting different kinds of transport network connections. The ISONI is able to adapt various transport interfaces, offering best effort, diffserv, intserv, MPLS, etc., and to share these link resources among the running SCs. To ensure QoS for various SCs common transport resources, the network traffic is scheduled according the scheduling strategy reflected by the ISONI QoS classes. In relation with the scheduling strategy of an ISONI domain the traffic is managed i.e. a

calculation in advance on how link capacity can be optimal used in relation to certain defined internal QoS classes available on this link. This will result in several possibilities of combination in order to maximize the usage of the capacity of link resources, which delivers the data traffic to the transport network in an adequate manner. This pre-scheduling functionality is located in the IXB.
The ISONI follows the principle of a managed network with respect to resources (link and processing resources). It customizes the link resources among nodes and inside nodes as well as the links to other domains. Instead of operating on individual streams, ISONI manages aggregated traffic on the virtual links to improve scalability. The IXBs perform traffic policing and shaping, thus avoiding performance impacts through greedy and/or defective applications and VSNs.
On the domain level, the ISONI has an aggregated view of available link resources between the nodes inside its domain and the available link resources towards other ISONI domains. This view represents the reported available link resources of an ISONI domain. It is used for managing the link resources and the selection of suitable nodes during the resource selection process in the VSN instantiation phase in which the ISONI selects "node sets" that meets the demands as defined in the VSN description. This selection process is complex and has to be performed observing, amongst others, the following aspects:

- available connectivity (considering the workload of the links at scheduled runtime)
- availability of ISONI QoS classes (at the scheduled runtime of the service)
- location/ propagation delay

The ISONI configures the IXB according to the policies in the VSNs' traffic profiles, e.g. bandwidth limitations, to perform policy enforcement.
All running SCs are monitored by their corresponding monitoring service at node level. When violations are detected, the ISONI triggers appropriate actions, e.g. informing the application service provider. (Live) Migrations are triggered to mitigate the effects of host failures.

5.2. Complete isolation

The service developer defines virtual addresses for all components in the VSN, these addresses are only valid in the given VSN. Effectively, every VSN provides an isolated namespace. The virtual addresses are mapped to physical locations by the ISONI as laid out in the following section. The ISONI is responsible for correct packet forwarding between the components along the defined interconnections of the VSN.
Access from the outside to the VSN is only possible through special components (in and out proxies).

5.3. ISONI Addressing Scheme

In section 3.2, we have introduced the concept of Virtual Service Networks and shown that there are no constraints on the L3 protocol used inside a VSN.

However, as one of the main goals of ISONI is to ease the integration of legacy components, i.e. components that are not aware that they are running in the context of a VSN, we have chosen to use IPv4 as L3 protocol inside the VSNs.
The VSNs are fully isolated; thus, from the global point of view, a VSN effectively is a namespace that is added to the VSN-L3-addressing.
The architecture also allows for IPv6, but, as most legacy applications still use IPv4 only, we have opted not to follow IPv6 in the implementations.
ISONI uses the combination of VSN-Identification (i.e. namespace) and VSN-internal address to determine the physical host that a SC is executed on and provide a virtual point-to-point link between source and target SC. This functionality is included in the ISONI eXchange Box (IXB) as laid out in section 5.3.1.

5.3.1. ISONI eXchange Box (IXB)

All virtual addresses used by VSN components are interpreted by the ISONI's component IXB (ISONI eXchange Box), which is responsible for connectivity and QoS enforcement. The IXB realizes the virtual network topologies for the services in the form of overlays and prevent crosstalk between them. Exactly one IXB is located at the edge of every node.
A node's IXB is configured by the ISONI and establishes VSN specific tunnels towards other nodes' IXBs. Effectively, the VSNs are spanned between the IXBs that perform routing, encapsulation, and decapsulation functions on the messages and data streams sent between the SCs. The IXB performs mapping of VSN virtual addresses of SCs to the physical addresses of the resources (see illustration in Figure 5). Through this mapping, the physical topology of the network and the resource distribution is completely hidden from the ISONI developer, as are as the services running on ISONI. This concept also ensures VSNs are completely disconnected, i.e. it prevents unauthorized information flows across VSN boundaries. It also prevents unwanted access to the VMUs from outside networks such as the Internet.
Within a VSN, each and every required communication link must be specified. As only such defined communication links are established by the IXBs, no communication event between two components can occur if it is not foreseen by the developer in the course of VSN creation.

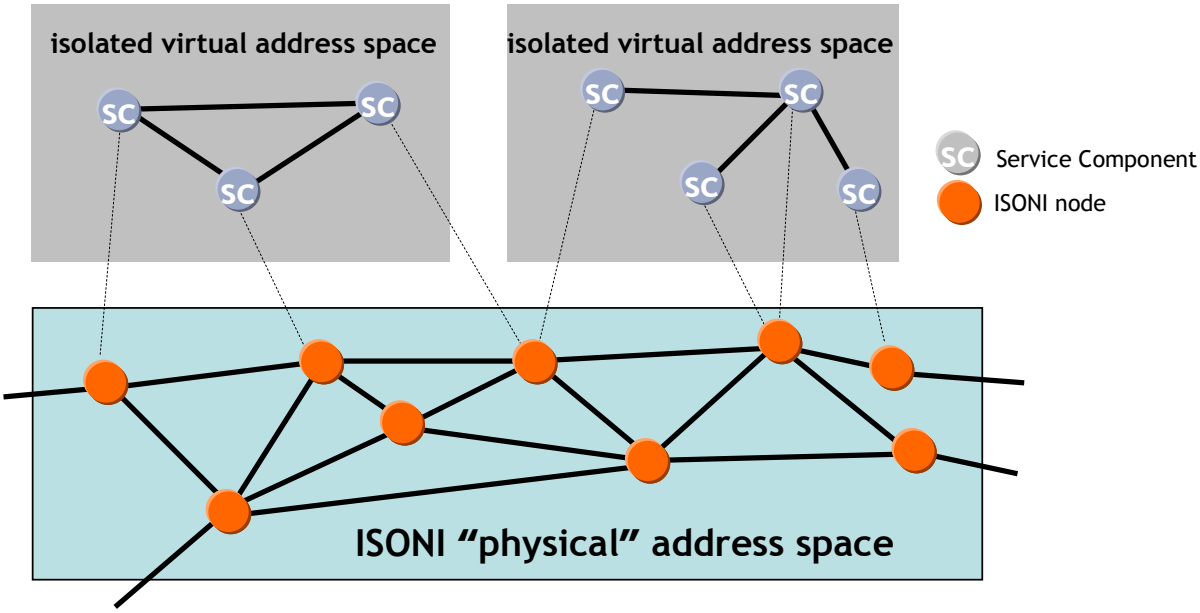

Figure 5 VSN, mapping of virtual to physical addresses

6. Service Component Lifecycle in ISONI

This chapter describes the lifecycle of dynamic ISONI Service Components, i.e. 3rd party software that is deployed and instantiated on demand.
First, the steps necessary to prepare the Service Component for execution are described, i.e. the scheduling, provisioning, and deployment of a SC.
Second, the SC's startup and runtime are presented. Finally, the cleanup process is outlined.

6.1. Automatic provisioning

The VMU image creation will be performed as an offline process on dedicated 'factories'. Working with VMU templates in the form of pre-created basic VMU images will considerably accelerate the process. At runtime, the ISONI autonomously deploys the prepared VMU images to selected physical hosts.
As stated in chapter 4.1, SCs basically are (3rd party) software that is executed on a tailored operating system. This implies that before a Service Component can be made available in the VSN, a VMU with the required operating system needs to be created and the SC's software needs to be installed.
A fundamental requirement to automatic provisioning is that the process must be completed and the VMU must be available on the physical node that is to execute the VMU before the scheduled VSN creation time is reached. Due to the installation processes, the VMU creation is an expensive process. This conflicts with the normal operation of the ISONI physical hosts, as the required resources are potentially allocated to other SCs: When only spare resources are allocated to the VM creation, the creation process can potentially take a long time through high resource usage for other applications, it may also be hard to schedule.
This issue is addressed by ISONI by using dedicated nodes for VMU creation. These nodes are called *VMUFactories* in ISONI terminology.
These VMUFactories perform preparation of the operating system required by the SC, either by customizing OS templates or by starting an automated installation. Afterwards, the SC's software is integrated into the VMU. To support this, the SCs are packaged to support unattended installation.
As the startup phase of the VMU, which includes operating system boot plus the initialisation time of the SC's software, is also an expensive and time consuming process, it is also carried out on the VMUFactories. After startup is completed, the Service Component enters a hold state and the VMU is put into hibernation, using power save mechanisms of the OS. The hibernated VMU is transferred to a *SC repository* where the provisioned Service Component is stored until execution time.

6.2. Automatic deployment

Well in advance of the required VSN startup time, the SC has been prepared as laid out in the previous section.
Shortly before the VSN startup time, the actual physical host that the SC will be executed on is chosen and the VMU is transferred from the repository to this host by the *Deployment Manager*. This allows for maximum flexibility as only running VMUs have to be considered when physical hosts have to be taken offline for maintenance or failures have to be compensated for. The VMUs are loaded into memory and released from hibernation state just before VSN startup time, thus minimizing the time window in which resources on a physical node cannot be sold to customers but are bound to ISONI-internal support actions. At VSN startup time, the SC is released from its hold state.

6.3. Service runtime actions

The trigger for starting and stopping the service may be given either externally by the service user, or automatically using criteria that are specified in the VSN. (Automatic start may be necessary in cases such as if the deployed service is part of a bigger workflow scenario and has to wait for proper parameters to be delivered by external services in order to fulfil its tasks.)
During runtime of the service, its resource usage will be. This monitoring will not be performed to check whether or not the service is trying to use more resources than requested. This will be enforced by scheduling mechanisms in case of CPU resources and by scheduling and shaping mechanisms in case of transport resources. The amount of storage available to the service is fixed too. The monitoring data are used for proving that the ISONI correctly provided the requested resource capacities.
This monitoring data will be made available to the ISONI customer (e.g. a service developer) to enable him to detect misbehaviour of the service. If predefined levels of resource usage are systematically exceeded during the service runtime a warning message for the ISONI customer will be generated. The monitoring data may also be used for re-dimensioning of the resource requests of follow up service instantiations (new VSN requests).

6.4. Service clearing

After runtime the VMU instances of the service will be deleted on the hosts and accounting and billing data will be produced. Some services need data generated during the lifetime of the service to be persistently stored after runtime, the ISONI takes care of this.

7. Conclusion

In this paper we presented ISONI, a framework allowing resource sharing among multiple services. We introduced the service concept of ISONI. Virtual Service Networks and Service Components are defined to enable reuse of components and even whole services, and to support the Service Oriented Architecture (SOA) approach. We described the general architecture of ISONI. It is designed to ensure maximum scalability for an ever growing number of resources and services.

Real-time services (e.g. conversational (multi-media) services, real-time multimedia streaming services, etc.) are our focus of interest. That is why, compared to others in the Grid community so far, we paid a lot more attention to the QoS properties of the connectivity of distributed resources. Properties of the network have a to be taken into consideration during resource and service discovery. The IXB has been introduced to provide this functionality and enforce given network QoS guarantees.

Security is an important topic that is implicitly dealt with, e.g. with sandboxed environments, traffic isolation, etc. Due to its complex nature, this topic will be addressed in a dedicated paper at a later stage of the project.

Future work in IRMOS will focus on conceptual work and proof of concept implementation of essential parts of ISONI like Execution Environment, Resource Manager, Path Manager and IXB. Future expansion stages will also allow necessary resource re-arrangements and (re-)allocations for running services that adapt better to changing resource availability.

Virtualization technologies have long been a part of the IT world with great payoffs. Initial experiments in a prototypical lab environment have been carried out using XenTM [4] technology. This has provided very promising results. We do not restrict ourselves to a Specific technology though, as the framework itself is independent of the actual virtualization layer used. Indeed, investigations in the use of KVM [5] as virtualization layer are under way. From that we conclude that the telecommunication world can profit immensely by introducing virtualization technologies. That is the logical continuation of the work on convergence of technologies and services observed over the last years. We consider ISONI to be a promising step in that direction.

Finally, there has to be mentioned that the outcome of our continuous research on ISONI within the IRMOS Project will be most likely included in a future updated version of this whitepaper.

8. References

[1] Jean-Jacques Moreau, Sanjiva Weerawarana, Roberto Chinnici, and Arthur Ryman. Web Services Description Language (**WSDL**) Version 2.0 Part 1: Core language. W3C Recommendation, W3C, 2007.

[2] Jonathan Rosenberg et al. SIP: Session Initiation Protocol. RFC 3261, IETF, June 2002.

[3] Pat R. Calhoun et al. Diameter base protocol. RFC 3588, IETF, September 2003.

[4] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles, pages 164{177, New York, NY, USA, 2003. ACM.

[5] http://kvm.qumranet.com/kvmwiki