# Component-based system for management of multilevel virtualization of networking resources

Robert Boczek Dawid Ciepliński

August 11, 2011

# Contents

# Chapter 1

# Introduction

In today's world every successful organisation is based on properly designed communication network. These networks must deal with delay-sensitive data such as video images, real-time voice or mission-critical data. Therefore must provide safe, predicatable and sometimes guaranteed services. Accomplishing the required Quality of Sevice(QoS) by controlling the delay, delay variation(jitter), bandwidth, packet loss parameters is deeply hidden secret of most successful end-to-end business applications. http://www.cisco.com/en/US/products/ps6558/products_ios_technology_home.html

Due to raising concern and importance of these issues in these paper we decided to have more insight into one of possible approaches to this matter which is Solaris OS and the Crossbow technology.

# Chapter 2

# Context

Constantly growing demand for bandwidth in networks (especially VoD, VoIP, RT) raises the question: 'Whether it is better to increase available bandwidth of networks or to build intelligent systems managing users traffic?'. As high bandwidth is just not enough because there are other equally important transmission parameters (delay, jitter, package missing tolerance) there seems to be just one correct answer for this question. Creating such systems is not an easy task and many groups such as IETF (The Internet Engineering Task Force) are working on this problem. Currently there are three existing models performing QoS in the IP network:

1. best effort,
2. Intserv,
3. Diffserv.

These models are discussed in more details in the following section.

## 2.1 QoS-aware networking

QoS (Quality of Service) is an issue in all types of networks such as IP, Token Ring, Frame Relay or even ATM. Each of them adapted specific approach towards this matter. IP network for instance is non-deterministic although there is a possiblity of packet classification using CoS (Class of Service) field. Token Ring on the other hand is deterministic and allows using priority to distinct traffic. Last but not least ATM creates virtual path between sender and receiver and sets QoS parameters. Despite the fact that all approaches are very interesting and meaningful this paper focuses mainly on the IP network and its approach to QoS.

Nowadays the IETF (The Internet Engineering Task Force) is working on two approaches beyond the basic best-effort service to provide more advanced handling of packets and providing requested level of bandwidth and delay which are:

1. integrated services — reserves resources necessary to provide the service along path,
2. differentiated services — do not require resource reservation along path

### 2.1.1 DiffServ

Due to clear need for relatively simple and coarse methods of providing differentiated classes of service for Internet traffic, to support various types of applications, and specific business requirements. The differentiated service approach to providing quality of service in networks employs a small, well-defined set of building blocks from which a variety of aggregate behaviors may be built. [?]

DiffServ works with traffic stream containing many complex microflows which among the same stream have:

- Same QoS requirements,
- Traverses domain towards the same direction.

Microflow is a flow between applications and is identified mainly by:

1. Source and/or Destination Address,
2. Transport protocol,
3. Source and/or Destination Port.

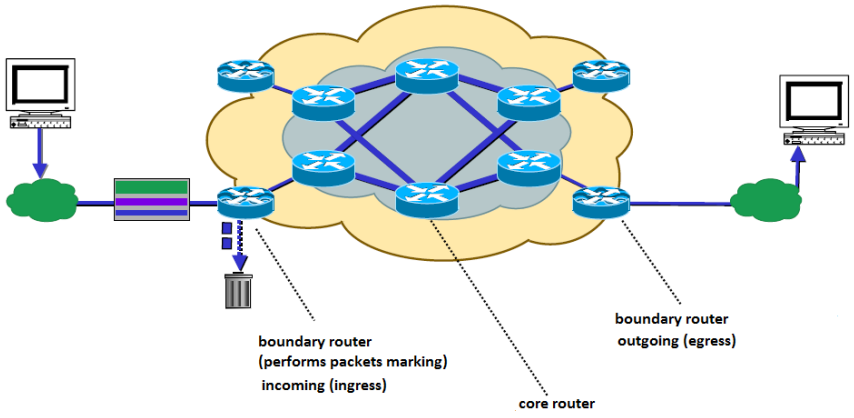


Figure 2.1: DiffServ domain

DiffServ domain is a set of nodes with consistent QoS policy (for example network managed by the same ISP or intranet). Routers within domain are didvided into two groups: boundary routers and core routers. Boundary router performs traffic classification, packet marking, traffic metering, traffic control (policing, shaping) where core routers pass packets according to PHB(Per Hop Behaviour - more about them later) and sometimes changes DSCP labels.

This architecture distinguishes two major components: packet marking using the IPv4 ToS byte or TC in IPv6 and PHBs(Per Hop Behaviour). Redefined ToS field now uses 6 bits for packet classification and is called Differentiated Services Codepoint(DSCP). PHB defines how packet should be treated due to its priority. Accepted policy should be consistent within all domain.

### 2.1.2 IntServ

Integrated services beyond the basic best-effort service defined in two RFCs provide two levels of service for IP which are called Controlled-Load Service and Guaranteed Service. Both require information about the traffic to be generated.

Guaranteed Service type should be used in applications with real time demands as it provides guaranteed bandwidth and delay, whereas Controlled-Load Service does not give full guarantee and should be used with application less sensitive for packets loss or delay.

IntServ requires resource reservation for certain flows or aggregated data streams. The reservation operation is available thanks to RSVP protocol. Due to this reservation session initialization lasts much longer than in DiffServ.

DiffServ vs IntServ

Table 2.1: DiffServ, IntServ comparison

| IntServ | DiffServ |
|---|---|
| Stateful | Stateless |
| Not scalable | Scalable |
| Stream oriented | Processing single packets |

## 2.2 Resource virtualization approaches

## 2.3 Multilevel network virtualization

### 2.3.1 Virtual network resources

### 2.3.2 Fine-grained QoS control

### 2.3.3 Virtual appliances

### 2.3.4 „Network in a box" concept

## 2.4 Applications and benefits of virtual infrastructures

### 2.4.1 Testing and simulations

### 2.4.2 Improving server-side infrastructure scalability

### 2.4.3 Infrastructure as a service

The IAAS( Infrastructure as a service ) sometimes also called Hardware as a Service (HaaS) is one of three cloud computing models, the other two are: Software as a Service (SaaS) and Platform as a Service (PaaS). This service is based on providing by the supplier whole scalable IT infrastructure depending on user demand such as virtualized hardware. The service provider owns the equipment and is responsible for housing, running and maintaining it.

At the beginning the IaaS was just renting dedicated servers services from supplier. Nowadays thanks to virtualization these are most often virtual machines. In the former case user paid for the concrete hardware (box), now client typically pays on a per-use basis.

### 2.4.4 The role of resource virtualization in the SOA stack

## Summary

# Chapter 3

# Requirements analysis

This chapter focuses mainly on earlier conducted requirements analysis and extracted conclusions. Established goal was to create system supporting creation of any requested, fully isolated network structure with virtual network elements and demanded resources recovered from selected (previously prepared) repository and fully monitored network traffic. This target implicated necessity of creating functional and non-functional specification in order to avoid ambiguities and skipping essential services. All these issues are described and considered in the first two chapters.

Section 3.1 presents requested behaviors (functions or services) of the system that support user goals, tasks or activities.

Section 3.2 describes adopted non-functional requirements towards newly created system.

## 3.1 Functional requirements

All services of the system have been divided into three groups: accounting, discovery and instantiation. These division have been performed based on the quality of the functionality.

### 3.1.1 Instantiation

Most of functionalities neccessary during process of network instantiation ( also alteration and deletion ) would be:

- Creation of any requested virtual network element,
- Creation of any requested resource from repository,
- Assignation any number of links to resource,
- Resource routing table modification,
- Defining QoS limits to links and flows ( limits traffic assigned to specific network traffic type ),
- Modification or deletion of each property and element previously mentioned.

### 3.1.2 Discovery

During the process of discovery following functionalities should be accessible:

- Discovery and assembly of links, resources from the same project,
- Detection of links assigned to a specified resource,
- Discovery of flows attached to a link.

### 3.1.3 Accounting

In terms of accounting, following functionalities for properties and elements should be provided:

- Monitoring of each link's bandwidth,
- Checking each link's, flow's network traffic ( input / output ) statistics,
- Displaying network traffic load chart of selected link or flow from specified time period.

## 3.2 Non-functional requirements

Non-functional requirements also called **qualities** of a system could be distincted between the executing system and the work products ( created during system development ). Executing system are related to user goals and are refered as run-time qualities, whereas work products are driven by the development organization's goals and refered as development-time qualities [**?**].

Requested run-time qualities:

- usability - usage should be intuitive, system should be efficient,
- operational scalability - system should support additional users or sites,
- configurability - configurable properties should be easy to set,
- adaptable - ability to adapt to changing conditions ( adding removing new nodes ),
- fault tolerance - system should be resistant for minor erros.

Requested development-time qualities:

- evolvability - support for new functionalities or adjustement to new technologies,
- extensibility - ability to add new earlier unspecified functionalities,
- composability - system should be created in form of composable components,
- reusability - ability to (re)use in future systems.

## 3.3 Underlying environment characteristics

Underlying environment should be a composition of a fully independent nodes. However, a few requisites must be fulfilled by each node:

- Each OS should have the crossbow functionality provided,
- Network accessibility between nodes must be ensured.

## 3.4 General approach and problems it imposes

Selected approach towards problem was

### 3.4.1 Load balancing / Deployment

Although at first load balancing was planned, created system does not support it. This matter is further discussed in **'Further work'** section in the **Summary** chapter.

In terms of deployment, potential problems are easy to indicate. Use of Jims functionality and related JMX features stresses the lack of transactional support. Although in our system in case of errors introduced changes are usually removed, there is no guarantee that it will acutally happen. Due to possible further errors caused during restoring previous system state. In that case these partial changes must be removed manually from each node involved in this failing deployment attempt which generally imposes specialistic knowledge of underlying environment.

### 3.4.2 Infrastructure isolation

Network isolation is provided thanks to the Crossbow functionality which is more accuratly described in the chapter about Solaris OS features.

### 3.4.3 Broadcast domain preservation

### 3.4.4 Constraints

# Chapter 4

# Solaris 10, Solaris 11 and OpenSolaris

The chapter provides an overview of Oracle Solaris operating system and evaluates it as a platform for resource virtualization. The chapter describes Solaris 11 Express release of the system, as it is the first release (together with OpenSolaris) with Crossbow technology integrated. Special emphasis is put on the networking-related aspects of virtualization. Thus, the Solaris Crossbow technology is described in detail.

Section 4.1 contains introductory information about the system. A short historical note is presented and general description follows. Main components of the system are introduced and described.

Each of the remaining sections describe in more detail these parts of the operating system that are extensively used by the implemented system. Section 4.2 investigates the Solaris Zones technology. After defining the concept of zones, zone lifecycle model is presented, the achieved level of process isolation is described and discussion of Zones advantages in comparison to non-virtualized environments follows.

Section 4.3 introduces Solaris Crossbow - lightweight network virtualization environment. The section starts with general description of the technology. Next, components crucial to efficiency improvement are presented in detail. Etherstubs, VNICs and flows are described. These are building blocks used to create virtualized network elements and apply QoS policies. The section ends with the comparison between Crossbow and DiffServ and a method of integration of these two solutions is presented.

Section 4.4 provides an overview of resource control methods offered by the Solaris OS. The types of resource management mechanisms (constraints, partitioning and scheduling) are identified and defined. Resource control hierarchy used by the system is depicted and explained. Also, the accounting facility is described. The types of resources extended accounting can work with are enumerated and examples of data that can be gathered are listed.

## 4.1 General information

Oracle Solaris is a *multiuser, multitasking, multithreading UNIX-like operating system* [**?**]. Since its release in 1992 (as Sun Solaris 1), the system became one of the most popular environments supporting enterprise software. Nowadays, big corporations and companies as well as individual developers use it to do their business and deliver reliable and scalable services.

The Solaris OS provides unique set of tools that support virtualization of practically all types

of resources at various levels. There is Logical Domains (LDOMs) technology for full virtualization and lightweight Zones, when all that is needed is the isolation of processes. Logical domains can be connected with complex virtual networks that are created with virtual switches (vsw) and virtual network devices (vnet) [?] and Crossbow can be used to enable lightweight and efficient networking for zones, exploiting capabilities of underlying hardware layer (network interface cards with virtualization level 1, 2 or 3 [?]).

Resource utilization can be managed with integrated administration tools. Resource access policies can be created with high level of granularity (per-process resource control) as well as in more general way (limiting resource access for LDOMs). Resource consumption can be subject of monitoring and accounting. With extended accounting subsystem enabled, it is possible to capture detailed accounting data even for single processes. Gathered data include CPU usage, number of bytes received or transmitted per DiffServ or Crossbow flow and more.

As far as multiple physical machines are considered, there is also support for VLANs (Virtual Local Area Network). Thanks to VLAN tagging support, it is possible to build systems that guarantee the quality of service from the lowest levels up, even for services belonging to different systems and consolidated within single physical machine.

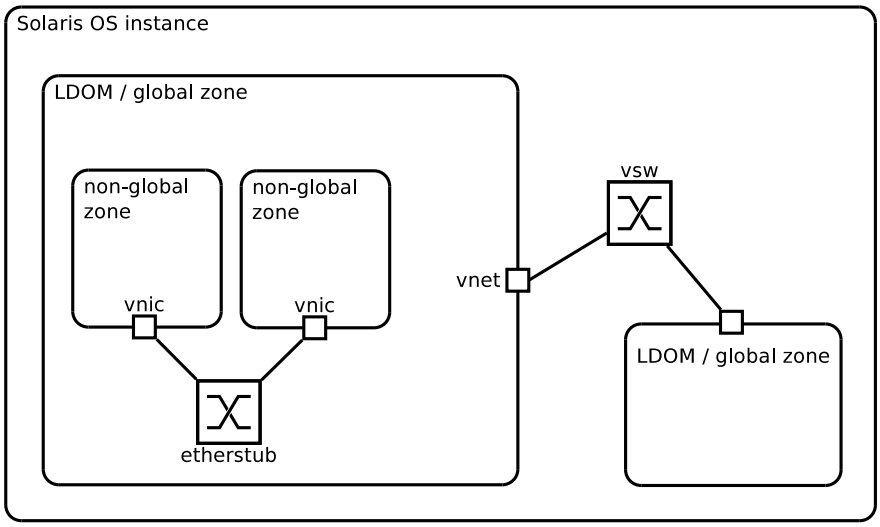


Figure 4.1: The variety of resources that can be virtualized with Solaris OS

As it can be seen, the Solaris operating system is accompanied by vast variety of virtualization-supporting subsystems. This multiplicity and flexibility makes it a promising platform for service provisioning and building even more abstract architectures on top of it. The following sections describe selected aspects of the system in more detail.

## 4.2 OS-level virtualization with Solaris Containers

The concept of lightweight (OS-level) virtualization is supported by most modern operating systems. The solutions are either integrated into the system's kernel and accessible as soon as it is installed (Solaris Containers, AIX Workload partitions, BSD jails [?]) or are provided by third-party

manufacturers as kernel patches and utility software (OpenVZ and LXC for Linux OS). Because of awareness of other system components and integration with them, it can be expected that Zones have more potential than other virtualization methods.

### 4.2.1 General information

Zones technology was introduced as of Solaris OS 10. It provides a way of partitioning system resources and allows for isolated and secure application execution environment [?]. Solaris Zones, together with resource management functionality, constitute the Solaris Container environment.

There are two types of zones: global and non-global. Global zone is the default one and is used to execute applications as well as to administer the system. Non-global zones can be created from within the global zone only. A single operating system instance with one global zone can host as many as 8192 non-global zones [?].

Zones can be assigned system resources such as CPU capacity, the amount of random-access memory or even maximum number of lightweight processes that can be running simultaneously. Also, network isolation is supported at two levels: basic, at the IP layer, and network isolation and advanced virtualization with fine grained quality of service control using the Crossbow technology.

Each zone can run a different set of applications, with optional translation of system calls (Branded Zones Technology) thus emulating different operating environments [?]. The user is able to create a branded zone with translation of Linux system calls and run Linux-specific applications without code recompilation.

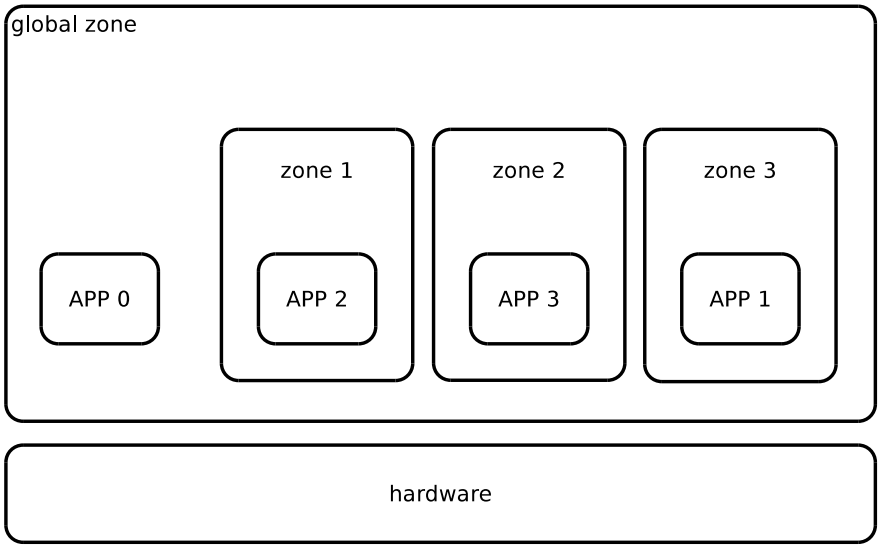


Figure 4.2: Solaris Zones high-level view

### 4.2.2 Zone lifecycle

A model was created to describe the states in which each zone must exist and its possible transitions. A non-global zone can be in one of six states: *configured*, *incomplete*, *installed*, *ready*, *running*, *shutting down or down* [?]. Figure 4.3 depicts the model.

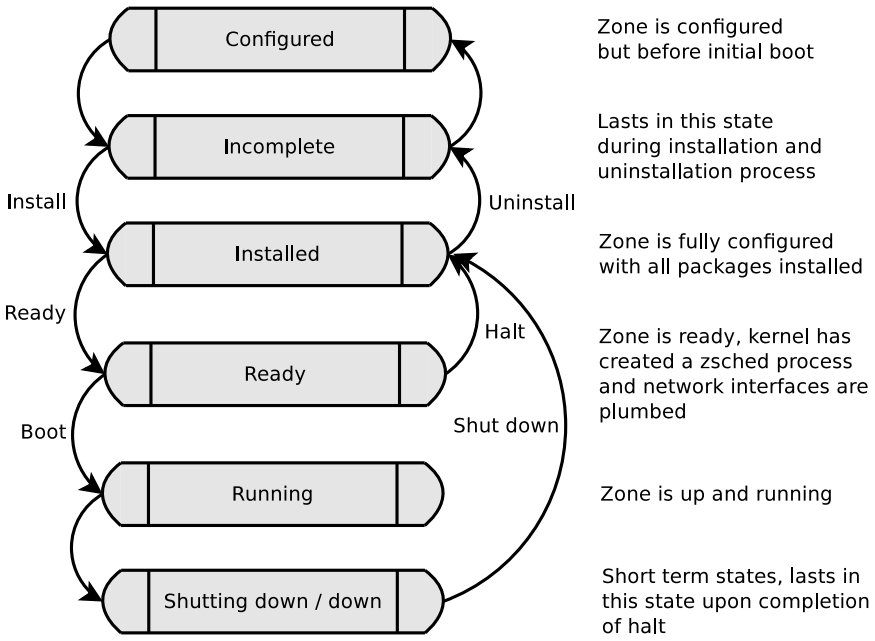


Figure 4.3: Zone states and possible transitions

### 4.2.3 Isolation of processes

The Containers environment offers a high level of application security and isolation. This is accomplished by imposing software bounds on the resource usage and introduction of additional abstraction layer over hardware.

Every process and its children are bound to concrete zone and the assignment cannot be changed. Moreover, it is impossible for processes in distinct zones to monitor each other operation. They are not visible to each other and no interprocess communication can take place, except for network-based one, if enabled by the administrator.

Because of the isolation, an application failure possibly affects only the processes in the containing zone. Assuming no interaction between processes in separate zones, the rest of the system remains intact and can operate normally.

### 4.2.4 Advantages of Containers technology when compared to non-virtualized environments

The architecture of Solaris Containers makes it a competitive solution as far as systems administration and operation efficiency is concerned. The technology, imposing negligible overhead [**?**], allows to perform tasks that would be impossible or very hard to accomplish if traditional setup is used. Examples of such tasks include dynamic resource assignment, instantaneous cloning and migration of systems between physical nodes.

The technology allows for running a number of isolated instances of operating system sharing CPU time, physical network bandwidth, filesystem contents and binary code. Sharing of these resources can greatly improve overall system efficiency and reduce the amount of occupied memory.

The speed of network communication between different zones can also be improved thanks to „short-circuited" traffic (ie. omitting the layers below IP in the OSI/ISO stack). The instances are able to execute applications with minimum overhead introduced mainly due to accessing commands and libraries through the lofs (loopback filesystem) [?, ?].

When using file system that supports snapshots (as, for example, ZFS), zones can be serialized (a snapshot of the file system can be taken) and sent over the network connection or other means of data transfer to another machine. There, the zone can be restored and operate as a part of the host system.

Another important aspect of building the infrastructure with containers is resource control. The Solaris system makes it possible to define resource controls (rctls) at various levels, also on per-zone basis. CPU shares, maximum number of lightweight processes and maximum swap size are examples of resource control properties that can be set for a zone. This can be further extended by providing fine-grained properties at project, task and process levels [?]. The resource control process is dynamic - assignments can be changed as the system is running, without interrupting the container's normal operation. This can be of extreme importance as far as high-availability systems are considered.

Containers facilitate service consolidation - all components of a system can be executed in a single machine with network-based communication handled entirely by the host operating system, thus eliminating the need for additional networking hardware and its management. The consolidated infrastructure becomes more flexible as the majority of administration tasks can be performed by issuing a series of terminal commands. All these factors make total cost of ownership lower [?].

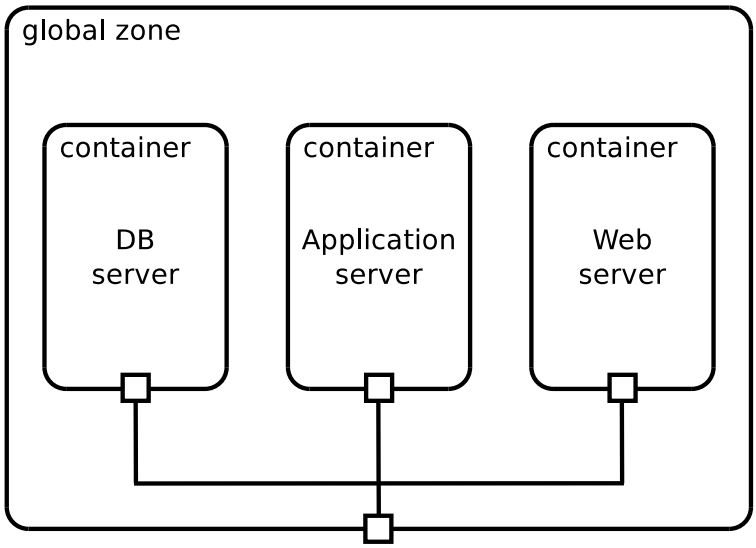


Figure 4.4: Service consolidation within a Solaris OS instance with internal network connectivity

### 4.2.5 Virtual appliances

Virtual appliance is a *pre-built, pre-configured, ready-to-run (enterprise) application packaged along with an optimized operating system inside a virtual machine* [?]. Solaris Zones, together with other components of the Solaris OS, constitute a complete framework that implements virtual appliance

approach to systems management.

The main problem virtual appliances can solve is the complexity and duration of application deployment process. In general, a service deployment can be described as comprising the following stages: preparation (learning the dependencies), pre-installation, installation and post-installation. With traditional (non-virtualized) approach, these stages have to be repeated every time a service is deployed on different machines.

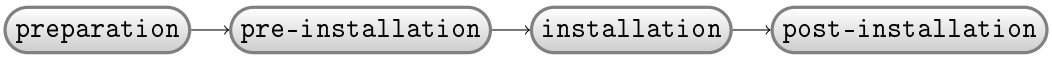


Figure 4.5: Traditional application deployment stages.

Virtual appliance approach makes it possible to reduce deployment time significantly [**?**]. This is achieved by performing most of the deployment stages once and storing the configured environment in a virtual appliance. The appliance can then be moved to publicly-available repository for actual deployment on host systems.

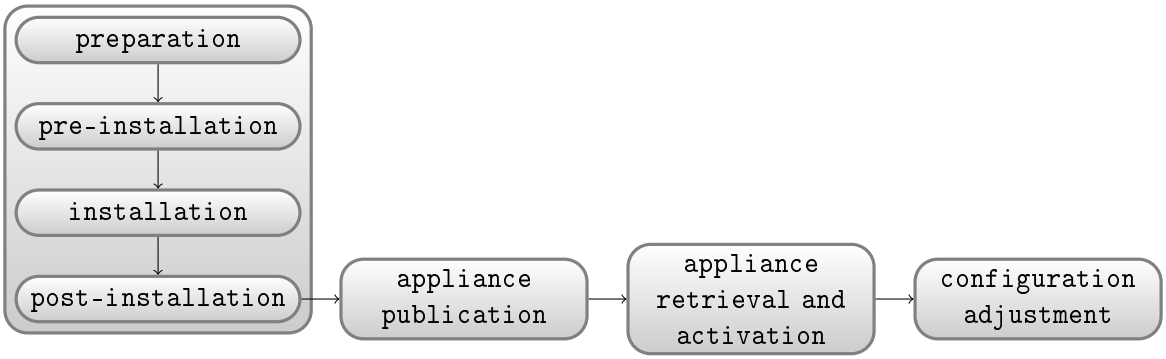


Figure 4.6: Deployment process with virtual appliances. Stage 1 is executed once.

It is possible to prepare sets of virtual appliances containing traditional services (such as application servers, database servers or media servers) as well as highly specialized networking-focused appliances that can act as routers, firewalls or load balancers. These Virtual (Network) Appliances, together with other components provided by Solaris OS, can be leveraged to build fully virtual network topologies.

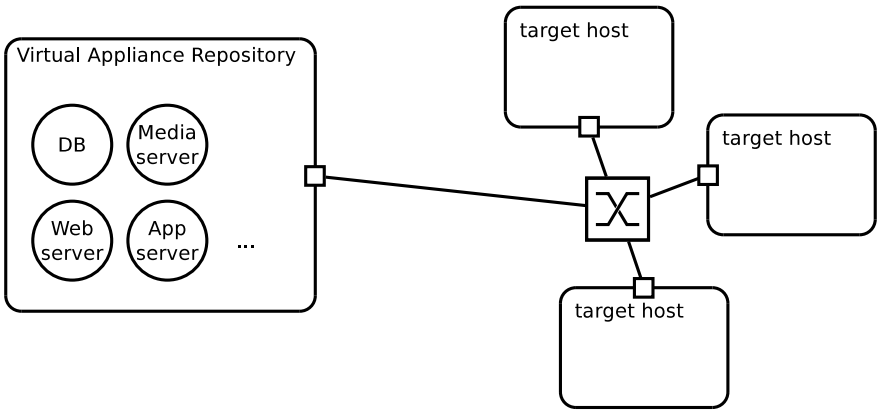


Figure 4.7: An example of infrastructure utilizing virtual appliances with appliance repository.

## 4.3 Crossbow - network virtualization technology

It is generally acknowledged that Crossbow was invented in China in 341 B.C but it was in middle ages when it earned its recognition. Very easy in use and simultaneously very effective. The Solaris Crossbow mechanism for QoS are just like real crossbows, very efficient in comparison to other existing QoS mechanisms and this similarity indicates the project name origin.

### 4.3.1 Crossbow architecture

One of the most important conditions in terms of network virtualization is that network traffic should be insulated between virtual machines. This kind of isolation can be achieved by having a dedicated physical NIC, network cable and port from the switch to the virtual machine itself. Moreover, switch must also ensure sustainability on every port. Otherwise, virtual machines will definitely interfere with each other [**?**].

In a particular case when a physical NIC has to be shared between virtual machines the most promising solution is to virtualize NIC hardware and the second layer of the OSI/ISO stack where sharing is fair and interference will be avoided. These approach was adapted in the Crossbow architecture in the Solaris OS [**?**].

Traffic separation is achieved with fundamental blocks of new architecture which are Virtual NICs (VNICs) created by partitioning physical NIC. A VNIC can be created over NIC or Etherstub and be dynamically controlled by the bandwidth and CPU resources assigned to it [**?**, **?**]. New architecture after introducing new networking features combined with existing features like Solaris Containers, resource control can be presented as following:

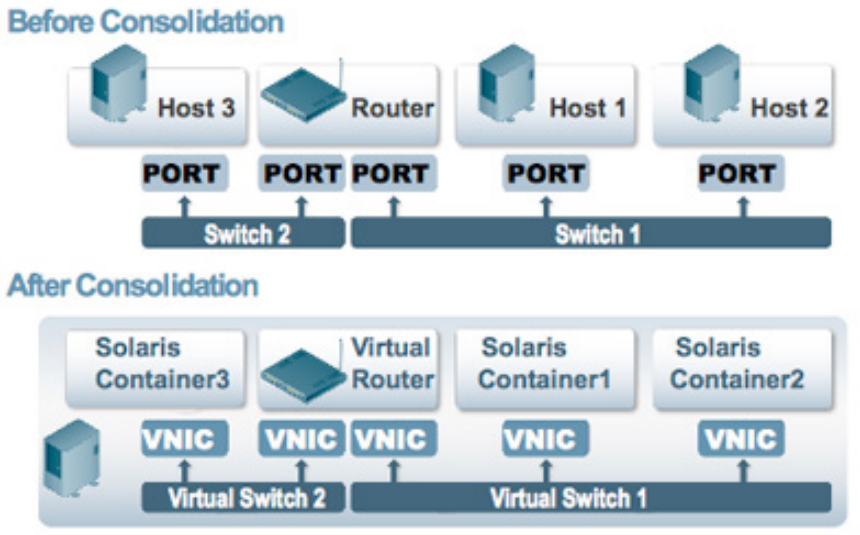


Figure 4.8: The Solaris Crossbow network virtualization enhancement, source: http://www.net-security.org/images/articles/crossbow.jpg

The crossbow architecture has introduced fully parallel network stack structure. Each stack could be seen as an independent lane (without any shared locks, queues, and CPUs) therefore network isolation is guaranteed. Key concept is hardware classification performed by the NIC over which VNIC was created. Each lane has a dedicated buffer for Transmit (Tx) and Receive (Rx) ring. In case when load exceeds assigned limit packets must be dropped as it is wiser to drop them than to expend OS CPU resources [?].

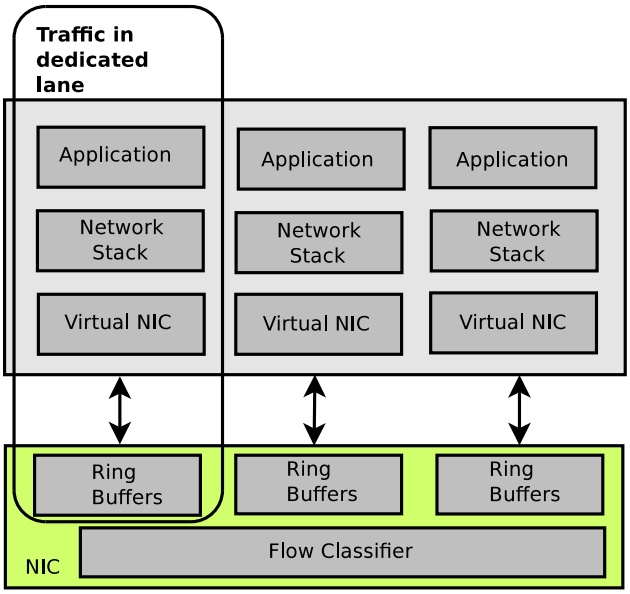


Figure 4.9: Dedicated lanes in the Crossbow architecture

### 4.3.2 Virtualization lanes

Virtualization lane is the most key component in the Crossbow architecture. Each lane consists of some dedicated hardware and software that might be used to handle specific type of traffic. It usually would be composed of:

1. NIC resources (receive and transmit rings, interrupts, MAC address slots),
2. Driver resources (DMA bindings),
3. MAC layer resources (data structures, execution threads, locks).

A virtualization lane can be one of two types, hardware-based or software-based.

#### Hardware-based virtualization lanes

This type requires ability to partitioning resources from NIC. The minimum requirement is that a hardware-based lane should must have a dedicated receive ring. Other resources such as transmit lane can be exclusive or shared between lanes. Each virtual machine could have one or more lanes assigned and the incoming packets would be distributed among them based on even scheduling unless some administrative polices where created, such as priority or bandwidth limit [**?**].

#### Software-based virtualization lanes

In case when NIC runs out of hardware-based virtualization lane, receive and transmit rings may be shared by multiple VNICs. The number of software-based virtualization lanes also often called softrings is unlimited. The main disadvantage of software-based lanes is the lack of fairness and isolation which in fact is provided in hardware-based lanes. The received and sent rings may work also in mix mode, whereas some of the rings may be assigned to software and some may be assigned to hardware based lanes [**?**].

### 4.3.3 Dynamic polling

The Crossbow architecture proposed two types of working modes. Currently used mode is determined by traffic and load. Under low load, where the rate of arriving packets is lower than time of packet processing, a lane works in the interrupt mode which means that receive ring generates an interrupt when new packet arrives. However, when the backlog grows, the line switches to dynamic polling mode in which a kernel thread goes down to the receive ring in the NIC hardware to extract all outstanding packets in a single chain. Key aspect is that every virtualization lane works independently and transparently from each other. Usually only three threads are used per lane [**?**]:

1. Poll thread which goes to the NIC hardware to get all packet chain,
2. Worker thread which is responsible for protocol processing (IP and above) or delivers packets to virtual machine. Thread performs also any additional transmit work which is a natural requirement some concrete protocol, such as processing TCP packets that require sending ACK packets,
3. Transmit thread that is activated when if packets are being sent after transmit side flow control relief discharge, or after retrieving transmit descriptor. Application or virtual machine can transmit any packets without performing queuing because of flow control or context switching.

### 4.3.4 Virtual switching

Virtual switches are always created implicitly when the first VNIC is defined under existing NIC and could never be accessed directly nor be visible by any user (even administrator) [?].

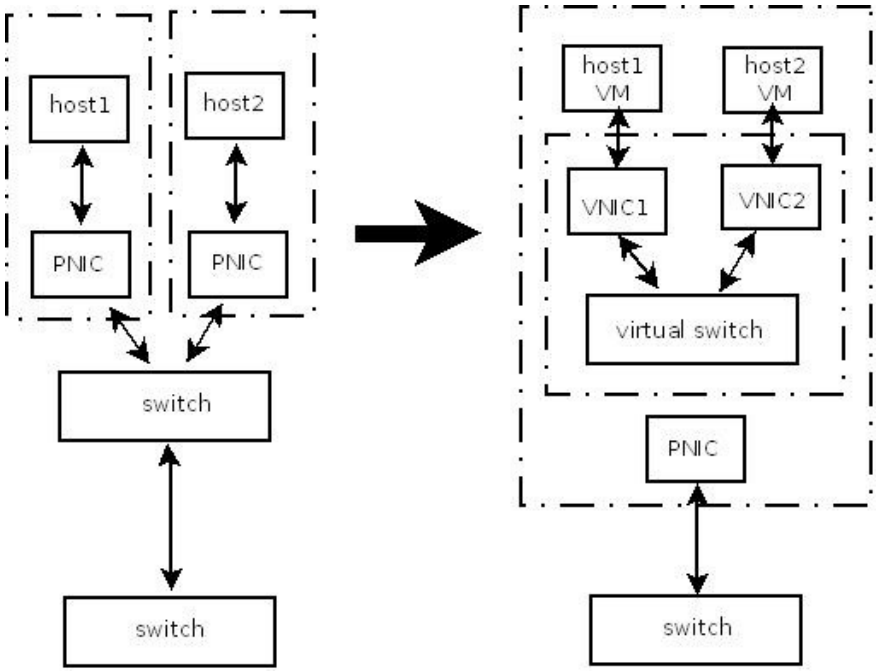


Figure 4.10: Mapping between physical and virtual network building elements

Semantics assured by virtual switches is the same as provided by physical switches:

1. VNICs created on top of the same NIC can send packets to each other,
2. Broadcast packets received by the underlying NIC are distributed to every single VNIC that was defined on the top of this NIC,
3. Broadcast packets sent by one of the VNICs is distributed to all VNICs defined on the top of the same NIC and to the NIC for further transmission as well,
4. In terms of multicast network traffic multicast group membership is monitored and used for distributing packets to appropriate VNIC.

Data Link Layer connectivity between VNICs is available only when they were defined on top of the same NIC.

### 4.3.5 Crossbow components

The Crossbow specification describes three major components: VNICs, Etherstubs and Flows. This section gives an insight into their application and usage.

### VNICs

Virtual NICs (VNICs) each containing their own lane are the key element in crossbow architecture. There is no difference between NIC and VNIC in administration, as they are all treated as data links. Every VNIC has an assigned lane and flow classifier which classifies received packets by VNIC's MAC address and sometimes by the VLAN tag. If created with a VLAN tag, protocols like GVRP or MVRP may be used to register the VLAN tag with the physical switches too [**?**].

In terms of sharing bandwidth, Crossbow enables administrative control of bandwidth for every single VNIC. The bandwidth of the link is implemented by regulating the periodic intake of incoming packets per dedicated lane. The network stack allows only as many packets as it was assigned to specific VNIC. The lane picks more packets when the next period begins. In case of regulating the speed of transmited bandwidth it is much easier as the network stack can either control the application that is generating the stream of packets or just drop the excessive amount of packets. These mechanisms are also used in flows QoS described and discussed later in this paper [**?**].

### Etherstubs

As it was mentioned before, the MAC layer provides the virtual switching capabilities which allow VNICs to be created over existing physical NICs. In some cases, creating virtual networks without the use of a physical NIC is more welcomed than creating over physical NICs. In that case VNICs would be defined on the top of pseudo NICs. The Crossbow provides these kind of elements which are called Etherstubs. These components could be used instead of NICs during creation of VNICs [**?**].

### Flows

Flows are additional instruments created to allow easier network traffic administration. They might be used in order to provide bandwidth resource control and priority for protocols, services, containers. Virtual networks can be described to maintain isolation and different network properties, and define flows to manage quality of service [**?**].

Defined flow is a set of attributes based on Layer 3 and Layer 4 headers of the OSI/ISO model which are then used to identify protocol, service or virtual machine. Flows assigned to a link must be independent therefore before adding new one its correctness is checked. Input and output packets are matched to flows in very efficient manner with minimal performance impact.

Crossbow flows can be created with one of the following sets of attributes:

- Services (protocol + remote/local ports),
- Transport (TCP, UDP, SCTP, iSCSI, etc),
- IP addresses and IP subnets,
- DSCP field.

For each flow the following properties can be set [**?**]:

- bandwidth,
- priority.

**flowadm** is the console command used to create, modify, remove or to display network bandwidth and priority limits assigned to a particular link.

### 4.3.6 Running examples of flowadm and dladm command

**dladm** and **flowadm** are two basic administrative commands for dealing with the Crossbow's components. Below a few general examples of their usage are presented.

**dladm** is the admin command for crossbow datalinks elements management. Below a few examples of VNICs, Etherstubs management commands are presented and how bandwidth and priority values might be assigned to these elements.

1. # dladm create-vnic vnic1 -l e1000g0 - creates new VNIC **vnic1** over existing NIC **e1000g0**,
2. # dladm create-etherstub ether00 - creates new Etherstub **ether00**,
3. # dladm show-linkprop vnic11 - lists all properties assigned to **vnic11** link,
4. # dladm set-linkprop -pmaxbw=1000 vnic11 - assignes 1Mbps bandwith limit to **vnic11** link,
5. # dladm set-linkprop -ppriority=low vnic11 - assignes low priority to **vnic11** link.

More examples can be found in **man dladm**.

**flowadm** is the admin command for flow management. It might be used as follows:

1. # flowadm show-flow -l e1000g0 - displays all flows assigned to link **e1000g0**,
2. # flowadm add-flow -l e1000g0 -a transport=udp udpflow - creates new flow assigned to link **e1000g0** for all udp packets.

More information about **flowadm** and **dladm** tools can be found in manual.

### 4.3.7 Crossbow and Differentiated Services - interoperability

The Crossbow technology is designed to work inside single operating system instance, there are no mechanisms meant to cope with problems that arise when dealing with installations spanning multiple physical machines connected with traditional (non-virtual) network. Crossbow's flows are, by design, relatively simple (when compared to DiffServ) but more efficient as far as receive performance is considered [**?**]. Crossbow, unlike DiffServ, does not require special hardware, although if it is present it can boost overall operation performance [**?**].

DiffServ, on the other hand, provides sophisticated QoS mechanisms that require proper hardware (DiffServ-aware routers) to be present for it to work. DiffServ is standardized (RFC 2475) and offers a multiplicity of classification, marking, policing and traffic shaping alternatives [**?**]. Special fields (called DSCP) contained in IP packet's header are used to carry processing-related information with packets. The approach can be used with complex networks, comprising a number of routers with QoS awareness.

These two environments complement one another rather than compete. Crossbow supports flow matching based on the DSCP field value. DSCP field generation is planned but not yet supported. It is possible (although, at the moment, only partially) to integrate these and build a comprehensive end-to-end networking solution with QoS support and virtualized components.

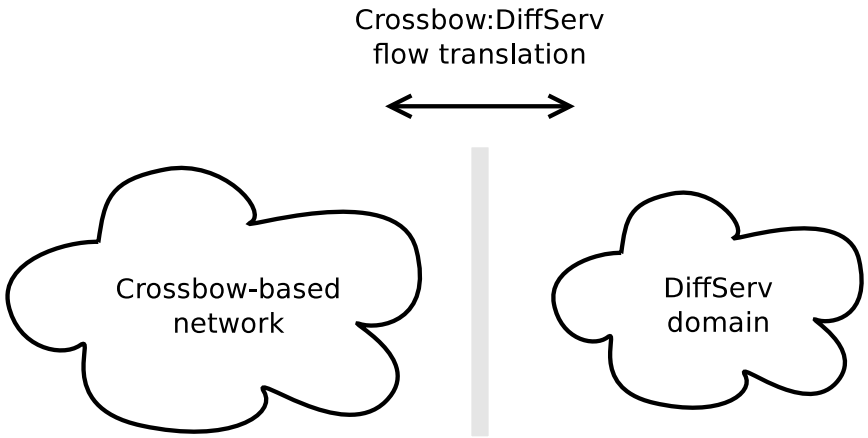


Figure 4.11: DiffServ integration using Crossbow-provided mechanisms

## 4.4 Resource control

Nowadays existing operating systems must provide mechanisms for response to the varying resource demands per workload which is an aggregation of processes of an application. By default resource management features are not used and system gives equal access to resources. When necessary, it is possible to modify this default behaviour with respect to different workloads. It is allowed to:

1. Restrict access to specific resource,
2. Offer resources to workloads on a preferential basis,
3. Isolate workloads from each another.

Resource is any part of computing system that may be modified in order to change application behaviour. Resource management enables more effective resource utilization and avoid wasting available ones due to load variability. Reserving additional capability during off-peak periods and effective sharing resources definitely increases application performance.

Most of the operating systems limited the resource control just to per-process control, whereas Oracle Solaris has extended this concept to the task, project and zone. Due to introducing granularity levels processes, tasks, and zones are efficiently controlled from excessive resource consumption. All these enhacements are available thanks to resource controls (rctls) facility [**?**].

Solaris Operating System introduced three types of resource management control mechanisms:

1. constraints - allows defining set of bounds on used resources for a workload,
2. partitioning - enables binding subset of system's available resources to specific workload,
3. scheduling - involves predictable algorithm making sequence of allocation decisions at specific intervals.

Hierarchical architecture allows defining set of resource control sets on each level. However, if more than one is assigned to a resource, the smallest container's control level is enforced [?].

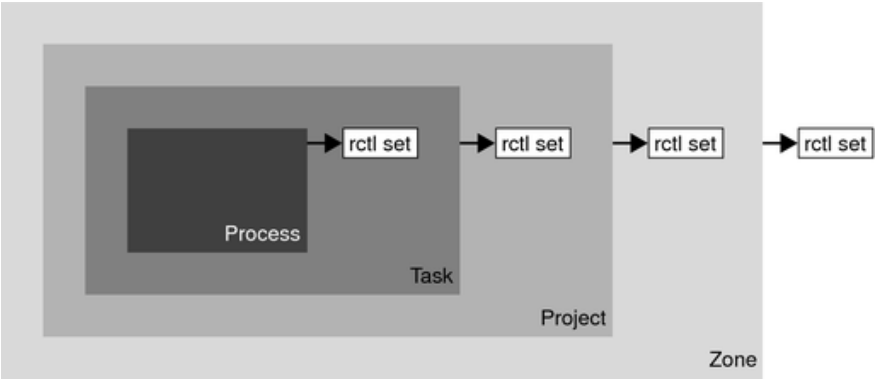


Figure 4.12: Solaris system multilevel architecture and its resource control sets (source: http://oracle.com)

### 4.4.1 Accounting

Highly configurable accounting facility is provided as part of the system. Its role is to gather historical usage records of system and network resources. There are two levels accounting can work on in Solaris OS - basic and extended. Basic accounting allows for per-zone and per-project statistics gathering while extended accounting facility makes it possible to collect the data for tasks and processes. Statistics gathered by the extended accounting can be examined using C or Perl interface of the libexacct library [?].

The extended accounting facility can gather data for:

- system resources usage (per-task and per-process),
- flows defined with the IPQoS tools,
- links and flows created with Crossbow.

## Summary

The chapter presented Solaris operating system with regard to resource virtualization. The stack of tools integrated into the system provides extensive support for virtualization techniques: Containers facilitate OS-level resource virtualization and Crossbow, shipped with Solaris 11, makes virtualization of networking resources possible. Resource control subsystem gives the administrator even more fine-grained control over resource utilization. Last, but not least, accounting functionality provides detailed view of resource usage history.

The features mentioned above make realization of flexible, scalable and efficient systems possible. With these foundations, it is possible to build and consolidate complex network-oriented infrastructures that prove to be reliable, relatively easy to manage and adjust to changing requirements.

Solaris 10 OS seems to be ideal cross-platform choice for customers dealing with management of high level services, complex system administration and high costs. It is the only open operating

system which has proven results running from every critical enterprise databases to high performance Web farms that is why Solaris OS is becoming strategic platform for today's constantly growing demands towards operating systems [**?**].

# Chapter 5

# The system architecture

The chapter discusses architectural aspects of the created system. First, the operating environment is discussed together with third-party components used to run the system. Then, general high-level view is described and layers of the system are presented. The remaining sections describe details of the layers.

Section 5.1 presents the context of the system. The distributed environment is described and basic requirements with regard to installed software are listed. Also, the way of extending the environment with specialized components is presented.

Section 5.2 introduces the design of the system. Main layers (or subsystems) are identified together with corresponding responsibilities. General aspects of layer interoperability are described.

Section 5.3 provides in-depth description of resource instrumentation layer. Its internal design is presented and main classes of objects analyzed. Interactions between the objects are depicted and, finally, the listing showing all the crucial classes belonging to the layer follows.

Section 5.4 describes the topmost layer of the whole system — virtual infrastructure management. The layer functionality is presented, then data model used is introduced and discussed and main components of the layer are described together with their interdependencies and interactions. Extensive description of the layer's three main use-cases — instantiation, discovery, monitoring — follows.

## 5.1 Operating environment

As depicted in figure 5.1, the system is designed to operate in a distributed environment — the bottommost layer of the operating environment is an IP network of physical machines. Each of the machines is running an instance of Solaris Operating System with support for Crossbow technology. None of the nodes is favoured over the others. The instances of operating system have Java Virtual Machine deployed and are capable of running JMX Agent which is used to host components of the system.

In addition to these basic specification, pure JMX Agents (i.e. agents without any components registered) can be enriched with third-party software and thus enable complete set of functionality implemented. With pure JMX Agents only single-node management is possible and limited to Crossbow networking resources. After integration with JIMS the system gains awareness of the whole distributed environment, has access to extensive mechanisms for controlling containers and can be used to create and manage complex virtual network topologies.

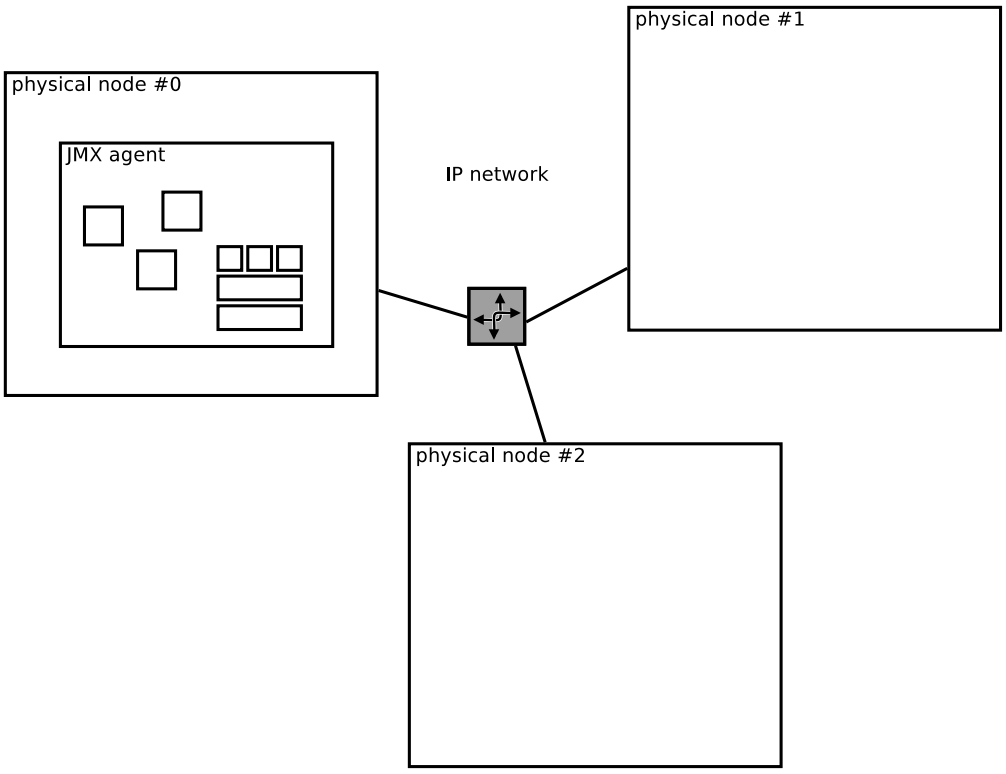


Figure 5.1: Deployment diagram for the system

## 5.2 Architecture overview

When considered at a very high level, the architecture of the system as a whole is layer-based. There are three main layers, each specifying a set of its own interfaces. The higher the layer is placed in the stack, the more complex interface it exposes. The three layers, as shown in figure 5.2, are:

- Infrastructure management layer

  contains components that help design, instantiate and manage network topologies,
  can possibly span multiple physical machines,
  requires JIMS installation to operate,

- Resource instrumentation layer

  is an abstraction layer over resources provided by the underlying system,
  present on each of the physical hosts,
  network-based interoperability between nodes is not supported,

- Underlying resources layer

  represents all the resources made available by host operating system,
  can be managed with vendor-supplied low-level utilities and libraries.

All the operations (e.g. topology modification) performed by infrastructure management layer go down the stack and, after necessary transformations, result in persistent changes to underlying resources. Conversely, the state of low-level resources can be expressed in terms of the domain model used by the highest layer (discovery process).

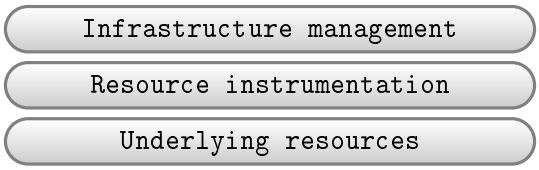


Figure 5.2: Layered system architecture

## 5.3 Crossbow resources instrumentation

The main responsibility of resource instrumentation subsystem is to provide a consistent way to create and manage resources of the underlying operating system. This general-purpose abstraction layer can easily be expanded when needed and further leveraged to build more sophisticated systems on top of it.

### 5.3.1 Separation of concerns

There are two classes of objects present at this level (figure 5.3) — Entity objects that are abstractions representing resources of specific type and exposing appropriate interfaces, and Manager objects used to perform basic coarse-grained operations (such as creation, deletion, modification) on resources they manage.


Figure 5.3: Manager and entity objects interoperability

#### Entity objects

Entity objects represent instances of a resource type. Each entity object class exposes an interface to manage the resource it is associated with. Fine-grained management is possible with entity objects — single properties can be accessed and manipulated (a subset of Flow public interface is presented in listing 5.1).

```
public interface FlowMBean {

  public String getName();

  public String getLink();

  public Map< FlowAttribute, String > getAttributes()
    throws NoSuchFlowException;

  public Map< FlowProperty, String > getProperties()
    throws NoSuchFlowException;

  public void setProperties( Map< FlowProperty, String > properties,
                             boolean temporary )
    throws NoSuchFlowException,
           ValidationException;

}
```

Listing 5.1: Selected methods of entity interface

### Manager objects

Each manager subtype is associated with a single class of resources. The subtype can be thought of as a gateway exposing methods to manage collection of resources. The responsibilities of manager objects include resource discovery, creation, modification and removal. Managers maintain lists of resources present in the system and provide ways to access them (as entity objects). The resources can also be published in external repositories.

```
public interface FlowManagerMBean extends GenericManager< FlowMBean > {

  public List< String > getFlows();

  public FlowMBean getByName( String name );

  public void discover();

  public void create( FlowMBean flow ) throws XbowException;

  public void remove( String flowName, boolean temporary )
    throws XbowException;

}
```

Listing 5.2: Selected methods of manager interface

Sequence diagram in figure 5.4 shows the process of creating new entity object. After creation,

the object is published in a repository to make it accessible for other components.

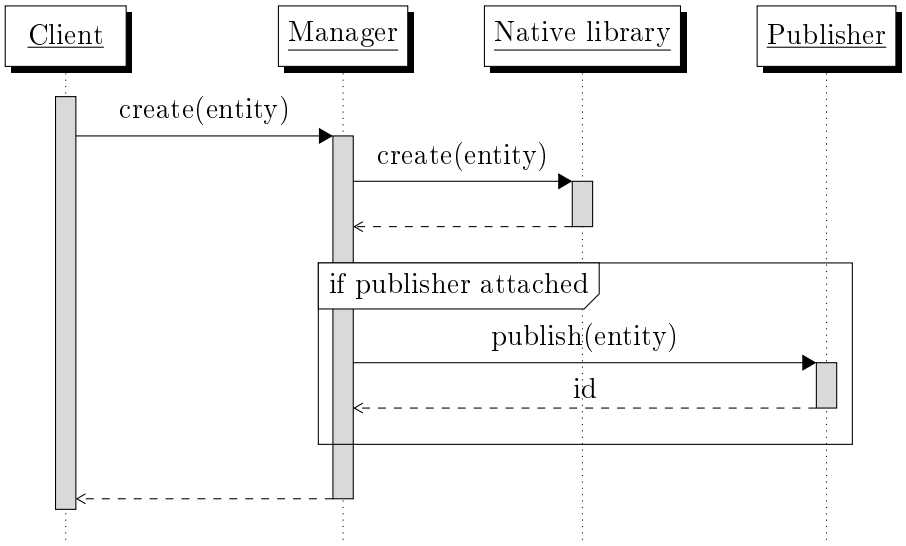


Figure 5.4: Entity creation scheme with optional publication

### 5.3.2 Layered design

Both the Manager and Entity objects share the same three-layer internal design as presented in figure 5.5. The objects themselves are exposed as Java Management Extensions beans. To implement the interface, either shell scripts or native libraries (or both) are used depending on the complexity of an operation. At the lower level, command line programs or native calls are executed, respectively.

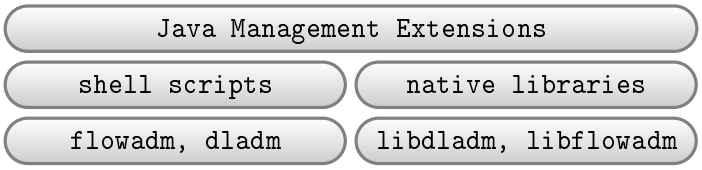


Figure 5.5: Layered system architecture

### 5.3.3 Instrumented Solaris OS resources

All the important Crossbow resources are instrumented (Manager:Entity pairs are created). This includes NICs (Network Interface Cards), VNICs (Virtual Network Interface Cards), VLANs (Virtual LANs), Etherstubs and Flows. All the components are loosely coupled and can be used independently.

## 5.4 Virtual infrastructure management

Virtual infrastructure management subsystem is built on top of the instrumentation layer. Leveraging entity and manager objects and components of the JIMS project, it provides high-level mech-

anisms to manage and monitor complex network topologies and quality policies associated with them.

### 5.4.1 High-level functionality overview

Figure 5.6 shows main stages of the management process together with general flows of data and is the starting point when identifying and designing coarse-grained components. The stages presented map to implemented components of the system that were implemented — User Interface to design, manipulate and monitor the topology, nodes responsible for discovery of available physical hosts, and mappers which translate between logical model and underlying resources.

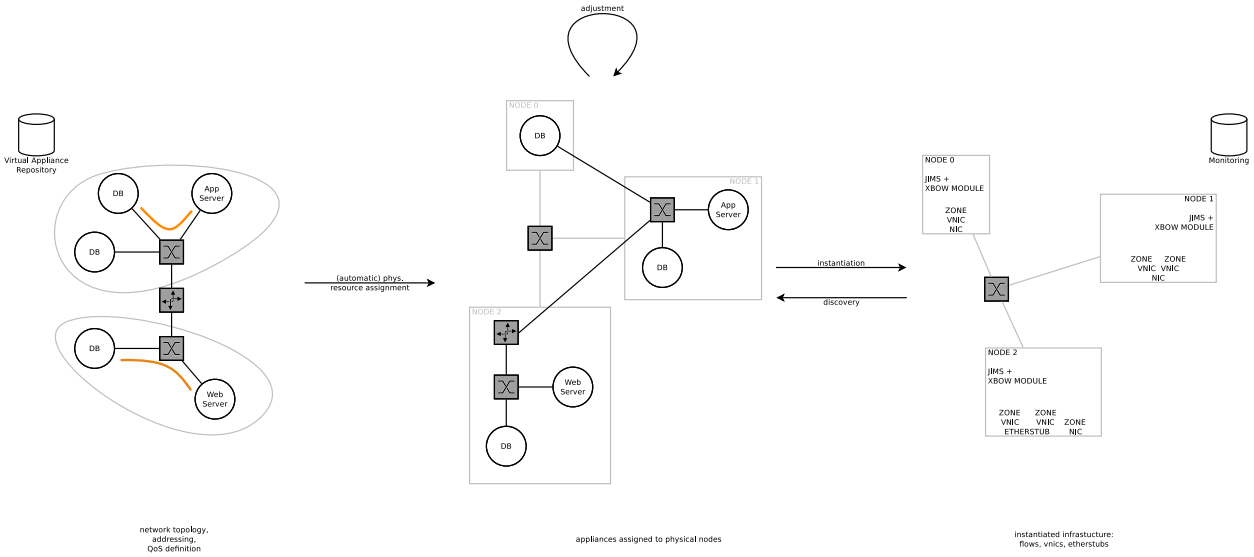


Figure 5.6: Main stages of operation

### 5.4.2 Domain model and data flows

Dedicated domain model was created for the virtual infrastructure management layer. It describes higher-level entities and operations that can be performed. The model is divided into three logical groups — static data describes resources and their interdependencies, assignment model is used to denote the association between parts of a topology and underlying physical resources, actions model is used to express the operations that can be applied to elements of a topology.

#### Static data model

As far as networking domain is considered, the model covers three main aspects:

- available entities — the set of resources that can be used to build a network topology,

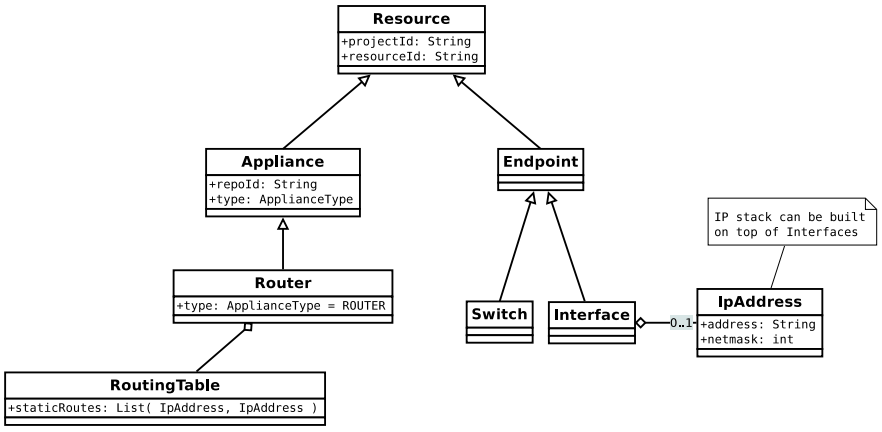


Figure 5.7: Object model — entities

- allowed interconnections — reflecting a subset of real-world connections between networking hardware,

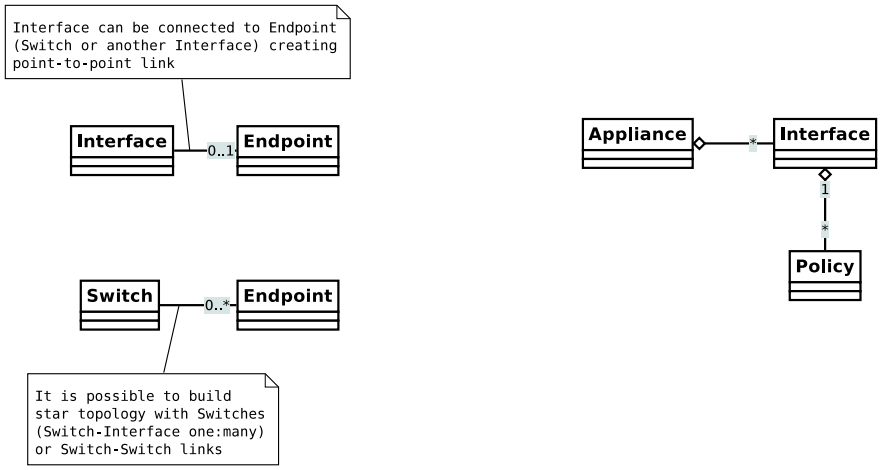


Figure 5.8: Object model — interconnections

- Quality of Service policies — classify traffic and determine priorities.

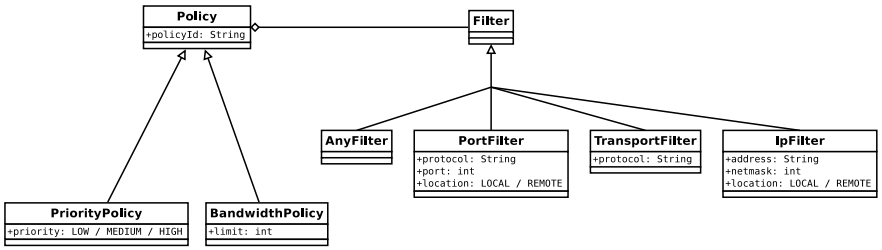


Figure 5.9: Object model — policies

### Modeling assignments

The assignment descriptor is used together with static data model to map entities to physical nodes. The descriptor is created automatically or by the user before instantiating the model as well as during the discovery stage when it is constructed by low-level components of the implemented system.

The annotation part of the descriptor can be used to carry additional entity-specific properties that have to be passed between components. It can also be used to hold auxiliary data while performing internal transformations of the model.

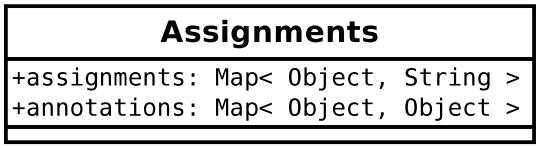


Figure 5.10: Assignments

### Actions

When managing the topology, actions play crucial role — they describe, for each element of the model, the operation that is going to be performed. Actions are assigned not globally for the model but on per-object basis — the approach that introduces more flexibility and efficiency.

There are three types actions designed. The object in the model can be created (ADD, as in instantiation phase), deleted (REM, typically performed after topology discovery) or modified (UPD, for entities that support on-line property adjustment).

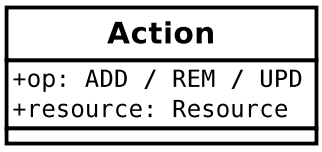


Figure 5.11: Actions

### 5.4.3 System components and their responsibilities

The specific character of the system — running in a distributed environment, moderate complexity — requires proper architectural model. The model should allow to design the elements of the system to be highly cohesive and maintain coupling as loose as possible. Each component has its own well-defined role and exposes a set of operations to interact with others.

The components presented in figure 5.12 reflect three main stages of operation shown in subsection 5.4.1. Boundaries were introduced to make the partition even clearer. The following subsections describe the components in more detail and provide listings with most important methods of exposed interfaces.

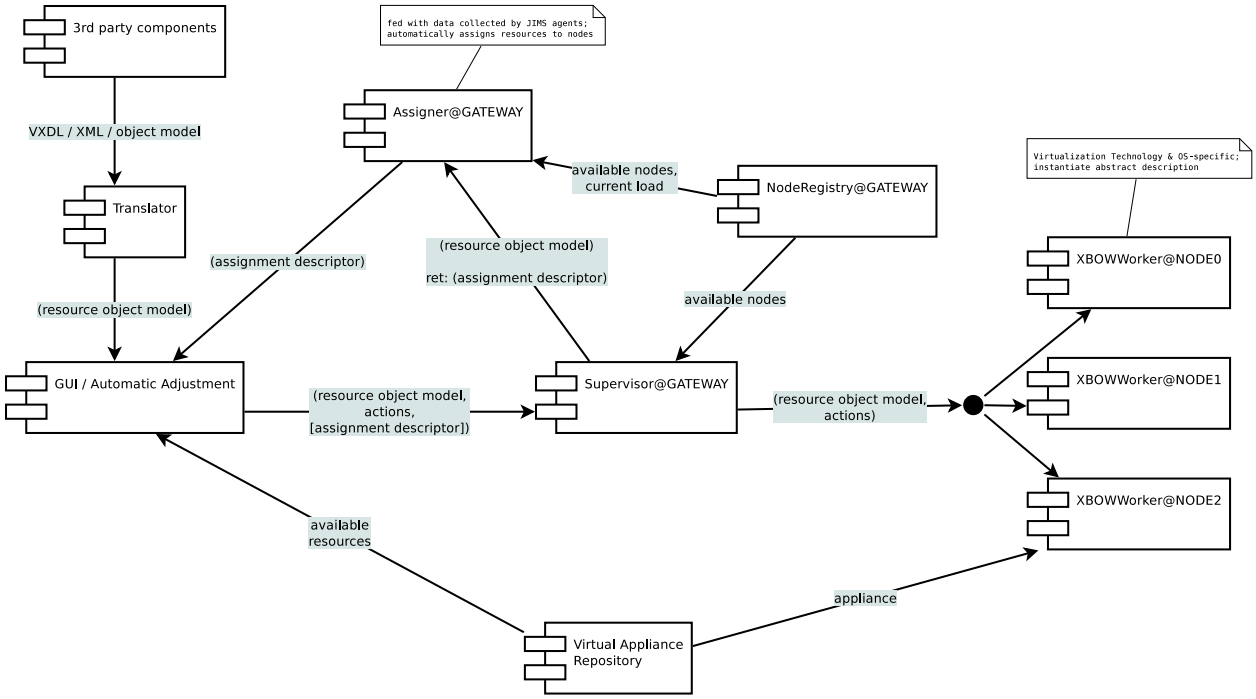


Figure 5.12: Components of the system

#### Virtual Appliance Repository

The main responsibility of Virtual Appliance Repository is to maintain the list of and provide access to virtual appliances created. It is used in two phases: the design phase to select appliances that are to be deployed, and instantiation phase to serve virtual appliance images.

```
public interface RepoManagerMBean {

  /**
   * Retrieves IDs of all appliances registered in the repository.
   */
  public List< String > getIds();

  /**
   * Returns filesystem path of the repository.
   */
  public String getRepoPath();

  /**
   * Sets the respository filesystem path.
   */
  public void setRepoPath( String path );

}
```

Listing 5.3: Virtual Appliance Repository public interface

#### Assigner

The optional assigner module can handle entire assignment stage and make it fully automatic. To be able to do this, it has to be configured with a set of rules and has to continuously collect data about physical nodes load. If the assigner component is not present, logical model has to be assigned manually to available host machines.

#### Supervisor

Supervisor component manages all the worker nodes present in the system. Its responsibility is to perform preliminary model transformations (if needed — for example, when using multiple host machines), divide the topology model according to the assignment rules and ask appropriate worker nodes to instantiate resulting parts.

Supervisor delegates most of the work to worker nodes. Transactional operations can be provided at this level by extending the supervisor's behaviour to rollback after one of the worker nodes fails.

```
public interface SupervisorMBean {

  /**
   * Performs actions on the supplied object model.
   */
  public void instantiate( ObjectModel model, Actions actions )
    throws ModelInstantiationException;

  /**
   * Performs actions on the supplied object model.
   * Uses provided assignment descriptor.
   */
  public void instantiate(
    ObjectModel model,
    Actions actions,
    Assignments assignments
  ) throws ModelInstantiationException;

  /**
   * Discovers all the topologies created.
   * Returns topology names together with domain model representation.
   */
  public Map< String, Pair< ObjectModel, Assignments > > discover();

  /**
   * Retrieves the list of managed workers.
   */
  public List< String > getWorkers();

}
```

Listing 5.4: Supervisor public interface

### Worker

Worker components perform all the low-level operations, including model to underlying entity mapping (and vice versa). Workers do not transform the model in any way — all they do is provide well-defined rules for instantiation (including naming schemes) and discovery. Multiple workers are managed by the supervisor.

Public interface of worker component is presented in listing 5.5.

```
public interface WorkerMBean {

  /**
   * Maps domain model to low-level resources.
   */
  public void instantiate(
    ObjectModel model,
    Actions actions,
    Assignments assignments
  ) throws ModelInstantiationException;

  /**
   * Analyzes present system entities and reconstructs the domain model.
   */
  public Map< String, Pair< ObjectModel, Assignments > > discover();

}
```

Listing 5.5: Worker public interface

### 5.4.4 Main data flows and cooperation of the components

#### Instantiation

There are three main stages identified when working with the topologies. There is a purely logical one that does not require any knowledge of the underlying environment — the operations involve manipulating the domain model to create or update the virtual topology. There is an assignment stage which results in association between the model and physical resources. Finally, the actual deployment takes place in model instantiation stage — logical elements are mapped to low-level ones after performing necessary transformations.

Instantiation is the process of transforming a logical model to fully operational virtual network. There are three main stages that constitute the complete process:

1. Logical model definition

   This is the first stage the user is exposed to. The main task is to create a virtual network topology comprising logical networking components (belonging to Layer 2 and 3 of the OSI/ISO model) and virtual appliances — specialized virtual machines. After the topology is created, IP addressing is provided and routing set up. Finally, Quality of Service policies are determined to classify and differentiate the traffic.

2. Physical resource selection and assignment

   After the model is defined, it can be associated with underlying physical resources. There are two possible ways of performing the assignment — it can be done manually with supplied utilities or special component can suggest optimal solution based on, for example, current workload and predefined set of rules. The latter approach is particularly useful when working with complex systems that should be able to adjust themselves automatically to balance the load.

3. Model instantiation

   The final, entirely automatic, step is to map the logical model and assignments to actual components created on the host machines. This involves a series of transformations to adjust the model to capabilities of the underlying environment and satisfy other requirements like topology isolation.

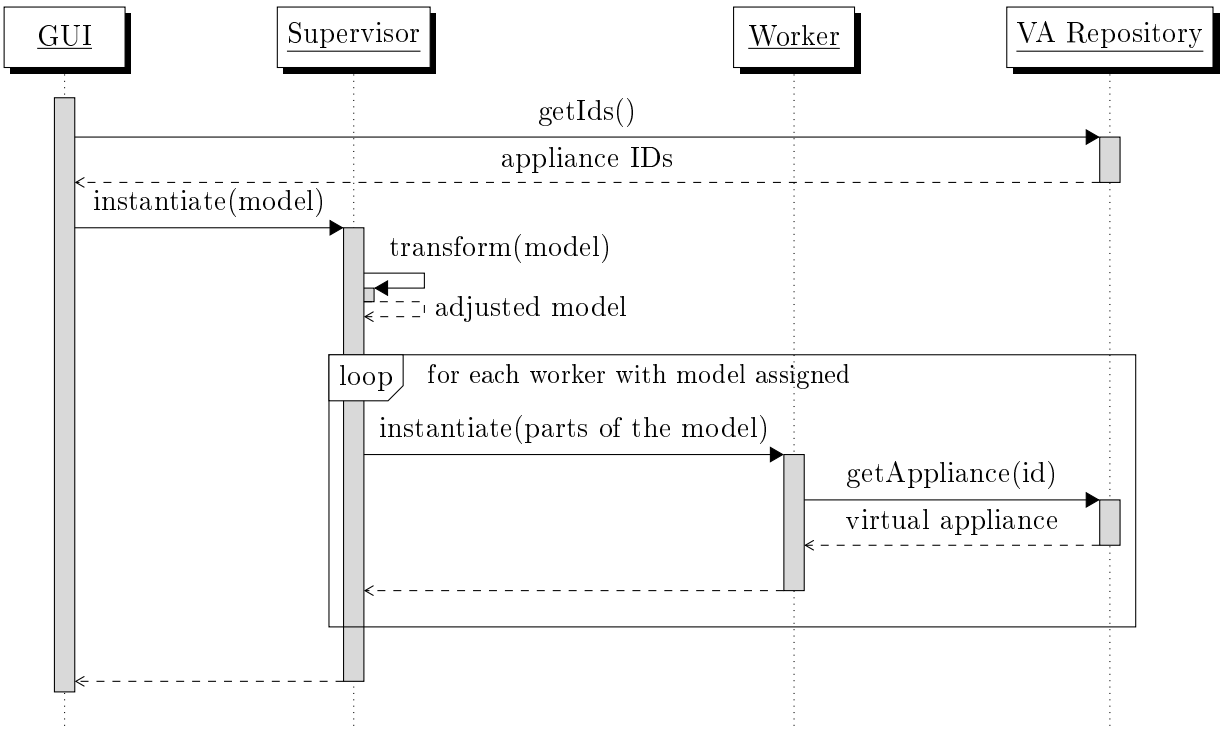


Figure 5.13: Topology instantiation

### Discovery

Instantiated topology, together with applied addressing, routing table entries and quality policies, can be discovered, i.e. object model that describes it can be recreated. The discovery process is an inverse of instantiation, it is composed of three phases (as shown in figure 5.14):

1. The system resources are inspected by each of the Worker nodes and parts of the model together with Assignment descriptors are created independently,

2. partial results are collected and merged by the Supervisor. Redundant data is removed and necessary transformations performed,

3. complete model is passed further (e.g. to the GUI component).

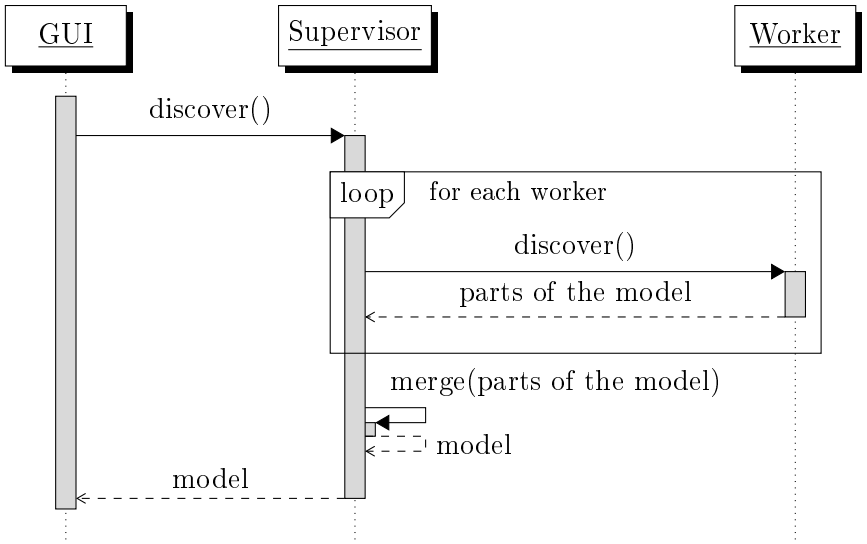


Figure 5.14: Topology discovery

### Monitoring

Topology operation can be monitored with high degree of granularity. Single flows can be inspected to see the amount of data transferred. Historical data is also made available.

The StatisticsGatherer component performs the translation between domain models, for example it is able to map Policy to corresponding Flow and retrieve traffic statistics. The operation in shown in figure 5.15.

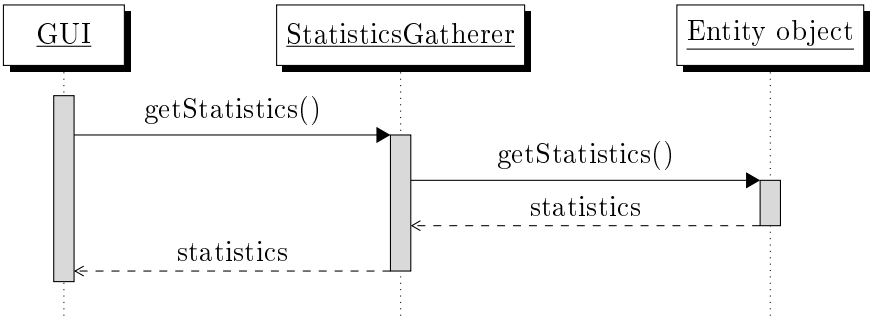


Figure 5.15: Topology monitoring

## Summary

# Chapter 6

# Implementation

## Chapter overview

This chapter focuses mainly on implementational details, especially on most interesting aspects of our system.

Section 6.1 informs about implementational environment aspects like requested operating system, libraries presence or necessary programs to be build and installed.

Section 6.2 describes in detail created components facilitating crossbow usage.

Section 6.3 outlines domain model transformation details.

Section 6.4 lists low level functions used in created system.

Section 6.5 presents all necessary steps required for platform build completion.

## 6.1 Implementation environment

JIMS Extensions for Resource Monitoring and Management of Solaris 10 provides general architecture for monitoring and management applications written in Java.

JIMS architecture is generally based on JMX ( Java Management Extensions ) - technology for distributed resource management. These managed resources in JMX are represented by MBeans which are simple Java objects registered at MBean Server under specific ObjectName [**?**].

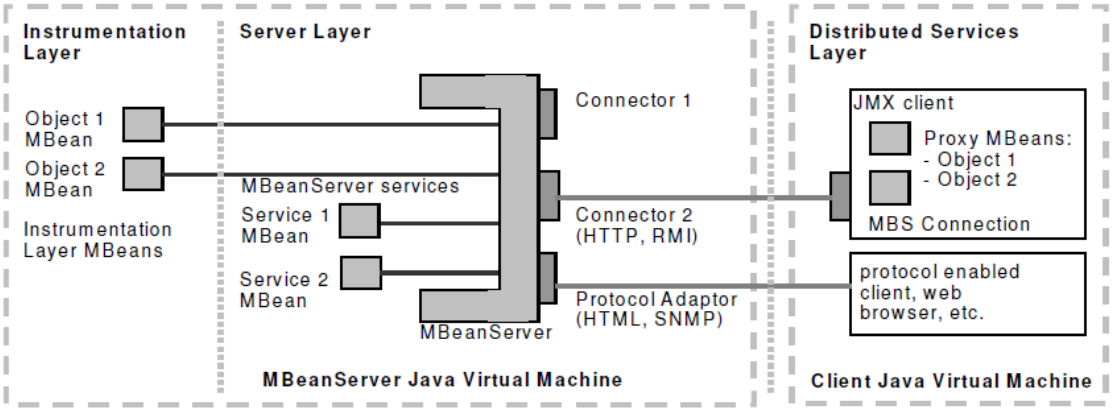


Figure 6.1: JMX architecture [?]

JMX provides also services such as:

- Notifications,
- MLet (downloading dynamic modules).

JIMS (JMX based Infrastructure Monitoring System) supports monitoring and management under both Linux and Solaris platforms. Due to Jims features such as: easy maintenance (automatic modules downloading), extensibility (possibility of adding additional modules) integrating with Jims newly developed module (jims-crossbow) was quite an easy task [?].

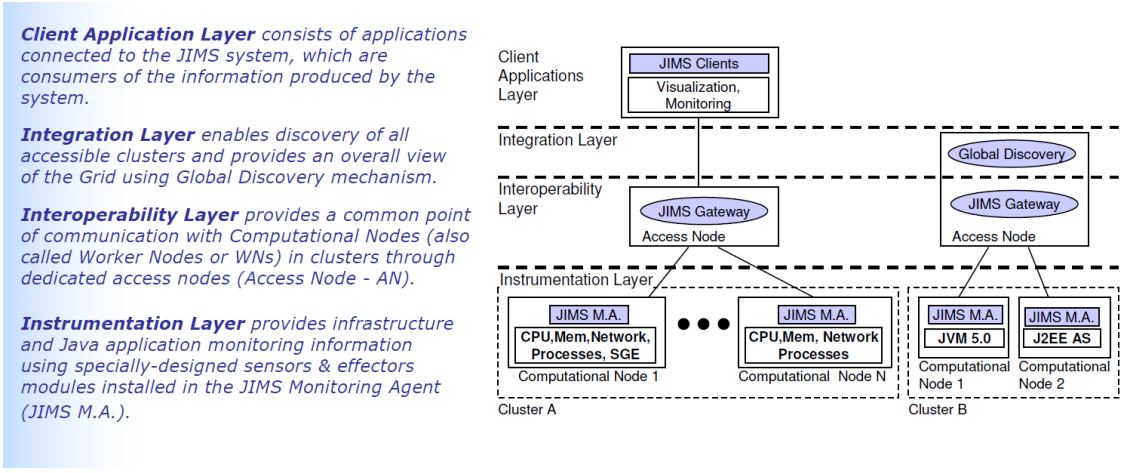


Figure 6.2: JIMS architecture [?]

Jims services(MBeans) enabling creating, reading and changing properties of zones and projects were extensively used in our system during deploying (creating) requested network structure. For more information about JIMS project please refer to the bibliography.

For the purpose of this paper two applications were implemented:

- Jims module for crossbow,
- GUI application.

In case of jims-crossbow module the implementation environment must consist:

- gcc compiler - for building jims-crossbow shared librarires,
- dladm, flowadm libraries.

The demand for crossbow libraries (flowadm, dladm) implicated that implementation environment must have been Solaris 11 or any other system supporting crossbow.

GUI application was developed in java using swt and swing graphic libraries. Project was managed with maven and thanks to maven profiles feature can be build and run on operating systems like:

- Solaris,
- Linux x86,
- Windows x86.

SWT core libraries for these operating systems were provided in repository, so the only requirements is to have one of the already mentioned operating system and installed: Java se 1.6 and maven 2.x.

## 6.2 Crossbow components implementational details

In terms of crossbow components two kinds of them have been distinguished:

- Managers,
- Entities.

Managers basically provide operations for entity discovery and general management. The following managers have been created:

- VNicManager,
- EtherstubManager,
- FlowManager.

Another component type is entity which is basically equivalent of single crossbow entity like VNic, Etherstub. These entities appear as MBeans and are registered at MBean servers. Each bean contains basic information about entity like name, assigned properties and other attributes depending on their type. Operations for altering properties, parameters, ip address are provided as well. Three figures below depicts in detail each manager and their corresponding entity.

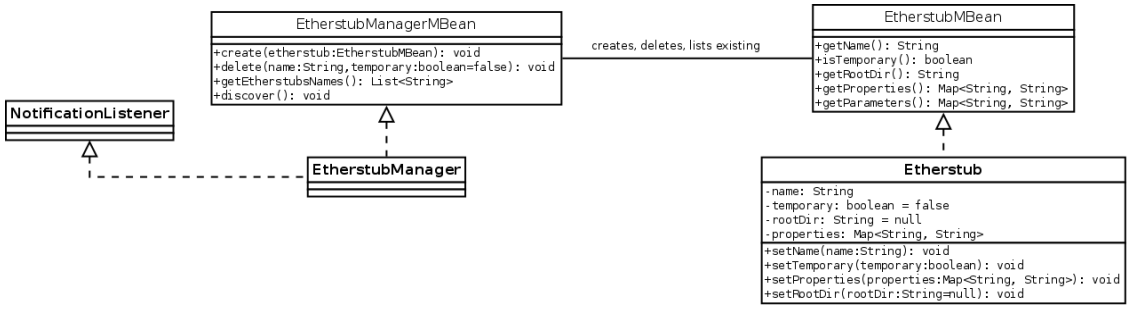


Figure 6.3: Etherstub class diagram

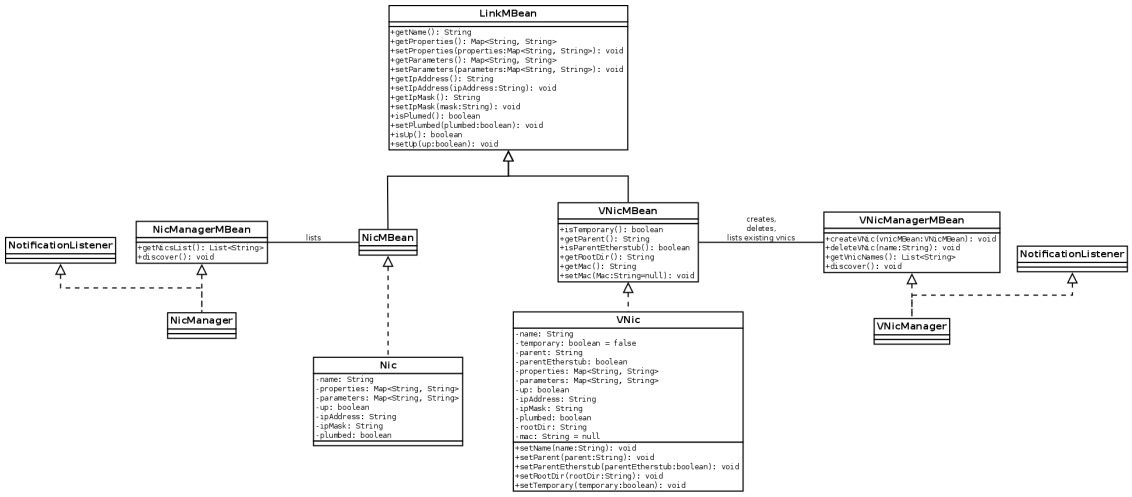


Figure 6.4: Link (VNic, Nic) class diagram

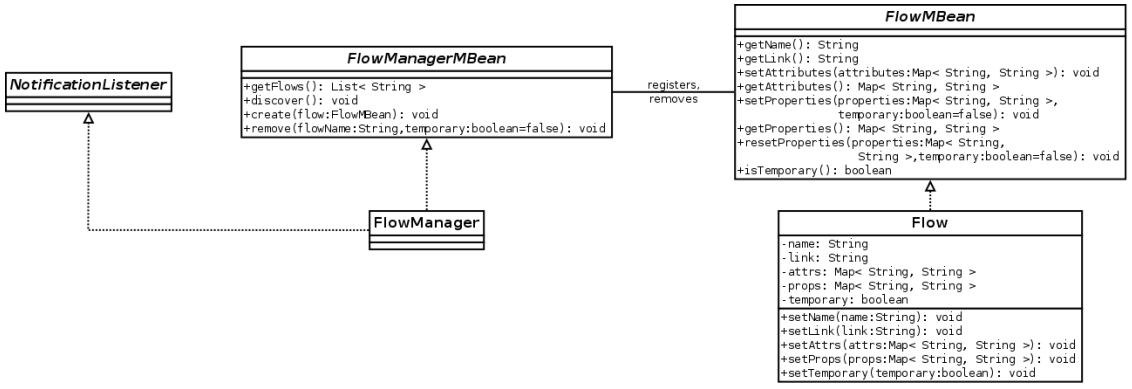


Figure 6.5: Flow class diagram

Due to the fact that managers and entities have been separated and that each entity is an individual MBean these entities can be accessed not only from java code but also be viewed and modified from jconsole.

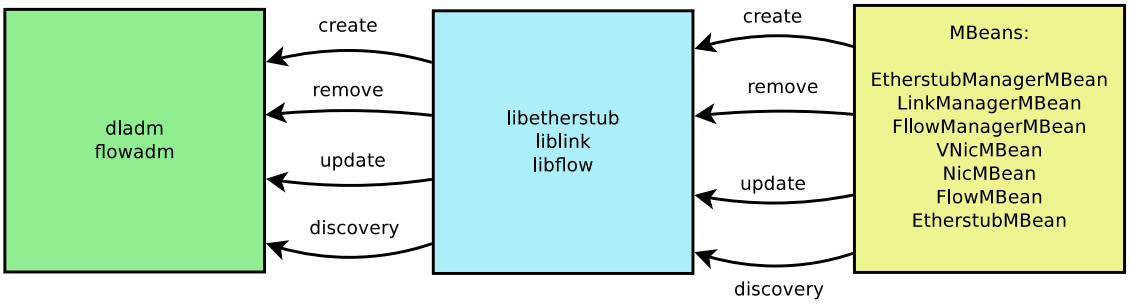


Figure 6.6: Jims-crossbow components, shared libraries and crossbow native libraries relationship

## 6.3 Domain model transformation details

To allow conversion between gui designed network structure (like in the figure below) to zones and crossbow entities on the server side new domain transformation model was proposed.

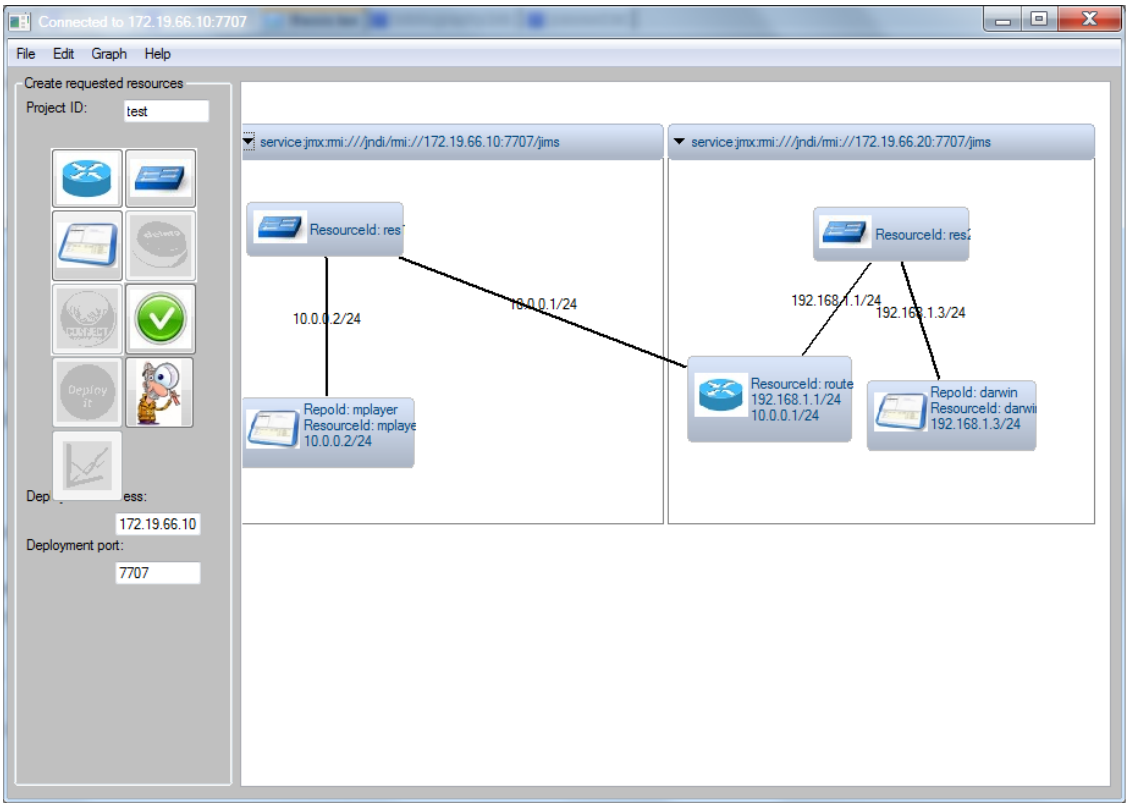


Figure 6.7: Example of a network structure to be converted to object model

In general four main class types were distinguished: Appliance, Switch, Interface, Policy.

**Appliance** is an equivalent of a zone to be created from one of existing snapshots. Appliances could be also extended to Routers (with assigned static routes) which on the server side are seen as two appliances created under different etherstubs with routing tables modified and assigned interfaces enabling connectivity between appliances working under these two etherstubs.

**Switch** is implied as crossbow etherstub.

**Interface** is converted to VNic and assigned to created zones.

**Policy** should be seen as a demand towards desired network traffic shape. Policy has assigned Flow Object with detailed values and flow type. These objects could be assigned directly to existing interfaces or exist as detached crossbow flows.

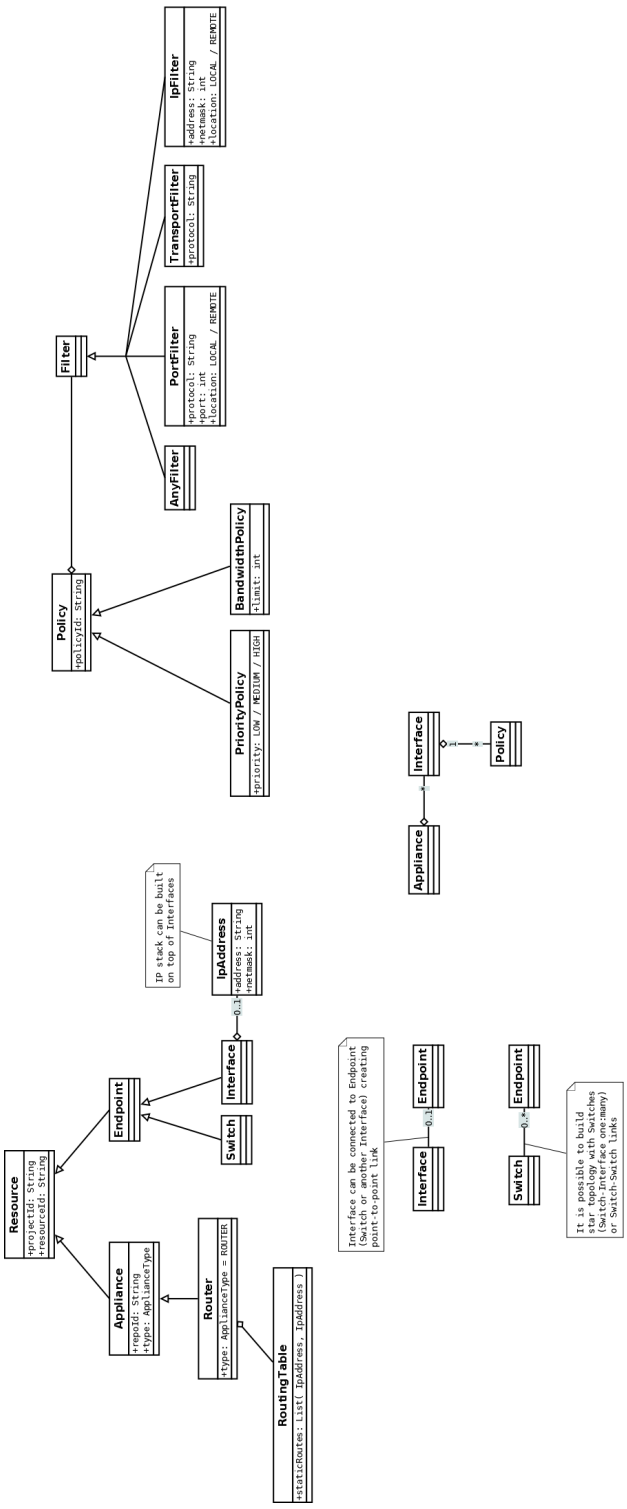


Figure 6.8: Resource object transformation model

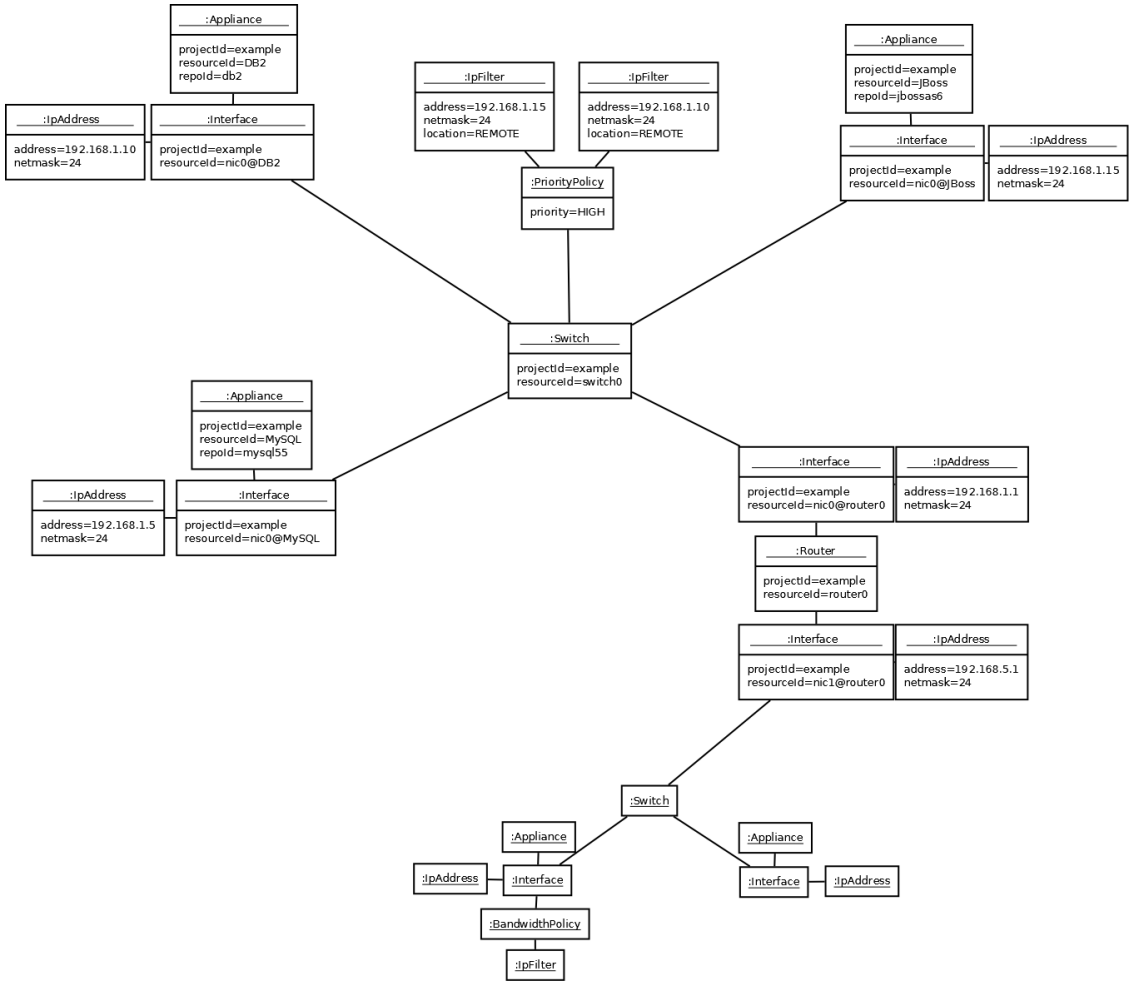


Figure 6.9: Resource object transformation model example

Created model was designed in order to comply requirement of multiple deployments so that only introduced changes were applied in already deployed project. Due to this issue to apply changes **instantiate(ObjectModel objModel, Actions actions)** method from **SupervisorMBean** must be invoked where ObjectModel class aggregates all domain model objects and Actions object encapsulates all actions to be done to every single domain model object. These actions would be:

- ADD - adds new resource,
- REM - removes this resource,
- UPD - updates existing resource,
- NOOP - no changes are applied.

## 6.4 Low-level functions access

JIMS layered architecture implicated the demand for an approach towards accessing low-level funtions. In developed application these approaches were adjusted to existing conditions so that for accessing functions from shared libraries JNA (Java Native Access) was used and for doing more complex operations on low-level scripts were written and Java ProcessBuilder invoked. Created jims-crossbow module containes shared library allowing CRUD operations which subsequently are being accessed by JNA, whereas most of jims low-level access was done through running shell scripts. Although running scripts by ProcessBuilder is faster in some cases accessing native libraries through libraries like JNA or JNI gives more configurational advantage and does not require shell script writing ability.

Listing 6.1 presents abbreviated example of JNA access to native libraries.

```
public interface LinkHandle extends Library {
  public int set_ip_address(String link, String address);
}

public class JNALinkHelper implements LinkHelper {

  protected LinkHandle handle = null;

  public static final String LIB_NAME
    = "libjims-crossbow-native-lib-link-3.0.0.so";

  /**
   * Creates the helper object and initializes underlying handler.
   *
   * @param libraryPath Path to native library
   */
  public JNALinkHelper(String libraryPath) {
    String filePath = libraryPath + File.separator + LIB_NAME;
    handle = (LinkHandle) Native.loadLibrary(filePath,
                                             LinkHandle.class);

    handle.init();
  }

  public int setIpAddress(String link, String ipAddress)
    throws LinkException, ValidationException {

    return handle.set_ip_address(link, ipAddress);
  }
}
```

Listing 6.1: Native library access with JNA

## 6.5 Building and running the platform

To build and run the platform the following prerequisites are required:

- Java SE 1.6, maven,
- jims sources downloaded,
- jims-crossbow module downloaded from https://github.com/robertboczek/solariscrossbow/tree/master/code.

Afterwards JIMS project must be built. Detailed description of how to build JIMS is in **README** file located in the main catalog of jims sources. One of the most common problems are missing jars that unfortunately must be manually downloaded and installed in the local repository.

Subsequently jims-crossbow module should be copied to the main folder containing jims and then build. The script **inst-crossbow-lib.sh** must be later executed to copy shared libraries *.so files to /usr/lib folder.

If everything went well another step is running JIMS:

- jims-gateway: .../jims-gateway/bin/jims-agent.sh [-b host_address] start|stop|restart,
- jims-agent: .../jims-agent/bin/jims-agent.sh [-b host_address] start|stop|restart.

We should run just single jims-gateway and on the rest nodes as many jims-agents as we require. For more information about **jims** and its architecture please refer to:

If jims-agent or jims-gateway did not started it is worth to see logs files, located respectively at target/.../jims-agent/var/jims/log/agent.log and target/.../jims-gateway/var/jims/log/agent.log

It also recommended to have logs opened during JIMS start to see whether any exception was thrown (**tail -f target/.../jimsgateway/var/jims/log/agent.log**)

JIMS has JMX-based architecture so each jims-agent and jims-gateway can be accessed through jconsole. In order to do that start jconsole, select remote process and enter type:
**service:jmx:rmi:///jndi/rmi://address:port/jims** where **address** and **port** is concrete address and port under which jims was started. **JConsole** allows browsing registered mbeans and performing CRUD operations. Especially in case of the crossbow module it allows these operations as the figure below presents:

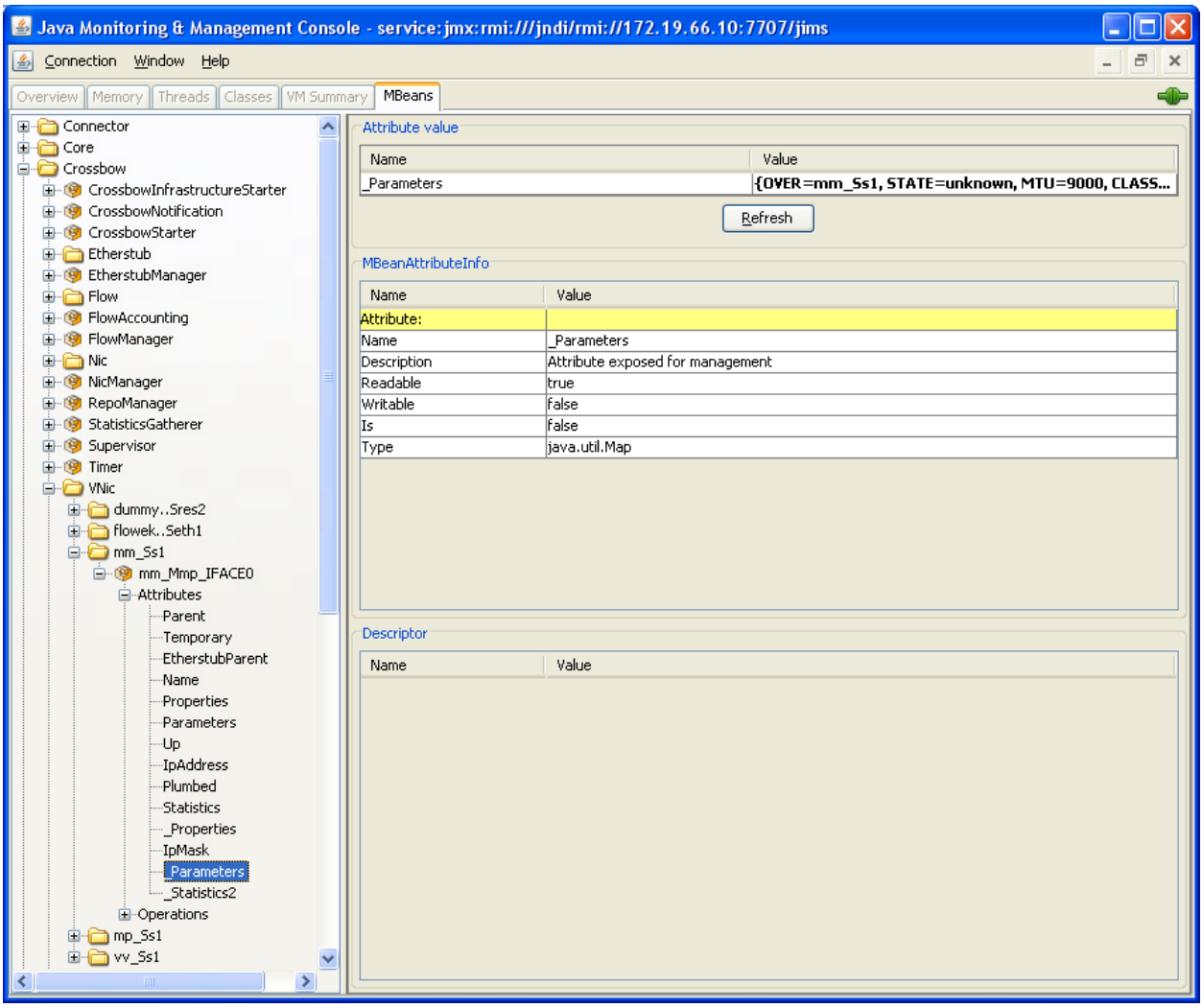


Figure 6.10: The Crossbow module registered MBeans example

To simplify working with our system GUI application was created. It is located in jims/jims-crossbow/jims-crossbow-gui catalog. Application may be imported to eclipse and then build and run or build using maven ( **mvn assembly:assembly** ) and run **java -cp target/jims-crossbow-gui-3.0.0-exe.jar org.jims.modules.crossbow.gui.Gui**

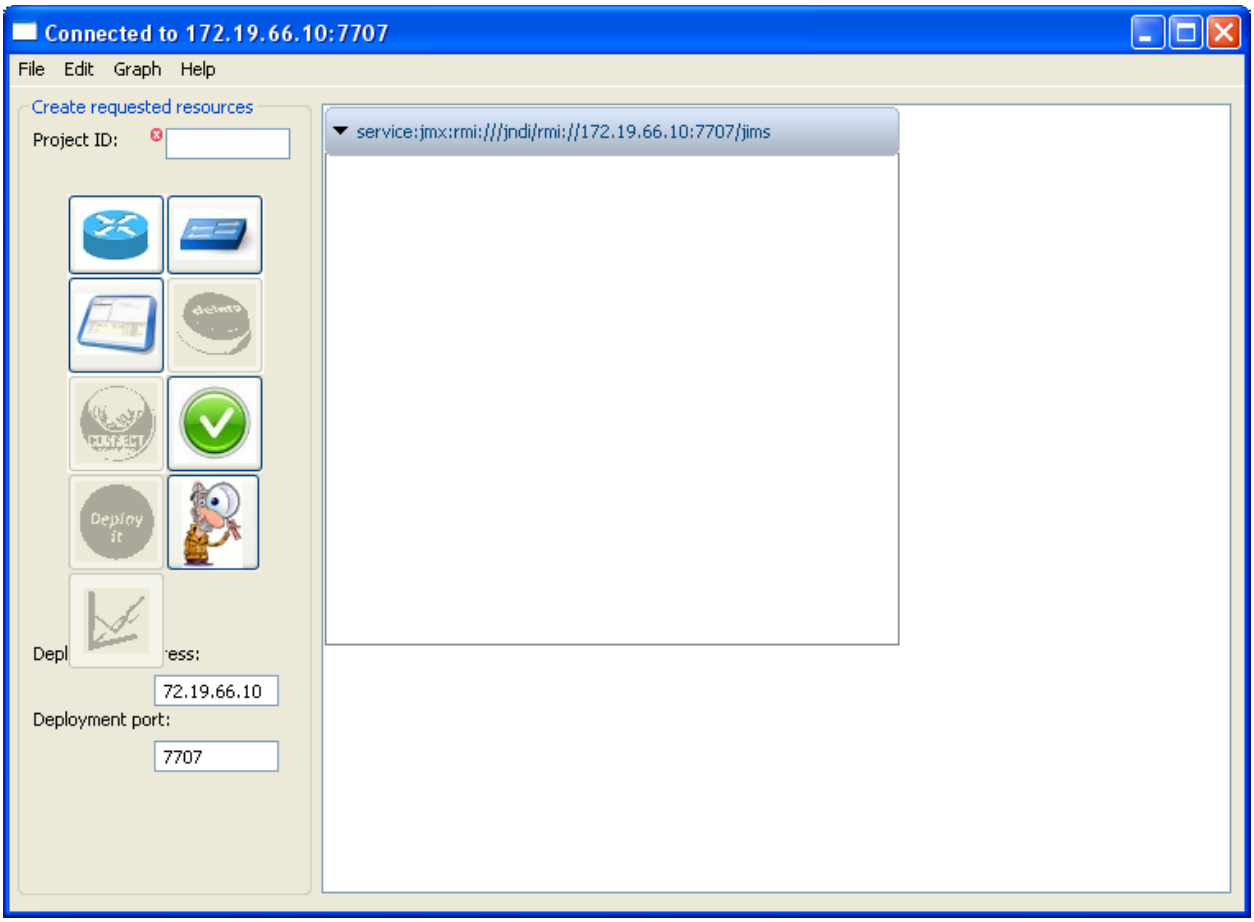


Figure 6.11: Gui application

Implemented gui application allows:

- Connecting to jims-gateway,
- Creating, modifying and removing desired network structure with requested virtual appliances,
- Discovering already created projects,
- Detailed information about links and flows like bandwidth load presented in charts from requested time periods or just simple numbers,
- Automatic logging using ssh to selected ( already deployed ) nodes and opening gui-type terminal.

## Summary

# Chapter 7

# Case Study

The chapter describes the infrastructure that was built using the implemented system. Steps necessary to restore the configuration are listed and described. A number of tests were performed to evaluate the created system and topologies it allows to create. Resulting experimental data is presented and discussed.

Section 7.1 introduces the overall view of the topology that was created. Main components are described and QoS requirements are discussed in more detail. Types of service are presented and appropriate network-level policies described. The policies are assigned to network components.

Section 7.2 describes the stages needed to set up the topology. The steps include virtual appliance creation and publication, topology design, determining the quality policy and instantiation. Domain model of the designed topology and resulting low-level entities are listed.

Section 7.3 presents the results of experiments performed to verify requirements the system has to meet. The section focuses on Quality of Service-related aspects — it verifies definition, management and operation of the policies.

Section 7.4 lists the advantages of using the proposed system and approach to create, manage and monitor QoS-aware virtual topologies. Aspects particularly helpful when preparing the case study are highlighted.

## 7.1 Scenario description

The test case is inspired by multimedia systems. The problems of quality-aware transmission arise naturally in multimedia-oriented networks. Moreover, there are easily-identifiable classes of traffic with non-uniform quality requirements. This characteristics make multimedia networks a reasonable choice when considering tests focused on quality requirements verification.

Also, complex topologies are built to enable multimedia transmission. The components that comprise these networks include, among others, specialized media servers, routers and client machines. This variety allows to demonstrate the usefulness of the created systems in the process of designing such topologies.

### 7.1.1 Types of service

As already stated, there are classes of traffic (or service types) that, by their nature, require different amounts of available resources (such as bandwidth, processing priority, etc.). The types of

multimedia services map directly to QoS policies required. Table 7.1 shows the mapping.

| service type | bandwidth | delay tolerance |
|---|---|---|
| real-time streaming | high | low |
| video on demand | high | high |

Table 7.1: Multimedia network traffic and its requirements

The most demanding type of multimedia data is undoubtedly real-time streaming. The high priority data has to be favoured in order to provide desired level of quality — precedence when accessing transport media and low-jitter transfer. This allows for low-delay streaming with small input data buffers on the client side.

Video on demand data is of less priority. It is assumed that the user is not going to utilize the data as it is being downloaded. This assumption loosens the transmission requirements and allows to treat VOD traffic as medium-priority or even best-effort data (no CIR).

### 7.1.2 Topology overview

The network topology built is an attempt to model simple yet real environment used to transmit multimedia data. There is a streaming server without user differentiation mechanisms (with respect to quality of transmission) and an HTTP server handling VOD requests. The clients connect to the streaming server and start streaming sessions. They can also download video served by the HTTP daemon.

Client and server components are placed in different subnetworks that, in turn, are connected with a QoS-aware router which provides one more level the policies can be defined at. Rules defined for the router's interfaces specify fine-grained policies for clients in subnetworks in addition to the ones defined at the server level.

Taking this approach, it is easy to enable QoS-aware networking leveraging relatively simple applications. The aspects of choosing server and client implementations and providing quality of service become orthogonal and the whole design remains clear and maintainable. Figure 7.1 presents an overview of the network structure and client configuration.

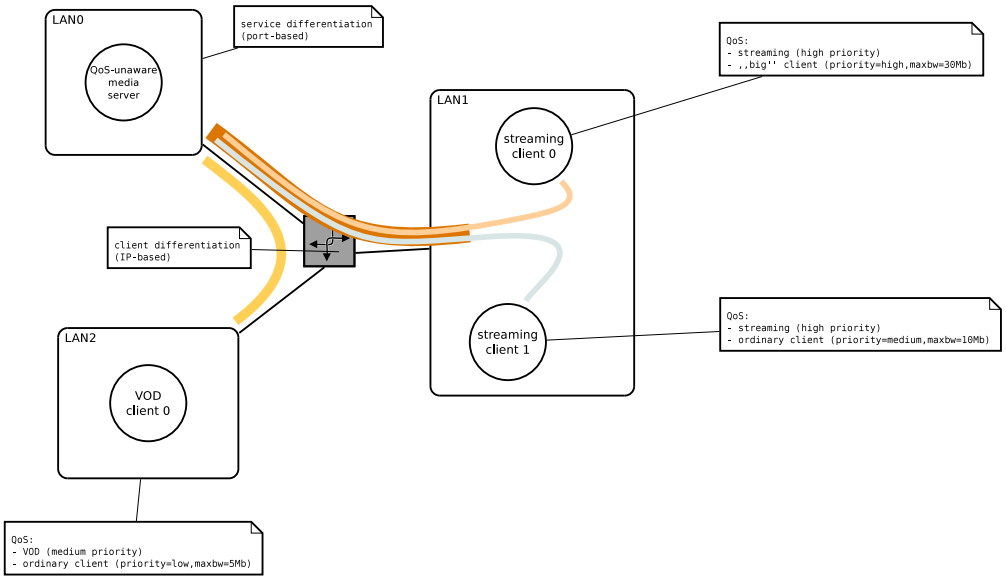


Figure 7.1: High-level view of the created topology

### 7.1.3 Service and client differentiation

The traffic is classified based on two properties: type of service (RTP, HTTP) and recipient address. Three traffic classes are distinguished with respect to service type:

- high priority RTP streaming

  UDP traffic with source ports 6970 and 6971, used by streaming client 0 and streaming client 1 in LAN1 subnetwork,

- low priority Video On Demand

  TCP traffic with source port 80, used by the VOD client 0,

- medium priority ordinary traffic

  all the other data.

Furthermore, the streaming clients inside the LAN1 subnetwork are assigned different priority values. Client 0 is favoured and has high priority for RTP data, whereas client 1 is assigned low priority. Client 0 in LAN2 is not assigned any priority explicitly.

## 7.2 Preparation of the environment

General steps while building the environment are: virtual appliance preparation, topology design and instantiation. Virtual appliances are created manually and published in an NFS repository. Then, they are used as building blocks when designing the network. Finally, the virtual infrastructure is instantiated, i.e. all the underlying low-level components, like zones, etherstubs, VNICs and flows, are created.

### 7.2.1 Virtual appliances

Listing 7.1 shows the initial steps when creating new zones. In this case, the zone is called `mplayer` and it contains a streaming client. At first, new ZFS pool is created to host the zone's filesystem, then basic configuration is performed and the zone is installed.

```
# zfs create rpool/Zones/mplayer

# zonecfg -z mplayer

zonecfg:mplayer> create
zonecfg:mplayer> set zonepath=/rpool/Appliances/mplayer
zonecfg:mplayer> set autoboot=true
zonecfg:mplayer> set ip-type=exclusive
zonecfg:mplayer> verify
zonecfg:mplayer> commit
zonecfg:mplayer> exit

# chmod 700 rpool/Appliances/mplayer
# zoneadm -z mplayer install
```

Listing 7.1: New zone creation

It may be necessary to modify `/etc/shadow` file as in listing 7.2 to be able to access the zone with `zlogin`.

```
# sed s/root::/root:NP:/ /etc/shadow
```

Listing 7.2: `/etc/shadow` file adjustment

After installation the zone can be booted and used after logging (listing 7.3).

```
# zoneadm -z mplayer boot
# zlogin mplayer
```

Listing 7.3: Booting and logging into a zone

All the required software should be installed now. When the zone is prepared, a ZFS snapshot can be taken and transferred to the repository (listing 7.4).

```
# zfs snapshot -r rpool/Appliances/mplayer@SNAP
# zfs send rpool/Appliances/mplayer@SNAP > /appliance/mplayer.SNAP
```

Listing 7.4: Publishing a snapshot in NFS repository

### 7.2.2 Topology instantiation

After all necessary virtual appliances have been created and stored in the repository, network topology can be designed and instantiated. Following steps comprise the whole process:

1. selection of virtual appliance templates from the repository,

2. designation of physical machine(s) to host the topology,

3. appliance-to-host assignment,

4. enabling network connection between nodes, addressing, routing setup,

5. defining QoS policies.

The topology consists of router (forwards IP packets between its directly-attached interfaces), server (with Darwin Streaming Server and thttpd HTTP server installed) and client appliances (mplayer compiled with RTP streaming support enabled). All the appliances are instantiated on a single physical host.

There are three subnetworks:

- 1.1.1.0/24 contains only the server appliance (addressed 1.1.1.2),

- 2.2.2.0/24 with one VOD client (addressed 2.2.2.2),

- 3.3.3.0/24 with two streaming clients (addressed 3.3.3.2 and 3.3.3.3).

The router appliance, with three interfaces (addressed 1.1.1.1, 2.2.2.1 and 3.3.3.1) links the subnetworks and provides network-level connectivity. Each of the appliances has additional entries in its routing table.

Quality of Service assurance is composed of two main stages:

- bandwidth of the links used to stream and download media is limited to 8Mbps (mainly to make the testing process easier),

- traffic is divided into classes and policies are assigned to the classes.

Service differentiation is based on local port numbers and users are differentiated with respect to their network addresses. To achieve this, PortFilter and IpFilter are applied. PortFilter specifies a triple (port number, location, protocol) — the example for RTP is (6970, LOCAL, UDP). IpFilter specifies a triple (IP address, netmask, location) — the example for a client in 1.1.1.0/24 subnetwork is (1.1.1.2, 24, REMOTE).

Relative bandwidth assignment is achieved with priorities. The PriorityPolicy instances determining traffic priority (LOW, MEDIUM, HIGH) have to be applied to specific interfaces.

Figure 7.2 presents the complete topology built with domain model elements. It includes virtual appliances (:Machine, :Router), connectivity (:Interface, :IpAddress, :Switch), policies (:PriorityPolicy, :BandwidthPolicy) and filters (:PortFilter, :IpFilter).

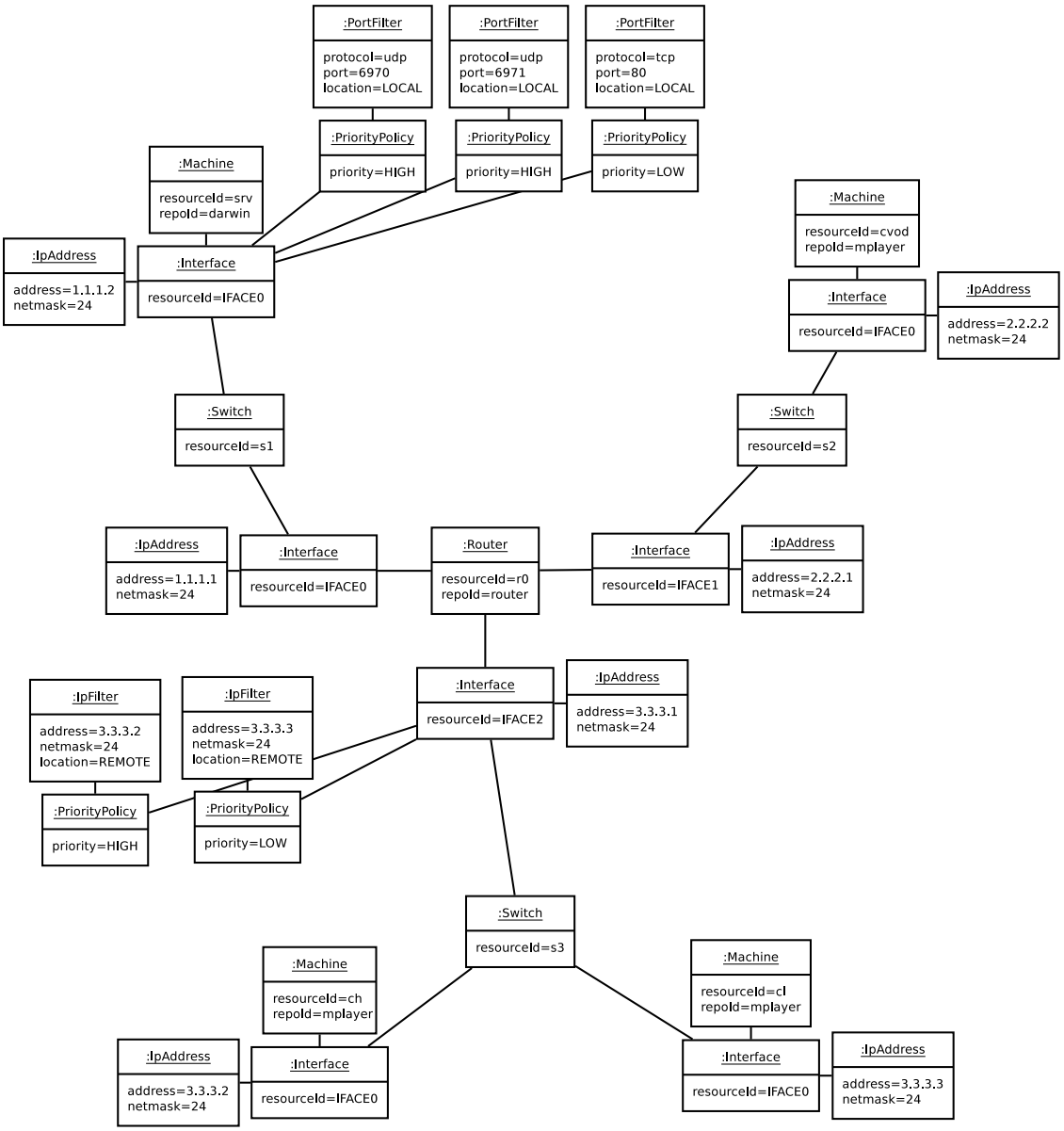


Figure 7.2: Network topology expressed in terms of the domain model

### 7.2.3 Resulting Crossbow and Solaris components

A successful model instantiation creates a number of Crossbow and Solaris entities. The resulting set contains zones, etherstubs, VNICs and flows that reflect the desired configuration. These entities work together and provide fully operational network topology.

All the entity names follow the same pattern — they are prepended with project identifier (`uc_`). There are five zones created, as shown in listing 7.5 (one media server, three clients and one router).

```
# zoneadm list -cv | grep uc_

NAME        STATUS    PATH                              BRAND    IP
uc_Mch      running   /rpool/Appliances/uc_Mch          native   excl
uc_Rr0      running   /rpool/Appliances/uc_Rr0          native   excl
uc_Mcl      running   /rpool/Appliances/uc_Mcl          native   excl
uc_Msrv     running   /rpool/Appliances/uc_Msrv         native   excl
uc_Mcvod    running   /rpool/Appliances/uc_Mcvod        native   excl
```

Listing 7.5: All the zones created after model instantiation

Each of the zones has virtual interfaces (VNICs) assigned. Flows are created for some of the interfaces. The server zone and all of the client zones are connected to the router zone with etherstubs. Listing 7.6 enumerates the etherstubs, VNICs and flows are shown in listing 7.7

```
# dladm show-etherstub | grep uc_

LINK
uc_Ss1
uc_Ss3
uc_Ss2
```

Listing 7.6: Etherstubs

```
# dladm show-vnic | grep uc_

LINK                 OVER      MACADDRESS          MACADDRTYPE
uc_Msrv_IFACE0       uc_Ss1    2:8:20:83:18:98     random
uc_Mcvod_IFACE0      uc_Ss2    2:8:20:8f:a8:d0     random
uc_Mcl_IFACE0        uc_Ss3    2:8:20:da:2a:12     random
uc_Mch_IFACE0        uc_Ss3    2:8:20:47:72:a2     random
uc_Rr0_IFACE1        uc_Ss2    2:8:20:31:2e:cd     random
uc_Rr0_IFACE2        uc_Ss3    2:8:20:ed:60:99     random
uc_Rr0_IFACE0        uc_Ss1    2:8:20:73:d1:22     random


# flowadm show-flow | grep uc_

FLOW                      IPADDR            PROTO  LPORT  RPORT
uc_Msrv_IFACE0_vod        --                tcp    80     --
uc_Msrv_IFACE0_stream0    --                udp    6970   --
uc_Msrv_IFACE0_stream1    --                udp    6971   --
uc_Rr0_IFACE2_low         RMT:3.3.3.3/32    --     --     --
uc_Rr0_IFACE2_high        RMT:3.3.3.2/32    --     --     --
```

Listing 7.7: Virtual interfaces and flows created on top of them

### 7.2.4 Media preparation

For a movie to be streamed, hint tracks have to be created. Hints are meta-data that provide information on how to stream audio and video tracks. This information is then used by the server when dividing the media into packets and sending them via the network.

The sequence of commands in listing 7.8 demonstrates how to prepare a movie to be streamed by Darwin Streaming Server (the example leverages ffmpeg[1] and mpeg4ip[2] utilities). First, the data streams are transcoded to MPEG4 (video) and AAC (audio) formats and saved in an MPEG4 container. Then, hint tracks are appended to the container.

```
$ ffmpeg -i movie.mpg -vcodec mpeg4 -acodec libfaac movie.mp4
$ mp4creator -optimize movie.mp4
$ mp4creator -hint=1 movie.mp4
$ mp4creator -hint=2 movie.mp4
```

Listing 7.8: Media preparation before streaming

## 7.3 The infrastructure operation

The traffic data is gathered as follows: each host node has tshark utility installed. It is set up to monitor all the traffic on the server interface and write it to a file (as shown in listing 7.9). When the gathering process is finished, the data is handed to wireshark and analyzed - graphs with throughput and jitter values are generated to show the interdependencies between the streams of data.

```
# tshark -i uc_Msrv_IFACE0 -w /tmp/dump.cap
```

Listing 7.9: Monitoring network traffic with tshark

The tests include verifying that the set up bandwidth limitations work on per-client basis (TODO), different service types are treated according to the policy and streaming client differentiation requirements are satisfied. The main metric used is bandwidth each of the streams is assigned. Also, jitter is analyzed to estimate the Quality of Experience the user gets when watching streaming media (TODO).

### 7.3.1 Limiting the bandwidth

Bandwidth is limited to 8Mbps for all the links. The limits can be changed online with immediate effects. Figure 7.3 shows bandwidth limitation for a VOD client downloading a movie. After a short period of transmission rate limited to 24Mbps, the link bandwidth is narrowed to 8Mbps (lowest supported value).

[1] available at `http://www.ffmpeg.org`
[2] available at `http://mpeg4ip.sourceforge.net`

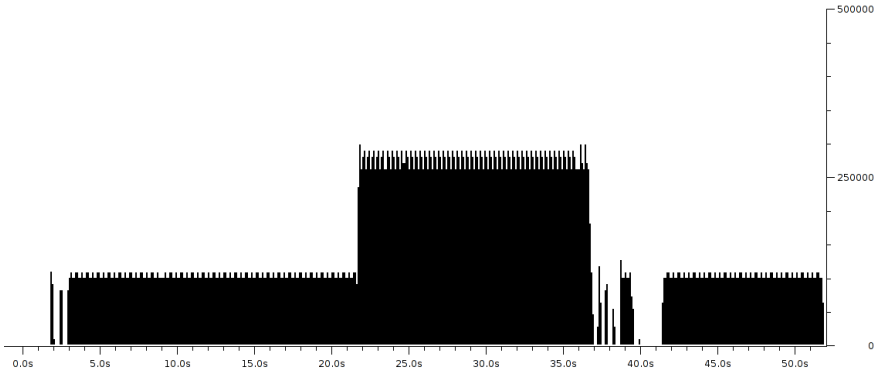


Figure 7.3: 8Mbps bandwidth limitation

### 7.3.2 Policies for different types of traffic

A VOD client is downloading a long movie. All the bandwidth is available. Another client connects to the streaming server and requests a number of video streams. As the RTP data is of high priority, the streaming client is favoured over VOD client and it gets most of the available bandwidth.

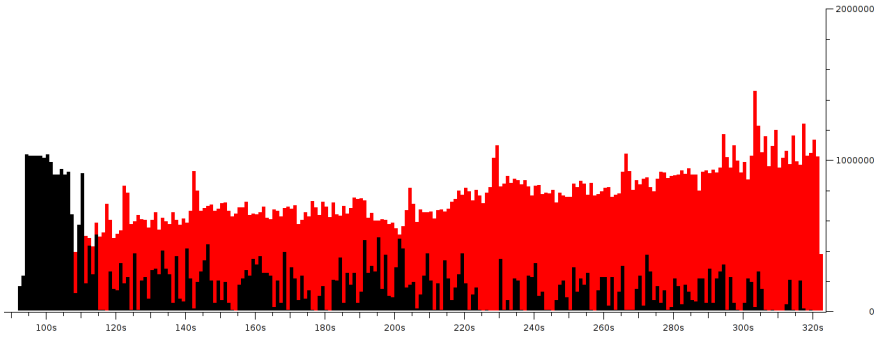


Figure 7.4: VOD traffic bandwidth consumption compared to high-priority RTP streams

### 7.3.3 Client-dependent quality of service

A VOD client is downloading a movie. Two clients connect to the streaming server and request a number of video streams. One of the streaming clients has low priority assigned, the other one is high priority. RTP streaming gets most of the bandwidth and high priority client is favoured over the low priority one.

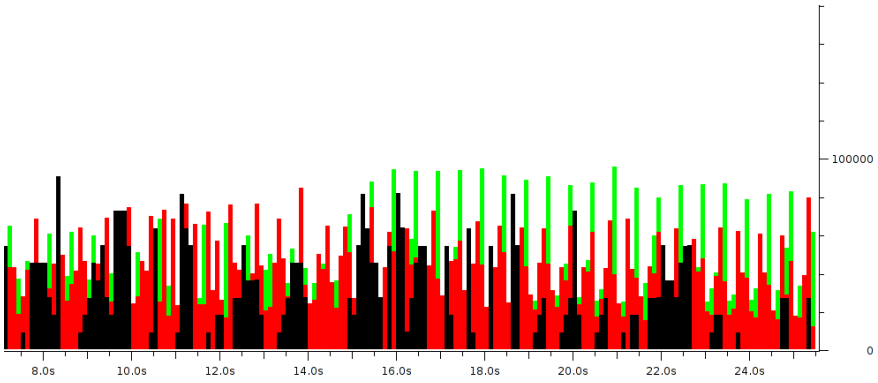


Figure 7.5: Distribution of available bandwidth between streaming clients

## 7.4 Enhancements provided by the solution

The implemented system provides extensive support for most of the stages that comprise virtual network management. There are two main goals the system is designed to achieve: to limit the time spent by the administrator to create and manage the topology and to make the process as intuitive as possible.

### 7.4.1 Topology design

The GUI console displays the topology in the form of a graph — with virtual appliances (or switches) as nodes and connections as edges. With the approach it is easy to visualize the structure of the network so that it can be quickly understood and adjusted.

All the essential aspects of the network design are configurable with GUI wizards. The system provides an easy way to set up addressing, routing and quality policies. Also, appliance repository access is integrated. Input data describing the model is validated while being entered by the user.

### 7.4.2 Infrastructure instantiation

By automating the instantiation process (ie. snapshot retrieval and transfer, zone attachment and configuration) a lot of user's time is saved. This does not only cover the time required to log in to a host system and enter commands manually — the instantiation stages, when possible, are performed concurrently and can save significant amount of time required by this process.

Input validation minimizes the risk of mistakes, especially when big topologies are considered and the whole design becomes complicated. A consistent naming scheme is provided so that the topology can be managed when the system is not available.

### 7.4.3 Online modifications

With online modification support it is easy to adjust the system without breaking its operation. The quality policies, for example, already instantiated can be changed, whenever needed.

Even more sophisticated control is possible. For example, additional rule-based component could be developed and integrated with the system to allow automatic adjustment based on statistical data.

### 7.4.4 Monitoring

Historical data can be accessed. There are customizable usage charts that can display bandwidth usage for policies and interfaces. Also, load on the host machine can be monitored to help designer assess which machines to choose when assigning the appliances.

## Summary

The chapter presented all the steps that were undertaken to ensure the system meets the identified requirements. The ability to create and manage complex network topologies was demonstrated by designing and instantiating multimedia oriented network with wide variety of components used. Validity and operation of deployed infrastructure was confirmed by the tests performed. It was shown that communication between virtual appliances is possible with properly configured routing tables. And, most importantly, the experiments confirmed that Quality of Service policies are preserved.

# Chapter 8

# Summary

## Chapter overview

Bibliography [?] test.

## 8.1 Conclusions

## 8.2 Achieved goals

## 8.3 Further work

In terms of the future work there are many improvements that might be implemented. Probably the largest component, which was initially plannned was automatic resource assigner, that would run and perform automatic resource assignments to nodes that run under lowest load. This assigner with attached rule based system would gather data about the load on each node and based on that decide what and where instantiate. Presented and discussed system in this thesis lacks that functionality. Instead, it offers manual assignments, where user selects on which node his virtual resources should be created.