Akademia Górniczo-Hutnicza
im. Stanisława Staszica

Wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki
Katedra Informatyki


Praca Magisterska

Robert Boczek, Dawid Ciepliński

# System komponentowy wspomagający zarządzanie wielopoziomową wirtualizacją zasobów sieciowych

Promotor:
prof. dr hab. inż. Krzysztof Zieliński

Opiekun techniczny:
mgr inż. Marcin Jarząb

Kraków 2011

Oświadczamy, świadomi odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonaliśmy osobiście i samodzielnie (w zakresie wyszczególnionym we wstępie) i że nie korzystaliśmy ze źródeł innych niż wymienione w pracy.

. . . . . .
podpis

. . . . . .
podpis

AGH
University of Science and Technology

Faculty of Electrical Engineering, Automatics, Computer Science and Electronics
Department of Computer Science


Master of Science Thesis

Robert Boczek, Dawid Ciepliński

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

Supervisor:
prof. dr hab. inż. Krzysztof Zieliński

Technical supervisor:
mgr inż. Marcin Jarząb

Kraków 2011

*We wish to acknowledge the support and assistance given us by professor Krzysztof Zieliński and Marcin Jarząb. Their generously contributed ideas, feedback and advice was of great help to us.*

*We also wish to thank our families and friends for their support and encouragement.*

# Contents


**1 Introduction** **10**
- 1.1 Thesis objective . . . 10
- 1.2 Organization of the thesis . . . 11
- 1.3 Individual contributions . . . 11

**2 Technological background** **12**
- 2.1 Resource virtualization . . . 12
  - 2.1.1 The need for virtualization . . . 13
  - 2.1.2 Common hardware virtualization techniques . . . 13
  - 2.1.3 Full virtualization . . . 13
  - 2.1.4 Paravirtualization . . . 13
  - 2.1.5 OS-level virtualization . . . 13
- 2.2 Multilevel network virtualization . . . 14
  - 2.2.1 Types of virtual networks . . . 14
  - 2.2.2 Virtualized network devices . . . 15
  - 2.2.3 Virtual appliances . . . 15
  - 2.2.4 „Network in a box" concept . . . 16
- 2.3 Applications and benefits of virtual infrastructures . . . 17
  - 2.3.1 Simulations and testing . . . 17
  - 2.3.2 Server virtualization . . . 18
  - 2.3.3 Infrastructure as a service . . . 18
  - 2.3.4 The role of virtualization in Service-oriented Architecture . . . 19
- 2.4 QoS-aware networking . . . 20
  - 2.4.1 DiffServ . . . 20
  - 2.4.2 IntServ . . . 21

**3 Requirements analysis** **23**
- 3.1 Domain specification . . . 23
- 3.2 Functional requirements . . . 24
  - 3.2.1 Virtual topology design and adjustment . . . 24
  - 3.2.2 Instantiation . . . 25
  - 3.2.3 Discovery . . . 25
  - 3.2.4 Monitoring . . . 26
- 3.3 Non-functional requirements . . . 26
- 3.4 Underlying environment characteristics . . . 27

**4 Solaris Operating System family** **28**
4.1 General information . . . 28
4.2 OS-level virtualization with Solaris Containers . . . 29
4.2.1 General information . . . 30
4.2.2 Zone lifecycle . . . 30
4.2.3 Isolation of processes . . . 31
4.2.4 Advantages of Containers technology . . . 31
4.3 Crossbow - network virtualization technology . . . 32
4.3.1 Crossbow architecture . . . 33
4.3.2 Virtualization lanes . . . 34
4.3.3 Dynamic polling . . . 35
4.3.4 Virtual switching . . . 35
4.3.5 Crossbow components . . . 36
4.3.6 Running examples of flowadm and dladm command . . . 37
4.3.7 Creating zone over VNIC . . . 38
4.3.8 Crossbow and Differentiated Services - interoperability . . . 38
4.4 Resource control . . . 39
4.4.1 Accounting . . . 40

**5 The CM4J system architecture** **41**
5.1 Operating environment . . . 41
5.2 Architecture overview . . . 42
5.3 Crossbow resources instrumentation . . . 43
5.3.1 Separation of concerns . . . 43
5.3.2 Layered design . . . 45
5.3.3 Instrumented Solaris OS resources . . . 45
5.4 Virtual infrastructure management . . . 46
5.4.1 High-level functionality overview . . . 46
5.4.2 Domain model and data flows . . . 47
5.4.3 System components and their responsibilities . . . 49
5.4.4 Main data flows and cooperation of the components . . . 53

**6 The CM4J system implementation** **56**
6.1 Java Management Extensions . . . 56
6.2 JMX-based Infrastructure Monitoring System . . . 57
6.3 Crossbow resources instrumentation . . . 57
6.3.1 Implementation environment . . . 57
6.3.2 Crossbow components implementation details . . . 58
6.3.3 Low-level functions access . . . 60
6.4 Virtual infrastructure management . . . 62
6.4.1 Implementation environment . . . 62
6.4.2 Crossbow infrastructure project . . . 63
6.4.3 Domain model transformations . . . 63
6.4.4 Data persistence . . . 65
6.4.5 Integration with JIMS . . . 65
6.4.6 GUI console . . . 65

6.5 Building and running the platform . . . 67
6.5.1 CM4J source code organization . . . 67
6.5.2 Building the source code . . . 67
6.5.3 Running CM4J . . . 67
6.6 System verification . . . 68
6.7 Deployment issues . . . 69

**7 Case Study** **70**
7.1 Scenario description . . . 70
7.1.1 Types of service . . . 71
7.1.2 Topology overview . . . 71
7.1.3 Service and client differentiation . . . 72
7.2 Preparation of the environment . . . 72
7.2.1 Virtual appliances . . . 73
7.2.2 Topology instantiation . . . 73
7.2.3 Resulting Crossbow and Solaris components . . . 76
7.2.4 Media preparation . . . 78
7.3 The infrastructure operation . . . 79
7.3.1 Limiting the bandwidth . . . 79
7.3.2 Policies for different types of traffic . . . 80
7.3.3 Client-dependent quality of service . . . 80
7.4 Enhancements provided by the solution . . . 80
7.4.1 Topology design . . . 81
7.4.2 Infrastructure instantiation . . . 81
7.4.3 Online modifications . . . 81
7.4.4 Monitoring . . . 81
7.5 Evaluation results . . . 81

**8 Summary** **83**
8.1 Conclusions . . . 83
8.2 Achieved goals . . . 83
8.3 Further work . . . 84

# List of Figures


2.1 Internal and external network virtualization . . . 15
2.2 Traditional application deployment stages . . . 16
2.3 Deployment process with virtual appliances . . . 16
2.4 An example of infrastructure utilizing virtual appliances with appliance repository 16
2.5 Logical Service-oriented Architecture model [14] . . . 19
2.6 Logical layers in the SOA Solution Stack [1] . . . 19
2.7 DiffServ domain [10] . . . 21

3.1 Important high-level objects and processes . . . 24
3.2 Functional requirements use-case diagram . . . 25

4.1 The variety of resources that can be virtualized with Solaris OS . . . 29
4.2 Solaris Zones high-level view . . . 30
4.3 Zone states and possible transitions . . . 31
4.4 Service consolidation within a Solaris OS instance with internal network connectivity . . . 32
4.5 The Solaris Crossbow network virtualization enhancement . . . 33
4.6 Dedicated lanes in the Crossbow architecture . . . 34
4.7 Mapping between physical and virtual network building elements . . . 36
4.8 DiffServ integration using Crossbow-provided mechanisms . . . 39
4.9 Solaris multilevel resource control . . . 40

5.1 Deployment diagram for the system . . . 42
5.2 Layered system architecture . . . 43
5.3 Manager and entity objects interoperability . . . 43
5.4 Entity creation scheme with optional publication . . . 45
5.5 Layered system architecture . . . 45
5.6 Instrumented resources . . . 46
5.7 Main stages of operation . . . 46
5.8 Object model — entities . . . 47
5.9 Object model — interconnections . . . 48
5.10 Object model — policies . . . 48
5.11 Assignments . . . 49
5.12 Actions . . . 49
5.13 Components of the system . . . 50
5.14 Topology instantiation . . . 54

5.15 Topology discovery . . . . . 55
5.16 Topology monitoring . . . . . 55

6.1 JMX architecture [15] . . . . . 56
6.2 JIMS architecture [15] . . . . . 57
6.3 Etherstub class diagram . . . . . 58
6.4 Link class diagram . . . . . 59
6.5 Flow class diagram . . . . . 60
6.6 Internal model transformation for model spanning multiple nodes . . . . . 64
6.7 GUI application . . . . . 66
6.8 GUI side instantiation process . . . . . 66
6.9 The Crossbow module registered MBeans example . . . . . 68

7.1 High-level view of the created topology . . . . . 72
7.2 Designing virtual topology using GUI console . . . . . 75
7.3 Applying QoS policies . . . . . 75
7.4 Network topology expressed in terms of the domain model . . . . . 76
7.5 Network topology transformed to Solaris components . . . . . 78
7.6 8Mbps bandwidth limitation . . . . . 79
7.7 VOD traffic bandwidth consumption compared to high-priority RTP streams . 80
7.8 Distribution of available bandwidth between streaming clients . . . . . 80

# Chapter 1

# Introduction

In today's world every successful organization is based on properly designed communication networks. These networks must deal with delay-sensitive data such as real-time audio and video or other mission-critical information. Therefore, safe, predictable and, very often, guaranteed services have to be provided. Accomplishing the required Quality of Service (QoS) by controlling delay, delay variation (jitter), bandwidth and packet loss parameters is a deeply hidden secret of most successful end-to-end business applications.

Many common networking standards did not foresee the future necessity of supporting QoS policies. Therefore, implementing QoS solutions over the Internet or even Local Area Network (LAN) is such a demanding issue. Incomplete resource reservation mechanisms provided by most common operating systems induced further research in the field of resource reservation and isolation.

The popularity of virtualization techniques is increasing. Virtualization is used heavily to partition server resources, create test bed environments for simulations and experiments and also as one of the components in Service-oriented Architecture (SOA). Moreover, grid and cloud computing models also leverage virtualized environments. All the applications encourage more research in the field.

## 1.1 Thesis objective

Having all the above facts taken into account the following thesis statement is proposed: *There exists a component-based architecture which enables construction of a system that would facilitate working with a fully isolated virtualized network resources grouped by project name.*

Due to substantial lack in similar products available on the market, research in this area has been planned. The main system responsibilities would be providing fast and efficient way of creating any requested, virtualized network structure with QoS guarantees. Solaris operating system is already a respectable solution as far as resource virtualization is considered. Recently supported by network virtualization mechanism contained in the Crossbow project, Solaris OS seems to be a suitable environment to proceed further research.

The thesis objective is to design the architecture, implement it and verify using a series of tests.

## 1.2 Organization of the thesis

The remaining part of this paper is divided into seven chapters, each containing description and discussion of particular stage of the system construction process.

Chapter 2 provides domain information about resource and network virtualization and supplying QoS guarantees for Internet Protocol (IP) networks. Virtual infrastructures are described together with an overview of technologies in common use.

Chapter 3 introduces crucial definitions of objects and processes discussed throughout the paper. Based on these definitions and constraints imposed, functional requirements and qualities of the system are identified and presented.

An overview of Solaris OS is presented in chapter 4. Special attention is paid to virtualization mechanisms present in the environment and ways to manage resource consumption policies. Crossbow — network virtualization subsystem — is then described thoroughly.

Proposed architecture of the system, named Crossbow Module for JIMS (CM4J), is introduced in chapter 5. The architecture is analyzed with respect to its two main layers — instrumentation and infrastructure — designed to provide different levels of abstraction and fulfill the requirements of the system.

Chapter 6 discusses implementation details of the system. It also lists problems encountered while implementing the system together with chosen solutions to these problems. Integration with JMX-based Infrastructure Monitoring System (JIMS) is described and system verification process is summarized.

The scenario that was designed and executed to validate the operation of implemented system is examined in chapter 7. The case study, inspired by multimedia systems, allows to verify that most of functionalities of the system are implemented properly.

Chapter 8 summarizes research performed in the field of network virtualization. Achieved goals are described together with possible directions of further improvements and functionalities that may be welcomed. Finally, objectives that were not completed are listed.

## 1.3 Individual contributions

The thesis is a result of joint effort of the authors. However, there are areas where particular author contributed majority of the content. Individual contributions are listed in table 1.1.

| Chapter | Author |
|---|---|
| 1 Introduction | Robert Boczek |
| 2 Technological background | Robert Boczek, Dawid Ciepliński |
| 3 Requirements analysis | Robert Boczek, Dawid Ciepliński |
| 4 Solaris Operating System family | Robert Boczek, Dawid Ciepliński |
| 5 The CM4J system architecture | Dawid Ciepliński |
| 6 The CM4J system implementation | Robert Boczek |
| 7 Case Study | Dawid Ciepliński |
| 8 Summary | Robert Boczek, Dawid Ciepliński |

Table 1.1: Individual contributions

# Chapter 2

# Technological background

The chapter provides background information about the domain the created system manages as well as the environment it runs in. Fields of applications and advantages of using virtual infrastructures are described. Subsequently, networking with various level of QoS guarantees, overview of virtualization techniques together with technology examples and detailed discussion of networking virtualization is presented.

Section 2.1 introduces basic information about resource virtualization. The reasons why virtualization is essential in contemporary systems management are discussed. Then, resources that can be virtualized are listed together with common virtualization techniques.

Section 2.2 presents network virtualization methods. The definition of virtualized network is provided and the types of virtual networks are listed. Also, virtualization methods for network devices belonging to various layers of the Open Systems Interconnection (OSI) model are described. The definition and application of virtual appliances to deploy services are discussed. Finally, the concept of „Network in a box" — encompassing virtualization, virtual appliances approach and QoS — is presented.

Section 2.3 enumerates some of possible applications of virtual network infrastructures. The section is focused on using the infrastructures to build complex test environments, providing scalable server-side solutions and the concept of Infrastructure as a Service (IaaS) used in cloud computing. A separate subsection is dedicated to virtualization as one of the core components in SOA and SOA Solution Stack (S3) models.

Section 2.4 describes the problem of providing QoS in computer networks of different types. The approaches chosen by Asynchronous Transfer Mode (ATM) and Token Ring networks are outlined. The remaining part of the section discusses Differentiated Services and Integrated Services — the solutions enabling QoS in best-effort IP networks.

## 2.1 Resource virtualization

Virtualization is a technique that *enables the sharing and/or aggregation of physical resources, such as operating systems, software and IT services, in a way that hides the technical detail from the end user and reduces the per unit service cost* [31]. There is a wide variety of resources that can be virtualized — whole computer platforms, software, memory, storage, data and network. Description of platform (hardware) virtualization is provided in the section.

### 2.1.1 The need for virtualization

Together with increasing hardware evolution problems with its better utilization arose. Low fault tolerance, no isolation and poor utilization of available resources implicated the demand of creating an approach overcoming this issues.

It was not until 1960s that virtualization was first used for better hardware utilization of some large, mainframe hardware. Today, over 50 years later when computers got more common than ever problems of rigidity and underutilization are still present. Numerous companies nowadays invest more and more funds in research as they foresee that virtualization will become even more popular and common in the future [32].

### 2.1.2 Common hardware virtualization techniques

There are multiple approaches to hardware virtualization (creating abstract computing platform [31] and using it to run software). The main three are full virtualization, paravirtualization and OS-level virtualization. The main difference between these approaches is the level of flexibility provided and imposed computation overhead.

### 2.1.3 Full virtualization

Full virtualization is a technological approach to virtualization also known as emulation of the underlying raw hardware. It allows to run unmodified instances of guest operating system in the virtualized environment.

Full virtualization offers the flexibility of running any guest operating system but at the same time it imposes the highest overhead [31]. Full virtualization solutions include Virtual PC, VirtualBox and VMWare software.

### 2.1.4 Paravirtualization

Paravirtualization is a technique that tries to avoid calls expensive to execute in a virtualized environments and use special Application Programming Interface (API) instead. The calls to para-API are run in non-virtual environment and therefore are more efficient [9].

Guest operating system has to be modified in order to run in a paravirtualized environment. Original calls have to be replaced with para-API ones. This substitution can be done at kernel or device driver level, or both.

Paravirtualization provides less flexibility than full virtualization, but at the same time the overhead is lower. Examples of paravirtualization solutions include Xen and Microsoft Hyper-V.

### 2.1.5 OS-level virtualization

OS-level virtualization may be often presented as slicing single OS to small partitions (sometimes called virtual environments, virtual private servers or zones). Multiple guest operating systems may be running under the same host system, but they must share the same kernel [31].

This approach is the least flexible of all presented here but it also imposes the lowest overhead. FreeBSD jails, Solaris Containers and OpenVZ are examples of OS-level virtualization implementations.

## 2.2 Multilevel network virtualization

Network virtualization is based on providing separate networking environments for defined groups of users. For each group logical environments are created over single existing physical network infrastructure. These created networks provide complete network services so that from the end-user perspective dedicated network with policies and resources is available. Network virtualization concerns not only logical segmentation of the network transport, but also network devices and other network services [19].

The problem of networking resources virtualization is multifaceted. There are many aspects at various levels that should be considered — virtualization at the Data Link Layer of the OSI model, providing software equivalents of equipment used by higher network layers (e.g. virtual routers), or creating fully virtualized networks with QoS guarantees. The section provides an overview of these techniques together with examples of technologies used.

### 2.2.1 Types of virtual networks

Virtualized networks belong to one of two main classes — internal or external [24]. Internal virtualization utilizes a single machine to create a logical network topology inside, whereas external virtualization means using existing physical infrastructure (comprising multiple nodes) to build subnetworks on top of it.

As far as internal virtualization is considered, the main building blocks are virtual network interface cards and virtual switches. These components establish logical links between virtual appliances (counterparts of physical machines). Examples of internal virtualization solutions include Xen domains, VirtualBox internal networks [8] and Solaris Containers with Crossbow technology.

External virtualization is implemented with Layer 2 switches aware of Virtual Local Area Network (VLAN) technology. With these devices, entire network is split into fully-isolated parts, each becoming a new independent network. Routers can then be used to establish connectivity between selected subnetworks.

The approaches listed can interoperate to create hybrid topologies with some parts of the network using internal virtualization and some using external one. Figure 2.1 depicts an example of a network topology that uses both internal and external virtualization.

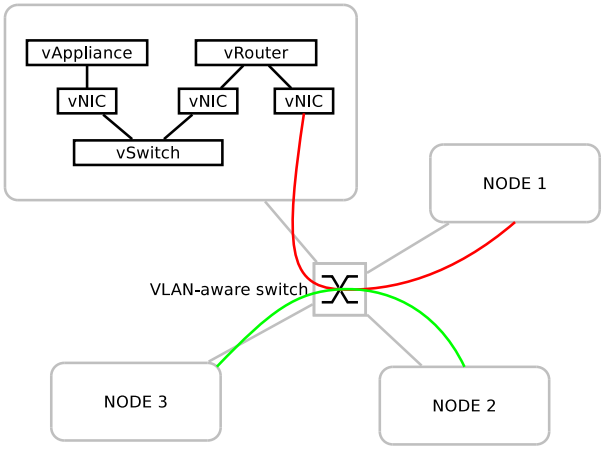



Figure 2.1: Internal and external network virtualization

### 2.2.2 Virtualized network devices

Virtualized network devices (i.e. nodes used to build network paths between boundary machines providing or consuming services) include mostly those belonging to second and third layer of the OSI model (e.g. switches and routers, respectively). However, devices that are assigned to higher layers are subject to virtualization, too.

Two virtualization approaches can be distinguished: partitioning of physical devices in order to create logical entities with assigned resources or software-based solutions used to build independent virtualized components with the same behaviour as their physical counterparts. Both methods can be applied at various layers of the OSI model. In the context of network devices, virtualization means creating both virtual as well as logical resources.

Data Link layer virtual resources can be created — these include virtual switches and virtual network interface cards built on top of them. Virtual routers connect network segments the same way as physical ones. More sophisticated examples include software implementations of firewalls and load balancers.

Logical devices require specialized hardware and the approach is used mainly in high-end solutions. VLAN technology is one of the most widely used partitioning techniques within Data Link layer — ports of a switch are grouped to provide separate broadcast domains. Another technique, used to merge geographically distributed segments and create single Layer 2 segment is Virtual Private LAN Service (VPLS) [18]. Virtual Routing and Forwarding (VRF) is leveraged to create logical routing entities on top of physical partitioning-capable routers.

### 2.2.3 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* [23]. 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 (figure 2.2). With traditional (non-virtualized) approach, these stages have to be repeated every time a service is deployed on different machines.

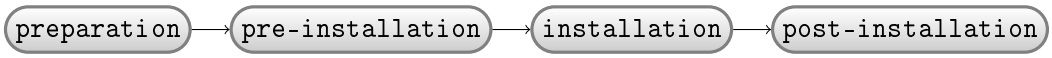



Figure 2.2: Traditional application deployment stages

Virtual appliance approach makes it possible to reduce deployment time significantly [23]. 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. Deployment process with virtual appliances leveraged is depicted in figure 2.3

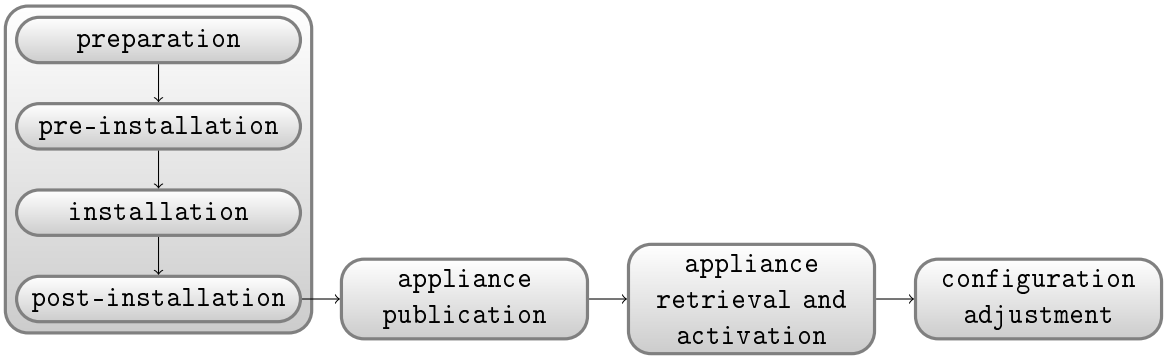



Figure 2.3: Deployment process with virtual appliances

It is possible to prepare sets of virtual appliances containing traditional services (such as application, database or media servers) as well as highly specialized networking-focused appliances that can act as routers, firewalls or load balancers.

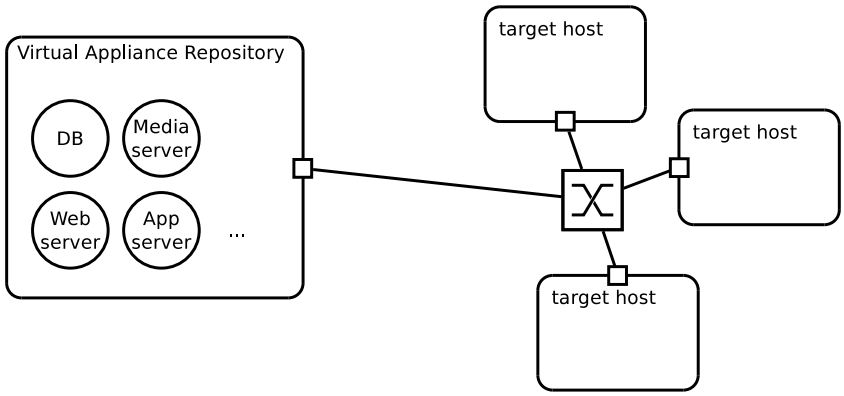



Figure 2.4: An example of infrastructure utilizing virtual appliances with appliance repository

### 2.2.4 „Network in a box" concept

The „Network in a box" term refers to virtual network topology hosted by a single machine (internal network virtualization). The topology consists of virtualized Layer 2 devices, virtual

network machines (routers, firewalls, etc.) and virtual appliances providing specific high-level services. Moreover, QoS policies are applicable to enable network traffic differentiation. The solution allows to easily consolidate network topologies of arbitrary complexity and size.

Thanks to the integration of multiple network-centric virtualization techniques, there are numerous advantages of the approach:

- the transmission is made more reliable and efficient with the use of virtualized data paths as physical links are not used and some packet processing stages can be omitted,
- policies driving the operation of virtual network machines can be easily changed with software-based implementation,
- services are deployed quickly and changed whenever needed as they are hosted on virtual appliances,
- fine-grained policies can be associated with specific classes of traffic to make the network QoS-aware,
- high level of flexibility (hard to achieve in purely-physical environment) is provided, including topology reconfiguration (virtual appliance reattachment), QoS policy management and control over physical resources consumption.

There can be multiple networks created inside single host system with guaranteed isolation. Also, the model can be extended to span multiple machines with topologies interconnected using VLAN.

There are existing virtualization solutions that support the model. The solutions differ in virtualization method used, QoS support and efficiency. Most notable examples include Solaris OS (together with Containers and Crossbow technologies), VMWare and Xen.

## 2.3 Applications and benefits of virtual infrastructures

Each Information Technology (IT) company some day faces the challenge of partially utilized servers, low level of performance, low scalability and high complexity. Although there is no perfect solution for all those issues, virtual infrastructure is an approach gaining recently more and more supporters.

Reduction of hardware, power and space requirements may be achieved thanks to accepting these approach. Apart from system efficiency growth, the effort put in system configuration is smaller.

### 2.3.1 Simulations and testing

Setting up networking environment when testing new protocols or topology operation itself can be hard or even impossible due to the costs. Even if successful, the topology is not flexible and adjustable — these two crucial requirements for simulations and experimental studies are not met.

Virtual infrastructures offer an alternative approach — the environment to be tested can be easily created, often within a single physical machine, and used to perform the simulations. The

parameters of the whole system can be adjusted whenever needed. Moreover, the approach does not require specialized hardware resources, thus it is cheaper. After necessary tests have been performed, the virtual model can be mapped directly to physical implementation, if necessary.

The impact of virtualization on software development and testing process is also important. Software development teams can use fully virtualized environments — reflecting the characteristics of actual ones — to build, test and deploy software. The resulting flexible environment can be easily duplicated or restored to some specific state when needed.

### 2.3.2 Server virtualization

There are multiple advantages of utilizing virtualization in server management. These benefits include, among others, reducing maintenance costs, flexibility and scalability.

A number of logical servers can run inside single machine at the same time with network connectivity and resource sharing (e.g. read-only file system) enabled, if needed. Thanks to this consolidation, hardware spendings are significantly reduced and the centralized infrastructure is easier to administer. Moreover, the resources can be assigned dynamically, depending on the demand, allowing efficient hardware utilization [4].

Depending on virtualization type, the host machine can run either multiple instances of the same operating system (lightweight partitioning) or independent systems provided by different vendors (full virtualization). In both cases, the instances can be fully isolated to prevent interference between applications running inside. The flexibility makes virtualizing complex heterogneous environments easier.

Replication and restoration on any target machine is simpler with virtualized servers. This property is particularly useful when designing and executing disaster recovery activities — in case of system failure the whole infrastructures can be reprovisioned immediately, allowing business continuity for crucial services to remain intact [13].

Finally, legacy systems can also be integrated as a part of virtual infrastructure. Specific software and hardware requirements of these systems can be reconstructed in a virtualized environment to allow seamless migration and interoperation with other virtualized components. The approach allows to reduce substantial expenses associated with legacy systems maintenance that many IT organizations bear [3].

### 2.3.3 Infrastructure as a service

IaaS refers to providing hardware (network, storage, computing resources) and software (operating system) as a service — the service provider owns the underlying equipment and is responsible for housing, running and maintenance. The infrastructure can be highly personalized with respect to network topology, QoS policies, computing power or provided software. The user is charged depending either on the contract signed or resource consumption (pay-per-cycle) [28].

The on-demand infrastructure has to fulfil the requirements of flexibility and scalability. Thus, ability to use virtualized resources when providing IaaS is crucial, both for customer and provider. The customer has access to isolated environment that — as far as usage is considered — is not different from a physical one and that can be expanded as needed, whereas the provider consolidates the services and makes efficient use of their hardware resources [28].

### 2.3.4 The role of virtualization in Service-oriented Architecture

SOA is an architectural approach with main focus on building IT systems as sets of loosely coupled services linked together and utilizing common communication infrastructure to inter-operate. Given a business process, a service can be thought of as a repeatable task within the process. Services expose their interfaces, hiding implementation-, organization- or time-dependent details. It is important that the design of the SOA-compliant system reflects the design of the business process [14].

Virtualization plays important role supporting the SOA. The Infrastructure Services part of the logical model (depicted in figure 2.5) uses virtualization extensively to provide secure, isolated, flexible and reliable execution environment to deploy and run services. Moreover, hardware usage can be optimized with single machine hosting multiple virtual environments [14].

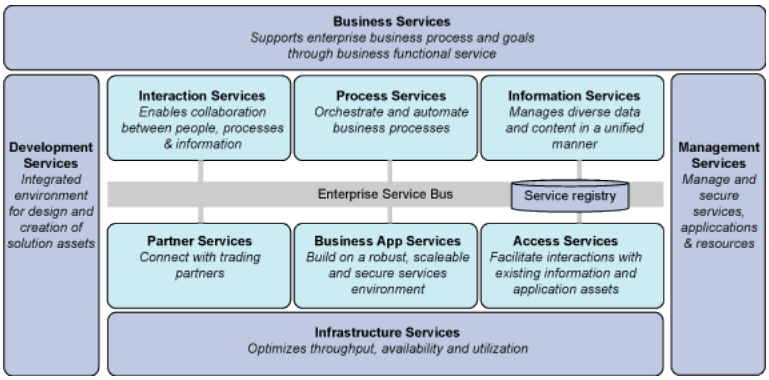



Figure 2.5: Logical Service-oriented Architecture model [14]

S3 (figure 2.6) is a high-level model depicting the functional and nonfunctional layers of an SOA solution. The S3 shows the separation of concerns of the nine layers and imposes a border between service providers and consumers. In S3, virtual appliances are leveraged to provide on-demand applications building the Operational Systems layer [1].

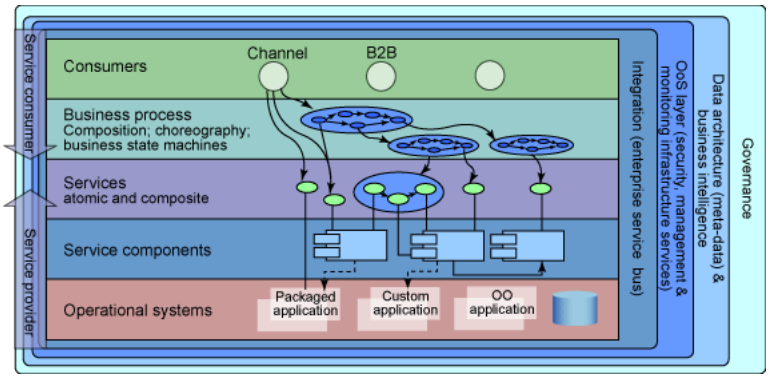



Figure 2.6: Logical layers in the SOA Solution Stack [1]

## 2.4 QoS-aware networking

QoS 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 Class of Service (CoS) field. Token Ring on the other hand is deterministic and allows using priority to distinct traffic. ATM creates virtual path between sender and receiver and sets QoS parameters. This section focuses mainly on the IP network and its approach to QoS.

Nowadays the Internet Engineering Task Force (IETF) 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 (IntServ) — reserves resources necessary to provide the service along path,

2. Differentiated Services (DiffServ) — do not require resource reservation along path

### 2.4.1 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 QoS in networks employs a small, well-defined set of building blocks from which a variety of aggregate behaviors may be built [5].

DiffServ works with traffic stream containing many complex microflows which among the same stream have the same QoS requirements and traverse the domain in the same direction.

Microflow is a flow between applications and is identified by:

1. Source and/or destination address,

2. Transport protocol,

3. Source and/or destination port.

DiffServ domain (figure 2.7) is a set of nodes with consistent QoS policy (for example network managed by the same Internet Service Provider (ISP)). Routers within the domain are divided into two groups: boundary routers and core routers. Boundary router performs traffic classification, packet marking, traffic metering, traffic control (policing, shaping) wehereas core routers pass packets according to Per-Hop Behaviour (PHB) and sometimes changes Differentiated Services Code Point (DSCP) labels [10].

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

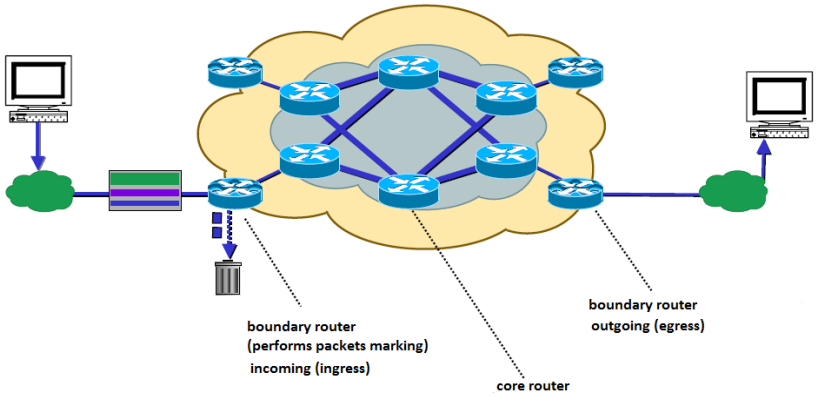



Figure 2.7: DiffServ domain [10]

### 2.4.2 IntServ

Integrated Services provide two levels of service for IP networks 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 Resource Reservation Protocol (RSVP). Thanks to this reservation session initialization lasts much longer than in DiffServ [10].

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

Table 2.1: Comparison of DiffServ and IntServ

## Summary

The chapter presented general information about resource virtualization together with examples of the approaches and implementations being in common use. The importance of network virtualization was emphasized and multiple aspects of network topology virtualization were covered — internal and external virtualization, partitioning and creating logical software-based networking devices. Also, one of the core concepts of virtual infrastructure management — virtual appliances — was introduced and its advantages were listed.

The selected applications of virtual network infrastructures confirm the importance of virtualization techniques in modern systems management, service provisioning, sophisticated software development and testing. The flexibility and scalability that virtualization offers compared

to hardware solutions makes it a natural choice when building enterprise class architectural frameworks, like SOA.

Despite the fact that the concept of virtualization is not new, it is still one of the most discussed and investigated fields in the area of computer science. Innovative ideas and solutions emerging incessantly and the impact they have on the way computing is perceived is what makes virtualization interesting both as a tool and a subject of research.

# Chapter 3

# Requirements analysis

The chapter presents initial phase of the system development — requirements analysis. The established goal is to design architecture and implement a system that supports creation of any requested, fully isolated virtual network topology with ability to monitor its operation. This objective implicates the necessity of creating requirements specification in order to avoid ambiguities and skipping essential services.

Section 3.1 introduces objects and processes discussed throughout the thesis. Formal definition is provided for each term. Also, high-level view of the domain is depicted to give better understanding of the subject.

Section 3.2 lists functional requirements the architecture has to fulfill. The requirements are grouped to reflect main coarse-grained processes.

Section 3.3 describes two classes of identified non-functional requirements. Run-time qualities (related to user goals) and development-time qualities are discussed.

Finally, execution environment requirements are gathered in section 3.4. These describe underlying environment characteristics such as isolation and preservation of broadcast domain.

## 3.1 Domain specification

To provide common vocabulary, make the requirements specification clearer and avoid ambiguities, definitions of crucial domain processes and elements are provided in tables 3.1 and 3.2, respectively. These definitions are referred to throughout the subsequent discussion.

| Term | Definition |
|---|---|
| Instantiation | The process of transforming virtual topology model to actual topology deployed on host machines |
| Discovery | Automatic process of examining existing virtual topology and recreating virtual topology model |
| Monitoring | Accessing historical information about topology operation (e.g. bandwidth consumption) |

Table 3.1: Processes and their definitions

| Term | Definition |
|---|---|
| Virtual topology | Computer network topology built using virtualized links, NICs, switches, appliances, etc. together with QoS policies defined |
| Virtual topology model | Abstract, implementation-independent description of a virtual topology |
| Flow | A class of network traffic |
| Host machine | Physical or virtual machine with an operating system capable of hosting a virtual topology |
| Underlying network environment | A group of host machines connected with LAN |

Table 3.2: Objects and their definitions

Figure 3.1 presents an overview of interoperation between objects and processes defined. The monitoring functionality provided is required to operate on the virtual topology model rather that the topology itself.

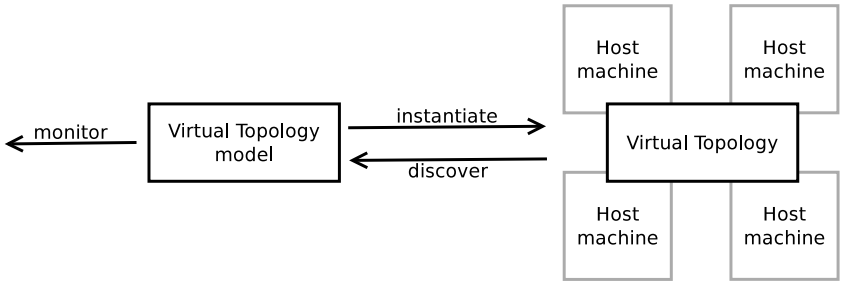



Figure 3.1: Important high-level objects and processes

## 3.2 Functional requirements

All services of the system are divided into four groups — topology design, instantiation, discovery and monitoring. This division reflects the main processes identified when working with virtual infrastructures. The groups of requirements are presented in figure 3.2. Detailed description of each group follows.

### 3.2.1 Virtual topology design and adjustment

The process of network design and adjustment is among the most important considerations. It is the main way the user is able to define and change virtual topologies. Virtual topology model is used to manipulate virtual topologies — no underlying details are exposed to the user or other high-level components.

The user designing a virtual topology from scratch or modifying an existing topology should be able to:

- create, modify, delete components of the topology (interfaces, switches, routers),
- access appliance repository and select appliances that should be a part of the topology,

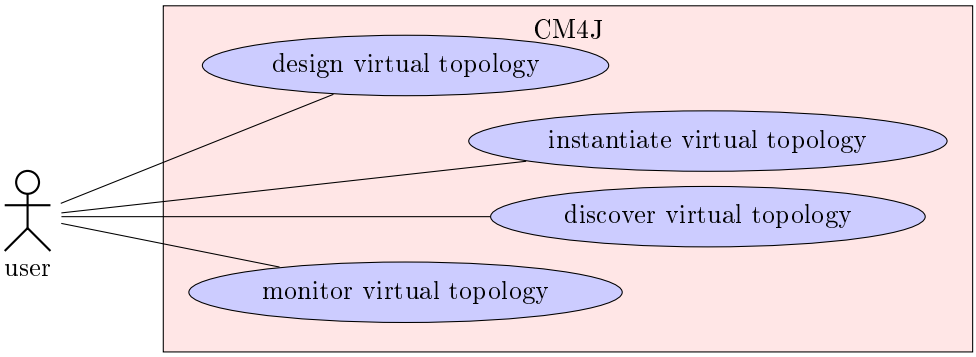



Figure 3.2: Functional requirements use-case diagram

- assign interfaces to appliances and routers,
- enable and alter routing tables to enable communication between different IP subnetworks,
- manage QoS policies (assign priorities and limit bandwidth of network traffic flows),
- assign parts of the model to available host machines.

### 3.2.2 Instantiation

The system should support virtual topology instantiation — for any correct virtual topology model it should be possible to create corresponding virtual components that implement the model. Instantiation process should:

- be automatic — no extra intervention is required from user to deploy the model,
- be atomic — either all the elements of topology are instantiated successfully or the process is rolled back,
- support deployment of multiple topologies on a single host machine,
- support topologies spanning multiple host machines,
- provide optional automatic topology assignment in accordance with predefined rules.

### 3.2.3 Discovery

Instantiated virtual topologies should be discoverable automatically. The discovery process is an inverse of instantiation. The required functionalities are:

- distinguishing multiple topologies deployed on a single host machine and returning separate models for them,
- support for topologies spanning multiple host machines.

### 3.2.4 Monitoring

When instantiated, a topology is subject to monitoring. The user still uses the model to query particular statistics so that implementation details remain hidden. Monitoring functionality should:

- allow to access interfaces, switches and routers and gather statistical data,
- provide multiple levels of granularity (bit, byte, packet),
- store the data so that it can be read and analyzed in the future,
- present statistics in a clean and readable way.

## 3.3 Non-functional requirements

Two main groups of non-functional requirements (qualities) of a system can be distinguished — there are run-time qualities (related to user goals) and development-time qualities (related to development organization goals) [17].

Main run-time qualities the system should meet are:

- usability — Graphical User Interface (GUI) should be intuitive and allow to perform operations effectively,
- configurability — configurable properties should be easy to set,
- adaptable — ability to adapt to changing conditions (unstable underlying environment),
- fault tolerance — system should be resistant to errors (incorrect input should not break the system's operation).

Development-time requirements to fulfill are:

- independent and reusable object model — the model should be general, independent of system-specific details,
- evolvability — support for new functionalities or adjustment to new technologies should be easy,
- extensibility — ability to add new, earlier unspecified functionalities; open/closed principle should be applied while designing the system,
- composability — system should be created in form of composable, highly-cohesive components,
- reusability — ability to (re)use and embed in other systems; API version dependent on required level of abstraction.

## 3.4 Underlying environment characteristics

Underlying environment should be a composition of fully independent nodes. However, there are requirements that must be fulfilled by each node to allow the system to work properly.

Chosen operating system must provide mechanisms for isolation among virtualized elements, so that one's processor or memory usage does not affect corresponding resources performing other operations. Also, there cannot be any interference between different topologies hosted on the same machine.

The broadcast domain has to be preserved, too — i.e. there has to be isolation starting from the link layer of the OSI model. Every instantiated infrastructure should preserve its broadcast domain in order to achieve network traffic isolation between different projects to fulfill the thesis statement. To achieve that, separate VLAN per project should be created or any other equivalent approach should be adopted.

## Summary

This chapter provided thorough analysis of system requirements. After the vocabulary of crucial terms have been established, all the requirements (both functional and non-functional) which the implemented CM4J is expected to fulfill were described. Also, expectations towards underlying environment were presented. In order to meet all the aforementioned constraints the system has to be implemented under OS supporting network and resource virtualization with ability to isolate infrastructure operation and restrict broadcast domains.

# Chapter 4

# Solaris Operating System family

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. Etherstub, Virtual Network Interface Controller (VNIC) and flow 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* [33]. 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 (LDOM) 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) [25] 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 [21]).

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 VLAN. Thanks to VLAN tagging support, it is possible to build systems that guarantee QoS from the lowest levels up, even for services belonging to different systems and consolidated within single physical machine.

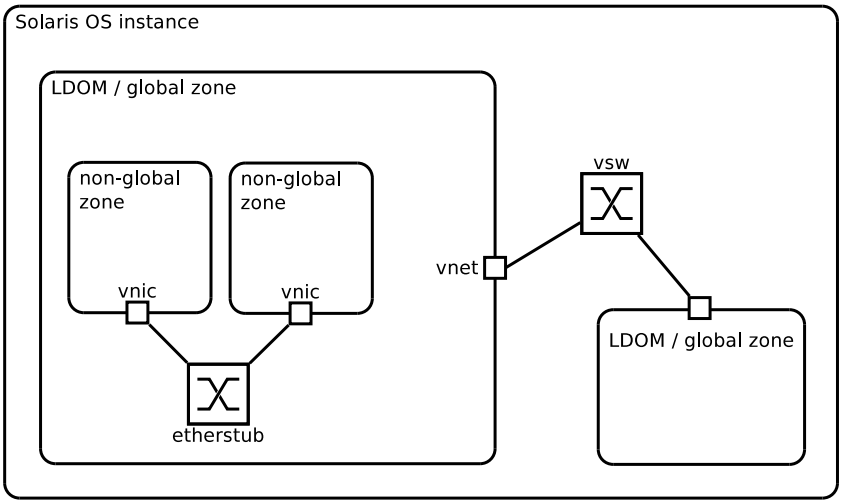



Figure 4.1: The variety of resources that can be virtualized with Solaris OS

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 [16]) or are provided by third-party manufacturers as kernel patches and utility software (OpenVZ and Linux Containers

(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 [27]. 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 [27].

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 QoS 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 [27]. The user is able to create a branded zone with translation of Linux system calls and run Linux-specific applications without code recompilation.

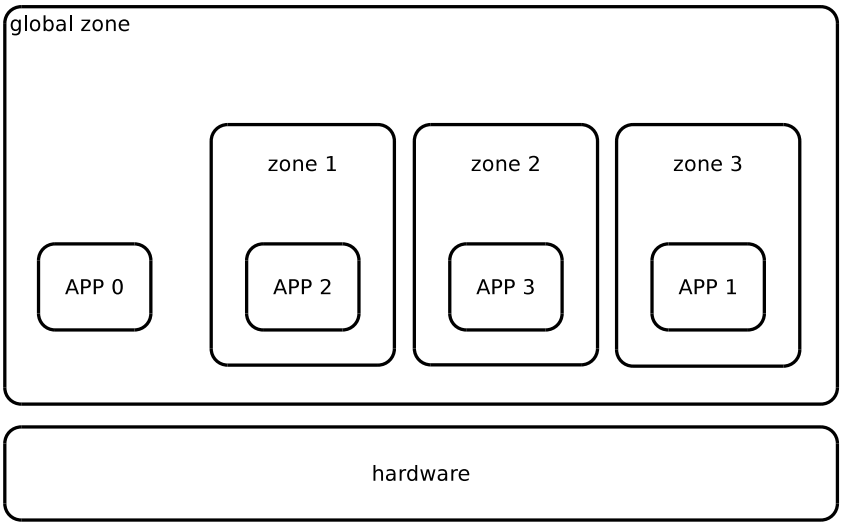



Figure 4.2: Solaris Zones high-level view

### 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* [27]. Figure 4.3 depicts the model.

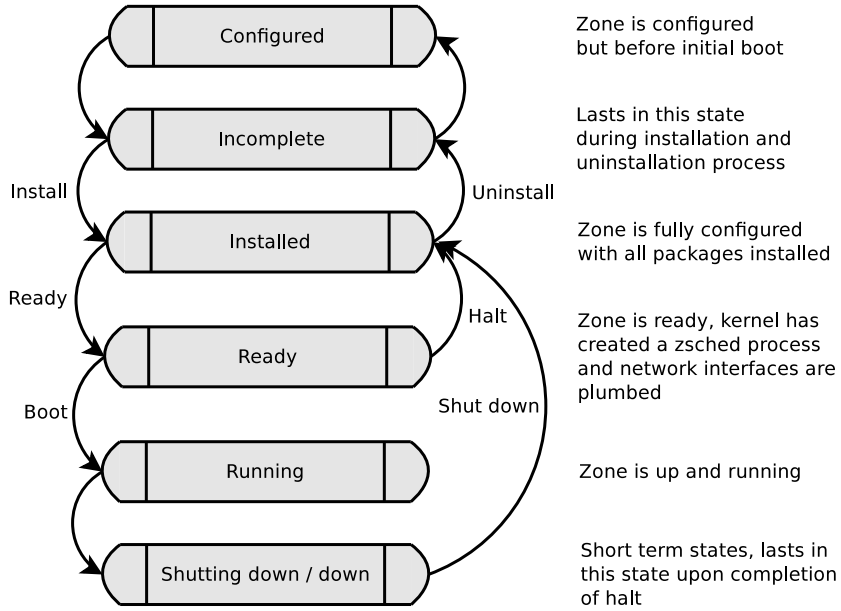



Figure 4.3: Zone states and possible transitions

### 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

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 [20], 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 stack). The

instances are able to execute applications with minimum overhead introduced mainly due to accessing commands and libraries through the lofs (loopback filesystem) [20, 26].

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 [27]. 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 [20].

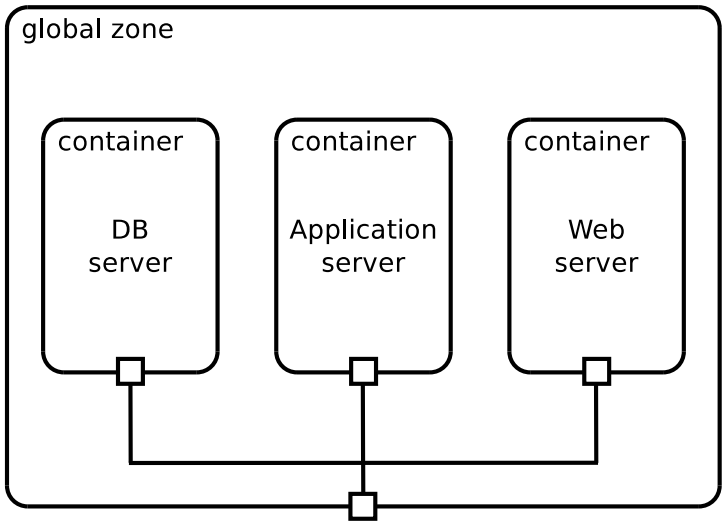



Figure 4.4: Service consolidation within a Solaris OS instance with internal network connectivity

## 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 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 [12].

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 stack where sharing is fair and interference will be avoided. These approach was adapted in the Crossbow architecture in the Solaris OS [12].

Traffic separation is achieved with fundamental blocks (VNIC) of new architecture 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 [12, 22]. New architecture after introducing new networking features combined with existing features like Solaris Containers, resource control can be presented as in figure 4.5.

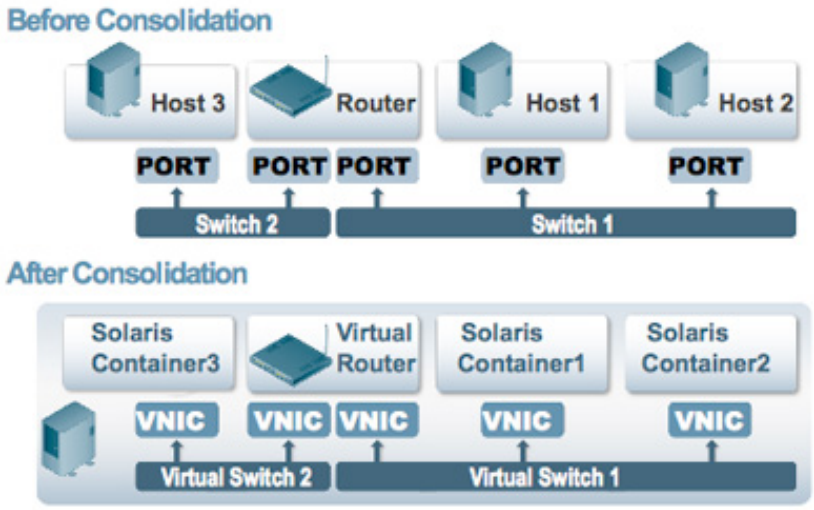



Figure 4.5: The Solaris Crossbow network virtualization enhancement[1]

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 [12].

[1] source: `http://www.net-security.org/images/articles/crossbow.jpg`

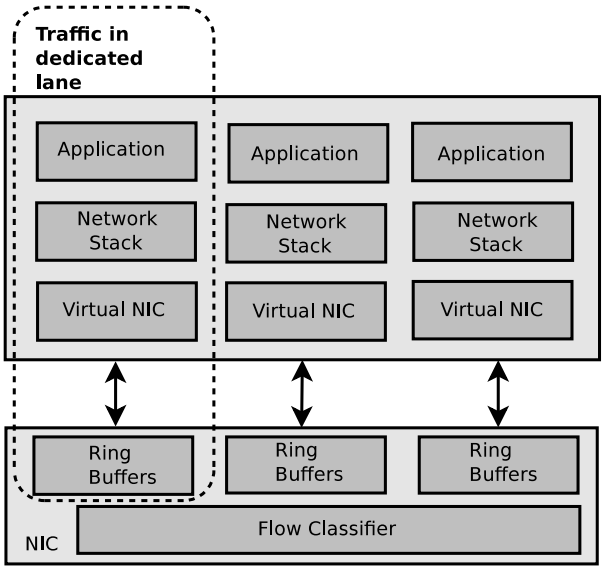



Figure 4.6: Dedicated lanes in the Crossbow architecture

### 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, Media Access Control (MAC) address slots),
2. Driver resources (Direct Memory Access (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 must have a dedicated receive ring. Other resources such as transmit lane can be exclusive or shared between lanes. Each virtual machine can 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 [12].

#### 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 [12].

### 4.3.3 Dynamic polling

The Crossbow architecture proposes 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 [12]:

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 over existing NIC and can never be accessed directly nor be visible by any user (even administrator) [11].

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 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 [11].

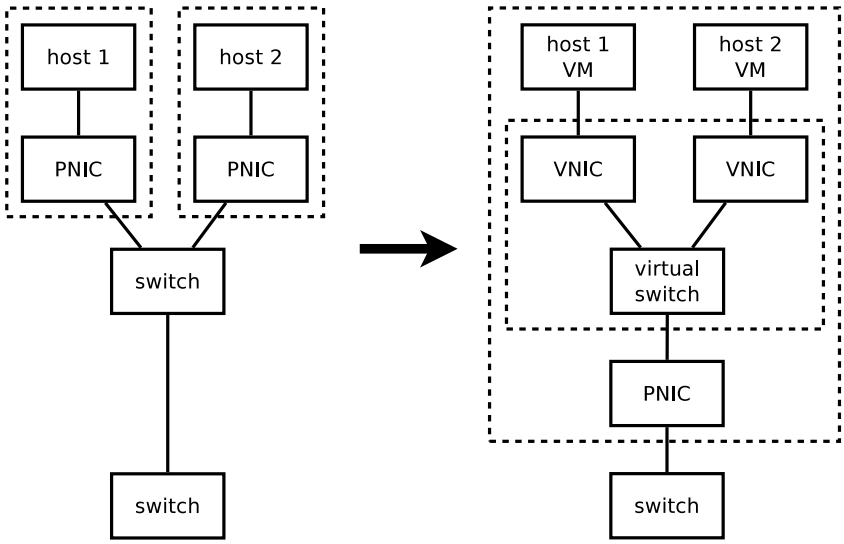



Figure 4.7: Mapping between physical and virtual network building elements

### 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

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 GARP VLAN Registration Protocol (GVRP) or Multiple VLAN Registration Protocol (MVRP) may be used to register the VLAN tag with the physical switches too [12].

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 transmitted 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 [12].

#### Etherstubs

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 [12].

**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 QoS [22].

Defined flow is a set of attributes based on Layer 3 and Layer 4 headers of the OSI 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 [29]:

- bandwidth,
- priority.

### 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.

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

Listing 4.1: **dladm** command usage examples

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

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

Listing 4.2: **flowadm** command usage examples

### 4.3.7 Creating zone over VNIC

Listing 4.3 shows an example of assigning VNIC to zones. Exclusive IP zones must be used when zones are to be used as containers for the virtual network.

```
# zonecfg -z zone1
zone1: No such zone configured
Use 'create' to begin configuring a new zone
zonecfg:zone1> create
zonecfg:zone1> set zonepath=/Zones/zone1
zonecfg:zone1> set ip-type=exclusive
zonecfg:zone1> add net
zonecfg:zone1:net> set address=192.168.1.101
zonecfg:zone1:net> set physical=vnic1
zonecfg:zone1:net> end
```

Listing 4.3: Creating an Exclusive IP Zone Over a VNIC example

### 4.3.8 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 [30]. Crossbow, unlike DiffServ, does not require special hardware, although if it is present it can boost overall operation performance [30].

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 (Request for Comments (RFC) 2475) and offers a multiplicity of classification, marking, policing and traffic shaping alternatives [2]. 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.

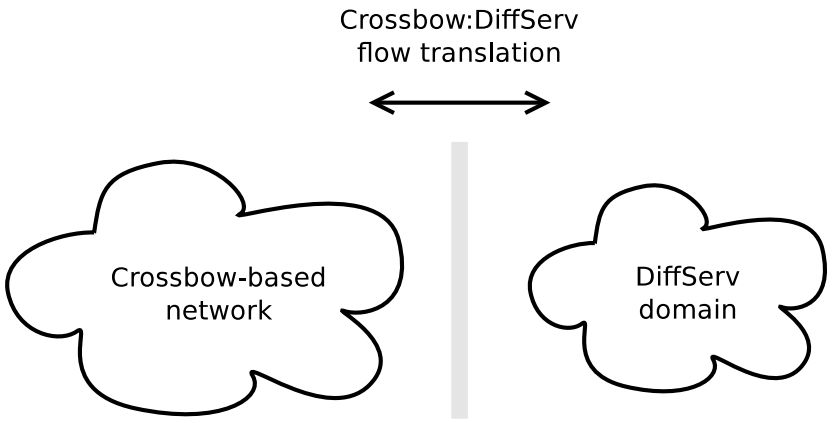



Figure 4.8: DiffServ integration using Crossbow-provided mechanisms

## 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 [7].

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 enhancements are available thanks to resource controls (rctls) facility [7].

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 [7].

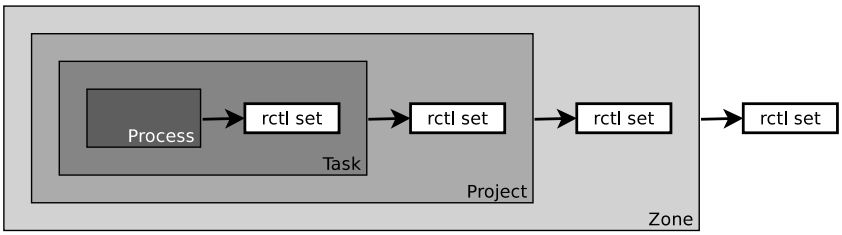



Figure 4.9: Solaris multilevel resource control

### 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 [27].

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 the Solaris operating system with regard to resource virtualization. The stack of tools integrated into the system provides an 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 a detailed view of the resource usage history.

The features mentioned above make the 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 an ideal cross-platform choice for customers dealing with the 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 on operating systems [6].

# Chapter 5

# The CM4J 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 (JVM) deployed and are capable of running Java Management Extensions (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.

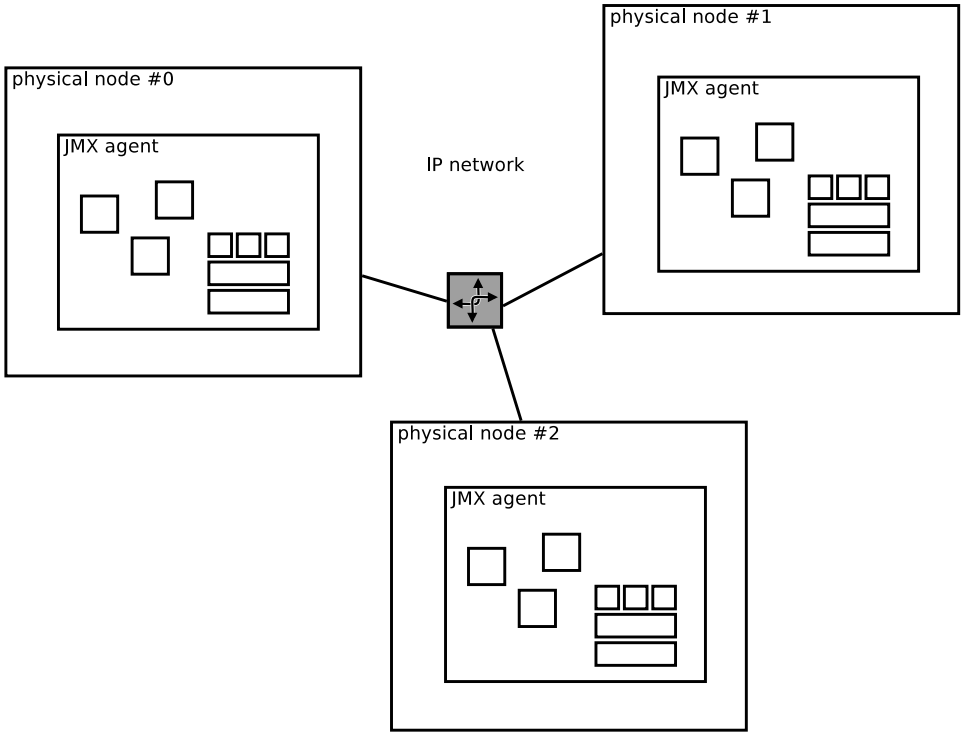



Figure 5.1: Deployment diagram for the system

## 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).

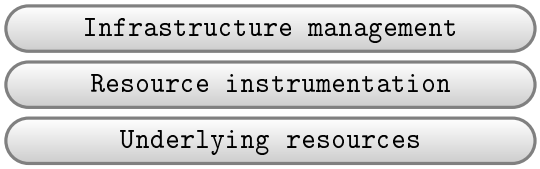



Figure 5.2: Layered system architecture

## 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.

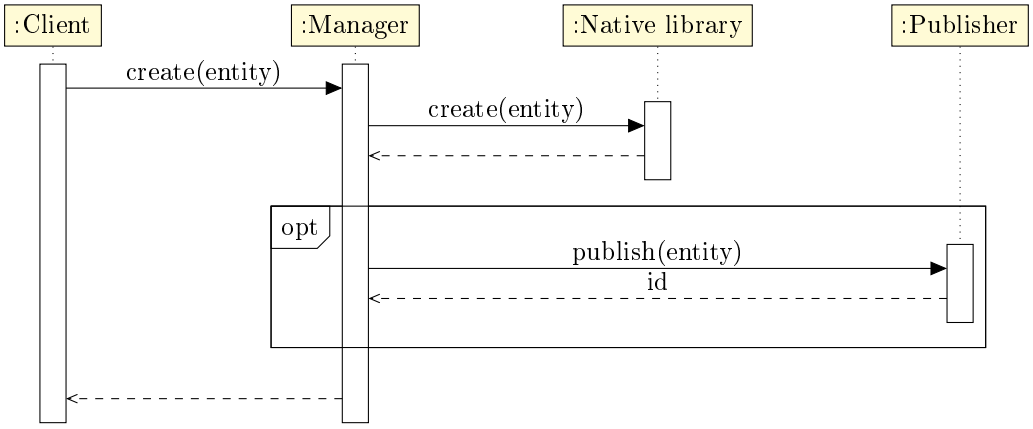



Figure 5.4: Entity creation scheme with optional publication

### 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 JMX 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.

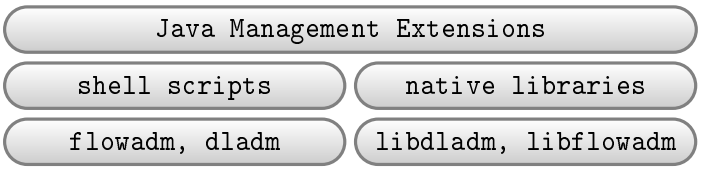



Figure 5.5: Layered system architecture

### 5.3.3 Instrumented Solaris OS resources

All the important Crossbow resources are instrumented (Manager:Entity pairs are created). This includes NICs, VNIC, VLAN, Etherstubs and Flows. All the components are loosely coupled and can be used independently.

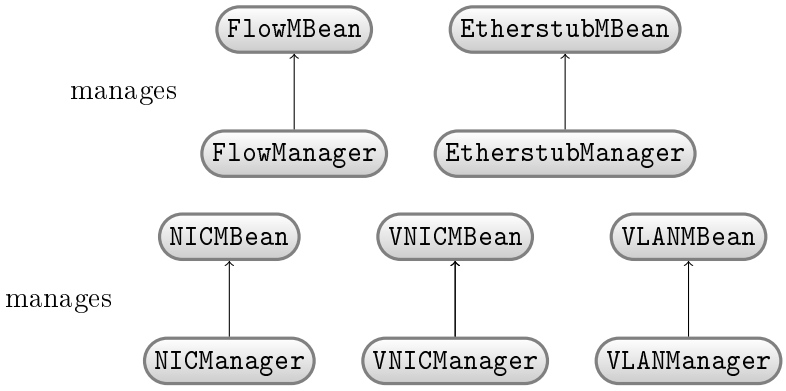



Figure 5.6: Instrumented resources

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

### 5.4.1 High-level functionality overview

Figure 5.7 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.

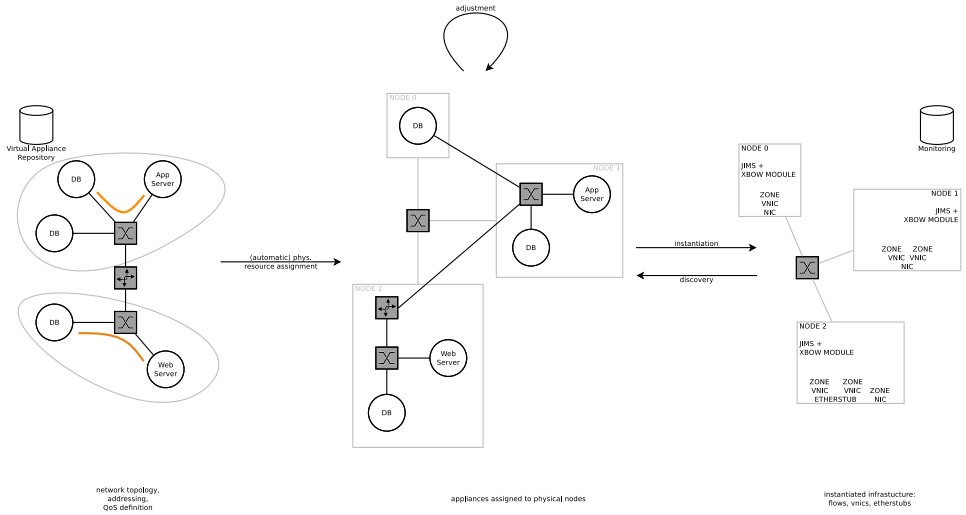



Figure 5.7: Main stages of operation

### 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,

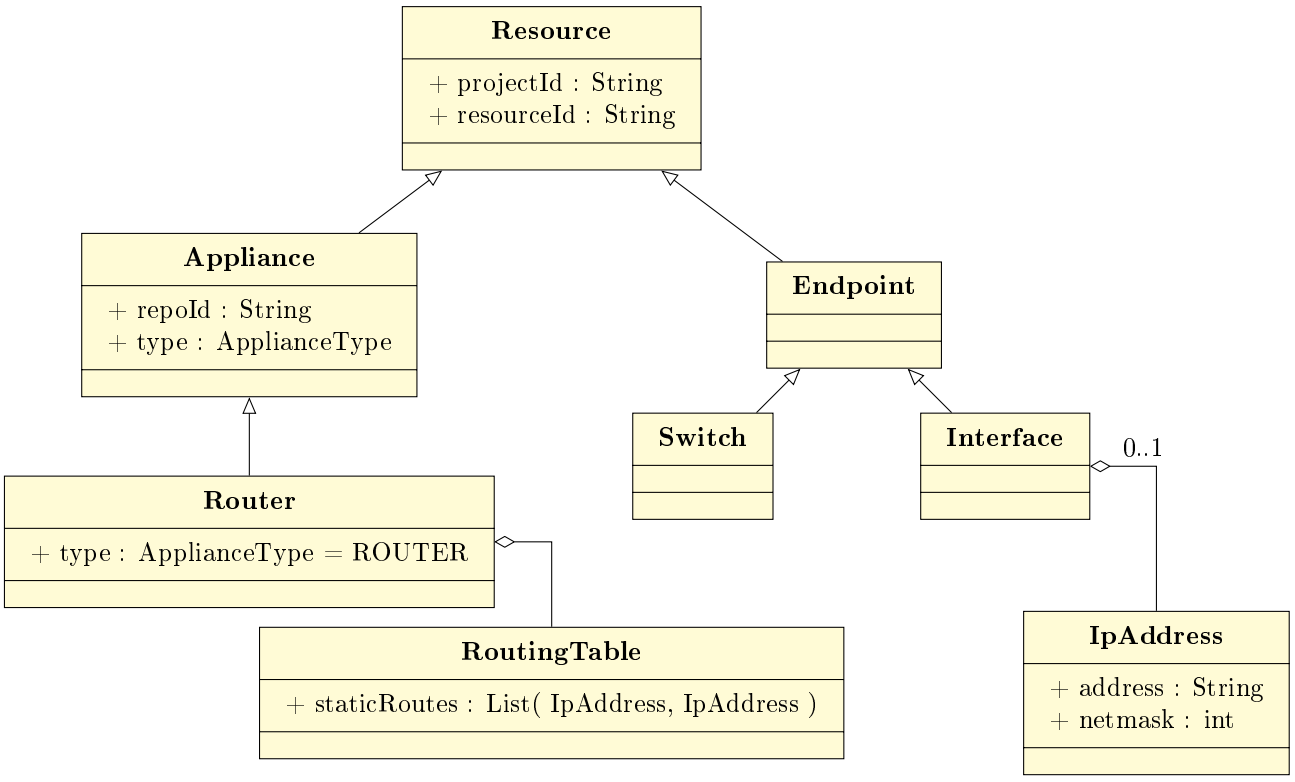
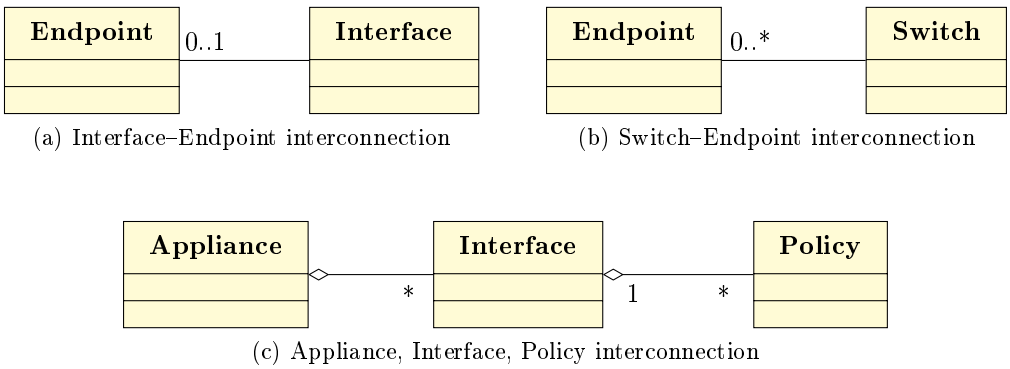



Figure 5.8: Object model — entities

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




(a) Interface–Endpoint interconnection

(b) Switch–Endpoint interconnection

(c) Appliance, Interface, Policy interconnection

Figure 5.9: Object model — interconnections

- QoS policies — classify traffic and determine priorities.

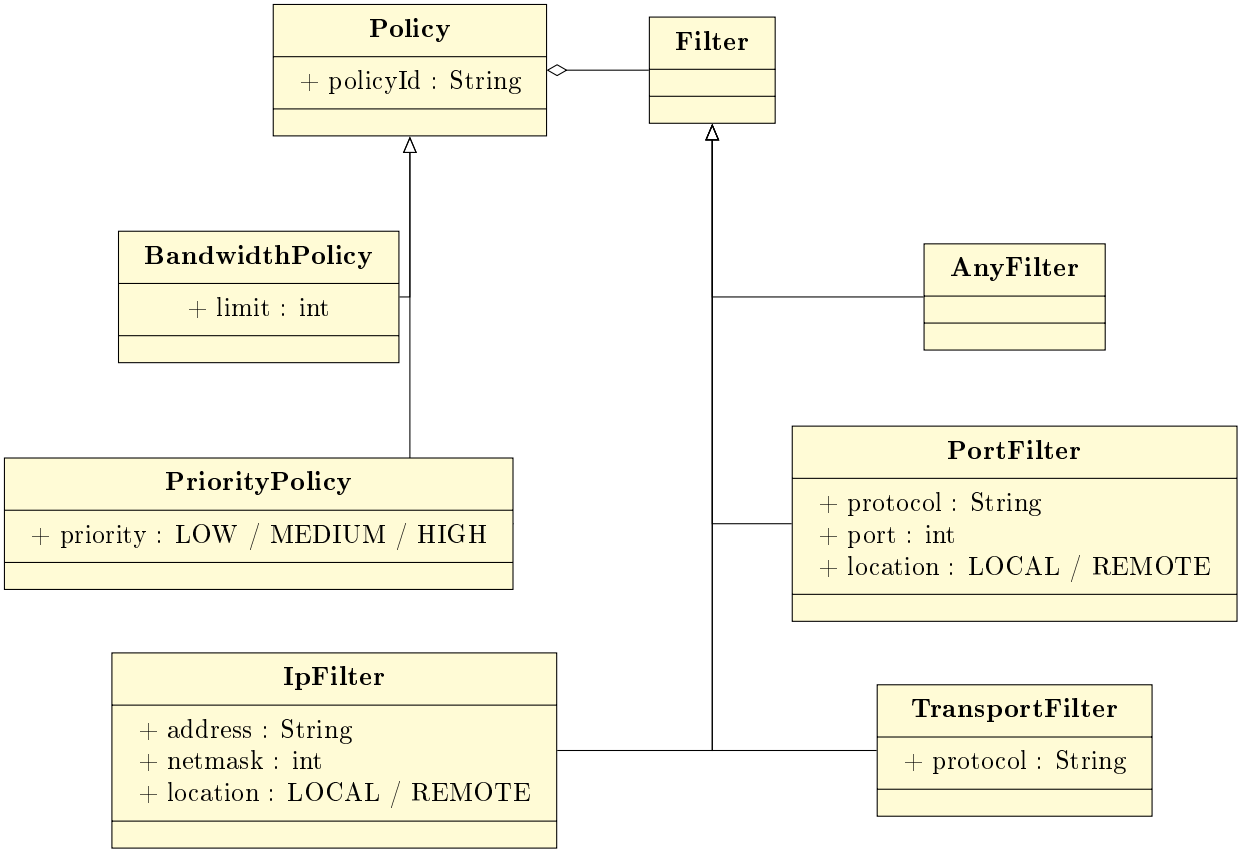



Figure 5.10: Object model — policies

### 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.

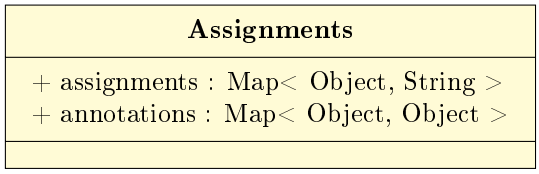



Figure 5.11: Assignments

#### 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 four types of actions designed. The object in the model can be created (ADD, as in instantiation phase), deleted (REM, typically performed after topology discovery), modified (UPD, for entities that support on-line property adjustment) or no action can be taken at all (NOOP).

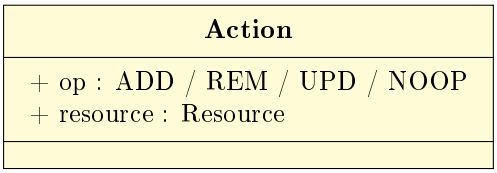



Figure 5.12: Actions

### 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 5.13 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.

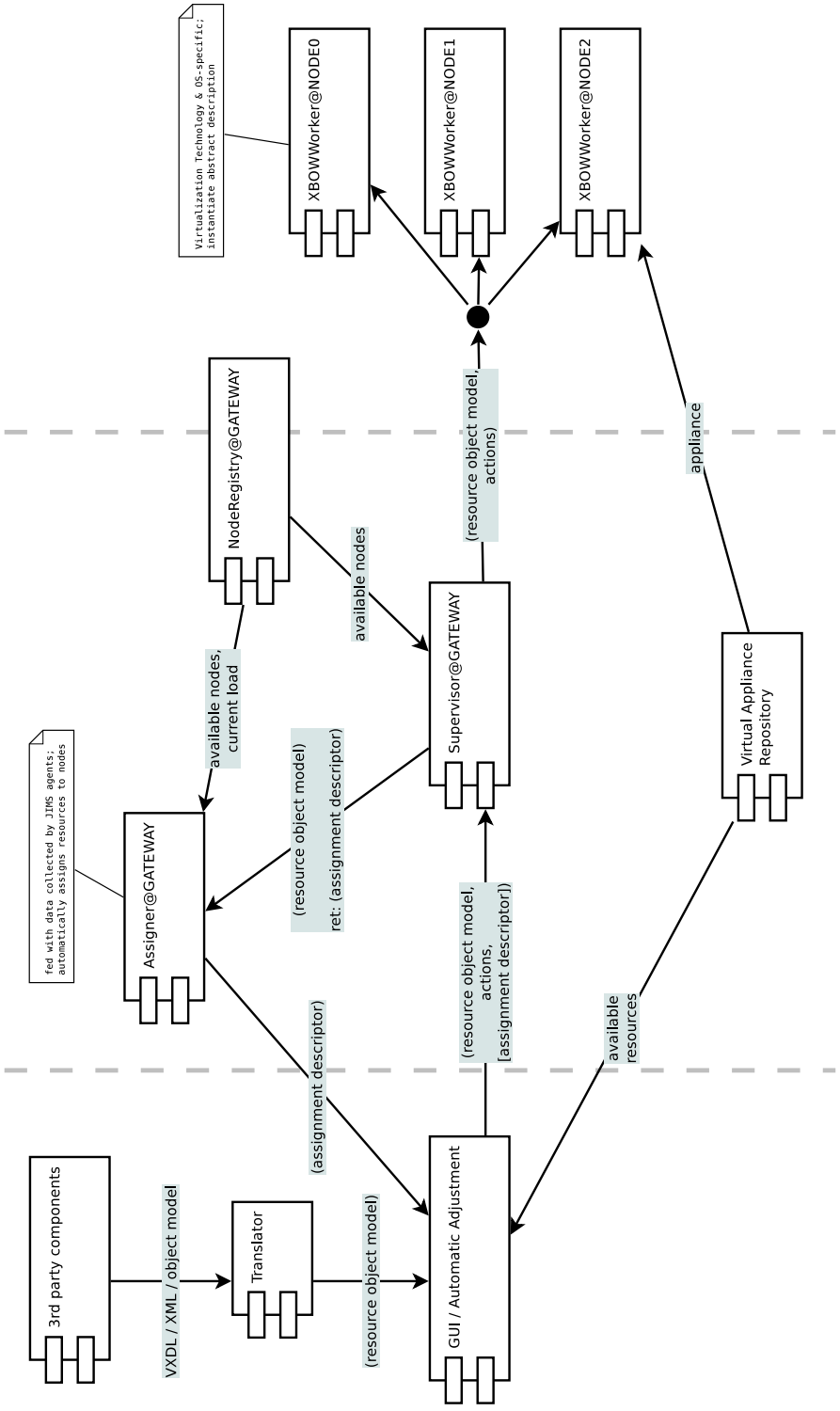



Figure 5.13: Components of the system

### 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 model) and virtual appliances — specialized virtual machines. After the topology is created, IP addressing is provided and routing set up. Finally, QoS 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.

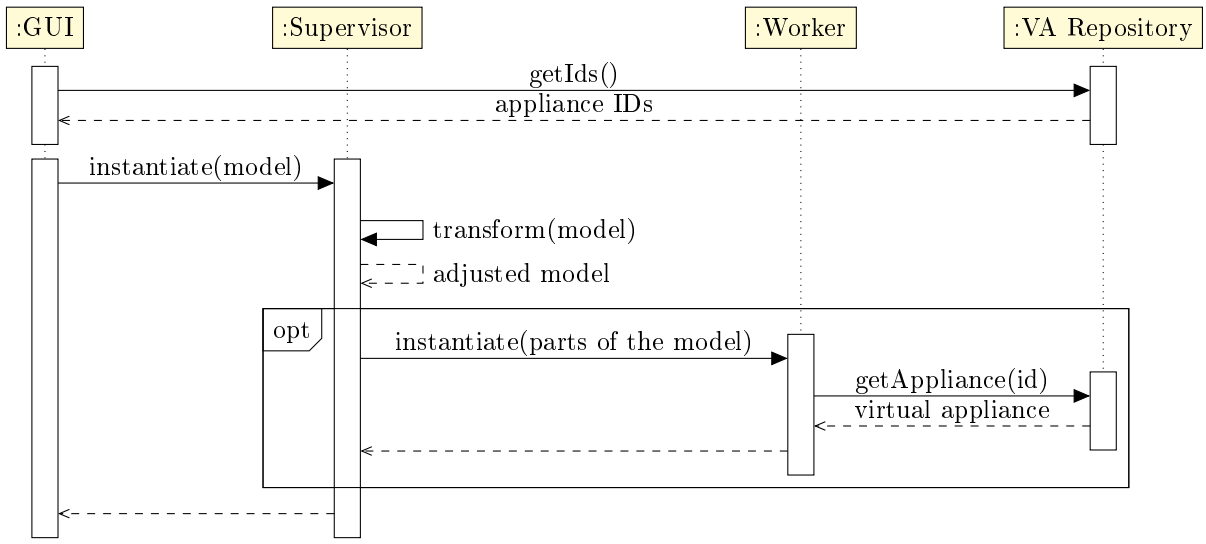



Figure 5.14: Topology instantiation

### 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 5.15):

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).

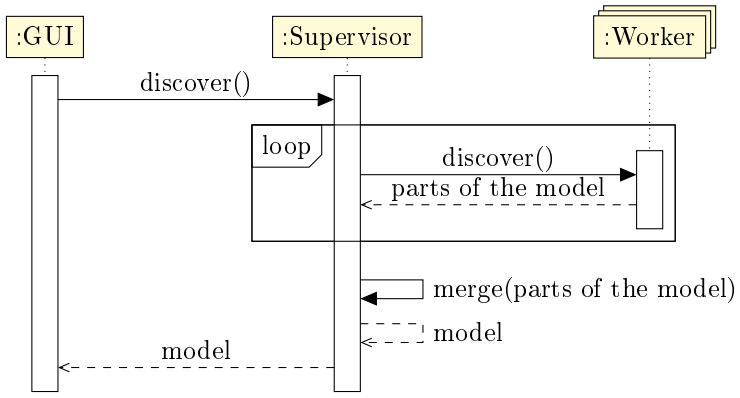



Figure 5.15: Topology discovery

### 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 5.16.

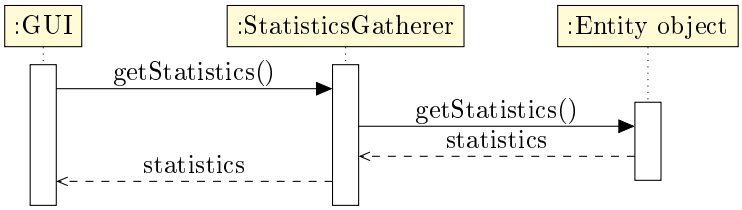



Figure 5.16: Topology monitoring

## Summary

The chapter provided the description and discussion of the system's architecture. The high-level layered design was presented and its advantages listed. Flexible and easily-expandable resources instrumentation layer was analyzed. The layer provides general-purpose abstractions over low-level system resources. Finally, the topmost layer (virtual infrastructure management) which used to build complex virtual network topologies was introduced.

The design satisfies requirements the system has to meet and ensures an easy expansion when necessary. Thanks to low coupling, new components can be integrated whenever additional functionality is needed. The created system can scale to support large topology models deployed on a number of physical hosts preserving a small amount of time needed for the instantiation process.

# Chapter 6

# The CM4J system implementation

This chapter focuses mainly on implementation details, especially on most interesting and complex problems encountered during the CM4J system implementation process like integrating with JIMS, accessing low level functions, providing multiple programming models etc.

The CM4J project's implementation structure is presented in section 6.5.1, whereas section 6.3.1 discusses implementation environment aspects like requested operating system, libraries presence or necessary programs to be built and installed. Issues like created components facilitating Crossbow usage, low level function access methods, prepared domain model transformation and adopted approach to data persistence are mainly discussed in this section.

Issues like implemented system verification, detailed description of necessary steps for build completion (6.5), load balancing problems and other potentially hard to solve issues (6.7) are described in the remaining part of this chapter.

## 6.1 Java Management Extensions

JMX is a technology for distributed resource management and control. Resources that can be managed, include applications, system objects, and physical devices. Managed resources are repre

These managed resources in JMX are represented by MBeans which are Java objects registered at MBean Server under specific ObjectName [15].

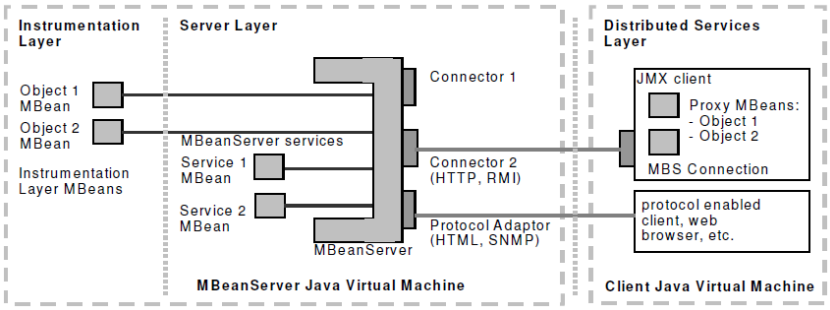



Figure 6.1: JMX architecture [15]

JMX provides also services such as:

- Notifications — allow MBeans to send asynchronous messages which inform about MBean's state change, event occurrence or other important issue,
- MLet (dynamic modules retrieval) — allow to instantiate and register one or more MBeans in the MBean Server downloaded from remote URL.

## 6.2 JMX-based Infrastructure Monitoring System

JIMS supports monitoring and management under both Linux and Solaris platforms. Thanks to JIMS features such as easy maintenance (automatic modules downloading) and extensibility (possibility of adding additional modules) the task of integrating CM4J with JIMS is straightforward [15].

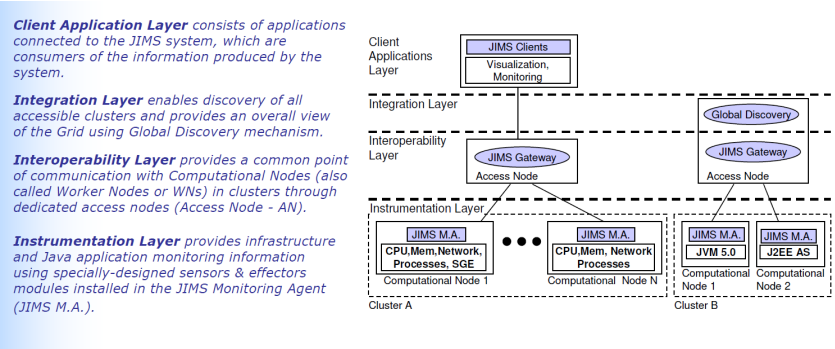



Figure 6.2: JIMS architecture [15]

The JIMS Extensions for Resource Monitoring and Management of Solaris 10 provides general architecture for monitoring and management applications written in Java. Existing JIMS services enabling creating, reading and changing properties of Solaris zones are extensively used in the system during instantiation of requested virtual topologies.

## 6.3 Crossbow resources instrumentation

### 6.3.1 Implementation environment

Crossbow components can be manipulated with either native libraries or command line tools (dladlm, flowadm) supplied with the operating system. Both methods are used in CM4J implementation — to provide coarse-grained operations, another layer of abstraction is added with higher-level C libraries and shell scripts.

To allow easy access to native code, Java Native Access (JNA) library is used. JNA mediates between Java and C code by performing all the necessary function call and data structure translations. Shell scripts can be invoked directly by Java Runtime Environment (JRE) — they are executed in separate system processes.

Classes that implement specific MBean interfaces use high-level operations exposed by JNA wrappers and shell scripts to provide the functionalities required.

### 6.3.2 Crossbow components implementation details

Figures 6.3, 6.4 and 6.5 present the way in which Crossbow instrumentation is designed and implemented. For each Crossbow component, appropriate manager and entity interface are provided. Implementations of manager interface realize NotificationListener JMX interface to provide periodic discovery operation.

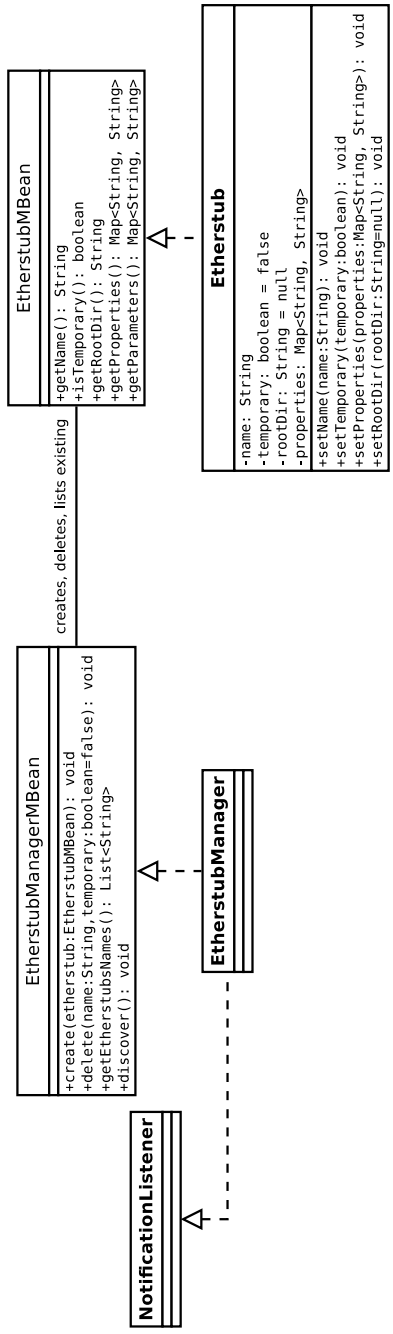



Figure 6.3: Etherstub class diagram

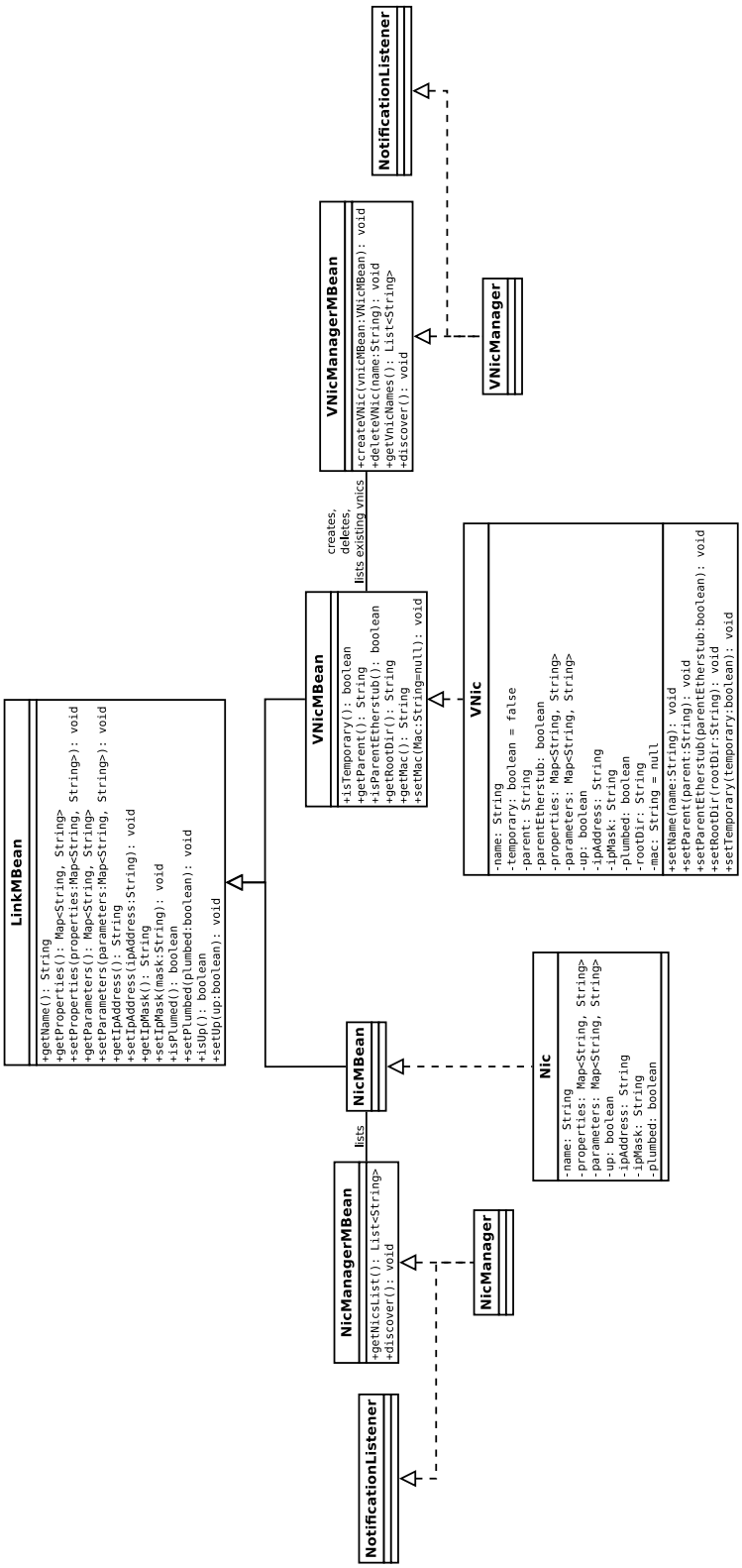



Figure 6.4: Link class diagram

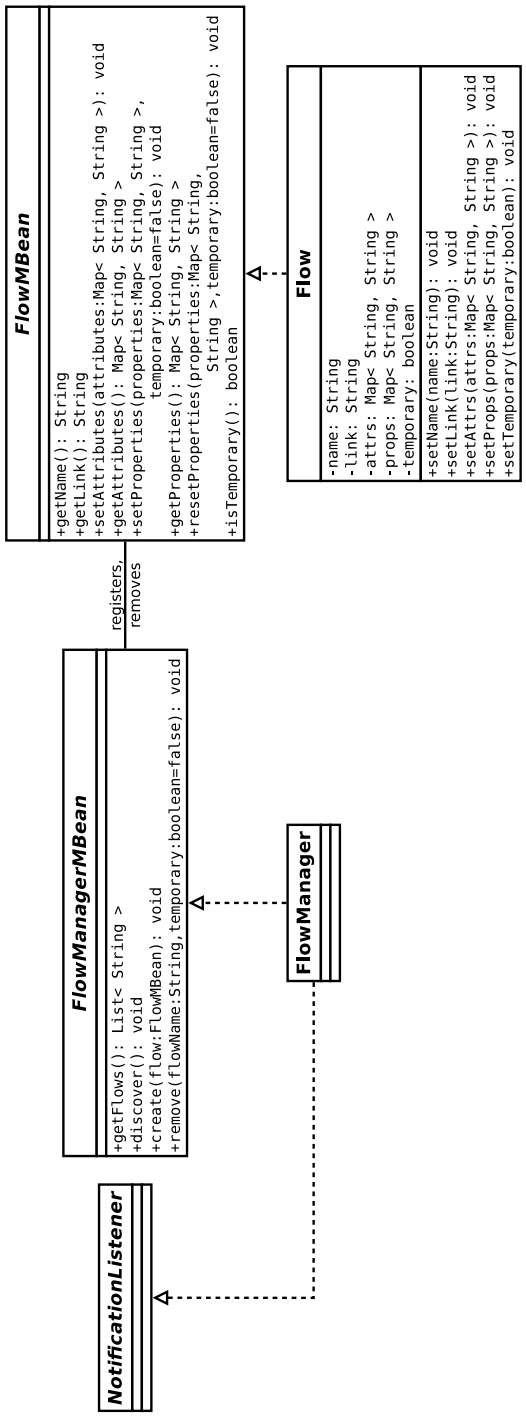



Figure 6.5: Flow class diagram

### 6.3.3 Low-level functions access

JIMS layered architecture together with crossbow dladm and flowadm library implicated the demand for an approach towards accessing low-level functions.

In developed application these approaches were adjusted to existing conditions so that for accessing functions from shared libraries JNA was used and for doing more complex low-level operations shell scripts were written. Created jims-crossbow module contains shared library allowing Create, Read, Update and Delete (CRUD) operations which subsequently are accessed by JNA, whereas most of JIMS low-level access is done through running shell scripts. Although running scripts by ProcessBuilder is faster in implementation, accessing native libraries through libraries like JNA or Java Native Interface (JNI) gives more configurational advantage and does not require shell script writing skills.

Listing 6.1 presents reduced example of JNA access to native libraries. The corresponding native code in C language is shown in figure 6.2.

```
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 (Java code)

```
#ifndef LINK_IP_H
#define LINK_IP_H

#include <link/types.h>


/**
 * \brief  Sets new ip address to link.
 *
 * \param  link      link name
 * \param  address   new address in a string format
 *                   ( for example: '192.168.0.13' )
 *
 * \return  XBOW_STATUS_OK                      on success
 * \return  XBOW_STATUS_OPERATION_FAILURE  when operation failed
 */
int set_ip_address(char* link, char* address);

#endif
```

Listing 6.2: Native library access with JNA (Native code)

## 6.4 Virtual infrastructure management

### 6.4.1 Implementation environment

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

- GCC compiler — for building jims-crossbow shared libraries,
- dladm and 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 is developed in Java using Standard Widget Toolkit (SWT) and Swing graphic libraries. The project is managed with maven and thanks to maven profiles feature can be built 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 requirement is to have one of the aforementioned operating systems together with Java SE 1.6 and Maven 2.x installed.

### 6.4.2 Crossbow infrastructure project

The jims-crossbow-module's subproject (The Crossbow Infrastructure) was designed in order to provide coarse operations like network structure instantiation, discovery, monitoring. Functionalities provided by MBeans may invoked from GUI console, JConsole or even from self written applications by just adapting to existing interfaces and object names.

Existing high-level MBean are described and discussed in table 6.1.

| MBean | Description |
|---|---|
| SupervisorMBean | Main Bean runs whole instantiation process (distributes model to parts and passes them to proper Worker |
| WorkerMBean | Responsible for instantiating on this node given model with regard to specified actions |
| RepoManagerMBean | Allows getting/setting path to projects placement, returns all existing projects from working path |
| StatisticGathererMBean | Provides statistics for interface or flow for specified time period |
| CrossbowNotificationMBean | Returns progress of deployment, logs with major information |

Table 6.1: High-level management MBeans

### 6.4.3 Domain model transformations

To allow conversion between network structure (as seen by the user in GUI console) and underlying Crossbow components, a series of transformations is performed by Worker and Supervisor nodes. These transformations include simple one-to-one mappings as well as more sophisticated multi-step operations.

#### Simple mappings

This is the class of transformations performed when instantiating or discovering abstractions that map directly to Crossbow components. These include switches, interfaces and policies. Worker component is responsible for performing the transformations. Table 6.2 contains detailed description of the mappings for each of the objects.

| Domain model class | Crossbow component | Details |
|---|---|---|
| Switch | Etherstub | The simplest mapping; no attributes set |
| Interface | VNIC | VNIC created over NIC or Etherstub |
| Interface | VLAN | Logical VLAN-specific interface; created when working with routers connecting multiple physical nodes |
| Policy + Filter | Flow | The set of flow's attributes depends on the Filter specified; Policy maps to Flow's properties |

Table 6.2: Domain model transformation (simple mappings)

### Appliance to Zone transformation

It is necessary for network interfaces to be instantiated before a zone is created. For an Appliance to be fully instantiated, a sequence of steps is performed:

1. a proper snapshot is retrieved from appliance repository,
2. the zone is configured — network interfaces are attached,
3. the zone is installed and booted,
4. routing table is populated with user-defined entries.

Appliances are discovered using the naming scheme — all the zones in the system are inspected and only these matching the naming pattern are converted to the domain model entities. Then, for each of the discovered appliances, following steps are performed:

1. routing table entries are discovered,
2. interfaces are attached (this is a part of Interface discovery process).

### Routers connecting multiple Worker nodes

Topologies that embody different subnetworks and span multiple Worker hosts require special treatment — traffic isolation has to be preserved and the network operation, from the end user's perspective, should be the same as for network deployed on a single physical machine. To satisfy these requirements, router zones with VLAN interfaces are created on Worker nodes. The transformation is performed by Supervisor component: the object model is modified by duplicating the routing entity and enabling isolated communication channels for communication between physical hosts. Parts of the model are then sent to Worker nodes to finish the instantiation process (figure 6.6).

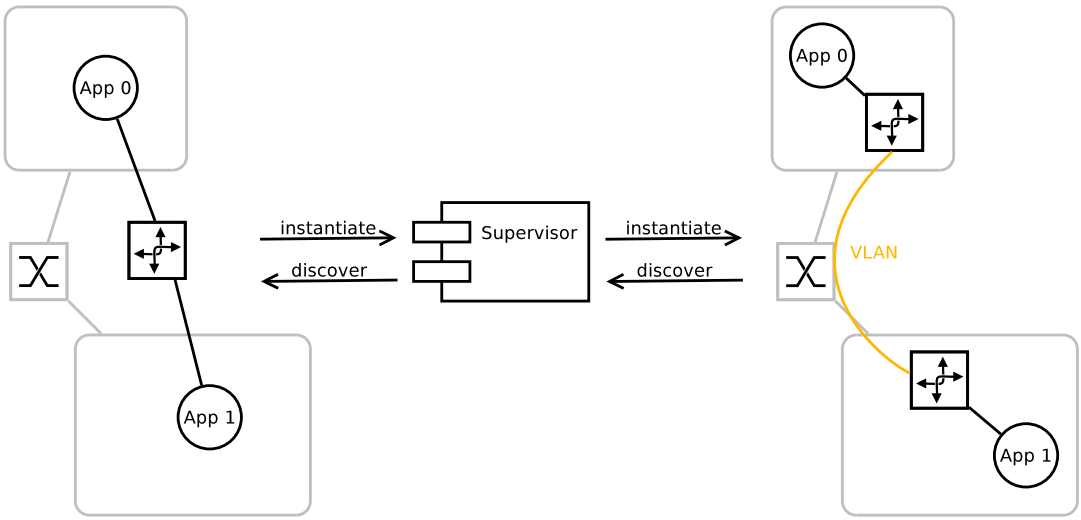



Figure 6.6: Internal model transformation for model spanning multiple nodes

In the case of discovery, the steps are performed in reverse order — Worker nodes discover, among others, router zones and VLAN interfaces and send parts of the object model back to the Supervisor. The routers (together with routing tables) are merged to create single router entity that is placed in the object model. Auxiliary routers and VLAN are then removed from the model.

### 6.4.4 Data persistence

Although created system does not provide data persistence in databases, persistence in a bit unusual way is provided. It is guaranteed on two levels: first is on GUI level where user may save created network structure that is then serialized into file and second is on the JIMS nodes. During creation, discovery, removing network elements precise naming conventions are preserved. Due to this convention whole network is persistent and does not require any additional databases or files. Since generated names are complex, manual modification or creation of elements is not recommended as it is easy to introduce make to find errors — created GUI is provided for this purposes.

With regard to statistics, flow's statistics are gathered and managed by operating system whereas VNIC's traffic load is gathered starting from every single JIMS restart.

| Domain model class | Solaris component | Naming scheme |
|---|---|---|
| Switch | Etherstub | `{project}_S{name}` |
| Interface | VNIC | `{project}_{appliance}_{name}` |
| Policy | Flow | `{project}_{appliance}_{interface}_{name}` |
| Appliance | Zone | `{project}_M{name}` |
| Router | Zone | `{project}_R{name}` |

Table 6.3: Crossbow components naming scheme

### 6.4.5 Integration with JIMS

### 6.4.6 GUI console

Developed GUI application facilitates usage of the JIMS and newly implemented the jims-crossbow-module. Apart from project verification performed in the pre-instantiation stage, GUI application is based on invoking JMX Bean operation's and presenting received responses.

Implemented GUI application allows:

- Connecting to JIMS Gateway,
- Designing desired network structure with requested virtual appliances,
- Discovering and modifying already created projects,
- Detailed information about links and flows like bandwidth consumption presented in charts for requested time periods,
- Automatic logging using Secure Shell (SSH) to selected (already deployed) nodes and opening GUI-like terminal emulator.

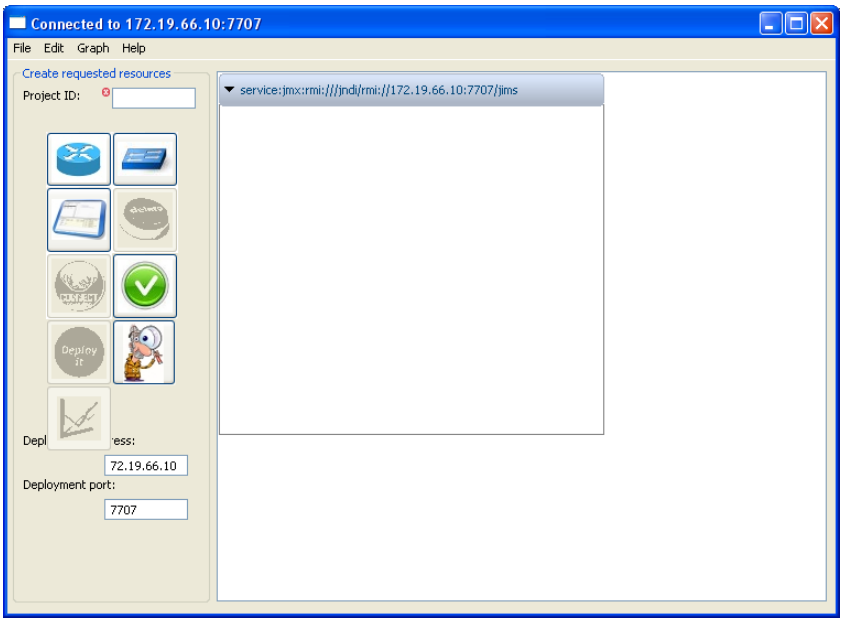



Figure 6.7: GUI application

In relation to verification on the GUI side of the system proper names, IP addresses, name duplicates and all necessary attributes are verified. Probability of future instantiation process failure is significantly reduced thanks to this verification, although not totally prevented.

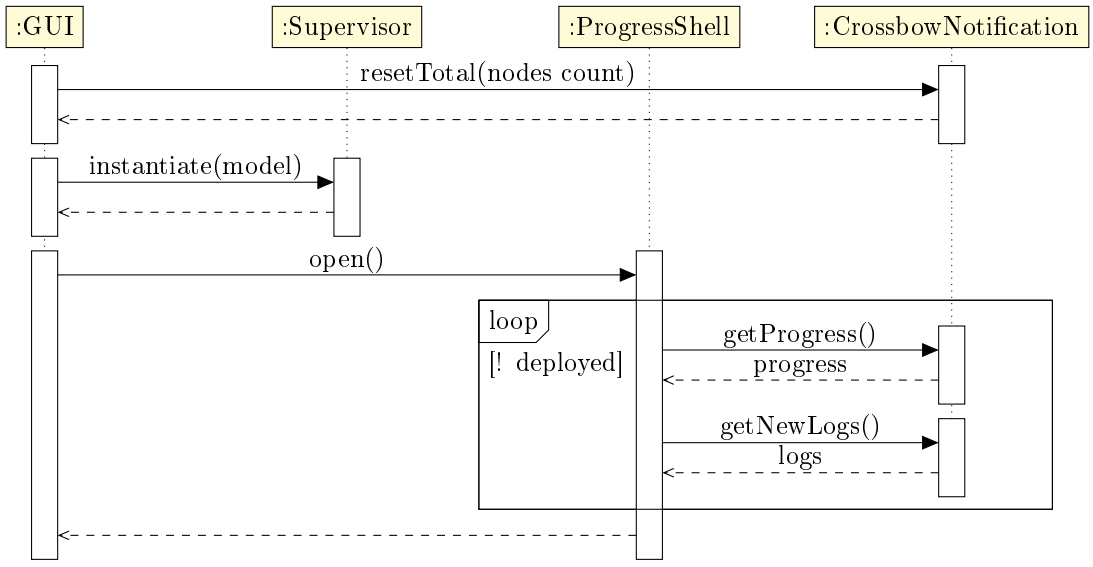



Figure 6.8: GUI side instantiation process

## 6.5 Building and running the platform

### 6.5.1 CM4J source code organization

The CM4J system consists of two subprojects: crossbow-gui and crossbow-jims-module. The crossbow-gui (based on Java, SWT and Swing technologies) was created to facilitate working with implemented crossbow-jims-module. Without the GUI, working with the system would be a lot more complex and would require specialized knowledge. Core functionalities are placed in the crossbow-jims-module subproject. During the implementation process of the crossbow-jims-module Java, JMX and C language were used.

### 6.5.2 Building the source code

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

- Java SE 1.6, Maven,
- JIMS sources downloaded,
- jims-crossbow module[1]

Afterwards JIMS project must be built. Detailed description of how to build JIMS is in README file located at the main directory of JIMS sources.

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.

### 6.5.3 Running CM4J

If everything goes 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`.

On the main node just single jims-gateway should be run and on the remaining nodes as many jims-agents as it is required. For more information about JIMS and its architecture please refer to bibliography.

If jims-agent or jims-gateway do not start, it is worth to see log files, located respectively at `jims-agent/var/jims/log/agent.log` and `jims-gateway/var/jims/log/agent.log`. It is also recommended to have the log file opened during JIMS start to see whether any exception was thrown (`tail -f 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:

[1] available at `https://github.com/robertboczek/solaris-crossbow/tree/master/code`

`service:jmx:rmi:///jndi/rmi://address:port/jims` where address and port is concrete address and port under which JIMS was started. The JConsole allows browsing registered mbeans and performing CRUD operations. Especially in case of the crossbow module it allows these operations as figure 6.9 presents.

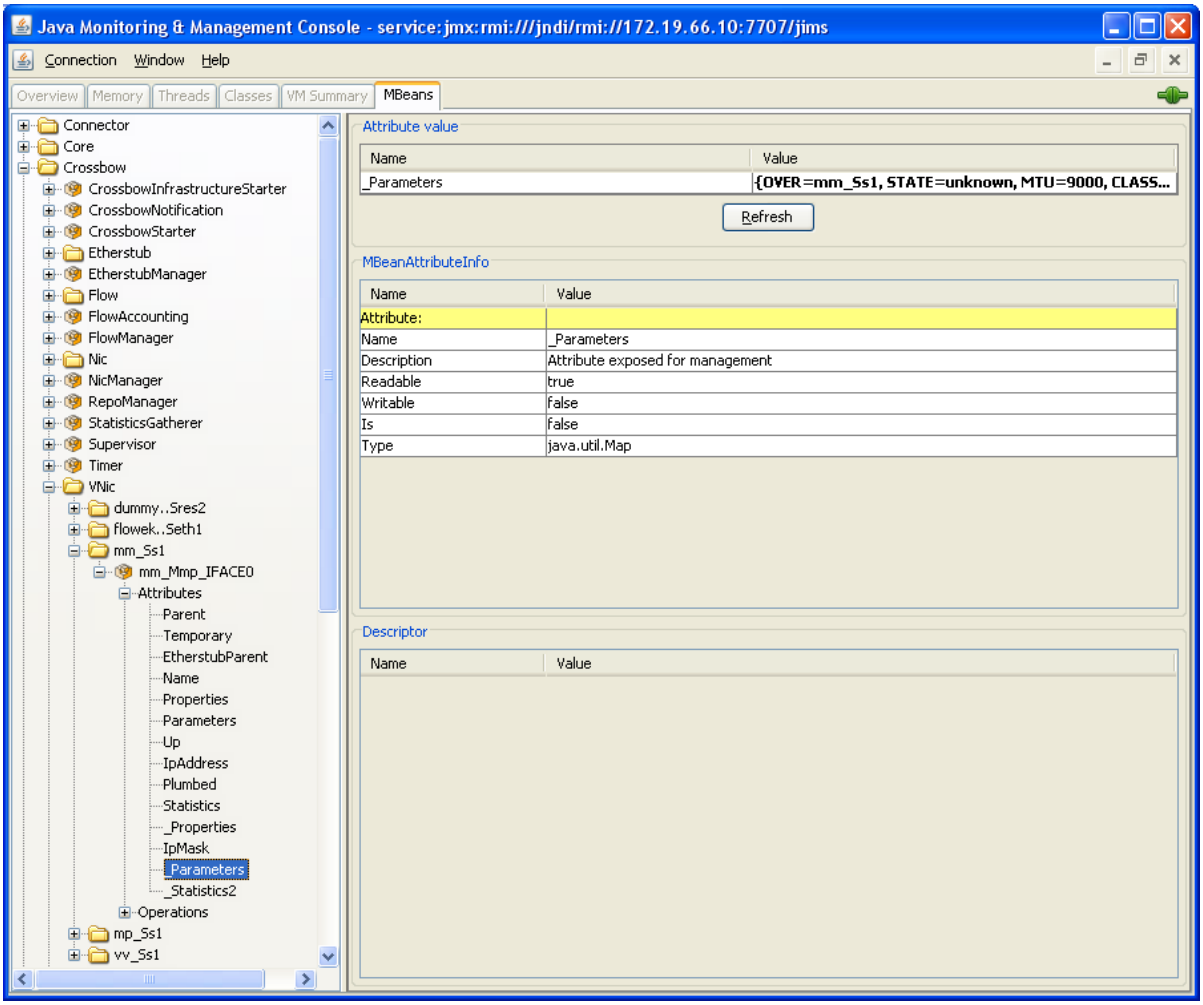



Figure 6.9: The Crossbow module registered MBeans example

Building GUI, which is located at 'jims/jims-crossbow/jims-crossbow-gui', requires Maven. Application may be imported to Eclipse and then built and run or built from console using Maven (`mvn assembly:assembly`) and executed:
`java -cp target/jims-crossbow-gui-3.0.0-exe.jar org.jims.modules.crossbow.gui.Gui`.

## 6.6 System verification

Complexity of created system implies necessity of preparing tools verifying correctness of the system. In order to achieve this verification tests on each level of our system were prepared. Starting from (low-level) shared libraries, where unit tests for methods in C were implemented,

through JMX level (with mocks) up to integration tests testing creating whole network structure. All prepared tests helped in finding most obvious mistakes and prepared system for testing with gui with lower risk of damaging underlying OS.

## 6.7 Deployment issues

wykopac do summary

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 actually 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.

## Summary

This chapter introduced chosen approach to system implementation. Despite the fact that the requested functionalities were demanding and complex, the final product meets all the previously listed functional requirements. Thanks to used technologies such as JMX, the system with fully independent business layer (loosely coupled MBeans) was implemented. These technologies also helped to fulfill the defined non-functional requirements like: evolvability, extensibility, composability, etc. Definitely one of the biggest advantage of this system (apart from all the delivered functionalities provided by CM4J) is reusability as four possible usage models exist. The System may be managed from an implemented GUI, a running JConsole or even from self developed (Java-based or native) application where developer can decide which part of CM4J would be used by just conforming to previously presented and described interfaces.

# 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 QoS-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.

Section 7.5 validates the system operation against defined requirements. All the major aspects of instantiation, discovery and monitoring processes are enumerated together with approaches chosen for CM4J design and implementation.

## 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 Video on Demand (VOD) traffic as medium-priority or even best-effort data (no Committed Information Rate (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 Hypertext Transfer Protocol (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 QoS become orthogonal and the whole design remains clear and maintainable. Figure 7.1 presents an overview of the network structure and client configuration.

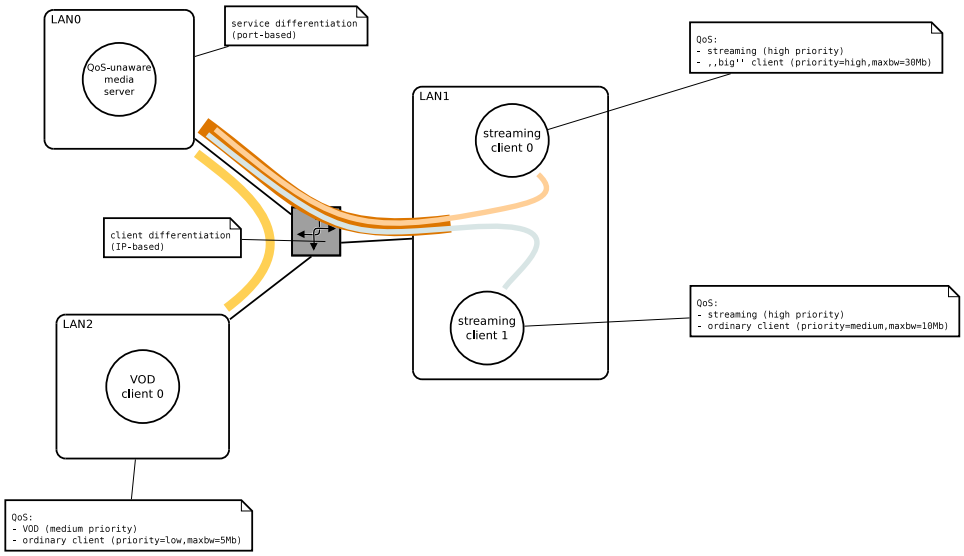



Figure 7.1: High-level view of the created topology

### 7.1.3 Service and client differentiation

The traffic is classified based on two properties: type of service (Real-time Transport Protocol (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 Network File System (NFS) repository. Then, they are used as building blocks when designing the network. Finally, the virtual infrastructure is designed and instantiated (using the GUI frontend), 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.

QoS 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 7.3 depicts the process of designing a virtual topology using GUI console. QoS policy assignment is shown in figure 7.3.

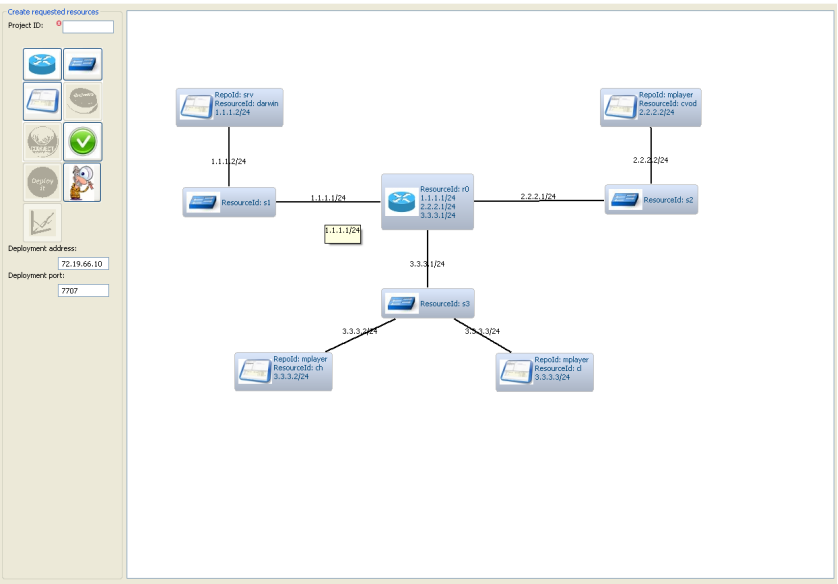



Figure 7.2: Designing virtual topology using GUI console

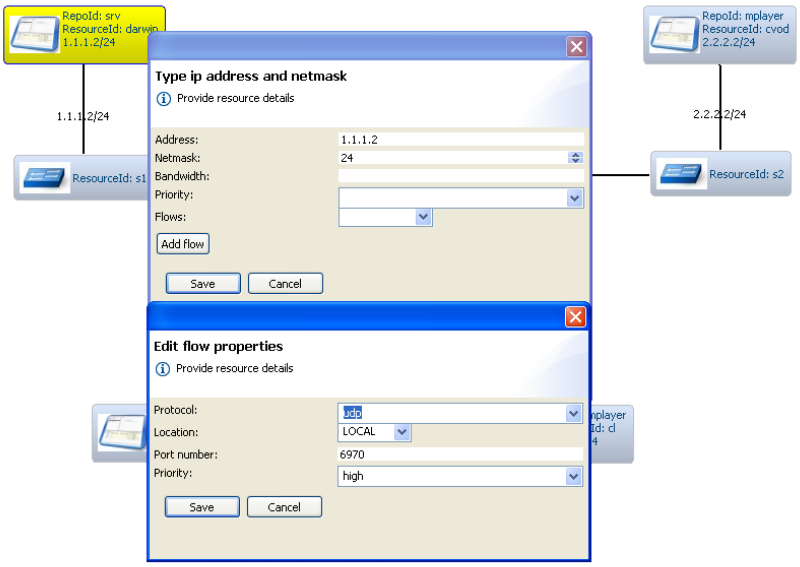



Figure 7.3: Applying QoS policies

Figure 7.4 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).

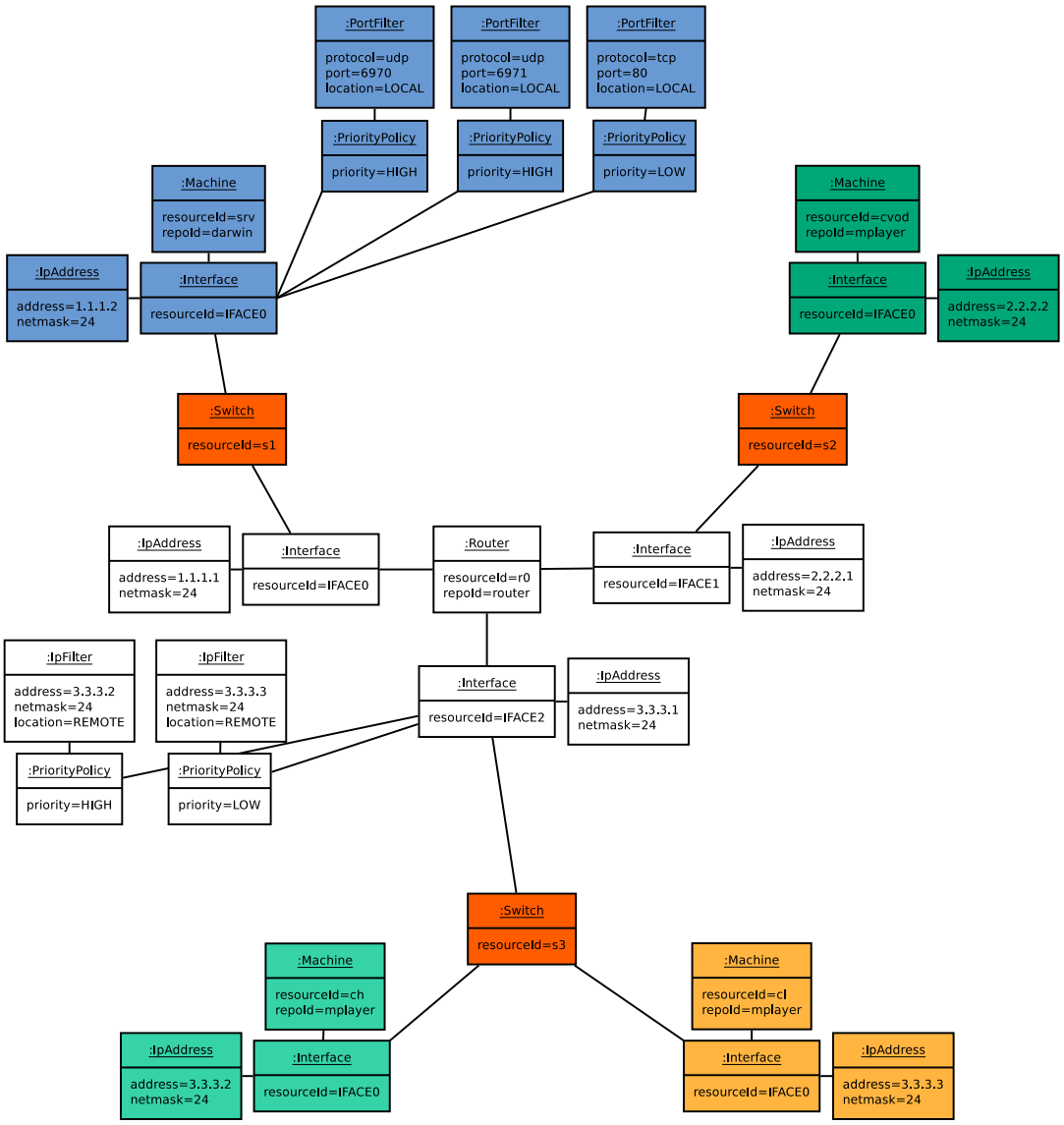



Figure 7.4: Network topology expressed in terms of the domain model

### 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

Figure 7.5 shows the interconnections between resulting Crossbow components. Colors

correspond to these in figure 7.4.

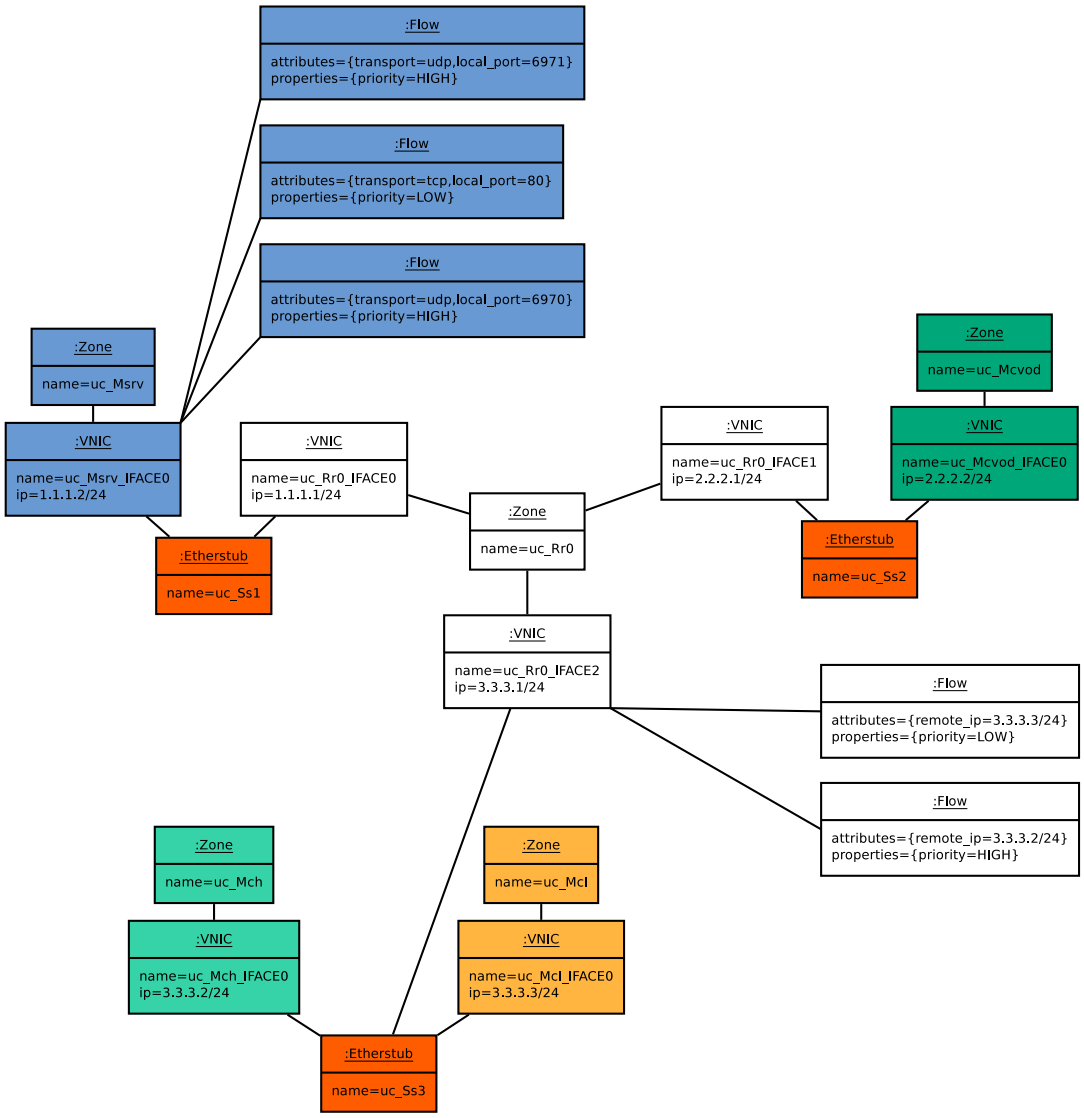



Figure 7.5: Network topology transformed to Solaris components

### 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 Advanced Audio Coding (AAC) formats

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

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 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, 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.

### 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 7.6 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).

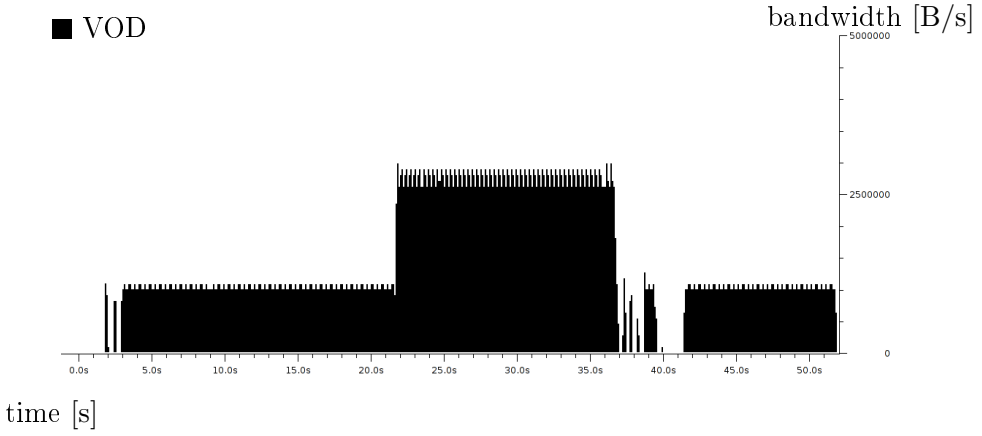



Figure 7.6: 8Mbps bandwidth limitation

### 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.

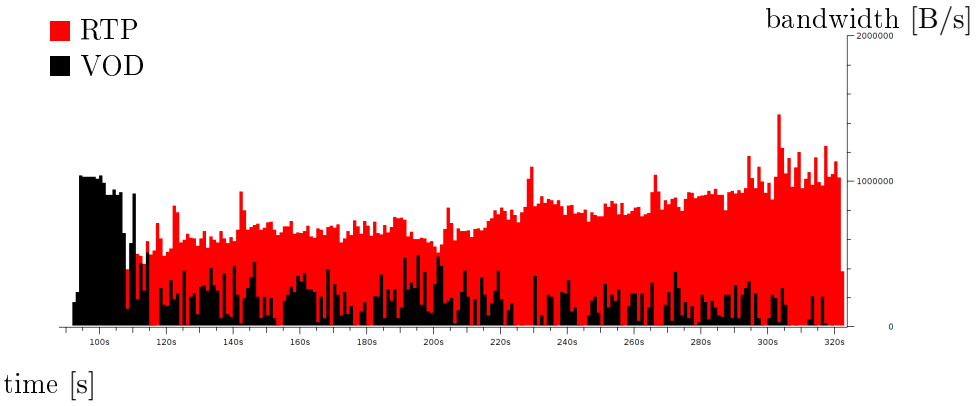



Figure 7.7: VOD traffic bandwidth consumption compared to high-priority RTP streams

### 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.

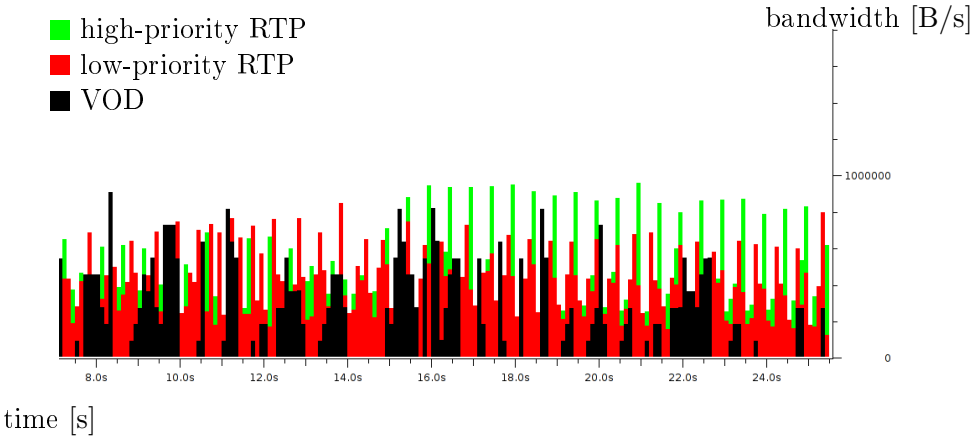



Figure 7.8: Distribution of available bandwidth between streaming clients

## 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 this 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.

## 7.5 Evaluation results

The case study confirmed the completeness of the implemented system with respect to requirements identified. All major groups of functional requirements — instantiation, discovery, monitoring — were verified while working with the topology under test.

The complete topology together with QoS policies was created with the GUI console provided. The console supports the user throughout the processes of design, instantiation, discovery, monitoring and adjustment. Graph network representation used by the GUI proved to be intuitive and allow easy management of complex networking structures.

Total deployment time could be substantially reduced thanks to virtual appliances. Created snapshots were published in a repository accessible from the GUI console and became reusable building blocks of complex topologies, minimizing the number of repetitive tasks administrator would have to perform otherwise.

The graph view user is presented maps directly to general, implementation-independent domain object model. Internal model transformations and model instantiation itself were verified to be correct. Also, the inverse process of discovery was tested — input topology was restored using only data persisted in target operating system.

As the CM4J does not use database for data persistence, topology adjustment using non-GUI tools may seem to be challenging. However, thanks to clear and consistent naming scheme, experienced user is able to tweak an existing topology manually, using utilities provided by the operating system.

Created topologies fulfill the requirements of isolation and traffic differentiation that respects defined QoS policies. This is achieved thanks to Crossbow technology itself as well as carefully designed object model transformations.

Resulting infrastructure can be managed in a multitude of ways. Low-level components are accessible with command line utilities provided by the Crossbow project (flowadm, dladm) and with higher-level native API developed as a part of the thesis. Also, JMX objects instrumenting Crossbow entities are exposed to allow easy integration with Java platform. Finally, the infrastructure, expressed in terms of the domain model, can be retrieved using the Supervisor component and manipulated with GUI console.

Although the topology presented in the chapter is complex, it does not take advantage of all the functionalities implemented. Creation of even more sophisticated designs (like topologies spanning multiple physical nodes with QoS policies and still preserving traffic isolation) is possible with CM4J.

## 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 a multimedia oriented network with a 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 the QoS policies are preserved.

# Chapter 8

# Summary

The chapter summarizes the outcome of the research conducted in the area of network virtualization with special regard to the thesis statement.

Section 8.1 briefly concludes the results of research and case study performed. Legitimacy and feasibility of the thesis statement is also discussed in this section. All the accomplished goals and objectives are listed in section 8.2. Possible improvements that might be added to the created system are considered in section 8.3.

## 8.1 Conclusions

The statement proposed at the beginning of the thesis claiming that: *There exists a component-based architecture which enables construction of a system that would facilitate working with a fully isolated virtualized network resources grouped by project name* was deeply considered throughout this thesis by the authors. In order to fulfill the proposed statement, a detailed requirement analysis was performed. Subsequently in order to comply with all the defined requirements mentioned in the thesis statement the layered architectural approach was chosen. This approach allowed to distinguish certain groups of independent, highly cohesive components. Together these components create the planned system, but each one can be used separately allowing to perform more specific operations.

The investigation performed while working on this paper allowed the authors to understand the concepts of complex network virtualization and resource reservation principles and sharing in more detail. The conducted tests additionally emphasized significance of these issues and proved how important they are in terms of successful operation of complex systems. Encountered problems confirmed that although very useful, virtual topologies management is also demanding and requires comprehensive knowledge in a variety of computer areas.

## 8.2 Achieved goals

Research in the area of networking virtualization approaches together with implemented CM4J system resulted in achieving major objectives:

- Significant improvements in system configuration in comparison to manual setup,
- Automatic update of given network parameters,

- Reusability provided thanks to loosely coupled components hidden under well-defined interfaces,
- Presenting network topology in a clear and natural way (graph format).

Hopefully the presented system and the whole concept of network virtualization, resource allocation and isolation would encourage the reader to do his own study in this area.

## 8.3 Further work

In relation to the achieved goals not all were successfully finished. Time shortage and the lack of people did not allow for as deep an insight at all issues as it was initially planned. There are many welcomed improvements that may be added to the CM4J system. The largest component initially planned was an automatic resource assigner that would run and perform automatic resource assignments to nodes that run under the lowest load. This assigner with an attached rule-based system would gather data about the load on each node and based on that it would decide what and where to instantiate. Unfortunately, the presented and discussed system in this thesis lacks that functionality. Instead, it offers manual assignments, where the user selects on which node new virtual resources should be created. Another functionality not implemented, yet welcomed would be a semi-automatic snapshot creation. Currently the user can create Solaris zone from existing snapshot and use it. The functionality where the created resource could be converted to snapshot and then transferred to the repository would definitely improve the system's usability.

# Acronyms

**AAC** Advanced Audio Coding. 78

**API** Application Programming Interface. 13, 26, 82

**ATM** Asynchronous Transfer Mode. 12, 20

**CIR** Committed Information Rate. 71

**CM4J** Crossbow Module for JIMS. 11, 27, 56, 57, 67, 69, 70, 82–84

**CoS** Class of Service. 20

**CRUD** Create, Read, Update and Delete. 61, 68

**DMA** Direct Memory Access. 34

**DSCP** Differentiated Services Code Point. 20, 37, 38

**GUI** Graphical User Interface. 26, 54, 62, 63, 65–69, 72, 74, 81, 82

**GVRP** GARP VLAN Registration Protocol. 36

**HTTP** Hypertext Transfer Protocol. 71, 72, 74

**IaaS** Infrastructure as a Service. 12, 18

**IETF** Internet Engineering Task Force. 20

**IP** Internet Protocol. 11, 12, 20, 21, 25, 30, 31, 35, 37, 38, 41, 53, 66, 74

**ISP** Internet Service Provider. 20

**IT** Information Technology. 17–19

**JIMS** JMX-based Infrastructure Monitoring System. 11, 41, 42, 46, 56, 57, 61, 65, 67–69

**JMX** Java Management Extensions. 41, 45, 56, 65, 67, 69, 82

**JNA** Java Native Access. 61

**JNI** Java Native Interface. 61

**JVM** Java Virtual Machine. 41

**LAN** Local Area Network. 10, 24

**LDOM** Logical Domains. 29

**LXC** Linux Containers. 29

**MAC** Media Access Control. 34

**MVRP** Multiple VLAN Registration Protocol. 36

**NFS** Network File System. 72

**NIC** Network Interface Controller. 24, 33–36, 45, 63

**OSI** Open Systems Interconnection. 12, 14, 15, 27, 31, 33, 37, 53

**PHB** Per-Hop Behaviour. 20

**QoS** Quality of Service. 10–12, 14, 17, 18, 20, 24, 25, 28–30, 32, 36–38, 48, 53, 70, 71, 74, 81, 82

**RFC** Request for Comments. 38

**RSVP** Resource Reservation Protocol. 21

**RTP** Real-time Transport Protocol. 72, 74, 80

**S3** SOA Solution Stack. 12, 19

**SOA** Service-oriented Architecture. 10, 12, 19, 22

**SSH** Secure Shell. 65

**SWT** Standard Widget Toolkit. 62, 67

**ToS** Type of Service. 20

**VLAN** Virtual Local Area Network. 14, 15, 17, 27, 29, 36, 45, 63–65

**VNIC** Virtual Network Interface Controller. 28, 33–38, 45, 63, 65, 72, 76, 77

**VOD** Video on Demand. 71, 72, 74, 79, 80

**VPLS** Virtual Private LAN Service. 15

**VRF** Virtual Routing and Forwarding. 15

# Bibliography


[1] A. Arsanjani, Liang-Jie Zhang, M. Ellis, A. Allam, and K. Channabasavaiah. S3: A service-oriented reference architecture. *IT Professional*, 9(3):10 –17, May-June 2007.

[2] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An architecture for Differentiated Services. RFC 2475 (Informational), Dec 1998. Updated by RFC 3260.

[3] Robert Boers. Virtualization of legacy systems: A holistic approach to datacenter planning. 2010.

[4] Manogna Chebiyyam, Rashi Malviya, Sumit Kumar Bose, and Srikanth Sundarrajan. Server consolidation: Leveraging the benefits of virtualization. 2009.

[5] Cisco, Inc. Diffserv: The scalable end-to-end QoS model. August 2005.

[6] Oracle Corporation. Solaris Operating System. 2009.

[7] Oracle Corporation. *System Administration Guide: Oracle Solaris Zones, Oracle Solaris 10 Containers, and Resource Management.* 2010.

[8] Oracle Corporation. *Oracle VM VirtualBox User Manual.* 2011.

[9] Simon Crosby and David Brown. The virtualization reality. 2006.

[10] Łukasz Czekierda. Zapewnianie jakości usług w sieciach IP. 2002-2010.

[11] Nicolas Droux. Virtual switching in Solaris. 2007.

[12] Nicolas Droux, Sunay Tripathi, and Thirumalai Srinivasan. Crossbow: From hardware virtualized NICs to virtualized networks. 2009.

[13] Exforsys, Inc. Virtual infrastructure benefits. 2nd Mar 2009.

[14] Rob High, Stephen Kinder, and Steve Graham. IBM's SOA foundation: An architectural introduction and overview. 2005.

[15] Marcin Jarząb, Damian Wieczorek, and Krzysztof Zieliński. JIMS extensions for resource monitoring and management of Solaris 10. 2006.

[16] Poul-Henning Kamp and Robert N. M. Watson. Jails: Confining the omnipotent root. In *In Proc. 2nd Intl. SANE Conference*, 2000.

[17] Ruth Malan and Dana Bredemeyer. Defining non-functional requirements. 2001.

[18] V. Moreno, V. Moreno, and K. Reddy. *Network virtualization.* Networking Technology Series. Cisco Press, 2006.

[19] Victor Moreno. Network virtualization. July 19, 2006.

[20] Daniel Price and Andrew Tucker. Solaris Zones: Operating system support for consolidating commercial workloads. 2004.

[21] Jose Renato Santos, Yoshio Turner, and Jayaram Mudigonda. Taming heterogeneous NIC capabilities for I/O virtualization. 2008.

[22] Jeff Savit. Flow control in Solaris 11 Express Network virtualization. 2010.

[23] Changhua Sun, Le He, Qingbo Wang, and Ruth Willenborg. Simplifying service deployment with Virtual Appliances. 2008.

[24] Sun Microsystems, Inc. *System Administration Guide: Network Interfaces and Network Virtualization.* 2008.

[25] Sun Microsystems, Inc. *Logical Domains 1.3 Administration Guide.* 2010.

[26] Sun Microsystems, Inc. *System Administration Guide: Devices and File Systems.* 2010.

[27] Sun Microsystems, Inc. *System Administration Guide: Solaris Containers—Resource Management and Solaris Zones.* 2010.

[28] Bhardwaj Sushil, Leena Jain, and Jain Sandeep. Cloud computing: A study of Infrastructure as a Service (IaaS). 2010.

[29] Sunay Tripathi, Nicolas Droux, and Kais Belgaied. Crossbow Virtual Wire: Network in a box. 2009.

[30] Sunay Tripathi, Nicolas Droux, Thirumalai Srinivasan, Kais Belgaied, and Venu Iyer. Crossbow: A vertically integrated QoS stack. In *Proceedings of the 1st ACM workshop on Research on enterprise networking*, WREN '09, pages 45–54, New York, NY, USA, 2009. ACM.

[31] E. Turban, D. King, J. Lee, and D. Viehland. *Electronic Commerce: A Managerial Perspective.* 2008.

[32] VMware, Inc. Virtualization basics. 2011.

[33] Paul A. Watters. *Solaris 10: The Complete Reference.* 1st edition, 2005.