AGH – UNIVERSITY OF SCIENCE AND TECHNOLOGY
KRAKÓW, POLAND

FACULTY OF ELECTRICAL ENGINEERING, AUTOMATICS, COMPUTER SCIENCE AND ELECTRONICS

PH.D. DISSERTATION

MARCIN JARZĄB, M.SC.

ADAPTABLE SOI PROVISIONING WITH LIGHTWEIGHT CONTAINERS VIRTUALIZATION TECHNOLOGY

Supervisor:
Prof. Krzysztof Zieliński

Kraków, 2011

AGH – AKADEMIA GÓRNICZO-HUTNICZA
KRAKÓW, POLSKA

WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI, INFORMATYKI
I ELEKTRONIKI

ROZPRAWA DOKTORSKA

MGR INŻ. MARCIN JARZĄB

ADAPTOWALNE UDOSTĘPNIANIE SOI PRZY UŻYCIU TECHNOLOGII WIRTUALIZACJI LEKKICH KONTENERÓW

Promotor:
Prof. Zw. Dr Hab. Inż. Krzysztof Zieliński

Kraków, 2011

I would like to dedicate
this doctoral dissertation
to my wife Aldona
and daughter Juliet.

Acknowledgements

I would like to express my gratitude to my supervisor, Professor Krzysztof Zieliński, for his support and assistance in carrying out the research described in this dissertation.

Special thanks to all my colleges from the Distributed Systems Research Group at the Department of Computers Science for their cooperation and valuable insight.

The research was carried out with the help Polish Ministry of Education and Science grant no.1583/T11/2005/29 and European Regional Development Fund program no. POIG.01.03.01-00-008/08.

Abstract

Modern computing infrastructures for applications must be created according to architectures that provide flexible platform and operational capabilities for provisioning of required computational resources by applications. Service Oriented Infrastructure enables moving from dedicated infrastructure for specific applications to architectures in which IT resources and infrastructures' system tools are exposed as services and allocated on demand using *virtualization* techniques. Lightweight virtualization is a very attractive approach for SOI provisioning, providing granular resource partitioning, but also poses challenges in areas of construction of platforms which realize an adaptable SOI provisioning process. The author of the thesis claims that a construction of such a platform is possible through exploitation of a component-based design, equipped with adaptable steering algorithms.

This thesis is organized as follows: *Chapter 1* introduces the research focus and explains key terms and concepts included in the dissertation. *Chapter 2* provides background for the research presented in the thesis, elaborating basic definitions related to SOI, SOA, adaptive systems and adaptable architectures. It also presents virtualization technologies, maturity levels and IT process frameworks for structured computer systems provisioning and management. It also describes already existing solutions for provisioning and adaptive management of virtualized infrastructures. *Chapter 3* specifies the adaptable SOI provisioning with a lightweight virtualization process model, with the most important elements of the solution that must provide for realization of such a model. *Chapter 4* elaborates the concept of the Adaptable SOI Lightweight Virtualization Provisioning Platform, including functional and non-functional requirements. *Chapter 5* elaborates how the model of adaptable SOI provisioning is reflected within the architecture of the platform. This chapter describes mechanisms that are useful for designing adaptable frameworks and presents a layered architecture of the platform. *Chapter 6* describes a reference implementation based on the requirements and the architecture. It necessitates using API and technologies to implement software components exposing SOI services. *Chapter 7* evaluates the Adaptable SOI Lightweight Virtualization Provisioning Platform developed using several complex use cases, focusing on different aspects of provisioning and management of SOI over Lightweight Virtualization Containers. *Chapter 8* summarizes the dissertation, verifies the thesis statement and analyzes the implementation of the Adaptable SOI Lightweight Virtualization Provisioning Platform with a focus on the limitations and possible improvements which could be considered in future work on the system created.

Typographical Conventions

ABBR	The first use of an abbreviation is written in boldface, indicating that the full name of this abbreviation can be found in the list in Chapter 9 (Section 9.1, titled “Abbreviations”, page 141). Some abbreviated terms are also explained in the text, in brackets or in footnotes.
`Source` *`code`*	All source code is printed using the monotype font, and placed in frames. Keywords and literals are highlighted according to the programming language used. Some additional source code fragments accompanying the thesis are placed in Appendix A.
Specification	Each italicized word carries a special meaning in terms of the related specification or context.
method()	Methods are denoted by empty brackets (regardless of their signature) and written in lowercase.

Table of Contents

List of Figures

List of Tables

Source Code Listings

1 Introduction

This chapter introduces the thesis and explains key terms and concepts included in the dissertation.

Currently, IT computational infrastructure is becoming a more complex system with many elements like middleware environments, such as Enterprise Service Bus (ESB), application servers, and Message Oriented Middleware (MOM), which must be effectively made available to users and managed. They can operate over virtualized hardware, an approach that enables consolidation and is more efficient in terms of energy consumption. In addition, such infrastructures can consist of heterogeneous tangles of hardware, middleware and software from multiple vendors. They have resource demands that vary over time due to changes in business conditions and user demands, and are deployed by service providers in data centers which must be designed to provide extra capacity demand to handle peak loads in case of some unexpected occurrences. This requires greater IT infrastructure flexibility through intelligent matching of computational resources to meet business requirements.

Provisioning addresses the activities required to prepare *Runtime Infrastructure*[1] to deploy and activate a particular application service. It is a more efficient form of change management and capacity planning with the ability to allocate or re-allocate a broad variety of computing resources, including: servers, middleware, applications, storage systems and network interfaces, to the applications and systems that need them. Such procedures are repeatable and can be defined in a structured process [1].

Modern computing infrastructures for applications should be created according to architectures that provide flexible platform and operational capabilities for provisioning of required computational resources by applications. This can be attained by introducing service orientation to the concepts of orchestration and management of computational infrastructure through a Service-Oriented Infrastructure (SOI) paradigm.

[1] Such infrastructure is defined as application containers within application middleware running on an operating system instance (can be virtualized).

Service Oriented Infrastructure is architecture which describes IT infrastructure in terms of Infrastructure Services and encompasses phases of the SOI life cycle i.e. *Design*, *Provisioning*, *Operation*, *Decommissioning* and *Management* of the particular Runtime Infrastructures. Infrastructure Services provide a virtualized pool of shared resources (Servers, Network, Storage, Infrastructure software) which are provisioned and managed in a highly automated way. SOI provides foundational support for a Service Oriented Architecture (SOA) or other application architecture [9] and enables moving from dedicated infrastructure for specific applications, to architecture in which infrastructure elements and tools are exposed as services, being defined in resources pools and allocated on demand using *virtualization* techniques.

Virtualization is a form of resource management, which manipulates devices, storage address spaces and data objects. It makes these resources more useable and more effective by aggregating resources together, subdividing resources, and providing resource substitutions. The most convincing example of virtualization is an operating system, which virtualizes computer resources. It is a first elementary example building the ground for higher layer of virtualization. This idea could be applied to computational resources such as servers, computer networks, storage and desktops.

1.1 Motivation

Provisioning and management of Runtime Infrastructure must be efficient in order to effective usage of computational resources to guarantee that a given *application service* is available for *consumers* with a specified Quality of Service (QoS) contained within a Service Level Objective (SLO) being part of a Service Level Agreement (SLA) contract. QoS might apply to many aspects associated with a service running, for instance: (i) *performance* (described by response time, network throughput), (ii) *reliability* (percentage of time service as available to a consumer, allowing for pauses connected with failures and administration activities). To meet the specified QoS, sufficient computational resources, according to those requirements, must be available. It is also necessary that resources on which services are running are equipped with mechanisms that will enable adaptation for changing requirements on *resource demands* and provision computational infrastructure on demand. This poses new challenges to systems and application management that did not exist in dedicated environments. Since each of the hosted services can have a resource demand that changes over short time scales (e.g. seconds or minutes), there needs to be a control system that can dynamically allocate the server's capacity to the Runtime Infrastructure in a real time[2]. The benefit of doing this is that it allows multiplexing between resource demands from co-hosted services, so that shared servers can reach higher resource utilization. An answer to such requirements is to provide IT infrastructure which enables reallocation of resources, depend on changes occurred in runtime used by applications' services. Such construction involves delivery of particular *Infrastructure Services*, which enable monitoring and management of compute resources and facilitates dynamic

[2] It is a synonym and not a real-time system in a strict sense.

allocation of required capacity, using virtualization of physical resources to create one or many operating environments - Virtual Execution Infrastructure (VEI). This consists of independent and distributed operating system instances - Virtual Execution Environments (VEE's). As virtualization provides mechanisms for partitioning compute resources within a physical server, the complexity of such virtualized system remains very high and setting up a suitable control algorithm ensuring the specified SLA is challenging. The efficiency and correctness of the control strategy depends on many parameters which must be very carefully identified. An important aspect is also the provisioning of VEI, which must be automated as much as possible using the SOI framework.

Lightweight virtualization techniques are a very efficient solution for provisioning VEI because a single instance of OS can host many instances of Lightweight Virtualization Containers (LVC). In comparison to other virtualization solutions, LVC are provisioned more easily and use fewer resources. The reason is that OS components like the kernel and native libraries are loaded in the RAM only once and *shared* between LVC instances. It is also faster because it does not spend time on emulation of system hardware. Existing technologies for lightweight virtualization are OpenVZ or Solaris Containers, which provide instruments helping users to install and manage virtualized systems with command line tools. Unfortunately, these solutions do not provide advanced provisioning frameworks that support effective management of distributed LVC and are not designed with SOI architecture in mind. They also do not enable integration of adaptive behavior.

Infrastructure Management Framework for SOI provisioning and management of lightweight virtualization must be modular with components that are configurable to fulfill the constantly changing requirements and evolution of IT technologies. They require solutions that not only correspond to current requirements, but also allow to evolution and adaptation, as technology advances and new requirements emerge. Exposed services must be fully discoverable, interoperable and easy to integrate with external provisioning frameworks.

1.2 Aim of the thesis

Adaptable provisioning of SOI calls for evolving open mechanisms for monitoring and management of the Runtime Infrastructure resources used by LVC and hosted services. In general, at least two type of resources can be distinguished that parameters must be subject to monitoring: (i) *Middleware, which provides containers (Application Execution Environments) for specific application services*, (ii) *Virtual Execution Infrastructure provisioned with Lightweight Virtualization Containers (Virtual Execution Environments) on which middleware are running*.

During VEI provisioning over LVC, a system administrator is obligated to specify runtime parameters like compute resources, storage capacity, networking interfaces and required software packages. For these resources, the big challenge is to determine their consumption attributes. The responsible process named *Capacity Planning*, according to [69] is defined:

Capacity planning in the context of computer performance analysis refers to the planning of computer resources to insure that workloads' Service Level Objective will be met. This tends to favor the use of analytic and simulation models. Once calibrated, capacity planning models should be tracked against future revisions of system software and hardware.

There are some tools, which are very helpful in this process like *TeamQuest Model* [69] or *PDQ* [69,71]. When using them, a system administrator provides some information about the application like resource demands for CPU, RAM, storage and target SLOs. Based on the information these tools estimate, some resource attributes values for a given domain but, in spite of such valuable information, this process is rather not a trivial task and must be done by system experts. Such computing infrastructures should be equipped with a mechanism which enables self-management and is realized by the vision of the Autonomic Computing [41] initiative. It has been created to enable building computing systems and services that are capable of managing themselves, anticipate their workloads, optimize performance and adapt to events occurring in the surrounded environment.

1.3 Thesis Statement

The complex issues of provisioning of Service Oriented Infrastructure, including the QoS requirements, can be solved using the techniques of lightweight virtualization enriched with subsystems allowing implementation of adaptable steering. Implementing this class of systems is possible due to employing elements of modern software engineering techniques as well as component programming.

Verifying the statement requires conducting extensive research in many fields related to design, implementation and practical evaluation of the implemented platform for adaptable SOI provisioning with LVC and special emphasis on following aspects:

1. Analysis of existing solutions for provisioning of Virtual Execution Infrastructure, deployment of applications, and determining weak points. The algorithms for adaptive management of computational infrastructure are discussed.
2. Model of adaptable SOI provisioning that suits open, distributed and virtualized environments with lightweight virtualization, supporting re-configuration and adaptively of Runtime Infrastructure used by application services.
3. The concept of the *Adaptable SOI Lightweight Virtualization Provisioning Platform* targeted for provisioning of Service Oriented Infrastructure with LVC and with elaboration of known adaptability paradigms ensuring flexible architecture for adaptive management of such infrastructure.
4. Implementation of the *Adaptable SOI Lightweight Virtualization Provisioning Platform* that combines the aforementioned aspects.
5. Practical case studies with evaluation of the platform in enhancing the SOI with LVC provisioning activities and ensuring the QoS of workloads.

1.4 Research Contribution

The author of the dissertation presents a component-based approach for implementation of Adaptable SOI Lightweight Virtualization Provisioning Platform for provisioning of Runtime Infrastructure over LVC designed according to the SOI paradigm. In the thesis, the author proposed the *model of an adaptable SOI provisioning with LVC*, including carrying out the requirements for development of the platform, which were mapped in the architecture and then implemented exploiting modern software technologies valuable to create this class of systems. Exploited *software engineering* techniques promote flexible architectures, maintaining separation of concerns between components implementing provisioning, management and adaptive behavior of particular elements of the Runtime Infrastructure.

Particular attention is devoted to Solaris 10 OS with LVC provisioning, workloads consolidation and QoS enforcements. The author also concentrates much effort on the applicability of LVC implementation provided by Solaris and policy-driven adaptive management. Subsequently, software architectures and construction techniques of adaptation loops for provisioning of VEI and coherent to Autonomic Computing Systems are described. Principles of controller implementation are proposed and practically (conditions of applicability) verified, including opened-loop systems, which exploit the model of controlled system behavior, in comparison to closed-loop systems exploiting feedback control.

2 Background and Related Work

This chapter provides background for the research presented in the thesis. It elaborates basic definitions related to SOI, SOA, adaptive systems and software engineering methodologies useful for construction of adaptable platforms for the provisioning of SOI. The chapter presents virtualization technologies with a focus on lightweight virtualization, maturity levels and IT process frameworks for structured computer systems provisioning and management. It also describes already existing solutions for provisioning of virtualized infrastructures and adaptive management.

Implementation of the A-SOI-LV-PP requires the use of multiple techniques in the field of software engineering, adaptable systems construction and algorithms used in adaptive management, each elaborated in the chapter. Some elements that must be available in the platform's architecture, supporting provisioning of Runtime Infrastructure with lightweight containers technology, are already defined in the reference architecture developed by The Open Group. Also a model for the SOA stack introduces some elements of which the author tries to fit into the architecture of the proposed solution. An important element is the analysis of the currently available platforms for provisioning of VEI and solutions offering support for adaptive management, with particular emphasis on the context of compliance for SOI architecture and support for lightweight virtualization.

2.1 Service Oriented Infrastructure

The SOI platform providing core Infrastructure Services to applications enables delivering more consistent service levels. This architecture is used for automatic allocation of resources on demand as an application's workload increases. The guidelines for Service Oriented Infrastructure must provide support for the essential elements [8]:

- Functionality to manage provisioning of Virtual Execution Infrastructure and deployment of application components used by end users,

- Service Management to assure the SOI solution provides the required service characteristics expressed with an SLA.

Figure 1 depicts the SOI reference model [9] defined by *The Open Group*[3] that merges Infrastructure Services with applications and consumers, thus capturing all the levels of activities related to management and monitoring of IT infrastructure. The model defines the conceptual building blocks that are provided in order to each service or other element can be used in SOI, when such an architecture is implemented.

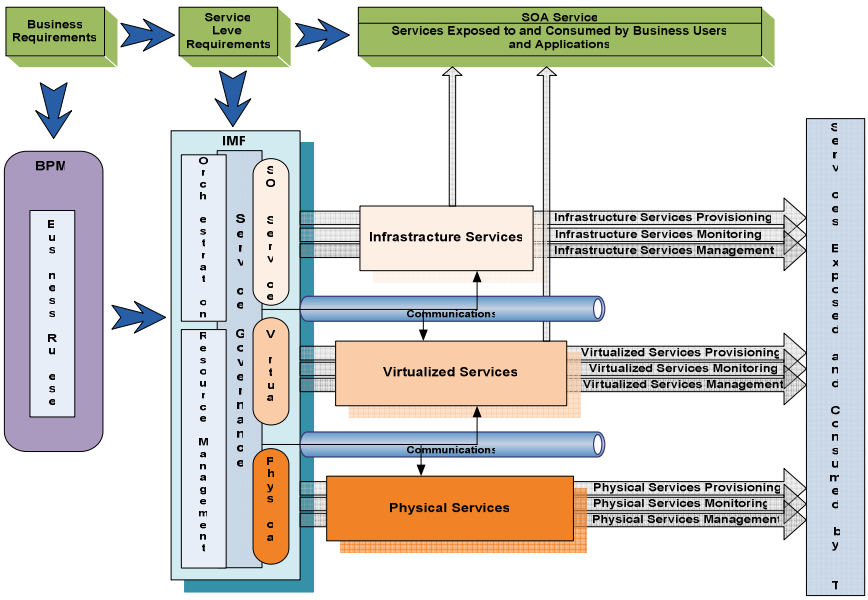


Figure 1. SOI Reference Model.

The elements of proposed architecture are:

- *Infrastructure Management Framework*: Collection of software tools that enable provisioning and management of IT infrastructure in an adaptive way, driven by business rules.
- *Infrastructure Services*: A collection of services that expose interfaces for management of particular infrastructure resources. Those services are implemented as combinations of *Virtualized* and *Physical* Services.
- *Virtualized Services*: A collection of tools that are part of the software provided by specific virtualization solution
- *Physical Services*: Represent hardware, storage and operating system resources. They can be mapped to low-level services related to identity management (role-based configuration) or Single-Sign-On (SSO).

[3] The Open Group (http://www.opengroup.org) is a vendor and technology-neutral consortium, whose vision of Boundaryless Information Flow™ will enable access to integrated information within and between enterprises based on open standards and global interoperability.

Successful implementation of the platform for SOI with LVC provisioning involves fusion of technologies, business models and infrastructure management techniques. Specifically, SOI accomplishes the goal of greater IT resource flexibility, efficiency and value generation through intelligently matching physical services to meet business demand on a pay-for-use basis. Intelligently matching refers to the combination of technologies, business processes or applications and Infrastructure Services. Such an approach for software and computational (physical and virtual) resources exposition would also enable provisioning of required resources dynamically as an application's workload increases. Construction of such tools involves incorporating many software technologies and analytical methods that together will form SOI for flexible computational infrastructure provisioning with lightweight virtualization and management.

The Infrastructure Management Framework (IMF), platform for adaptable SOI provisioning must be flexible and give the system administrator flexibility and increase operational efficiency. The goal is to fit the system not only to requirements defined during the design phase, but also to prolong the ability of the platform for adaptation during the usage phase. Such a platform must be a *flexible system*, where two variations are known, *adaptable* and *adaptive* systems. An *adaptable* system provides users with tools that make it possible to change the system characteristics. If the user changes these characteristics automatically according to specified goals, then the system is adaptive. The architectural vision of this solution must be reconfigurable, built from modularized components that can be re-organized and extended with new functionality to meet new business requirements. The *self-adapting* system is the most common conception of adaptivity [95]. The goal of adaptive systems is to provide facilities that enable IMF to handle specific requirements, like optimizing workloads according to SLA contracts. The following sections introduce aspects related to virtualization, maturity levels in system management and architectures of adaptable systems.

2.2 Reference Architecture of SOA Solution Stack

The existence of tremendous number of computer-based business applications created with traditional software technologies often delivers a need for migration of existing systems to *Service Oriented Architecture* (SOA), which is architecture for the creation of dynamic associations between services, interfaces and their implementation. According to the *Institute for Enterprise Architecture Development's* (IFEAD)[4] concept, SOA architecture is defined as follows:

Services Orientation as well as SOA is an architectural style whose goal is to achieve loose coupling among interacting services. A service is a unit of work done by a service provider to achieve desired results for a service consumer. Both provider and consumer are roles played by organizational units as well as software agents on behalf of their owners.

[4] Eric Sweden, *Service Oriented Architecture: An Enabler of the Agile Enterprise in State Government*, Report NASCIO, May 2006

SOI is founded on the SOA concept with notions of service provider and service consumer to the virtualized computing, networking and storage infrastructure themselves. When resource demand for computing resources for a given application changes, like more CPU, RAM, network or I/O bandwidth, a request in the SOA style is made to the IMF. The reference architecture of SOA systems provides a blueprint for how concrete elements are organized and what the relationships are between the specific building blocks i.e. services, components and flows that support business processes. It is also important that the proposed solution ensures a means for reusability of required assets for rapid development, deployment, and management of business processes, SOA applications and IT infrastructure used. The SOA Solution Stack (S3) [20] proposed by IBM elaborates the process of SOA applications development and deployment. The S3 model presented in Figure 2 provides a detailed description of architectural elements divided into nine layers. Each layer has a physical and logical aspect and lets the organization define the degree of consumer-provider integration. The existence of both functional and non-functional service requirements is also assumed, which establish the SOA's objective. S3's nine layers are *operational systems*, *service component*, *services*, *business process*, *consumer*, *integration*, *QoS*, *information architecture*, and *governance and policies*. The five *horizontal* layers relate to the overall functionality of the SOA solution, which are cut-crossed by the *vertical* layers that are non-functional in nature.

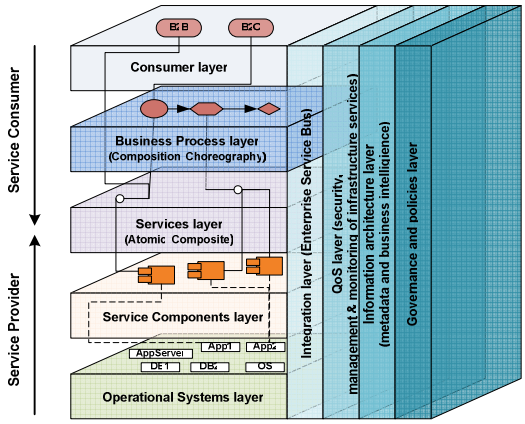


Figure 2. SOA reference architecture.

A description of horizontal layers in more detail is as follows:

- *Operational Systems*: The foundation of IT operating environment, including existing application software systems, databases, application servers, transaction processing systems and operating systems. This layer also includes virtualized infrastructure resources that results in better resource utilization and manageability. These combined elements constitute the Runtime Infrastructure required by particular application service.
- *Service Components*: The layer guarantees that the implemented components conform to service description, reflect the required functionality and provide enforcement points for realization of aspects defined in SLA contracts.
- *Services*: Provides services defined within a specific organization. Each service is described with sufficient detail to invoke the business functions by any business process

or directly by a consumer. Service specification consists of interface signatures, network location, communication protocols and semantics.

- *Business Process*: Contains a definition of business processes, which are divided into smaller tasks, each mapped to a specific service, which fulfill specific business functionality. These processes capture the activities needed to accomplish a specific business goal. The layer communicates with the B2B or B2C channels that provide inputs from business users and returns the results to them.
- *Consumer*: Provides mechanisms for handling interaction between users and other applications in SOA ecosystem. Clients' communication channels might be implemented with portals, rich clients and other technologies.

Vertical layers concentrate on aspects related to integration, QoS, availability, metadata about exchanged information's and policies that define IT manageability process.

- *Integration*: The layer is equipped with facilities like mediation, routing, and transport protocols for ensuring communication between services and consumers. In the layer, binding of services and components occurs for process execution and exposition of services for communication channels managed by the ESB.
- *Quality of Service*: This layer provides tools that enable realization of non-functional requirements and is responsible for observing other layers and emitting signals or events when non-compliance, according to defined SLA contract, occurs. The provided functionality ensures that a given SOA solution would conform to requirements like reliability, availability, manageability, scalability and security.
- *Information architecture*: This layer defines inter-organization and cross-organization data structures related to a specific industry, for instance XML schema (meta-data) for exchanged information and business protocols used.
- *Governance and policies*: Provides management facilities for making decisions about SOA's aspects, related to capacity planning, security, performance and monitoring. This layer is applied to other S3 layers because it integrates the operational life-cycle management activities of the SOA ecosystem. Since it is connected with the QoS layer, it helps in enforcing QoS though intelligent capacity planning according to defined policies.

Although computational infrastructure built according to the SOI paradigm is used for hosting a wide spectrum of applications, the S3 reference model of SOA specifies some elements like Operational, QoS and Governance and Policies layers that can be transformed into the SOI conceptual architecture. Such architectures must operate on adaptable Runtime Infrastructure that enable flexible provisioning of VEI over LVC, monitoring the health and making adjustments to meet defined SLA automatically through dynamic resource allocation.

2.3 Virtualization Technologies Overview

The nature of individual services that can cause some computational resources to be underutilized or overwhelmed depends on some conditions like time of a day or other regular events. Services consolidation into groups, called workloads, onto fewer larger servers, which computational resources might be virtualized, leads to better flexibility and manageability. A workload is an aggregation of all processes of an application, or group of applications, which makes sense from a business perspective. Another factor is logical isolation between workloads, which is especially important for scenarios when applications belong to different business units and when namespace conflicts, security, and administration issues occur. The big challenge when consolidating multiple applications onto single physical server is to provide mechanisms of control over the resources (e.g. CPU, memory portions or network bandwidth) utilized by those applications. Hardware and software advances introduce concepts of *virtualization*, which provides mechanisms for advanced management of resources e.g. devices, storage address spaces and OS instances providing resources consumption granularity, some degrees of isolation and flexibility (Figure 3).

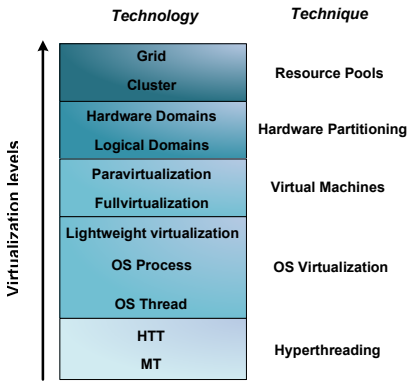


Figure 3. Virtualization levels.

Virtualization techniques allow multiple OS and application instances to run on the same server, having the illusion that it is running within its own hardware making physical resources more usable and more effective by aggregation and the ability to partition them together.

- *Resource Pools*: Groups of prepared servers with a provisioned operating environment that can be assigned to serve a specific application service. These groups are also called server pools and are managed with a concept of service with the usage of appropriate provisioning software tools. This architecture assumes that computers, called blades, are plugged into a rack-mount shelf equipped with a scalable communications subsystem (shared I/O channel to Network Area Storage (NAS) and several gigabit Ethernet ports) and control logic that supports system redundancy. Blades architecture allows server construction to be further partitioned using the following virtualization techniques.
- *Hardware Partitioning*: Domains are a solution which involves hardware and firmware interfaces to divide a computer's hardware resources so they can host multiple OSes. They usually have some architectural limitations like a fixed number of OS instances,

amount of resource granularity or sharing. This approach is mostly used on high-end servers.

- *Virtual Machines* (VM) are a technology that enables running operating system instances in domains that share a physical server resources. Such domains are isolated environments to be managed by *hypervisor* (a.k.a. *Virtual Machine Monitor* - VMM). Hypervisor performs emulation of some real hardware resources enabling it to run some number of *Virtual Machines*. Each VM has the illusion of owning hardware and is able to manage an arbitrary guest operating system. Hypervisors usually do not have fixed limits on the number of OS instances and enable specification of fine-grained resource controls settings.
- *OS Virtualization*: Another technology is LVC, in which there is only one underlying operating system kernel, which the containers enhance by providing distinct borders offering increased isolation between groups of processes. Software that virtualizes an operating system environment enables it to run some number of LVC instances, which do not emulate any of the underlying hardware. Instead, the container or application communicates with the host OS to share resource usage, which then makes the appropriate calls to real hardware. *Process* virtualization is a concept used for domain specific solutions like *K Virtual Machine* [16], *Multitasking Virtual Machine* [17, 18] or *.Net Application Domains* [134], which enable running several applications in one process instance of Virtual Machine and also enable secure isolation between applications. The advantages of the hosting application environment are reducing the memory footprint, better start-up and control of how resources are used [18, 19], which is very important in case of small and portable devices. *Thread* virtualization is a technique long-used by operating systems to provide the illusion of exclusive access to shared system resources. Multiple process threads running on single CPU are assigned time by the OS scheduler, which provides some form of virtualization.
- *Hyperthreading*: Virtualization of CPU processing resources where the rationale is to maximize throughput performance by utilizing idle cycles. Intel, for example, refers to this form of processor virtualization as hyper-threading technology (HTT) or multithreading (MT) on its Xeon and Pentium 4 product lines.

Virtualization techniques differ in the complexity of implementation, ease of administration and common use. Important aspects are the performance impact in comparison to the standalone OS, level of access to resources and, of course, breadth of OS support. The most important and widely-used technologies in the industry are described in more detail bellow.

2.3.1 Hardware Partitioning

Hardware partitions, introduced by Amdahl Corporations' *Multiple Domain Facility* (MDF) [13], are architected as self-contained servers, logically isolated from each other in a highly secure and reliable manner. Each partition is an "electrically" separated computational domain, contained in multiple system boards, CPU's, memory, I/O, network resources and boot disks,

which, as whole, is able to run a single OS instance. Based on the MDF model, Sun Microsystems created a new class of servers, SUN StarFire and SUN M8000, which also support hardware partitions through the *Dynamic System Domains* (DSDs) [14] mechanism. They also introduced Logical Domains (LDoms) virtualization technology, limited to the UltraSPARC T1 and T2 processors. LPARs are another virtualization technology based on the domains concept provided by IBM, which can only be run on PowerPC based machines [34]. The IBM boxes have an abstraction layer between the hardware and the operating system, called *Power hypervisor that* allows for logical partitioning. The LPAR partitions can run different kinds of PowerPC-based operating systems (Linux, AIX, OS/400). The domains usually have high reliability and strong isolation between them with low overhead, but are limited in number and rather financially expensive, thus used by big enterprises.

2.3.2 Virtual Machines

Virtual machines date back to the late 1960s with the release of IBM's VM/CMS operating system. VM works in several modes: *Full-virtualization* allows the VM to manage an arbitrary guest operating system without its modifications because it is not aware that it is not running on real hardware. *Native-virtualization* is a hybrid of full virtualization and acceleration techniques used for increasing performance of I/O operations and exploits extensions built-in directly into the hardware like Intel Virtualization Technology (IVT) and AMD-Virtualization (Pacifica) families. *Para-virtualization* requires rather radical modifications of the operating systems managed by the VMM and equips each VM with an abstraction layer of the hardware, which is not identical to the real physical hardware, but such approach allows near native performance. VM's technology also allows for live-migration (moving instances between hypervisors) to another physical server[5]. The known virtualization products that are based on VM technique are VMware[6], Microsoft Virtual Server[7] or Xen[8].

2.3.3 Lightweight Virtualization Containers

Lightweight Virtualization Containers (LVC - named also *OS-Containers*) are operating system virtualization technologies managed only by a single Operating System Kernel (OS Kernel), on top of which virtual OS instances are created. This approach is very efficient and flexible if there is a need to provide a similar set of operating system instances used by some number of workloads that need to be isolated. *Solaris Containers* [27] technology is an operating system virtualization that prevents underutilization of computer resources by consolidating multiple workloads on a single computer. The Solaris Containers functionality comprises two main

[5] If the source and target servers are located within the same LAN and share NAS that contains particular VM instance.

[6] http://www.vmware.com

[7] http://www.microsoft.com/windowsserversystem/virtualserver/

[8] http://www.xen.org

components to provide these features: *Solaris Zones* partitioning technology (for applications isolation) and resource management tools. Isolation prevents processes that are running in one zone from monitoring or affecting processes that are running in the other zones, while *Solaris Resource Management* framework provides mechanisms for system resources allocation, management and accounting. *OpenVZ* [32] is a virtualization technology based on containers developed by SWsoft. It provides multiple isolated Virtual Private Servers (VPSs), running on a single physical server, to share hardware and management effort with maximum efficiency. Each VPS has its own life-cycle, which means that it can be started and shut-downed independently and also maintains its own file system, networking stack, memory, processes, files and system services. The basic OpenVZ VPS capabilities are: *Dynamic Real-time Partitioning* for partitioning a physical server into set of VPSs, each with full dedicated server functionality and *Resource Management* for controlling how host OS resources are shared between VPS instances. Lightweight virtualization imposes little performance impact on the virtualization capability. Nevertheless, the limitation is a choice of operating system that can cause problems, since applications are often certified for only a certain OS or its version.

2.3.4 Multilevel Virtualization

Domains and virtual machines host OS instances that also can be virtualized through OS virtualization techniques and founding the so-called *multilevel virtualization* depicted in Figure 4. Lightweight virtualization containers can be running in VMs and created in large numbers, even on machine with a single CPU. LVC are available for many platforms, including UltraSPARC, Intel or AMD architectures, but because they are running in single OS kernel, it is not possible to mix and match software that requires conflicting OS versions[9].

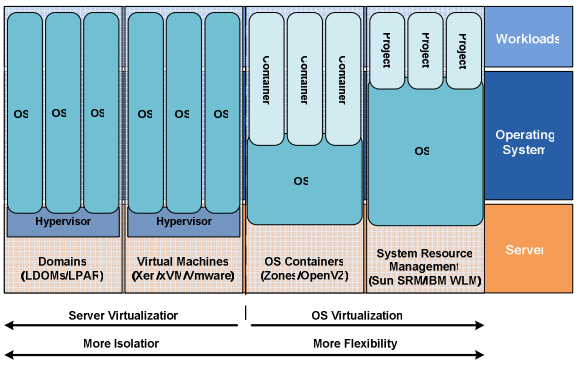


Figure 4. Multi-level virtualization overview.

When multiple users or software application workloads are sharing the same domain, VM or LVC container, it is possible that one workload would monopolize access to computational resources. To prevent such scenarios, facilities implemented in modern system resource management tools, for instance by Sun's SRM or IBM's WLM, are used. Workloads that are

[9] Solaris 10 Zones provide support for Linux Branded and Solaris 9 zones.

more important can be prioritized along with other, less critical workloads and ensure that business critical applications have access to required resources.

2.3.5 Summary

Virtualization is a basic element of Service Oriented Infrastructure and increases IT agility and flexibility because such environments can be provisioned on demand, allowing deployment of application services. Lightweight virtualization is a very attractive approach, which can be used together with VM and Hardware Partitions and provide more granular resource partitioning facilities through utilization of system resource management frameworks; but it also poses challenges in areas such as provisioning of complex VEI configurations and management to fulfill specified QoS goals. Solutions based on lightweight virtualization are also considered to be more efficient compared to the VM [37]. However, due to lower isolation, require greater effort in the area of tuning parameters that define the allocation of physical resources.

2.4 Provisioning Challenges of Virtual Execution Infrastructures

Virtualization poses management challenges that must be considered and analyzed during the practical application of any virtualization technology. Some are defined by Enterprise Management Associates (EMA) and, described with blueprints in [36], are related to VEI environments and can specify general aspects of SOI provisioning with lightweight virtualization.

2.4.1 Maximizing IT Operations Resource Efficiency

Complex deployments might be based on many virtualization technologies sprawling across physical servers. When considering multi-level virtualization scenarios, such installations become even more challenging in terms of provisioning and further management because of the duplication of the tools provided by each platform. This can be avoided by integration and automation of physical and virtualized systems management into an Infrastructure Management Framework platform, which automates regular provisioning and management activities (virtualized computational resource monitoring and allocation, or live migration). The solution enables proactive reaction for unexpected events by an automatic detection, diagnosis and remediation process, equipped with knowledge based on some process frameworks and policies. Such an approach, in turn, helps to overcome major difficulties in expanding IT infrastructure and the provisioning of new elements on demand.

2.4.2 Managing Virtual and Physical Assets

Virtualized Execution Infrastructure that consists of many heterogeneous elements like Domains, VM’s or LVC allocates distributed physical resources and makes such infrastructure harder to control to ensure that computational resources are used efficiently. The previously mentioned platform must improve visibility with continuous, automatic discovery and recording for both physical and virtual infrastructure elements. Particular VEI elements must be tracked in terms of current activities i.e. how workloads consume resources and if they can be consumed

more efficiently. This can improve hardware resource utilization and increase server consolidation ratios.

2.4.3 Establishing Performance and Availability Requirements

Effective application service provisioning requires effective SLA contracts fulfillment. VEI is a dynamic environment with continually changing workloads and resource allocation. Thus providing services for continuous measurement of application services performance metrics is especially vital, allowing for workloads' adaptation through resource allocation to maintain service levels. Monitoring and management services must enclose the Operational Systems Layer of the S3 model to enable predictive and preventative performance management. Provided functionality must be able to detect, diagnose, and correct performance problems. Performance metrics for particular VEI and application middleware elements must be consolidated and events correlated. This will enable location of potential bottlenecks in the entire infrastructure and isolate performance and availability issues.

2.4.4 Integrating Network, Storage and System Management

Virtualization also necessitates requirements in the area of networking communication and storage. The physical server, which consolidates a specific number of workloads running within domains, VMs or LVC, intensifies network traffic on the device. Scenarios in which application services running are physically moved between devices is realized through real-time migration of particular VEI elements and requires maintaining network connections, topology and flow limits to provide the required bandwidth between clients and other interconnected systems [39, 40]. Migration and dynamic provisioning of VEI elements also requires external storage (NAS or SAN devices), shared between physical servers, increasing rate and data transfers. Virtualization, therefore, produces much greater requirements for solid, continuous and high-bandwidth network and storage infrastructure.

2.4.5 Summary

The described challenges must be reflected in the provided IMF platform, which incorporates a toolset for network, storage and virtualization platforms to automate and integrate particular elements to optimize the whole provisioning and management process. It focuses on orchestration of compute, network, and storage virtualized resources, ensuring these resources are available to the upper layers of S3's stack.

2.5 Evolution of Maturity Levels in Computer Systems Management

Adaptive and autonomic management of computing resources is not a new problem for computer scientists. For decades, system components and software have been evolving to deal with the increased complexity of system control, resource sharing, and operational management. It is the evolution of past trends to address the increasingly complex and distributed computing environments of today [43]. Evolution of maturity levels related to IT

lifecycle management, regarding autonomic computing solutions, affects the level of decision making. Five levels can be defined, which are explained in [42,46]:

- *Basic Level*: IT professionals manage each system element. Activities related to *configuration*, *optimization*, *healing*, and *protection* of IT infrastructure components are performed manually.
- *Managed Level*: System management technologies can be used to collect information from different systems and analyze this information. In this scenario, IT professionals do most of the analysis, but it is a starting point for the automation of IT tasks.
- *Predictive Level*: Individual components monitor themselves, analyze changes and offer advice. Therefore, dependency on people is reduced and decision-making is improved.
- *Adaptive Level*: IT components can, individually or group-wise, monitor, analyze operations and offer advice with minimal human intervention.
- *Autonomic Level*: System operations are managed by business policies established by the administrator. In fact, business policy drives overall IT management, while at the adaptive level; there is an interaction between humans and the system.

In the context of SOI provisioning, availability and performance are important topics that are the foundation for the basic requirements to support the business operations smoothly. Unfortunately, the general state of the industry is rather at *basic* and *managed* levels. Switching to higher levels requires a particular IT organization following appropriate management processes that are defined in order to ensure the availability and performance of Runtime Infrastructures.

2.6 Informational Technology Process Frameworks

Provisioning of SOI is a very complex administration process, because it can grows in complexity and dynamics, raising requirement for self-sufficiency, which means that human administration of these systems is reduced to a minimum. Aspects related to infrastructure management can be described with the usage of some maturity model, illustrating the evolution phases that lead from a resource management-focused approach to a service management-centric approach [22]. The described process frameworks define a set of best practices that can be used for delivering of SOI to end users in a controlled and disciplined way.

2.6.1 Information Technology Infrastructure Library

Information Technology Infrastructure Library (ITIL) is a collection of IT best practices, created by the U.K. Office of Government Commerce (UOGC), that are designed for organizations to have efficient service management. Those best practices are the result of experience acquired by major IT organizations in the area of IT services, infrastructure and application management, and contributed to by the UOGC. ITIL blueprints are a good source of patterns used for adopting best practices to be implemented in any IT environment. They describe the goals, general activities, inputs and outputs of various processes that establish IT

services with the current and future requirements of business customers and improve the quality of these services. These activities are part of the *Service Lifecycle* (Figure 5) that bridges the five key stages of a service: *Strategy*, *Design*, *Transition*, *Operations* and *Continual Improvement* [23].

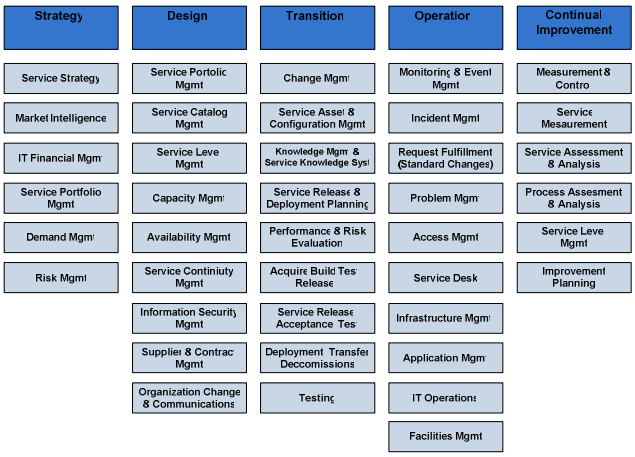


Figure 5. Information Technology Infrastructure Library v3 overview.

2.6.2 Operational Management Capabilities Model

Operations Management Capabilities Model (OMCM) [24] is based on the Sun IT Management Framework (Sun ITMF) and concentrates on aspects related to *people*, *processes* and *tools*.

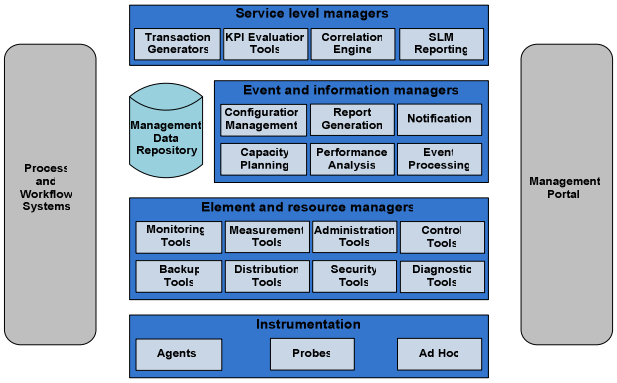


Figure 6. Operations Management Capabilities Model overview.

Similar to ITIL, it is also a comprehensive methodology for continuous improvement of IT management and provides best practices at all levels of management of IT environments. The management tools framework (Figure 6) consists of a layered combination of management applications that are tied together, when appropriate, through the integration of specific components [138]:

- The *Instrumentation* layer consists of all management elements that enable the various management tools to gain access to managed resources through the appropriate agents, probes, or other ad-hoc scripts and executables.
- The *Element and Resource management* layer consists of management applications that directly interact with the execution environment to query or modify managed resources.

- The *Event and Information* layer consists of applications that manage events and information generated by the lower layers of the framework.
- The *Management Data repository* is a logical representative of the storage and management of operational data.
- The *Service-level managers* are applications that provide the tie-in between business requirements as defined by SLAs and the technical status of the execution environment as determined by the lower layers of the framework.

2.6.3 Summary

Both ITIL and OMCM are written guidelines describing *effective* provisioning of IT services to end users according to required levels of service performance and availability. IT process frameworks suit requirements of SOI governance well and might be integrated with the S3 reference model that seems to conform to the *predictive* level of systems management. They also support architectural decisions that describe artifacts, capacity planning and management policies. These decisions are categorized into four common functions founding a *control-loop*: *collect (monitor)* the details to identify a need, *analyze* the details to determine what should done to fulfill the need, *create* a plan to meet the need, and *execute* that plan. These control loops are performed by the system administrator, nevertheless automation of process frameworks activities can be delivered with a software platform, which implements automatic behavior promised by vision of Adaptive or Autonomic Computing.

2.7 Adaptive and Autonomic Systems

To achieve automation in SOI management framework, Infrastructure Services must be orchestrated by a coordinator, which uses business rules. Such rules map from higher business application levels down to the Infrastructure Services, based on application consumption [12]. Introducing autonomic computing architecture for SOI provisioning and management would reduce manual operations related to administrative functions through predefined policies. Self-management of IT infrastructure adds the following characteristics [44]:

- *Self-configuring*, the ability for dynamic configuration itself, on the fly, with minimal human intervention
- *Self-healing*, any improper operations caused by system or application failures are detected automatically and corrective action is initiated without disrupting application services.
- *Self-optimizing*, the IT environment is able to efficiently maximize resource utilization to meet SLA with minimal human intervention.
- *Self-protecting*, the goal is to provide the right information to the right users at the right time through actions that grant access based on the users' role and pre-established policies.

A model of *Autonomic Computing Systems* (ACS) is presented in Figure 7 [46]. The basic elements of ACS are Autonomic Elements (AEs), which are software agents interacting together and ensuring self-managing behavior forming a hierarchy of *self-management* subsystems. Each AE consists of two building blocks: Autonomic Manager (AM) and Managed Elements (ME), which are system components, e.g. hardware or software applications. Autonomic managers execute according to defined policies and are responsible for management of specific managed elements that must expose sensor and effector interfaces. Sensors are used by the AM to acquire monitoring data and then compare it with a specified goal. The resulting action is performed on a given managed element by effectors.

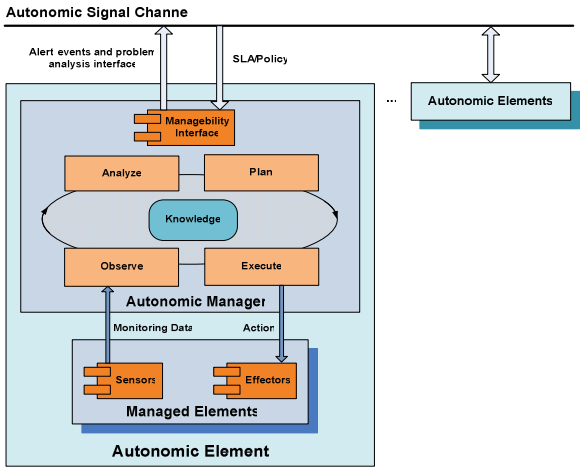


Figure 7. Autonomic element architecture.

The autonomic manager performs *monitoring*, *analyzing*, *planning* and *execution* activities that provide *control-loop* functionality and require knowledge about managed resources built into the manager or collected during runtime. In the former case, such information can be represented as a mathematical formula, i.e. a managed resource model describing a relation between its state and control action that should be performed to achieve desired system behavior. The latter case corresponds to a situation where such a model is difficult to identify. The manager may search for suitable control parameters during an iteration process performed directly over managed resources. AMs can communicate together, via an autonomic signal channel, and form a relationship between them. Autonomic and adaptive managers must use some knowledge to govern the behavior of control loops responsible for steering underlying resources to meet specified goals. The ability of an autonomic manager to extract knowledge from the observation of the system activity is distinguished as a characteristic feature that differs *autonomic* from *adaptive* system. Adaptive systems follow close-world assumption and typically have a knowledge element represented by a model or a set of rules. Such knowledge can be defined with such means as policies and is used by autonomic managers to determine what actions should be taken for the managed resources that they are responsible for. This knowledge is stored in a repository and accessible via well-defined interfaces.

2.8 Construction Methods of Control Loops for Computer Systems Management

System administrators use heuristics, best practices provided by software and architectural patterns and system tuning guidelines [109,110,111]. In the following examples, to obtain the best performance of an application server middleware, the specific application must be considered with its expected load (number of users), database which store applications' data and the VEI used by the application server in context of resources' settings. Quality assurance tests, performed in an iterative way, might produce a set of configuration settings that satisfy the specified SLA, but they are rather time consuming. In such a scenario, using efficient systematic analytical methods for finding the close-to-optimal systems' configuration might be very efficient and useful.

2.8.1 Queuing Networks

A queuing network is defined as a set of interconnected queues where each queue represents physical resource that services customer requests [112]. In a typical scenario, a resource is made from the server and a waiting line for clients requests (Figure 8). When a request arrives, it is serviced immediately, but only when and if the server is available, otherwise the given request joins the *waiting line*. Customers arrive at the system with some value of *arrival rate* (λ). The time needed by a server to serve the request is referred to as the *service time* (S), the time spent in a waiting line is named the *waiting time* (W) and sum of service and waiting times is named *response time* (T). This approach is used for predicting the performance of virtualized environments in [30] and other computer systems [67].

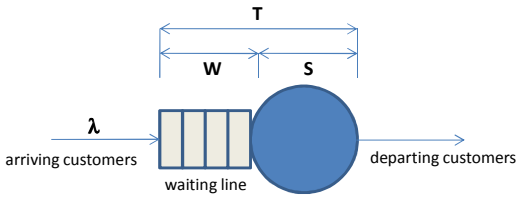


Figure 8. Single queue with one resource server structure

2.8.2 Control Theory

Modern control theory is a discipline that is concerned with the formal basis of analysis and design of computer systems for control and management. Its scope includes challenges and methods of designing *control algorithms,* i.e. formal rules (equations, procedures, applications), to determine control decisions, which are performed by specially designed and implemented plants able to process data and perform actions [77]. Management systems based on control theory, develops architecture for controlling target system to achieve a desired objective. Summarizing control theory enables analysis and construction algorithms and components for creating systems that achieve some predefined objectives and describes design techniques for determining the values of parameters in control algorithms. A practical evaluation is presented in [73, 74].

2.8.3 Fuzzy Logic

Similar to feedback control, there is a controller and a target system. However, there is no notion of reference input since the goal is optimization. Second, the feedback consists of both the measured output and the control input. Fuzzy control employs qualitative descriptions of systems to make it easier to specify controller actions, which are so-called linguistic variables that exist in one-to-one relation to numeric values that can be between 0 and 1. These variables indicate the direction and magnitude of change. Practical evaluation of fuzzy logic in the context of SLA management is elaborated in [75, 76].

2.8.4 Heuristics

Heuristics stand for strategies using readily accessible, though loosely applicable, information to control problem-solving processes in human beings and machines [79]. Such techniques are rules used to manage computer systems, solve problems and produce good solutions, without proving to be correct. When designing *control loops*, heuristic methods can supplement queuing networks, control theory and fuzzy logic algorithms to produce more accurate and efficient implementation. Practical exploitation of heuristics in the contexts of ACS is presented in [80].

2.8.5 Summary

The example methods of control loops construction described focus on the analytical mechanism used in performance an engineering process that might be automated by introducing an adaptive manager, which manages a self-adapting process driven by adaptation loops. During the analyzing and planning phases, actions are performed through the evaluation of control algorithms defined with the methods and implemented by the policies, which consist of three elements: an event, which triggers policy evaluation, a condition that makes a policy applicable, and a decision (or action) to be executed when the condition is met. The input data used in the evaluation process are acquired from resources using a sensor interface and for invoking actions, an effector interface exposed by the IMF platform.

2.9 Policy Representation

Adaptive and autonomic managers accept a goal specified as a policy, analyze the current situation, in the context of existing knowledge, and execute decisions. They manage their internal behavior and relationships with other elements in accordance with established business goals. The objectives can be defined within the *Governance and Policies* Layer. In the context of the S3 model, *policy governance* provides the ability of so-called policy-based management, enabling high-level definition of directives to be translated into specific actions and is a solution used to define adaptive or autonomic system behavior. The goal is specified by *policies*, which define a principle of the system management activity and relates the system state with appropriate *actions* (specifies a high-level goal of the system functioning), and is mapped to a form represented, for instance, by a mathematical model understood by an adaptation process. The policy is defined as a set of rules and may contain additional information like the rules'

evaluation order and descriptions. In such a context, according to [58], three types of policies can be distinguished:

- *Action Policies*: The lowest level of specification, they are in the form *IF (Condition) THEN (Action)*, where information about the current system state is recognized as a pattern (condition) defined in policy and, if current state matches pattern, then appropriate actions are taken. The example of such policy might be *IF (Transaction_Throughput < 100 TPS) THEN (Assign_CPU to Container_Pool).* The adaptive system is supposed to deliver input data used for condition evaluation and invoke actions on managed elements. These types of policies are described in [59,60,61].
- *Goal Policies*: Describe goals to be achieved without specifying how to attain them, for instance, *"Transaction_Throughput must be greater than 100 TPS"*. They are more powerful and are specific for autonomic systems, because the administrator defines only directions without exact knowledge how to influence runtime parameters. Construction of autonomic environments using these types of policies is more challenging and requires advanced modeling and planning services, which translate goals into actions [62,63,64].
- *Utility Function Policies*: These policies are the next level in the evolution of policy definition for autonomic systems, because they automatically specify the most optimal goal for the managed element. The employment of such techniques demands utilization of sophisticated modeling and optimization services that would translate utility functions into actions. The outgoing research activities are described in [65,66,68].

Policies can be represented with dedicated programming languages or more common formats like XML whose advantage is simplicity, popularity and universal way for description of structured data.

2.9.1 Policy Description Language

Policy Description Language (PDL) [48] was one the early policy languages invented in Bell Labs. It specifies policy as a declarative rules or propositions. The information model of the PDL consists of four main abstractions: *policy rule*, *event* and *condition* and the interpretation is that when the event occurs and the condition is true, then the specified action will be executed. The proposed model was later adopted by many subsequent policy languages and implementations [54].

2.9.2 Autonomic Computing Policy Language

Autonomic Computing Policy Language (ACPL) is a policy language created for IBM Policy Management for Autonomic Computing [49]. The concept extends the Policy Core Information Model (PCIM) [57] to support distributed IT systems management. Each policy rule can specify a condition and an action executed if the condition is true. Such a rule is also specified with a priority that specifies the relative importance of a particular policy in a policy group. There can

be two formats used to specify policy rules: an XML with an appropriate tooling support and a simple text format to define policies through regular text editors. The latter format has been standardized by DMTF[10] into CIM-SPL (Simple Policy Language for CIM).

2.9.3 CIM-SPL

CIM-SPL fully incorporates Core Information Model (CIM) [56] constructs, and includes many features and lessons learned from the design and use of other languages. The data used for evaluation are contained within CIM data structures and accessed via CIM Object Manager (CIMOM) defined within CIM Objects. CIM-SPL supports all intrinsic data types defined by CIM Meta Schema with *arithmetic*, *boolean* and *casting* operators. In general, this policy language is useful in creating policies based on CIM data model.

2.9.4 Rule Engines

Rule engines use rules that typically have the structure: *when <some conditions> then <do some action>*. They evaluate rules based on provided *facts* (describe state of the system) with an *inference engine* using pattern matching algorithms like RETE, Linear, Treat, Leaps. The most popular are RETE and solutions like CLIPS[11], Jess[12], Soar[13] or Drools[14] are based on this kind of pattern matching algorithm. Some of the advantages of using rule engines instead of developing independent solutions are *scalability and efficiency* (pattern matching algorithms are scalable even for big numbers of facts) and providing a *knowledge container* that allows collecting all rules in centralized way. Rules can be defined with languages that are similar to natural and easy to understand.

Some other policy languages exist. For example, eXtensible Access Control Markup Language (XACML) [51] can be used in the context of access control policies. Ponder [52] is good solution for specifying rules but very problematic in case of SLA contracts definition. A extensive comparison can be also found in [53,54].

2.9.5 Summary

The SOI provisioning process model is an example of a policy-driven system. The management of the Operational Systems Layer elements within such Runtime Infrastructure is automated via the usage of policies that express key decisions satisfying QoS goals. They can express adaptability strategies of already provisioned particular SOI elements i.e. managed resources.

[10] Distributed Management Task Force - http://www.dmtf.org/

[11] http://clipsrules.sourceforge.net

[12] http://www.jessrules.com

[13] http://www.zilonis.org

[14] http://www.jboss.org/drools

2.10 Technologies of Building Adaptable Systems

When considering requirements for adaptive management of SOI, there must be means provided that enable integration to intercept and redirect interactions between these virtualized resources and hosted middleware used for running application services. The adaptability of the IMF can be introduced on different levels of internal architecture. Elements used to provide adaptive resource management, such as sensor and effector interfaces and the Adaptation Manager, can be separated from the parts responsible for provisioning. There are a few concepts that can be used to achieve that.

2.10.1 Component-Based Design

Component-based design is a result of the evolution of concepts raised from the level of maturity attained in object-oriented design that promotes software reuse. Requirements specified by interfaces definition enable developers to implement required functionality independently. Such an interface is a contract specified by a service provider that clients might implement in components that can be developed, assembled and deployed by third parties using component-based middleware platforms e.g. *.NET* [134], *COM/DCOM* [133], *Corba Components Model* (CCM) [136] or *Enterprise Java Beans* (EJB) [137]. Two types of composition are supported, namely *static* and *dynamic*. When dealing with static composition, components are combined together during *compilation* phase to provide required services functionality. In contrast, dynamic composition allows components that are part of given service to be added, removed, and reconfigured at run-time. Such a process requires late binding support from the component framework that allows dynamic coupling of compatible services through well-defined interfaces.

2.10.2 Reflective Middleware

Middleware refers to a distributed platform of interfaces and services that reside ‘between’ the application and the operating system and aim to facilitate the development, deployment and management of distributed applications [114]. Figure 9 depicts middleware decomposed into four layers proposed by Schmidt [113]:

- *Host-Infrastructure Middleware*: Provides low-level interfaces for encapsulation of communication with OS, concurrency, memory management or network services. These interfaces are implemented by components dedicated for a given operating system and assure portability by elimination of many tedious, error-prone and non-portable aspects related to development and maintenance of networked services.
- *Distribution Middleware*: Specifies higher-level API's for distributed programming models and components that acts as a facade for automation of OS network services exposed by the host-infrastructure middleware. These interfaces enable clients to program distributed applications services by invoking procedures on dependant objects more transparently without dependencies like location, programming language and OS platform in mind.

- *Common Middleware Services*: Extends distribution middleware by providing higher-level services that allow service providers to concentrate on programming business logic without a need for "plumbing" code responsible for transactions, concurrency, resources pooling (number of threads or database connections) and others.
- *Domain-Specific Middleware Services*: Provides services tailored to the requirements of particular domains, such as e-finance, e-markets, health-care and aerospace, that are rather domain-specific and targeted at vertical markets.

Because the role of middleware is to hide low-level aspects related to resource distribution and platform heterogeneity from the business logic of application services, it is logical to extend middleware platforms with services that provide adaptive behavior related to aspects concerned with QoS. Reflective techniques are seen as the base for the development of the next generation of middleware platforms [116, 117] as they enable adaptation that is adjusting to meet the requirements of the environment or the application. This adaptation may proceed autonomously or half-autonomously on the base of the application deployment within the limits of a defined course of action of the users and the administrator [118]. When considering the middleware platforms, the reflectivity causes the accessibility of the mechanisms to the internal manipulation of the system services during their work. Reflective middleware systems are therefore the proper base to construct an adaptable and flexible management tools to implement the IMF. Reflection in the context of middleware environments enables the system to reason about and act upon itself through inspection and adaptation at run-time. Inspection must provide extensions that allow observation (e.g. to monitor current memory usage) while adaptation e.g. would allow specifying a number of objects in the in-memory cache to have a better match the system's current operating environment.

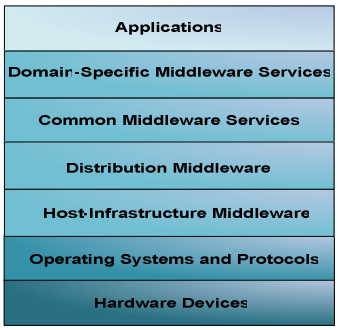


Figure 9. Middleware layers.

Dynamic adaptation in middleware can be realized with *parameter* and *compositional* adaptation described in much detail in [119].

2.10.3 Parameter Adaptation

Parameter adaptation involves the modification of environment variables or configuration parameters to adjust services behavior [121]. This adaptation is widely used in many solutions like the adjustment of values of the internet TCP protocol control window in response to apparent network congestion [125] or context-aware systems [126]. Parameter adaptation is rather easy to utilize/achieve; it is common to perform configuration changes to set parameters

of various service components to meet specific functional or non-functional requirements. The disadvantage of the technique is the lack of mechanisms that enable adopting algorithms or components that were not provided during the implementation or assembly phase; only existing adaptation strategies can be used by such a service.

2.10.4 Compositional Adaptation

Compositional adaptation enables algorithmic or structural parts exchange of an application by modifying its component composition at run-time i.e. new components can be exchanged, removed or inserted [121]. Such a technique, in contrast to *parameter adaptation*, enables *dynamic recomposition* of an application to use newly defined adaptation strategies, which is the glue when services are running in an environment with limitations on CPU, memory or network resources. Dynamic decomposition might also support choosing appropriate components in *context-aware* management systems to be equipped with needed sensor and effectors interfaces, depending on the type of instrumented resource. The realization of compositional adaptation has been aided by the confluence of three key technologies [119]: *separation of concerns*, *computational reflection* and the already discussed *component-based design*.

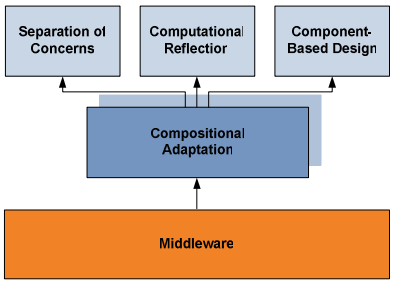


Figure 10. Main technologies contributing compositional adaptation [100].

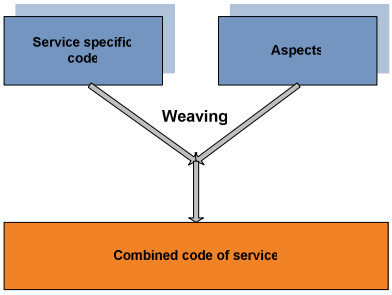


Figure 11. Aspects weaving structure.

Separation of Concerns enables developments' separation of the functional behavior (business logic) and the crosscutting concerns (e.g. security, quality of service and fault tolerance) of services. Such separation provides an effective means for services maintenance and promotes reusing, which are very important principles in software engineering. The most widely used approach today seems to be AOP (aspect-oriented programming) [132] that promote modularization of software. Modules that contain the business logic of a given service (*service specific code*) and non-specific code (*aspects*) are then merged during the weaving phase to create the final services' code (Figure 11).

Computational Reflection is a technique that allows a system to maintain information about itself (meta-information) and use this information to change its behavior (adapt) [122, 123]. Reflection enables a system to reveal (selected) details of its implementation without compromising portability [124]. It hides unnecessary implementation details at a level of abstraction through the compromise of two activities i.e. *introspection* and *intercession*; this enables changes to the system behavior. Introspection enables an application to observe its own

behavior and intercession enables a system or application to act on these observations and modify its own behavior [115].

2.10.5 Summary

Adaptable architecture is supposed to use a component-based approach in its implementation that integrates well into custom solutions driven by well-established industry standards. These components should use reflective middleware techniques with parameter and compositional adaptation enabling the implementation of adaptable SOI provisioning over LVC solution in a systematic and principled manner. However, architecture analysis must be performed that defines requirements for SOI platform and identified based on infrastructure elements and their representation.

2.11 Related Work

Provisioning of application middleware and components deployment can be performed using package managers (*RPM*, *dpkg*) and application installers (*Windows installers* or *InstallShield*) [2][15]. Provisioning in distributed environments is more challenging, which introduces several additional issues to be resolved like heterogeneity of resources, coordination of deployment actions, support for distributed sequencing of installation synchronization and dependency resolution activities [3].

Currently there are many platforms which support provisioning and management of VEI based on VM virtualization technologies (Table 1). OpenNebula[16] orchestrates storage, network, virtualization, monitoring and security technologies to enable the dynamic placement of multi-tier services (groups of interconnected virtual machines) on distributed infrastructures, combining both data center resources and remote cloud resources, according to allocation policies. Nimbus[17] provides a toolset for infrastructure provisioning with emphasis on the needs of science, but many non-scientific use cases are supported as well. It allows a client to lease remote resources by deploying virtual machines on those resources and configuring them to represent an environment (so-called "Virtual Workspace Service") desired by the user. Eucalyptus[18] has its origins in the VGrADS[19] project, which developed software systems that simplify and accelerate the development of Grid applications and services, while delivering high levels of performance and resource efficiency. Eucalyptus provides services to control environment required by an application that is a container with a set of data center virtualized resources (machines, storage and network), which are provisioned by user on demand. The

[15] pp. 18-20

[16] http://opennebula.org/about:technology

[17] http://www.nimbusproject.org/about/

[18] http://www.eucalyptus.com/

[19] http://vgrads.rice.edu/

Tashi[20] project aims to build a software infrastructure that enables the Big Data that are stored in a cluster/data center to be accessed, shared, manipulated, and computed by remote users in a convenient, efficient, and safe manner. There is a plan for enabling on-demand provisioning of storage and compute resources through definition of open, non-proprietary interfaces for management tasks such as observation, inference, planning, and actuation.

Table 1. Comparison of selected platforms for VEI provisioning.

	OpenNebula	**Nimbus**	**Eucalyptus**	**Tashi**
Virtualization Technologies	Linux distros with Xen, KVM, VMware hypervisors. Not supporting LVC.	Linux distros with Xen and KVM hypervisors. Not supporting LVC.	Linux distros with Xen, KVM, VMware's vSphere hypervisors (ESX, ESXi are optional). Not supporting LVC.	Linux distros with Xen, KVM, Qemu, and Kqemu hypervisors. Not supporting LVC.
Programming Interfaces	Rubby, C++, JAVA	Python, JAVA	Frontend implemented with Java, and helper components in Python. Core components implemented with C language.	Python
Adaptable Architecture	Enables the deployment of any cloud architecture: private, public, hybrid and federated; customizable plug-ins to access virtualization, storage, information, authentication/authorization and remote cloud services; new plug-ins can be easily written in any language.	Possible integration with other platforms like OpenNebula and Nagios. Proprietary platforms are integrated through implementation of modules in Spring framework.	Not designed with extensability in mind, because there is no standardized mechanism for writing custom extensions.	Extensible end-to-end system management (non-proprietary interfaces for management tasks such as observation, inference, planning, and actuation). Flexible storage models and machine models i.e. VMM-agnostic, able to run different virtual machine.

[20] http://wiki.apache.org/incubator/TashiProposal

Policy Management	Integration with HAIZEA[21] providing scheduling backend. Policy engine supports Python scripts.	No available modules but 3rd-party extensions exist [96].	Not supported. It only provides some pre-defined strategies related to VM set by administrator in configuration files.	Not mentioned[22].
Frontend Interface	XMLRPC API and Amazon Elastic Compute Cloud (EC2).	EC2	EC2	EC2

Many of the discussed platforms integrate with the specific virtualization technologies using LibVirt [38] library aimed at harmonizing the management of any virtualization mechanism. Regardless of the technology used (Xen, VMware, OpenVZ, and others), LibVirt provides a uniform access interface, based on XML files, and programming in different languages. In the context of lightweight virtualization, only Linux implementations are supported and Solaris Containers are not. However, as these platforms currently provide a subset of specific Infrastructure Services, they are rather Virtual Machines technology oriented and cannot be used in the context of adaptable SOI provisioning over Lightweight Virtualization Containers (even LibVirt exploitation does not necessary enable support). In addition, they do not provide integrated framework for provisioning, monitoring and management of the Runtime Infrastructures representing elements of the Operational Systems Layer and are not accessible with a uniform interface. They also offer limited support for policy languages.

N1 Provisioning Platform (N1PP) is SUN Microsystems vision of architecture and services for optimizing computing infrastructure provisioning and provides all the services for provisioning heterogeneous hardware and software environments. It allows an automation of tasks related to provisioning computational infrastructure and resource configuration to meet fluctuations in the computing load that can be performed by the system administrators. N1 Provisioning Server System (N1PSS) [5] provides a comprehensive infrastructure automation solution and enhances the management, visibility, and control of the Sun Blade System. The services provided allow the complete design configuration, deployment and management of multiple secure, independent, logical server farms. Another component, N1 Service Provisioning System (N1SPS) [6] automates deployment, configuration and analysis of distributed applications. It provides centralized control over deployments and configurations and enables organizations to manage applications as distinct units, rather than as a great set of installation files. N1SPS defines an object-model for a set of *Components* and *Plans* for system configuration, service provisioning and application deployment automation. It also includes a component library with

[21] http://haizea.cs.uchicago.edu

[22] At the time of writing the thesis, Tashi is in incubation phase.

templates for most common middlewares, databases and operating systems and enables development of their own plugins, which can be integrated with other provisioning tools.

There are also projects that are related to aspects such as policy-based management of Runtime Infrastructures. The Policy Management for Autonomic Computing (PMAC) [98] framework was developed as an outcome of IBM's research in autonomic systems. Although it is designed with the ability to manage various resources, in the opinion of the author, the services provided are very complex, heavy and an inefficient approach. Management of distributed heterogeneous SOI will require tremendous effort related to the managed resource descriptors definition modeled as stateful Web Services with WSDL with WS Resource Framework[23] documents. Autonomia [99] provides users with all the tools required to specify the appropriate control, management schemes and the services to configure and deploy the required software and network resources and then manage their operations to meet the overall system requirements. In the Adaptive Distributed Virtual Computing Environment (ADVICE) [100], the users are provided with a seamless parallel and distributed computing environment that provides all the software tools to develop, schedule, run and visualize large scale parallel and distributed applications. Users are provided servers, Visualization and Editing Server (VES) and Control and Management Server (CMS) to develop parallel and distributed applications that can run on fixed or mobile computing resources.

These frameworks provide all the tools required to specify appropriate control and management schemes, deploy and configure the required software and hardware resources, run the application and provide online monitoring and management to maintain the desired autonomic attributes of applications as well as system services. Unfortunately, their architecture is not suitable for managing complex Runtime Infrastructures with many heterogeneous elements built according to the SOI paradigm and provisioned with lightweight virtualization.

2.12 Summary

As stated in this chapter, Service Oriented Infrastructure introduces a modern conceptual model of the Infrastructure Management Framework for provisioning and management of Runtime Infrastructure that combines virtualization technologies and policy based management to ensure specific QoS levels for running application services. SOI concentrates on orchestration of virtualized compute, network and storage resources, providing an abstraction layer through Infrastructure Services exposition, which enables easier sharing of physical infrastructure and dynamic adjustment of resource attributes. This evolution of IT management towards service orientation provides a modular infrastructure fabric managed as Service Oriented Architecture, ensuring that resources can be provisioned. Currently many provisioning tools can be used for application installation, component deployment and VEI provisioning. Unfortunately they are not standardized and integrated into a unified stack of tools, and do not provide support for SLA

[23] http://www.ibm.com/developerworks/library/specification/ws-resource/

enforcement (though manual monitoring and management is supported). Applying the SOI paradigm to integration can deliver a comprehensive Infrastructure Management Framework that extended with self-managing services, enable on-demand provisioning of Runtime Infrastructure over Lightweight Virtualization Containers. Adaptable SOI provisioning with LVC requires a framework implemented with software components, which promote adaptable architecture through the evaluation of design patterns, reflective middleware and adaptation techniques. Introducing policy-based management with sophisticated methods of control loop definition provides effective facilities for ensuring QoS. The remaining part of the thesis elaborates the possibilities and requirements for the design and implementation of the IMF for adaptable SOI provisioning with lightweight virtualization.

3 Model of Adaptable Provisioning of Service Oriented Infrastructure with Lightweight Virtualization

This chapter specifies the model of an adaptable SOI with a lightweight virtualization provisioning process with its most important elements of the Infrastructure Management Framework that must be present for realization of the model. It elaborates a particular group of SOI Virtualized and Physical Infrastructure Services for lightweight virtualization, Lightweight Virtualization Containers Application Appliances and discusses the Solaris platform. It explains how lightweight virtualization and the SOI paradigm can also be exploited in the context of Cloud Computing.

Adaptable provisioning of SOI with lightweight virtualization requires provisioning of VEI, Application Execution Environments (middleware), deployment of application components and continuous active monitoring and management activities. Such a process requires exposition of applicable interfaces for acquiring data and effecting virtualized and physical elements (services) of SOI, which are indispensable in the case of adaptable provisioning and management of Runtime Infrastructure to meet specified QoS parameters. It is also important to provide means that enable a definition of adaptive SOI behavior by adaptation strategies (control loops) expressed with policies. The model of the provisioning process should define such activities, which require well-structured and organized SOI Infrastructure Services implemented by software components.

3.1 Elements of Adaptable SOI Provisioning with Lightweight Virtualization

Adaptable provisioning of SOI providing Runtime Infrastructure for application services that require both operating system and middleware is more complex because there can be many infrastructure elements involved to make the application successfully available to users. Some elements must be provisioned and some are already present like databases in which data can be stored, load-balancers or LDAP services.

3.1.1 Required Elements by the SOI Provisioning Process

In opinion of the author, the natural order of implementing an adaptable SOI provisioning solution is to first devote attention to Managed Resources construction. This is justified by the fact that a management policy and an adaptation strategy of an adaptable system must be expressed in the context of the capability provided by Managed Resources. The concept of the IMF realizing adaptable SOI provisioning platform can be defined as set of components with architectural building blocks, decisions and interactions among elements. It must promote adaptable architecture, enabling realization of adaptive behavior that follows the typical control loop's construction patterns and consists of *Manageability Components* described below (Figure 12).

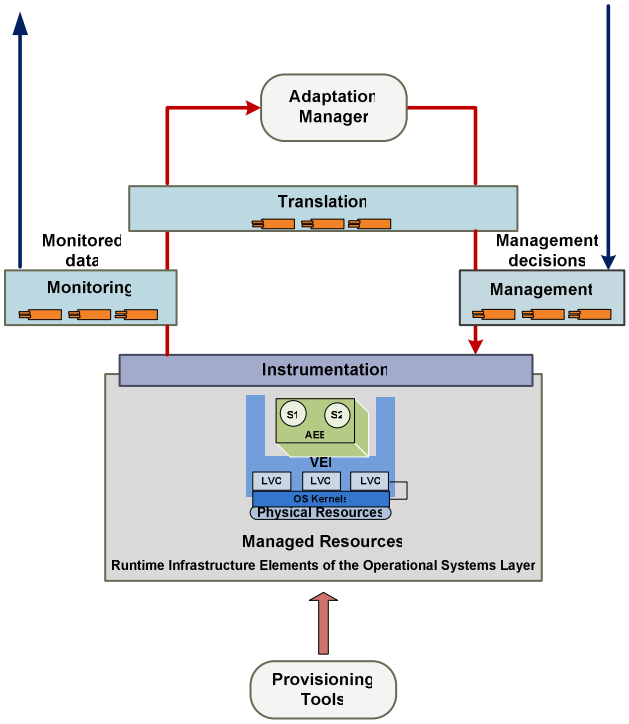


Figure 12. Elements of adaptable SOI provisioning process.

The *Managed Resource* is particular element of IT infrastructure (e.g. lightweight container, middleware or application services) instrumented with sensors (*Monitoring Components*) and effectors (*Management Components*) that can be monitored and managed. The instrumentation process exposes the resources, through Infrastructure Services, with the ability to acquire the data required to characterize its activity and install mechanisms for changing the parameters or configuration, according to decisions imposed by the control loop. Such instrumentation should be non-intrusive, selective and dynamic (performed during runtime). Monitoring Components are responsible for selected metrics calculation and events processing.

To enact the adaptation strategy, the Adaptation Manager designed according to ACS architecture uses output data from the Monitoring Components, which must be transformed into a format used by the given implementation of a policy engine, thus the *Translation Component* plays the role of a *Translation Layer*. The AM ensures policy based management and might be based on the standard control theory, fuzzy logic or heuristics, which is suitable when a

decision has to be based on many different parameters and a precise mathematical model of the Managed Resource is not available. The outcome of policy evaluation, which is an action selected by the Adaptation Manager, is converted by the Exposition Component to a format acceptable by the *Management Component*. This component enforces management actions using effectors, which instrument the Managed Resource.

Provisioning components must expose functionality for creation of Runtime Infrastructures over lightweight containers and deployment of application services. Tools for SOI infrastructure provisioning, rather than operating on pure installation distributions, must also capture knowledge about the application to be deployed with special emphasis on *installation*, *registration*, *configuration methods* and *dependency requirements*. The following elements of a platform supporting provisioning can be distinguished, which capture knowledge about provisioned Runtime Infrastructure elements, which are technology neutral, shared and used in any IT environment [4]:

- *Infrastructure and Application model* is a data-driven representation of an application and infrastructure (target server and middleware) for deployment and configuration. It is applicable to different server environments that might require different deployments or configuration choices (for example, testing versus production). The model is configurable through variables specified by an admin with values that can be generated when initiating the provisioning procedure.
- *Automated model building* should be equipped with a model builder module for the automation of model creation. Applications installed on reference servers can be "mirrored" and checked into a repository as a template, which can also be versioned. These templates can be used later for provisioning on any computational node.
- *Application Deployment management* requires each target computational node to be equipped with a *provisioning agent* that is provided with a runtime environment in which modeled methods can be executed.
- *Resource Configuration management* capabilities are required to determine the desired configuration of an application’s runtime environment. This includes computational resources (number of CPUs and amount of memory assigned to the target computational node) and middleware (database resource including number of connections or thread pool). It would then generate the configuration by setting values in appropriate text or XML files or by replacing variables with actual values in modeled methods.
- *Dependency managemen*t ensures that after the initial configuration required by particular elements, activities are performed that verify that all dependencies for a successful deployment have been met.

Some products, like J2EE application servers[24], contain some of these procedures as built-in functionality. Custom solutions must have their own framework implemented which deploy, install, upgrade and start applications based on knowledge of the underlying system, the business domain and exposed interfaces supporting the SOI provisioning process.

3.1.2 Classification of Infrastructure Services

An administrator of IT infrastructure designed according to SOI must be able to monitor and manage the configuration of particular elements within the S3's OSL through specification of QoS goals (*QoS Layer*) for provisioned elements, definition of provisioning procedures and adaptation policies (*Governance and Policies Layer*) stored in a central repository. The IMF solution for adaptable SOI provisioning with lightweight virtualization must enable provisioning of Runtime Infrastructure elements within the *Operational Systems Layer* like databases, transaction-processing systems and middleware's (CORBA, J2EE, .NET) with application services running over Lightweight Virtualization Containers. The proposed approach must support aspects such as:

- The VEI environment creation through provisioning of Lightweight Virtualization Container instances with allocated compute resources distributed over physical servers,
- The provisioned VEI elements can be modified during runtime so the running application services can obtain or release access to physical resources during execution,
- The provisioning process of Application Execution Environments must support arbitrary vendors to allow deployment of application services implemented with any technology, which is possible if the current exhibition is supported by the appropriate Infrastructure Services.

The aim of adaptable SOI provisioning with LVC is a *dynamic* management of the life cycle of infrastructure elements, like adding, removing or halting LVC instances and deployment of application components, which help to react to changing business requirements and utilize computing resources more efficiently. During adaptable provisioning, defined portions of physical compute resources are allocated to LVC instances with specific number of CPU's, RAM, storage and network resources, which are monitored and managed to support scaling out and then rapidly released to scale back. Such a process is driven by *policy-based resource allocation,* allowing computing infrastructure elements to be assigned to particular resource pools, provisioned on-behalf of a particular VEI instance or instances. Monitoring and management interfaces provided by virtualized and physical services are supposed to monitor particular LVC instances within provisioned VEI instances. It includes observing hosted workloads represented by application services, enhancing utilization of computing resources,

[24] The Java 2 Platform, Enterprise Edition (J2EE) Deployment API Specification - version 1.1, is defined under JSR 88 in the Java Community Process. It defines standard APIs to enable deployment of components and applications based on the J2EE platform, as well as the development of platform-independent deployment tools.

checking for performance degradation and determines the appropriate actions to be taken related to resource allocation needed by specific application components or processes. A reference model of SOI delivers virtualized and physical services that can be described in categories such as *Platform as a Service* (PaaS), *Infrastructure as a Service* (IaaS) and *Hardware as a Service* (HaaS) - Figure 13.

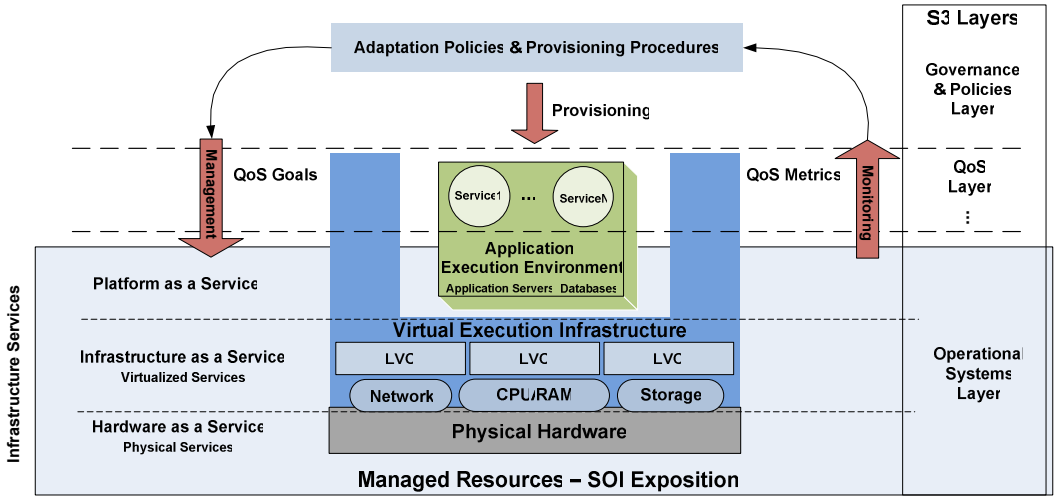


Figure 13. Virtual and physical Infrastructure Services for SOI with LVC provisioning.

PaaS exposes services on top of a given software platform that is provided as a "service" to build high-level service enabling provisioning of Runtime Infrastructure for *application deployment management*. PaaS might produce a platform that contains a properly configured OS, databases and application server with already-deployed application components, often serving tenant application[25], and enabling software delivery through an on-demand business model.

IaaS enable monitoring of SLA, resource utilization and capacity and controlling of CPU assignments, network and I/O resources with a usage of appropriate standardized services over the network. Physical infrastructure such as servers, storage systems, switches, routers, and other physical devices systems are pooled and made available to handle workloads that range from application components to high-performance computing applications. This leads to a case in which hardware is becoming an integral part of SOI infrastructure, named HaaS, under which physical devices are exposed in the IT infrastructure with service-oriented components[26].

[25] There is a distinction between multi-tenancy, isolated tenancy and mega-tenancy. Multi-tenancy uses the same instances of database or other environments with the same version of an application. Isolated tenancy devotes a dedicated and unique database to each application installation, offering flexible software customization with many applications' version. Mega-tenancy is a sort of combination of the multi and isolated tenancies where execution environments are dedicated with a shared database.

[26] This can be accomplished for instance with the Intelligent Platform Management Interface (IPMI) or Lights-Out Management (LOM) specifications that defines a set of common interfaces to a computer system that system administrator can use to monitor system health and manage the system.

PaaS, IaaS and HaaS provide *resource configuration management* facilities enabling orchestration of Infrastructure Services to manage compute resource configuration and provisioning of Runtime Infrastructure. This organization of SOI's Infrastructure Services enables *Runtime Infrastructure* provisioning where Lightweight Virtualization Containers can be provisioned over distributed physical compute, network and storage resources. These Infrastructure Services are orchestrated during the whole process of adaptable provisioning, steered by adaptation loops and expressed by policies governing access to the Runtime Infrastructure. These adaptation loops support dynamic infrastructure capacity management with guaranteed reservations and enable both manual and on-demand (triggered by events occurred) SOI provisioning of complex multi-tiered applications over LVC with all required Infrastructure Services. Such construction of the required elements for SOI provisioning enables automation of the management activities for specific QoS contracts and, in case some problems occur, it attempts to reconfigure system to fulfill the contracts.

3.1.3 Lightweight Virtualization Containers Application Appliances

SOI provisioning with lightweight virtualization can be carried out using Lightweight Virtualization Container Application Appliances (LVC-AA), which are pre-engineered and validated units of IT infrastructure (*Infrastructure and Application model*) that conform to standards and best practices related to security, availability and performance. Such an approach would enable Runtime Infrastructure acquisition, deployments and operations in a very efficient way and support expanding capacity as the architecture is very flexible and extensible. Prepared packages of LVC-AA can be bundled as units of assemblies within LVC configuration templates defined by administrators, stored in repositories and next provisioned on demand. LVC-AA is the basic deployment object that is partially configured to serving database, application server or CRM/ERP functionality with various OS vendors and their versions. Reference configurations are prepared on *gold* servers and tested. After validation and an approving certification process, snapshots are created and stored in a repository with metadata describing installed software and the LVC version. The definition of such assemblies can be performed according to design patterns facilitating deployment, scalability, security and ease of integration with other services. Such a picture of SOI provisioning can be used in the context of infrastructure provisioned over LVC (Figure 14); although the solution can use any virtualization technique as requirements dictate.

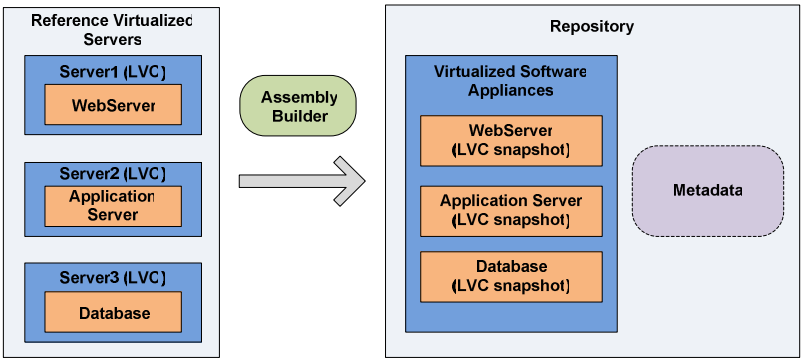


Figure 14. Creation process of LVC-AA.

An *Assembly Builder* tool realizes *automated model building* services and performs snapshot of particular reference configurations and places it in a repository with metadata. Often, LVC-AA configuration templates prepared contain only pre-installed middleware with the core functionality exposing administration services used for configuration of new instances during provisioning on target execution nodes.

Each assembly provides specific elements of the OSL optimized according to capacity guidelines based on matched performance and operational characteristics during the definition of a particular LVC-AA configuration template. When provisioning such LVC-AA, a system operator specifies configuration of compute resources (RAM, number or portions of CPUs) and network connections and bandwidth. Dynamic provisioning requires functionality for performing activities related to adjusting the internal configuration of middleware and should be *adapted* to assigned virtualized resources without manual intervention, ensuring compute resources are utilized more efficiently. To provide availability and scalability, Runtime Infrastructure elements within assemblies can be clustered (more than one assembly provisioned), ensuring the ability to handle peak loads and add extra LVC instances to particular VEI on demand to fulfill SLA contracts.

The assemblies must be equipped with agents implemented by software components, exposing SOI Infrastructure Services providing support for SLA management, resource monitoring and accounting, activated after VEI provisioning and application deployment. If needed, the provisioned LVC can be extended dynamically with components required for consistent policy enforcement and IT operational processes.

3.1.4 Summary

Provisioning of SOI computational infrastructure with LVC is a sophisticated process, due to the large number of logical elements that have to be managed and the complexity of contemporary application installation. The process is dependent on many parameters, which must be customized for each computational element of the system. The IMF realizing adaptable SOI provisioning process must also be equipped with many automation services supporting adaptable provisioning.

3.2 Solaris Lightweight Virtualization Platform

Efficient realization of the SOI provisioning process with lightweight virtualization requires using a virtualization platform that provides functionality supporting advanced resource management with low-level system interfaces should be easy enough for instrumentation and exposed as SOI Virtualized Services. Solaris 10 provides an integrated stack of tools, which besides *Solaris Containers* virtualization technology, offers powerful facilities, which can play an important role in the SOI provisioning process and may be an element exploited by the IMF [28]. Solaris platform is the most representative example of implementation of the lightweight virtualization.

- *DTrace*: Infrastructure for flexible and extensive instrumentation of probes that allows observation of system performance for tuning. A framework helps to determine *hotspots* in running applications. Solaris OS contains about 40,000 kernels of built-in probes that are points of instrumentation which enable monitoring of applications without recompilation, restarting an application and rebooting the system. Each native application can also define its custom probes, which are sent to *consumers* and analyzed to find potential bottlenecks. Such consumers can be encapsulated within *sensors* implemented with software components exposed as SOI's Infrastructure Service.
- *Zeta File System* (ZFS): Integrates volume management functions, but there is no volume manager notion, rather a concept of a storage pool, where capacity is provided on demand and can be dynamically resized. Tasks related to administration are very easy to perform with the tools provided and can be done even by users who are not experienced in volume management. ZFS enables creation of file system snapshots, cloning, recovering and restoring them later on other destinations. Such functionality is highly serviceable when constructing an environment for delivering and later provisioning LVC-AA.
- *Process Rights Management*: Extends security functions known commonly in UNIX into set of privileges, allowing granting processes as many rights as needed and make super user rights more granular. Individual processes are granted permissions that they absolutely need to work. When providing Infrastructure Services responsible for managing various parts of Runtime Infrastructure, they can be assigned profiles that contain privileges to perform only a specific set of operations. In the event of specific services' compromise, the attacker would have access only to limited functionality.
- *Solaris Fault Management (SFM)*: Provides services, which allow diagnosing problems automatically and predicting them in advance. It contains a Predictive Self-Healing component, which provides automatic recovery from software and hardware failures as well as administrative errors. These errors are correlated according to failed components and recommended actions are performed, ensuring fault and service isolation in case of unexpected failures and anomalies.
- *Service Management Facilities (SMF)*: Offers a unified model of services and management activities. Services are represented as objects that have defined dependencies. When errors (administration, bug or hardware) occur, services are restarted in a well-defined dependency order. It is also a core service of Predictive Seal-Healing technology. Workloads performing computations managed by SMF are described via XML descriptors that handle boot-up, access management and self-healing. Particular elements of the SOI provisioning framework can be controlled through the SMF introducing self-healing.

- *Crossbow*[27]: Network components, which virtualize a physical NIC (Network Interface Card) into multiple Virtual NICs (VNICs) that can be assigned to virtualized OS domains such as xVM, LDOM and LVC. Each Virtual NIC can be assigned its own priority and bandwidth on a shared NIC without causing any performance degradation.
- *Extended Accounting*: Enables collection of statistics at the process, task and network flow levels inside of a given LVC. A system administrator, for calculating the cost of running specified users' computations, can next analyze these statistics for billing purposes or predict future requirements for computational resources.

Solaris OS offers an advanced environment for *hierarchical* virtualization (Figure 15) and allows specifying resource consumption limits on the following elements:

- *Zone*: Lightweight Virtualization Container is virtualized OS instance, also named Solaris Container,
- *Project:* Identifies running workloads,
- *Task:* Workload component,
- *Processes:* Running processes within a task.

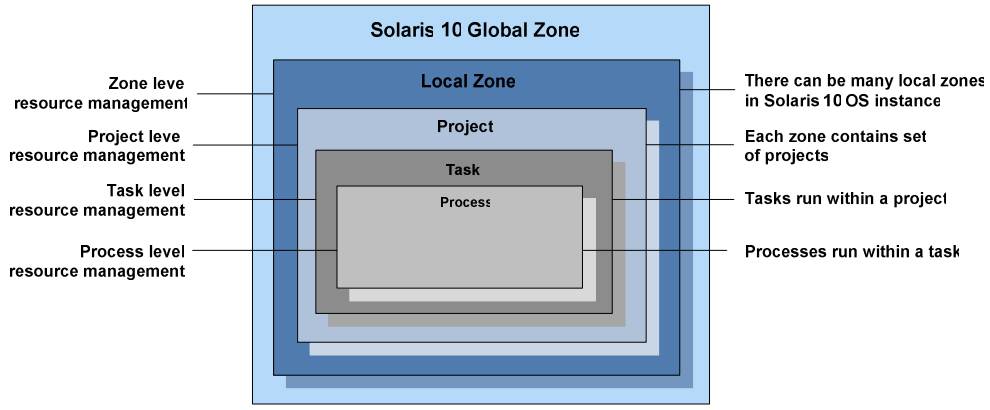


Figure 15. Solaris virtualization technology overview.

Solaris also integrates heavyweight virtualization based on LDOM and xVM technologies that both can be combined with LVC and offering even more flexibility for hardware resources partitioning, founding multi-level virtualization[28]. The availability of physical and virtualized SOI services can be managed by a SUN Cluster framework [29], [31] which ensures such core infrastructure elements, containing a single point of failures, are under the full protection of a cluster framework integrated with the Operating System.

[27] Since OpenSolaris 2009.6 and Solaris 11

[28] Section 2.3.4

3.2.1 Solaris Lightweight Virtualization Containers

Solaris LVC combine two technologies, namely: Solaris Zones and Solaris Resource Manager (SRM). The Solaris Zones technology provides virtualized isolated computational domains managed by *global zone*, which is present by default; there is always one single global zone and many other local zones (LVC's)[29]. Each LVC has its own host name, IP addresses, file system, user database and booting capabilities, thus retaining full security for hosted applications. Applications in a single container cannot detect, monitor or affect processes running in another zone. One exception is the global zone, which encompasses the entire system and is comparable to a normal Solaris OS instance. It has access to physical hardware and can see and control all processes. The administrator of the global zone can control the system as a whole, easily performing the administration tasks related to VEI provisioning and management and each LVC can have its own administrator as well.

3.2.2 Solaris Projects

In Solaris, workloads are identified by entities called *projects* and *tasks*. A project is an "*administration tag"* of group-related computations. A task is a group of related processes, executing in the same project, which represent a workload component. Each task is associated with one project and each process can only be a member of one task, but processes might be migrated between tasks. Once workloads are identified and labeled using *projects*, the next step is specification of how compute resources are assigned and monitored, which involves measuring workload resource consumption.

3.2.3 Solaris Resource Manager

The Solaris Resource Manager enables fine-grained measurement and dynamic control of workloads through a mechanism of *resource controls* (rctl) available for resources such as CPU, memory, ports[30], message queues[31] and lightweight processes (LWPs) [23] - summarized in Table 2. The SRM provides OS support for QoS guarantee of the elements within the OSL. Most resource controls are configured through the project database, except for zones' resource controls stored in the descriptor of each configured LVC. Fair Share Scheduler (FSS) divides CPU resources according to assigned shares and allocates CPU time for zones and projects. In Solaris 10, resource controls *project.cpu-shares* and *zone.cpu-shares* are used to specify CPU shares for projects and containers respectively, providing the so-called feature *Two-Level Fair*

[29] Notion of Solaris zone and Ligtweight Virtualization Container will be used interchangeably.

[30] It is a generic API for dealing with event sources (sockets, timers, pipes, message queues etc). It provides a scalable way for multiple threads or processes to wait for multiple pending asynchronous events from multiple objects.

[31] The maximum number of message queue IDs allowed for the project is controlled by project.max-msg-ids. For the process, there is process.max-msg-qbytes resource control that manages the maximum number of bytes of messages on a message queue and process.max-msg-messages – the maximum number of messages on a message queue.

Share Scheduling. Solaris LVC entitlement for CPU shares is partitioned between projects in the zone. The values of these resource controls can be set statically during provisioning of LVC and changed dynamically when QoS requirement is modified or other LVC instances negatively affect system performance. *Lightweight processes* are controlled with limits on the maximum number of processes within particular project, while others provide constraints on CPU time used or file descriptors. The resource controls also enable limiting memory and network bandwidth usage.

Table 2. Solaris Resource Manager controls.

Entity	Resource Controls
Zone	*zone.cpu-share* *zone.cpu-caps* *zone.max-lwps* *zone.max-locked-memory*
Project	*project.cpu-shares* *project.cpu-caps* *project.max-port-ids* *project.max-shm-ids* *project.max-sem-ids* *project.max-msg-ids* *project.max-lwps* *project.max-tasks* *project.max-contracts* *project.max-shm-memory* *project.max-device-locked-memory* *project.max-crypto-memory*
Task	*task.max-cpu-time* *task.max-lwps*
Process	*process.max-cpu-time* *process.max-file-descriptor* *process.max-file-size* *process.max-core-size* *process.max-data-size* *process.max-stack-size* *process.max-port-events* *process.max-sem-ops* *process.max-sem-nsems* *process.max-msg-qbytes* *process.max-address-space* *process.max-msg-messages*

A particular action identified with a specific resource control can be assigned some reactions expressed with following policies. They are useful for scenarios in which monitored resource usage exceeds assigned limits and can affect the progress of an application.

- ***none*** - No action is taken on resource requests for the application. It is also possible to enable a global message that displays when the resource control is exceeded, although the process exceeding the threshold is not affected. Messages can be observed in a system's global log file. System administrator to find the correct value of a resource limit can use such information, which can be reported automatically through the internal services of the IMF.
- ***deny*** - Resource requests for an amount that is greater than the threshold are denied. For example, a *task.max-lwps* resource control with a "deny" action causes a fork system call to fail if the new process would exceed the control value. Such a policy could help to avoid excessive resource usage, useful in such scenarios as denial of service attacks.

- ***signal*** - A signal is sent to the process when the threshold value is exceeded, allowing to proactively manage resource contention problems.

Beside resource controls, SRM also provides *Dynamic Resource Pools* (DRP) that enable a set of CPUs to be assigned to specific LVC. They can also be operated using separate schedulers and policies. Solaris LVC with configured projects can be assigned only specific portions of CPU resources when FSS is used by a particular DRP.

The functionality provided by the SRM allows dynamically changing a resource setting for the containers, projects, tasks and processes. Such a mechanism is suitable when the requirement for dynamic provisioning of SOI is crucial, because there is direct support for such facilities in the Operational Systems Layer running over the Solaris lightweight virtualization containers layer. As an example, test results are presented with SRM and two instances of Tomcat application server running within separate LVC instances (Figure 16). The discussed test was evaluated with two scenarios with a generated load of 75 virtual users targeted for each workload concurrently. The purpose was to present the impact of dynamic resource control on the performance of an application server in terms of request throughput.

1. *Workloads tomcat1 and tomcat2 assigned two and one CPU shares appropriately*. Because *tomcat1* has twice as much assigned share, at the same time it is able to process proportionally more requests. After *tomcat1* is finished, the remaining CPU resources are available to *tomcat2* in which a substantial rise in throughput is observed.
2. *The tomcat1 and tomcat2 workloads during the test duration were switched CPU shares*. The dynamic change of assigned CPU resources influences the substantial rise and drop of tomcat2 and tomcat1's throughput appropriately and such a result is up to the pattern of the FSS.

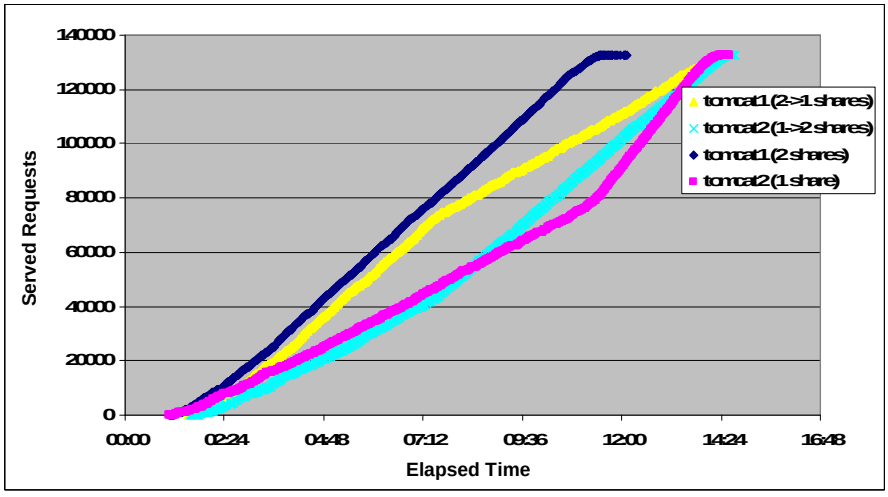


Figure 16. Dependence of the number of served requests on the allocation of CPU shares.

The elaborated scenario presents a useful mechanism for managing QoS of workloads running over LVC. Although the steps related to the adjustment of resource control values were executed manually, such activities can be automated by introducing adaptation loops equipped with knowledge of how to adapt the provisioned LVC to running workloads.

3.2.4 Provisioning Challenges of Solaris Lightweight Virtualization

The presented advanced virtualization and resource management framework provided by Solaris 10 introduces a number of logical elements that must be provisioned, configured and monitored. Operations related to lifecycle e.g. shutdown, halt, boot, ready, modifying LVC configuration (devices, file systems, attributes, resource controls) and adding or removing containers are required. Also adding or removing projects, modifying existing projects' attributes (name, comment, assigned users and groups) and controlling resource usage with resource controls are needed

The virtualized services of SOI must expose interface for native SRM integration to secure resources for an assured QoS and must be able to collect and publish various kinds of information about provisioned LVC instances on distributed physical nodes. The SRM provides many resource controls, which can be treated as manageable entities, and enables easy control of how many system resources are allocated to individual applications running with LVC or projects, but finding the right resource setting ensuring required QoS is very challenging. This can be solved by the introduction of control loops specified by the ACS model and required sensors and effector components. The SOI virtualized services for LVC monitoring must expose real-time information about resource usage of running workloads, which are measured periodically. Software components, which provide virtualized services, must instrument native Solaris interfaces to acquire such data.

3.2.5 Summary

Solaris lightweight virtualization provides a very effective framework for workloads management in terms of isolation and resource management facilities. Provided services enable granular resource partitioning for LVC instances within single OS instance and for workloads within LVC instance. Such functionality, when comparing to VM or domains virtualization technologies, is very effective because reduces the number of provisioned elements when resource guarantees for QoS enforcements are crucial. Components exposing services for SOI provisioning with Solaris-LVC must be automated as much as possible and require only minimal configuration with decent default options encapsulating the complexity of the provisioning process of Solaris 10 virtualized infrastructure. The implemented components will expose Infrastructure Services used by provisioning tools and adaptation managers, which are responsible for adaptable management of Runtime Infrastructure running over LVC.

3.3 Cloud Computing Integration with Lightweight Virtualization

The concept of exposition of computational infrastructure in the form of services is also used by *cloud computing* [127] providers, which use virtualization, on-demand deployment and delivery of services. Cloud platforms expose elastic environments to host specific workloads and be

expanded on demand to handle specified performance parameters[32]. In cloud computing, the infrastructure itself is programmable where each operation related to IT infrastructure lifecycle is programmable and exposed via a well-known interface defined by the cloud provider. Cloud platforms are proprietary solutions and not standardized, thus some initiatives like Cloud Resource Model API [128], Open Cloud Computing Interface (OCCI) [129], Amazon WS [131] define the common elements of a cloud implementation by specification of the relevant machines, storage volumes and networks. Operations that deal with the provisioning of VEI, supporting running application middleware and services, are exposed via API to users. These users are able to specify how virtual components are configured and interconnected. They describe how such an infrastructure can be provisioned i.e. how VM can be provisioned from an image, how a storage volume can be attached and how VM connect to a network. Unfortunately, the specifications are not relevant to other aspects like management of application middleware (PaaS) and actually require service-oriented components available through this API.

In addition, existing cloud environments and standards do not prescribe support for lightweight virtualization that integrated with VM's technology and provide flexible and scalable VEI through the multilevel virtualization technique. Besides horizontal and vertical scaling features, which support fluctuating resource demands of workloads, resources assigned to VM's can be partitioned between the hosted lightweight virtualization containers. This approach enables deployment of VEI, based on LVC over public clouds, in a more integrated fashion (Figure 17). The cloud platform for the end user does not provide functionality to allow flexible allocation of resources for already existing VEIs[33]. The reason for this is the sharing of physical infrastructure by other users who are not members of the organization. Introducing mechanisms of lightweight virtualization provides more granular resource management.

Such integration of clouds and physical infrastructure within the enterprise also allows dynamic sizing of physical resources and using compute resources from the cloud. Adaptable SOI provisioning with lightweight virtualization integrated with cloud platforms provides fully automated provisioning of compute resources and adaptive management of such Runtime Infrastructure through *adaptation loops* with logic defined through the ACS model. Applying service-oriented framework for integration of cloud platforms and internal IT infrastructure is

[32] According to National Instituite of Standards and Technology (http://www.nist.gov/index.html). four cloud deployment models can be defined [128]. Public cloud services are characterized as being available to clients from a third party service provider via the Internet. A private cloud offers many of the benefits of a public cloud-computing environment, such as being elastic and service-based without the restrictions of network bandwidth, security exposures and legal requirements that using public cloud services might entail. A hybrid cloud is a combination of a public and private cloud that interoperates. A community cloud is controlled and used by a group of organizations that have shared interests, such as specific security requirements or a common mission.

[33] Amazon EC2 for example does not provide services for dynamic compute resource allocation for VM instances instead auto-scaling services scale provisioned VEI with number of VM instances.

also very valuable when considering use cases such as *Enterprise-to-Cloud*[34] or *Enterprise-to-Cloud-to-Enterprise*[35]. Implementation of such deployments using the SOI paradigm provides a framework for provisioning arbitrary *application service models*[36], *universal access*[37] and *scalable services*[38].

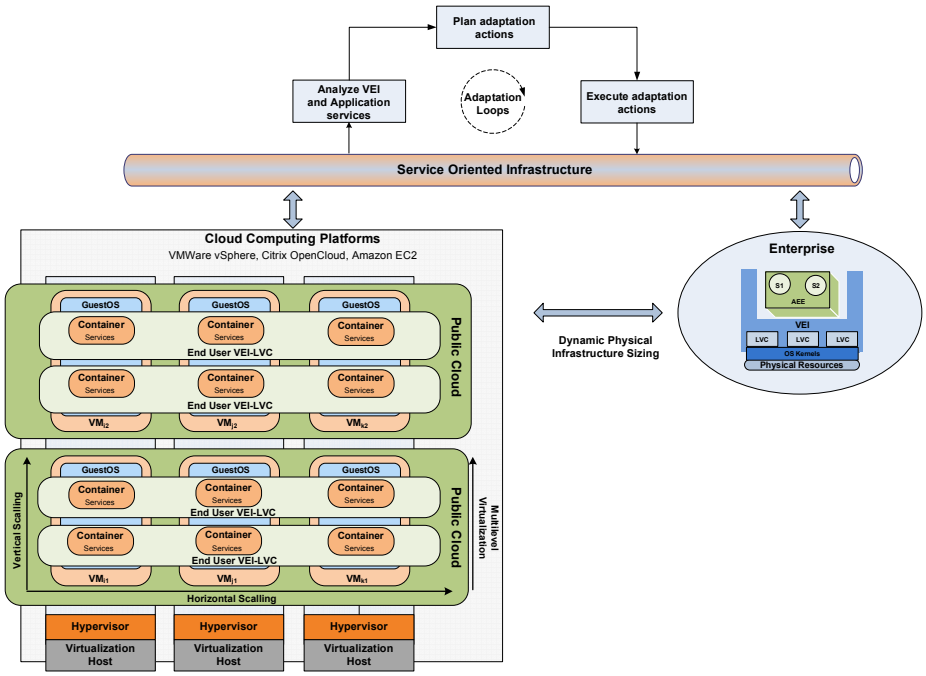


Figure 17. Adaptable Service Oriented Cloud Computing Infrastructure with Lightweight Virtualization Containers.

3.4 Process Model of Adaptable SOI Provisioning with Lightweight Virtualization

SOI provides shared distributed computing infrastructure running over virtualized physical resources. Due to the shared nature of these resources and the dynamic nature of application services, the resource demands satisfying a defined SLA are hard to predict. Provisioned application services are multilayered, spreading many LVC instances and being elements of a particular VEI domain, and involve many infrastructure elements, thus they have complex relationships with each other. The key challenge is identification of managed resources within SOI, including provisioned elements' relationship like VEI, middleware and application services. The proposed model of the provisioning process must support LVC installation and

[34] Using cloud services for internal processes of the enterprise like using cloud-storage for databases, backup operations or provisioning additional VM's to handle peak loads.

[35] Two enterprises use the same cloud where interoperating applications from these enterprises share compute resources, in B2B scenarios like supply chain for example.

[36] Different persistence models based on NoSQL databases or multi-tenancy.

[37] Ubiquitous services accessed from any device or consumer services.

[38] Elastic infrastructures enable on-demand provisioning of extra capacity or reduce resources used.

configuration and provide a management model for virtualizing resources on distributed physical servers, enabling flexible allocation of compute resources according to specified SLAs. The process model structures the required steps to provision VEI consisting of LVC instances hosting middleware with application components (Figure 18). It also facilitates SOI Infrastructure Services orchestration and assumes generic architecture of the IMF to support provisioning of the arbitrary elements of the OSL.

- ***Step 1*** - *Self-Managing Requirements Specification*: Specification of SLA documents, which contains QoS metrics for provisioned elements of the OSL running over Lightweight Virtualization Containers.
- ***Step 2*** - *Providing Manageability Endpoints*: Each provisioned element of the OSL can become a Managed Resource which must be exposed by components, enabling management and monitoring services to realize the adaptation process.
- ***Step 3*** - *Adaptation Policies Definition*: Once computational infrastructure is provisioned, continuous management to fulfill the SLA is a must. Each adaptation loop is driven by some control algorithms which is expressed with policies.
- ***Step 4*** - *Provisioning Procedure Initiation*: Before provisioning procedure activation, a system administrator specifies initial runtime parameters required to start particular provisioning procedure.
- ***Step 5*** - *Provisioning Procedure Activation*: A provisioning procedure is being activated and performs activities for LVC installation, with the required configuration of application resources and components deployment, while taking into account the initial tuning phase of Application Execution Environment. The final activity is to activate the adaptation loop.

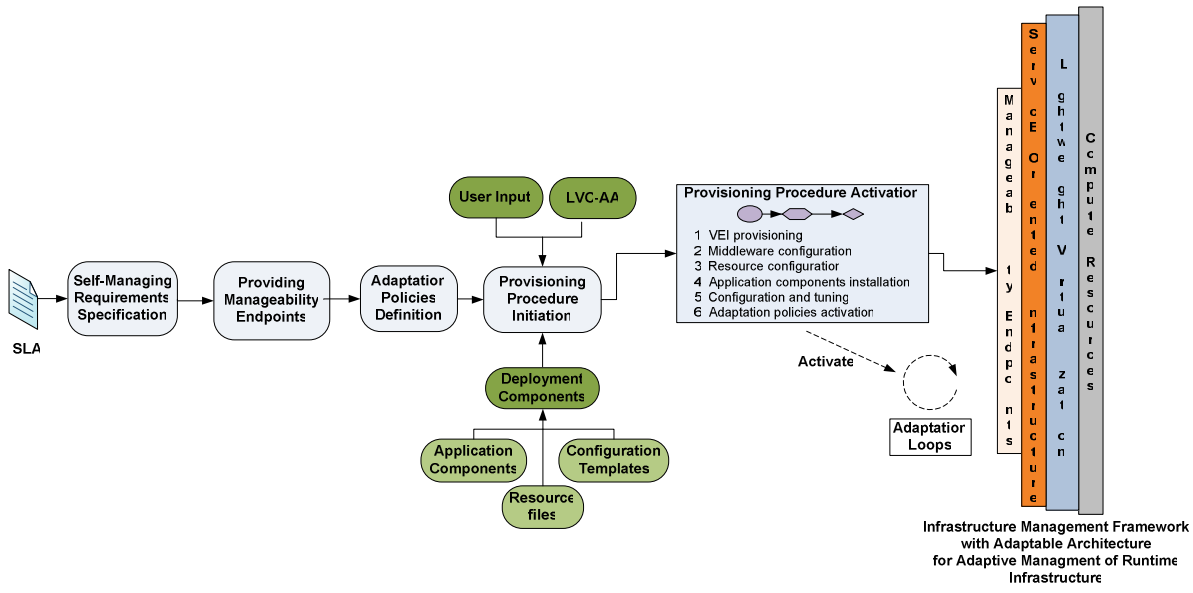


Figure 18. Model of adaptable SOI lightweight virtualization provisioning process.

3.4.1 Self-Managing Requirements Specification

These requirements are associated with SLA documents that deal with performance, reliability, security or other quality aspects, also defined as self-* properties, as discussed in [47]. This step must clearly specify and model these aspects with a self-managing requirement specification. The method these requirements are modeled on put a lot of effort in current research activities being done. As an example, goal-oriented modeling [126] can be utilized, which is a technique for engineering adaptation requirements. Simple metrics can include the maximum average response time that the application should offer. This, of course, requires sufficient compute resources which are specified in the following steps of the process and the number of these resources is supervised by the adaptation process once Runtime Infrastructure is provisioned.

3.4.2 Providing Manageability Endpoints

Management operations on the underlying infrastructure's resources require manageability endpoint components (provisioning, monitoring and management). In the context of SOI, these are gateways for interacting with Infrastructure Services, encapsulating a particular managed resource. Appropriate endpoints are identified as a result of the self-managing requirements specification. They can be designed as simple objects or JMX[39] components exposed with WSDM[40], SOAP[41] or REST communication protocols. The management components provided can use compositional adaptation and are deployed within the IMF, which is implemented using reflective middleware techniques promoting adaptable architectures.

3.4.3 Adaptation Policies Definition

It is a crucial element of the process to deliver a self-managing system, in which an adaptation solution is expressed through policies, processes and anchor objects (*Manageability Endpoints*) that define control loops used by the self-managing process of the particular Runtime Infrastructure elements of SOI. The policies include strategies for scheduling and parameters' control affecting availability of physical resources used by the provisioned LVC instances. Multi-layered applications are spread over distributed LVC instances and depended on various Runtime Infrastructure elements, creating a hierarchical structure. Such applications for instance depend on other application services, databases or Enterprise Information Systems (EIS). Adaptive management of such applications requires controlling of particular SOI elements through continuous exchange of monitoring metrics (performance, availability) managed by the control loop. The policies provided might use a

[39] *Java Management Extensions* - http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement/

[40] *Web Services Distributed Management* - http://www.oasis-open.org/committees/wsdm/

[41] *Simple Object Access Protocol* – http://www.w3.org/TR/soap

performance model of the provisioned system (defined through queue network techniques for example) and explain the current performance of a particular service, but also predict its future performance if compute resources are assigned to a hosting LVC instance.

3.4.4 Provisioning Procedures Initiation

Workflows defining provisioning procedures must specify some input data like port numbers, server names and application components installed in provisioned VEI based on specified LVC-AA configuration templates. Inserting input variables gives a system administrator fine-grained control over SOI provisioning operations. For instance, defining a port number variable in the configuration template would standardize a port configuration across all the servers in a deployment. Setting the port number in a plan at runtime could customize the port number configuration for an individual VEI element. In the case of SOI provisioning with Solaris lightweight virtualization, an effective solution will be using ZFS cloning and a snapshot mechanism, which would allow obtaining Solaris LVC-AA from a repository and attaching it at the destination global physical server[42]. These configuration templates are assigned computational resources (CPU, RAM) through DRP (including CPU scheduling policies), resource control limits and a specified network configuration. Such parameters must be specified in this step.

3.4.5 Provisioning Procedure Activation

Division of physical resources between VEI domains is associated with providing computational resources and Runtime Infrastructure that will satisfy the requirement of deployed components during the adaptable SOI provisioning process, aiming to increase operational efficiency and optimization of workloads running over virtualized resources. Therefore, delivered requirements specify a pool of physical resources (physical hardware) which constitute the basis for determining the number of LVC instances within provisioned VEI and policies defining the optimal resource allocation. When virtualized infrastructure, middleware and applications spread across many servers, the provisioning process becomes more complex and challenging. It must orchestrate the bare-metal hardware working under the control of a particular virtualization platform and middleware with applications running. Each of these infrastructure elements is exposed as software components implementing Infrastructure Service, which is instructed during provisioning of the LVC to assign computation, network and storage resources, activate and reconfigure middleware and install applications. It is also important that during the provisioning of individual elements of infrastructure, the management environment must also be deployed; in fact, these are software components that expose SOI virtualized and physical services. The above actions make the application available to end users using the already provisioned VEI and required middleware. They can be generally split into the following deployment categories:

[42] For Solaris 10 lightweight virtualization particular physical location means the *global* zone.

- *Middleware configuration*: After provisioning configuration templates that contain specific middleware there must be a requirement to create a new instance of server or cluster, create or use a database instance and create tables used by the application.
- *Resource configuration*: The application depends on many services that enable acquiring data from databases or legacy systems. Access to these sources is possible via application server resources configured separately for each system. Such construction provides transaction support and better utilization.[43]
- *Application installation*: Bundles containing application services code must be installed into the application server middleware. Very often, the activity takes advantage of built-in deployment services in middleware using PaaS services.
- *Configuration and tuning for a specific environment*: This activity customizes the configuration of each infrastructure component to suit a specific environment, for instance when development needs more verbosity in auditing files and less security checking. In some cases, tuning can be performed automatically; for instance, Glassfish application server provides *Performance Advisor* [26]. Well-known database middleware like DB2[44], Oracle[45], MySQL[46] database provide tools used by administrators to tune particular database instances through setting configuration parameters related to performance. Such tools exposed as an infrastructure service are executed automatically during the provisioning process.
- *Adaptation policies activation*: In the final stage, there is an activated adaptation process that includes actions such as modification of operational parameters, physical re-allocation of resources and update/replacement of application components. It defines adaptive resource provisioning by adding resources when necessary to maintain the specified QoS metrics and removing resources when possible without violating the SLA. Such adaptation processes are implemented with adaptation loops expressed with policies.

This procedure is an execution plan that specifies which actions should be performed on physical servers when provisioning an LVC instance based on specified information. The defined procedure uses elements provided in the previous steps and assumes the existence of SOI Infrastructure Services (provided by manageability endpoints components), which would enable invocation of each defined action. Each provisioning

[43] Connection pooling and security services.

[44] DB2 Service Request Manager provides db2tuning.sh script.

[45] Oracle Enterprise Manager contains tunning pack.

[46] MySQL Advisors: administration, schema, memory, performance and custom advisors.

procedure can invoke other sub-procedures, for instance provisioning a clustered application server domain and database server can be separate procedures with extra steps that deploy application components.

3.4.6 Summary

The provisioning process of SOI with lightweight virtualization discussed can be realized by introducing an architectural framework equipped with best practices like design patterns and methodologies (IT process frameworks). Applied together with flexible software techniques for construction of the SOI platform would deliver the right technological product that aligns business goals and objectives. Provisioning procedures define actions to be performed on specific physical resources that are virtualized and exposed via corresponding SOI services, which are orchestrated for particular activities triggered by a specific action on a target host. A workflow task can define an entire provisioning process, affecting multiple servers or a single step in a larger re-configuration process. It must be emphasized that policies are more effective in terms of adaptive management of already-provisioned SOI infrastructure to manage SLA contracts than they are for the provisioning of such infrastructure. It is due the fact that provisioning is a process which orchestrates service (similarly to Business Process Management - BPM) processes and can instantiate a policy for activities related to self-adaption of already-provisioned infrastructure elements. Moreover, a policy, as a result action, can invoke some provisioning procedure, thus such transient dependencies involve separate services for policies and provisioning process management.

3.5 Summary

In this chapter, elements of the adaptable SOI provisioning process were elaborated with an emphasis on aspects related to management of particular elements of the OSL. The adaptable SOI provisioning process assumes a/the number of Infrastructure Services that support the adaptation process. Elements of the Runtime Infrastructure running over VEI provisioned with Lightweight Virtualization Containers are instrumented with software components, which are Manageability Endpoints exposing Infrastructure Services for provisioning, monitoring management. These components provide the facilities required to filter and analyze available data about the elements in the OSL layer. The proposed structure of Infrastructure Services consists of HaaS, IaaS and PaaS that automate the provisioning and management of computational infrastructure resources running over Lightweight Virtualization Containers. The model of adaptable SOI provisioning process described is general in nature and can be exploited in arbitrary use cases related to the provisioning of SOI computing infrastructure with lightweight virtualization required by application services. The allocation of computational resources is based on adaptation decision algorithms for control loop construction defined with a particular policy language. This poses many challenges related to requirements to be fulfilled by the platform implemented and internal construction, which influence architecture and implementation. Provided functionality must automate provisioning and management activities across computing infrastructure, including virtualized resources, operating systems, application

and database servers, storage and networks through user-defined workflows that implement “best-practices.” These are elaborated in the next chapter.

4 Requirements for Adaptable SOI Lightweight Virtualization Provisioning Platform

This chapter elaborates the concept of the system for the adaptable SOI provisioning that realizes the process described in the previous chapter. Functional and non-functional requirements that must be taken into consideration to create such a system are introduced. Generic architecture with appropriate modules that provide all required elements and services for SOI provisioning and adaptive management is also presented.

In this chapter the author presents the requirements for the *Adaptable SOI Lightweight Virtualization Provisioning Platform* (A-SOI-LV-PP), which enables realization of the already-defined provisioning model. In order to define the requirements for the system, there is a need to consider the specific elements of the SOI provisioned over the LVC. The A-SOI-LV-PP must be designed with ease of administration tasks and operate in a heterogeneous environment (supporting various implementations of LVC) comprised of physical servers that are further virtualized - all of which must be provisioned and managed to satisfy defined SLA contracts. To support the various elements of the adaptable SOI provisioning process, modules should be distinguished within the A-SOI-LV-PP that will provide certain services, thus making it easier to identify requirements independently of each module.

4.1 Overview of Adaptable SOI Lightweight Virtualization Provisioning Platform

Adaptable provisioning of SOI with lightweight virtualization requires Infrastructure Services that provide managing and monitoring functionality for the Virtualized Execution Infrastructure used by the AEE and application services (Figure 19) which are SOI Managed Elements. VEI instances span a group of prepared servers, called server pools, managed with the concept of service with the use of appropriate provisioning software tools. They host provisioned operating environments (OS Kernel) supporting lightweight virtualization and advanced resource management techniques.

The A-SOI-LV-PP must provide an abstraction layer for the computing infrastructure provisioning, allowing managing resource allocation and configuration to automate scaling by adding or removing physical resources, detecting current configuration of network topology and current resource assignment (list of available CPU's, memory, number of NIC's) which can be partitioned between LVC instances. Many other complex infrastructure related tasks, like discovery of physical servers, OS Kernel and LVC instances, must be supported without manual intervention by a system administrator. Tenancy support requires each tenant provisioned with a dedicated VEI instance for application deployment and a provided complement LVC configuration template equipped with all PaaS elements.

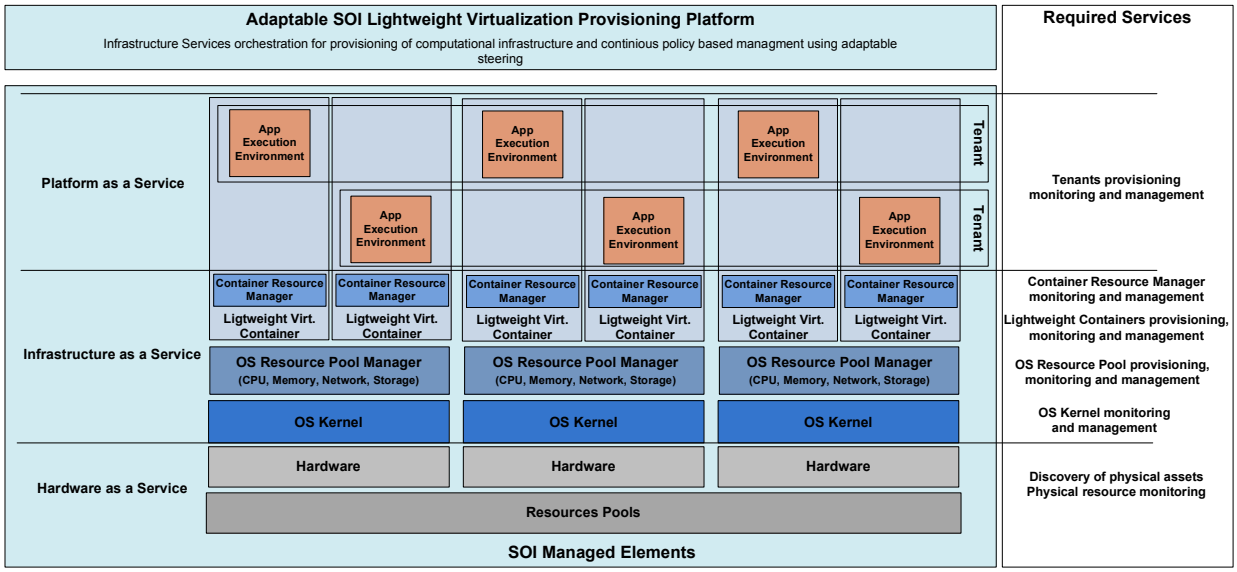


Figure 19. Required services for provisioning of Runtime Infrastructure.

Infrastructure Services exposed by software provisioning and management agents installed within Lightweight Virtualization Containers and OS Kernel instances manage these containers. The agents monitor resource usage and influence parameters that specify limits on virtualized compute resource consumption controlled by the OS Resource Pool Manager (OS-RPM) and Lightweight Virtualization Container Resource Manager (LVC-RM)[47] by the hosted middleware as specified by the control loops, ensuring specified QoS. Separate elements of Runtime Infrastructure running over a lightweight virtualization platform must automatically reconfigure itself to meet specific SLA contracts, thus reducing administrative overhead, and provide adaptation to handle the dynamics in the infrastructure. They must provide centralized management of the IT infrastructure exposed with services with controlling functionality of LVC instances provisioning over physical nodes.

[47] In the remaining part of the thesis, the terms *Container Resource Manager* and *Lightweight Virtualization Container Resource Manager* may be used interchangeably.

Ensuring realization of the adaptable SOI lightweight virtualization provisioning process also requires flexible architecture and incorporation of some concepts from the Autonomic Computing System architectures [41]:

- *Self-Defining*: A system must be aware of the domain and elements to be managed and must know its surrounding environment accordance to components, status and relations with other systems. This capability is sometimes called *self-organization,* since the managing functions of the system are automatically assigned to manage resources during bootstrapping. To accomplish this, the system must provide *adaptable* software agents that are governed by embodied policies.
- *Self-Configuring and Re-Configuring* (also named *auto-configuration*): The diversity of managed elements requires not only support for proper versioning and initial parameter configuration selection, but also the composition of suitable management entities. Support for dynamic *auto-configuration* is a necessary condition for easy deployment of the system.
- *Self-Optimizing*: The managed elements must be constantly monitored and tuned to achieve predetermined system goals, which often can be very complex and even conflicted. This can be achieved by introducing advanced techniques from *adaptive control* theory that uses monitored metrics and, based on a control algorithm or system model, takes appropriate actions.
- *Self-Healing*: Events occurring related to system errors or improper states must be automatically recognized and diagnosed. In complex computing environments, finding a process of the roots' cause is very challenging and might involve correlation of information from many managed elements. Initially this can follow rules defined by system administrators, but, as more intelligence is embedded, it must learn and generate rules to be stored in some knowledge repository.
- *Self-Protecting*: Each attack on a system must be detected and identified and then appropriate actions must be performed to remove the security threats and ensure the whole system integrity. The specially designed intrusion detections systems alert system administrators to possible malicious attacks by hackers. The algorithms implemented would detect suspicious code, which upon the analysis process will be reported and "cured".
- *Context-Aware*: The system must be equipped with functionality that enables it to find and generate rules for how to best interact with other systems. Very often, the best response time depends on algorithm used, specially implemented and designed for specific context, and must be automatically activated when switching to such a context. The system also must be able to provide its description and available resources to interested parts that can also be automatically discovered.
- *Open*: The system must be *interoperable,* which means that heterogeneous environments must be supported and open standards implemented. Moreover, it

should be flexible enough to support new software technologies and application components to be introduced and implemented in the future.

- *Anticipatory*: The system, during work, will collect information about events which have occurred related to performance, errors or frauds and store them in a persistent repository for auditing purposes. This information is interpreted in a way that allows understanding correlations and proposing optimized solutions for those events. The system will anticipate workloads and support delivery of ready to use decisions that are performed automatically.

The A-SOI-LV-PP abstracts hardware and software across a pool of physical servers with many virtualized OS instances through incorporation of lightweight virtualization technologies. It provides transparent horizontal and vertical scaling, unlimited AEE's and pooled compute resources exposed with services supporting adaptable SOI provisioning process.

4.2 Modules of Adaptable SOI Lightweight Virtualization Provisioning Platform

The essence of the A-SOI-LV-PP lies in the architecture, which is supposed to answer to constantly changing business requirements and ensure flexibility and agility which are crucial for adoption of new technologies. Working on construction of the A-SOI-LV-PP conceptual model must result in the architecture of a system consisting of major modules presented in Figure 20, which is also coherent with the SOI Reference Model[48].

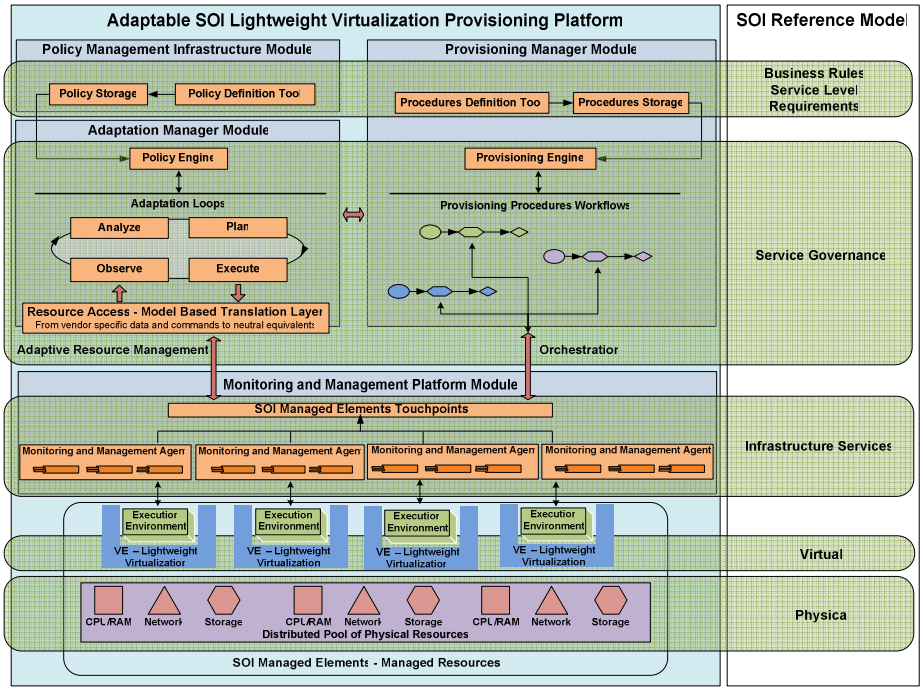


Figure 20. Software modules of the Adaptable SOI Lightweight Virtualization Provisioning Platform.

[48] Elaborated in Section 2.1

Particular modules are integrated by the common model of understanding of infrastructure components that describes the relation between VEI and AEE, supporting the realization of the process of adaptable SOI provisioning with LVC (Table 3).

The *Monitoring and Management Platform* (MMP) module exposes a unified interface (*touchpoints*), which are Manageability Endpoints, to manage or provision SOI Managed Elements. These Manageability Endpoints provide Virtualized or Physical Services, exposing the elements of the Operational Systems Layer, constituting physical, virtualized and Application Execution Environment resources, which also make their observation and control possible. The platform can be modular in the sense that monitoring and management agents are equipped with software components delivering sensors and effectors access interface. Sensors enable access to the state of a SOI Managed Element while effectors provide functionality to change the state of the element.

The *Provisioning Manager* (PM) module provides a *Procedure Definition Tool* for the definition of *provisioning procedures* that orchestrate Infrastructure Services to create Runtime Infrastructures with allocated compute resources as specified by a system administrator. The Provisioning Engine manages many instances of procedures concurrently. After provisioning, the infrastructure must be constantly monitored to check for SLA contract fulfillment; the *Adaptation Manager* supervises this process.

The *Adaptation Manager* (AM) is a module responsible for controlling the SOI Managed Elements. It also senses degrading performance and determines actions to be taken. The typical architecture of adaptation loops is based on the ACS and consisting of monitor, analyze, plan, and execute phases, which requires comprehensive support in IT infrastructure, tools and management systems. The self-adapting process is driven by adaptation loops, which are defined on the basis of a specific adaptation policy. The module is equipped with a Translation Layer internal service, representing uniform abstraction formalism, providing access for SOI monitoring and controlling functionality. This layer translates vendor-specific data and commands into their neutral equivalents and exposes SOI Managed Elements.

The *Policy Management Infrastructure* (PMI) module provides tools for adaptation policy definition that administrators can deploy inside the Adaptation Manager. A particular policy defines adaptation loop activities activated by the provisioning procedure and uses a particular control algorithm suitable for a particular self-adapting process. Each policy is stored within storage to be reused by provisioning procedures activating a specific adaptation loop.

Detailed architecture of the A-SOI-LV-PP depends on initially defined requirements, which must be fulfilled by the platform. These requirements are divided into two categories - functional and non-functional. Functional requirements normally deal with the functions that the software is supposed to offer. Non-functional requirements in Software Engineering present a systematic and pragmatic approach to `building quality into' software systems. Systems must exhibit software quality attributes, such as accuracy, performance, security and modifiability [140]. These requirements can be considered separately in context of each module of the A-SOI-LV-PP.

Table 3. Role of the A-SOI-LV-PP modules in the adaptable SOI provisioning process.

	MMP	AM	PM	PMI
Step 1				Enables handling of SLA documents encapsulated within adaptation policy.
Step 2	Exposes SOI Managed Elements with Manageability Endpoint components.			
Step 3				Provides a framework for process definition of adaptation loops with particular policy language.
Step 4			Handles initiation of the particular provisioning procedure allowing specification of all required input.	
Step 5	Infrastructure Services are used by the PM during provisioning procedure invocation and by the AM during adaptation loops processing.	Management of the adaptation loops processing.	Provides a graphical tool for definition of provisioning procedures. Coordinates activities within the provisioning procedure.	Used by the PM when activating a particular adaptation process to obtain a specified policy.

4.3 Monitoring and Management Platform

Monitoring and Management Platform provides functionality for acquiring performance metrics and configuration data from each SOI Managed Element and solves the challenge of heterogeneity of native resource control interfaces offered by contemporary computer systems and creates the need for a uniform open interface. It also distributes and responds to signals

from Adaptation and Provisioning managers through exposed services used for common management operations like provisioning and policy-based management of particular SOI elements like LVC instances of VEI (IaaS) and application middleware (PaaS).

Accurate information must be provided about performance metrics related to the computational resources utilization used by LVC. These metrics must be available for system administrators through an integrated console and presented in a uniform way. Measuring and tracking the consumption of virtual resources enables planning when to reserve resources for existing LVC or new instances must be provisioned. Many middleware products are equipped with a built-in management interface with sensors and effectors, thus should be easily integrated with the MMP. The interface allows obtaining metrics about not only itself, but also deployed applications. Such processes must be automated as much as possible and can be realized by introducing instrumentation tools, which would attach the middleware to the MPP.

When defining detailed requirements for the MMP, some already existing architectures specified for the purpose of Grid infrastructures can be considered. Such an architecture, for instance, is a Grid Monitoring Architecture (GMA) [21] announced by the GGF[49] Performance Working Group. According to GMA, a system that collects and distributes monitoring metrics must provide internal services that satisfy following requirements:

- *Low latency*: When acquiring monitoring data about particular SOI Managed Elements, the metrics become stale quickly, creating a need for efficient read access and "fast" transmission from the Managed Resource to the sensor component. In some cases, for instance "trend analysis" or accounting, such data with well-defined timestamps must be archived in persistent storage.
- *High data rate*: In the case of fluctuating workloads and constantly occurring noises, performance metrics are updated more frequently and generated at higher rates. The processing of monitoring data must be implemented to handle such operating conditions.
- *Minimal measurement overhead:* The previous requirement means that measuring can take place very often and must be implemented in a way which is not intrusive.
- *Scalability:* SOI is a very complex system with many managed computational resources, middleware and application services elements, so a platform that is responsible for monitoring and management must provide scalable services for measurement, transmission of information, provisioning and affecting internal configuration. This can be achieved by introducing a hierarchical structure of SOI Managed Elements.

[49] The Global Grid Forum (GGF) is a group formed by individuals from within the community of researchers and practitioners engaged in research, development, deployment, and support activities related to high-capability distributed software systems, or "grids."

Following, crucial functional and non-functional requirements that ensure usability and promote adaptability of the A-SOI-LV-PP are defined. Some of these requirements are specific to Solaris OS platform with lightweight virtualization technology.

4.3.1 Functional Requirements

Support for Management and Monitoring of Physical Resources - Hardware as a Service (MMP-FR1)

HaaS must expose an interface for gathering information related to the virtualized physical resources infrastructure layer. Sample metrics illustrate data about available computational resources like the number and type[50] of physical CPUs, amount of memory, hard disks and network interface cards. These physical resources are used by OS Kernel instances[51] that further virtualize resources between LVC (Solaris or OpenVZ containers).

Support for Management and Monitoring of Solaris Lightweight Virtualization Containers - Infrastructure as a Service (MMP-FR2)

IaaS exposes an interface for LVC provisioning and re-configuration activities like network configuration and increasing or decreasing compute resources. It sprawls across many physical servers containing Solaris operating systems[52], which are virtualized with LVC. Implemented SOI virtualized services must support creation, deletion, and modification of containers across multiple physical nodes. Adaptive management requires constant feedback so information about running processes (workloads) running within a particular container and project instance must be provided. Exposed functionality calculates resource usage be specific workloads, which represent application middleware and are running within the given LVC instance. This information is monitoring metrics specifying compute resources contention problems and enhancing resource usage accounting, useful for capacity planning. Examples of metrics collected through this interface include I/O utilization such as disk, NIC, memory and CPU portions used by LVC instances.

Extended Accounting facilities provide resource usage history; however, the drawback of the current implementation is the lack of information about resource usage at specified intervals, instead of just providing summarized information for the time equal to the specified interval period, from which accounting information is obtained. This can be solved by introducing persistence services and enabling calculation of long-term resource usage trends helping to understand workload characteristics and tune Runtime Infrastructure performance. Such a design enables it to analyze resource usage in detail according to the exact period and provides

[50] Beside vendor name, flags identify CPU's capabilities, other flags specify if PVM (para-virtualized) or HVM (hardware assisted) guests are supported.

[51] Solaris global zone for managing Solaris containers or Linux instance with OpenVZ containers.

[52] Such Solaris OS instance can be running within VM or hardware domains (LDOM's).

abilities to anticipate required compute resource demand assigned to LVC instances used by particular application services.

System administrators must be also be equipped with services allowing the setting alarms of varying severity, for every managed LVC and notifying administrators that particular thresholds have exceeded defined limits. This is the case for resource controls observations like the number of threads in particular container's task or used CPU time. Provisioning of Solaris Containers with LVC-AA also requires exposition of Infrastructure Services for ZFS management. A particular instance on LVC on a target host (global zone) must be attached to ZFS pools and configured with appropriate attributes like networking configuration and resource controls settings.

Support for Application Execution Environment Management and Monitoring - Platform as a Service (MMP-FR3)

Supported middleware platforms can be application servers, database middleware and others. When applying the S3 model, many other assets must be considered like composite applications, business processes that are deployed on top of the Operation Systems Layer. In such a context, management of services includes service providers that expose application services (functions) to be consumed and service consumers, which invoke these functions. Such application services might be implemented with a variety of technologies like COBOL, CORBA, J2EE, Microsoft .NET or even pure C/C++. These elements must gather information about the application layer through the middleware monitoring and management interface. Examples of metrics collected through this interface include throughput and average time of database and web transactions and the number of users. Also many resources used by application services are managed by the supporting middleware like J2EE application servers (JBoss, Tomcat, and Glassfish), Message Oriented Middleware (JMS implementations like HornetQ, ActiveMQ) and database middleware (MySQL, PostgresSQL). Such middleware must be monitored and managed with a special emphasis on the aspects related to the health of infrastructure and problems correlation describing, for instance, the exhaustion of thread pools, database connection pools and CPU and memory usage. When managing SLA contracts for application services, the key measurement is business transactions performance, which is also absolutely crucial in analysis, troubleshooting and finding potential hotspots that cause bottlenecks. It is very significant to isolate resources that are not performing or are failing. Capabilities should be provided that allow decomposition of a single transaction (client request) into smaller units associated with specific resources and obtain a relationship between the request and implementation artifacts, such as application components with invoked operations on a database or other services.

Context Awareness (MMP-FR4)

The MMP must be equipped with internal *adaptability services* implementing context-aware features, which provide the ability to activate appropriate software components that expose SOI services according to the type of managed resources. Adaptability services detect the vendor

and version of underlying resources automatically without the need for manual reconfiguration and administration activities. The properties of the surrounding environment are automatically discovered during start-up through reflectivity techniques. The next dedicated group of software components (modules) with sensor and effector interfaces for the underlying resources management is deployed for SOI Managed Elements representation. Expected adaptability services should support any vendor of infrastructure elements within the OSL like differential hardware platforms, types and versions of operating systems, lightweight virtualization platforms and application middleware.

Self-Configuration (MMP-FR5)

SOI is a complex system with many resources that are composed of virtualized computational nodes and other dedicated nodes such as *infrastructure registry*[53], *naming service*[54], *storage and Access Nodes*[55]. The particular resources have their own life cycle, can be created, activated/de-activated or even migrated. This must be reflected in the MMP with a special emphasis on the addition of new physical nodes and the removal or failure of existing physical nodes with LVC instances. Such requirements are fulfilled by *self-organization* services of the *MMP's agents*, which can be running within the OS Kernel instance[56], installed directly on a physical node. Self-organization refers to:

- Discovery and registration of new physical nodes running OS Kernel instances,
- Discovery and registration of new created/activated elements of VEI including number of LVC instances,
- Discovery and registration of AEEs running on top of VEI.

Discovered elements must be identified and described with information stored in the *Infrastructure Registry* accessible to provisioning tools when provisioning VEI with LVC and deploying application services, which require compute resources and AEE platforms.

Dynamic Instrumentation (MMP-FR6)

Context-awareness and self-configuration requires support for dynamic instrumentation of managed resources with components exposing the provisioning interface. The MMP must be able to manage the life cycle of the modules during runtime because initially loaded components exposing Infrastructure Services can be modified. Such dynamic instrumentation of

[53] Registry service stores informations about particular SOI elements that for instance expose sensor and effector interface or any other functionality used by Adaptation Manager or Provisioning Manager.

[54] DNS or NIS services.

[55] Access node is exposed in the public network for communication purposes and which acts as a gateway between the public wide area network and the monitored resources, hidden behind the firewall and using a private addresses space.

[56] Global zone in Solaris 10 OS.

particular resources enables runtime modification of SOI Infrastructure Services like activation, de-activation or upgrading appropriate components without the restart of the MMP.

Monitoring Methods: Pulling, Tracing and Notifications (MMP-FR7)

Monitoring and management of particular resources within SOI can involve different scenarios for interactions between Adaptive and Provisioning Managers and system administrators; *these are consumers*. Such scenarios define use cases that obtain monitoring metrics in a request/response pattern, logs tracing for particular events occurred in a specific resource and subscription for notifications about occurred events (reaching some monitored thresholds, activation or de-activation of resources). The following are well-known monitoring methods to be implemented by the MMP:

- *Pulling*: Sampling method initiated on the consumer side
- *Tracing*: Involves log analysis and uses parsing techniques for obtaining information about events which have occurred
- *Pushing*: Notifications about an occurring event in a specific resource are sent periodically by a producer and obtained by the consumer.

Each method must be implemented within a monitoring agent (information producer), exposed by the sensor's interface and accessible to external consumers.

Automatic Installation and Updates (MMP-FR8)

Because SOI spreads many physical servers and contains software agents installed within VEI, the installation process might become a challenging task. Installation should be automatically performed, which can be accomplished with LVC-AA that will contain such agents, but an agent must only have "core" components installed. During start-up, the appropriate components are downloaded from the Components Repository, the initiated and activated. Updated components can be placed into the repository and the older versions of already-activated agents can be restarted with new functionality.

4.3.2 Non-Functional Requirements

Uniform and Interoperable Interfaces (MMP-NFR1)

The concept of SOI is to expose each managed resource as a service that must be well defined and independent on any access method used by client applications for acquiring monitoring metrics and performing management tasks. Such an access method is associated with protocols used like SOAP, RMI, SNMP or even REST. This can be accomplished by introducing a *connector* concept that handles the request for a specific protocol and is responsible for marshalling, redirecting to proper service exposed by sensor or effector components and un-marshalling. The connectors can be added dynamically at runtime without the need to perform any modifications of the MMP.

Portability (MMP-NFR2)

The implemented platform must be portable, thus providing support for a wide spectrum of hardware, operating systems and virtualization platforms. There Common parts of components should be available that are independent of any OS platform and these, which are assigned for specific OS and its version, that can be deployed inside MMP on demand. These specific components can be accessible via the *Components Repository* used by management agents and loaded dynamically.

4.4 Adaptation Manager

Adaptation Manager must provide services responsible for the implementation of control loop activities. Based on the ACS model, the *Analyze* and *Plan* activities are related to policy processing and require policy engines compliant with specific policies, which are activated by a system administrator. The remaining *Observe* and *Execute* modules' activities implementation rely on interfaces provided by the MMP exposing SOI Infrastructure Services for managing elements of the OSL. The AM's system activity model makes it possible to specify functional and non-functional requirements for such system implementation and is summarized as follows.

4.4.1 Functional Requirements

Support for Control Loops Definition (AM-FR1)

An internal structure should be proposed and implemented as a framework ready to be refined for the particular activity goal implemented with a specific control loop and expose functionality necessary to perform each step of such a control loop. Its representation and relevant tools for its processing should be flexible in design, allowing defining different strategies and algorithms for adaptivity process of SOI. It should be possible to change an activity goal at any time, even during runtime.

Support for Discovery and Translation of Sensor and Effector Interfaces (AM-FR2)

The sensor and effector components must be available to successfully perform control loop activities related to observing resources' state and applying adaptation actions. Particular SOI Managed Elements components exposing the functionality must be automatically discovered by the AM and available to evaluate a particular adaptation policy. Specification of particular sensors and effectors depends on the domain and are specific to a given managed resource. The transformation of resources to SOI Managed Elements should be straightforward and performed automatically according to specified requirements regarding sensors and effectors. This process must be performed at runtime if necessary.

4.4.2 Non-Functional Requirements

Support for Flexible Deployment (AM-NFR1)

There is no obligation for collocation of the Adaptation Manager and SOI Managed Elements in the same process. It means that AM could be deployed as a remote process to the MMP operating somewhere in the network. The deployment procedure should be lightweight enough and performed on demand during the system activity and its knowledge should be stored in persistent storage to be available after system restart.

Support for Arbitrary Policy Engines (AM-NFR2)

A tool for control loop definition, and the AM which manages them, must be general enough to serve the most typical adaptation strategy without any modifications, e.g. the system should be compatible with many policy engines because, in practice, it is impossible to find one, best solution well-suited to all problems. This fact requires the AM to support many policy languages (policy engine specific) and be able to incorporate others in the future through the requirement of supporting plugability of policy engines.

4.5 Policy Management Infrastructure

The Policy Management Infrastructure provides the required services for storing, deploying and evaluating policies [50,55] and must ensure consistency of defined policies for management of individual elements of SOI. Policies define operational governance control of runtime aspects of SOI, including service monitoring and management. The PMI is responsible for definition, management and distribution of policies spanning elements of the OSL, enabling administrators to manage policy for large-scale systems as IT infrastructures designed with SOI in mind. The *Policy Definition Tool* (PDT) enables system administrator *create*, *delete*, *modify* and *save* operations on policies, which are persistent in the *Policy Storage* that acts as a repository where policies are stored for any future use and are accessible for evaluation during the system runtime.

4.5.1 Functional Requirements

Support for Policy Deployment (PMI-FR1)

The Policy Definition Tool must provide functionality for on-demand deployment of policies within the instance of the particular AM to realize defined adaptation loops. This should be performed through distribution of policy and, later, activation within a particular policy engine of the AM. Manageability Endpoint components must be present in the MMP to enable realization of the strategy defined by the policy.

Support for Policy Lifecycle Management (PMI-FR2)

Each policy can be deployed into the AM and, once loaded, it can be enabled, disabled or unloaded from the policy engine. Unloaded policies can be either reloaded or deleted. These

life-cycle services should be centralized within the PDT and act as a centralized control point for managing concurrent access.

4.5.2 Non-Functional Requirements

Support for Arbitrary Policy Languages (PMI-NFR1)

The Policy Definition Tool must provide functionality for different policy representation based on the requirements of a particular adaptation loop defined according to control algorithm and using underlying Infrastructure Services.

Centralized Storage (PMI-NFR2)

The access to policies must be centralized through the *Policy Storage,* accessible by distributed elements of the platform (like Adaptation Managers for instance) requiring dynamic activation of a particular policy.

4.6 Provisioning Manager

The Provisioning Manager provides services that deal with the execution of repeatable workflows that automate the configuration and allocation of particular compute resources for provisioned LVC. It can both receive a provisioning request from a system administrator or the Adaptation Manager and translated it into an executable workflow instance. These workflows are *provisioning procedures*, which represent transitions, or steps, each associated with a particular Infrastructure Service. Strictly connected with notion of workflow is the concept of the previously discussed informational technology processes, specify services and methods, required to provision, use and operate the SOI according to process frameworks which, if required, can be defined through the PM.

4.6.1 Functional Requirements

Provisioning Procedure Design Tool (PM-FR1)

The first step is to design a workflow that specifies how particular Infrastructure Services are orchestrated, using tools like the graphical modeling tool of provisioning procedures. It should be easy enough to allow drag-and-drop of elements likes nodes, transitions and actions and specify the relation between appropriate Manageability Endpoints. Such a provisioning procedure should be saved as *archive,* passed to an execution engine.

Provisioning Procedures Execution Engine (PM-FR2)

A system administrator must be allowed to request the engine to execute a defined procedure's action. They are coordinated through the execution engine, which sequentially invokes provisioning tasks within a particular provisioning procedure and provides multiple working threads to enable simultaneous execution of provisioning procedures on multiple infrastructure elements, maintains state and logs all events.

4.6.2 Non-Functional Requirements

Easy Programming Model (PM-NFR1)

Facilitating agile methods used for the definition of provisioning procedures expressed in a more common dialect, which combine programming languages and declarative techniques[57] will be an easier approach. However, technology must allow interaction with SOI Managed Elements that are exposed with components accessible with various communication protocols.

Unified Set of Tools (PM-NFR2)

Elements of the PM module should be integrated into a coherent framework that unifies the definitions, implementation and management of provisioning procedures. This will enable it to centrally perform the interaction between system administrators and Infrastructure Services responsible for providing the Runtime Infrastructure on demand.

4.7 Summary

The presented conceptual model of the A-SOI-LV-PP promotes modular design and exposes Infrastructure Services for flexible provisioning and adaptive management, supporting the adaptable provisioning process model described in the Chapter 3. These services are discussed by the author in the form of functional and non-functional requirements in the context of each module independently. In the opinion of the author, only the most important requirements for the realization of the process to be implemented by the A-SOI-LV-PP are presented.

The Provisioning Manager supports definition of provisioning procedures and orchestrates the Infrastructure Services. The Policy Management Infrastructure provides tools for policy definition and persistency supporting operational governance of SOI. It stores policies related to the adaptive management process of already-provisioned Runtime Infrastructure elements used by application services. The Monitoring and Management Platform exposes uniform and interoperable Infrastructure Services and, internally, is responsible for many complex services dealing with context-awareness, self-configuration and instrumentation of Managed Resources. The Adaptation Manager must provide tools for control loops management, defined with control algorithms expressed with arbitrary policy languages (obtained from the PMI) that rely on specific services possessing the semantics required by the Policy Engine.

Meeting the requirements for each module of the A-SOI-LV-PP is necessary to propose an architecture and technology for its implementation, which are elaborated in the next chapter.

[57] Currently existing standards like Bussines Process Execution Language (BPEL) introduce complexity in terms of process definition and requirements. BPEL requires endpoints accessible with Web Services and detailed technical expertise is mandatory during definition phase.

5 Architecture of Adaptable SOI Lightweight Virtualization Provisioning Platform

Once the requirements for the Adaptable SOI Lightweight Virtualization Provisioning Platform have been specified, this chapter elaborates how the model of adaptable SOI provisioning can be reflected with the architecture of the platform. This chapter describes mechanisms which are useful for designing adaptable frameworks. Next, it presents the hierarchical organization of SOI Managed Elements with managing modules and layered architecture of the platform.

As elaborated in Chapter 3, the process model of the adaptable SOI provisioning with LVC was a description of the A-SOI-LV-PP, together with the initial specification, and its IT environment. The model was a drawing with sequential steps expressed in a natural language and was an initial activity to define the A-SOI-LV-PP's requirements. The requirements for the platform are described in the context of Runtime Infrastructure provisioning and adaptive management.

This Chapter describes the architecture of the Adaptable SOI Lightweight Virtualization Provisioning Platform and is a concept of environment, based on the adaptable SOI provisioning model. Description of the architecture of the system was preceded by presenting the foundation of concepts used in the design phase including a detailed explanation of software components realizing functionality in the provisioning process. These assumptions allowed the construction of the platform in accordance with the requirements specified in Chapter 4 and, as such, allow the practical verification of the thesis statement.

5.1 Adaptable Architectures Foundation

Architecture can be defined as a set of structuring principles and patterns that, when applied to a problem, provides the framework for a solution, which can then be assembled from a set of simpler subsystems or components [7]. It describes the internal structure according to a list of major elements (logical components) and the relationship between them. The elements of Adaptable SOI Lightweight Virtualization Provisioning Platform must be based on flexible architectural framework and share common essential elements that promote service

independence, modularity, availability and performance. This can be achieved through introducing:

- *Open standards and common interfaces*: Requirements for integration and interoperability call for using well-documented, industry-known standards. Such an approach would maximize integration and reuse of existing solutions within SOI from various service providers and infrastructure management frameworks.
- *Patterns*: These have many groups such as *design patterns* [141], *architectural patterns* [142] and *analysis patterns* [143]; their role is to document a solution for a recurring problem defined in a particular context. These catalogs prevent working out a solution for specific problem repeatedly and continue to be improved and discussed through discussions in open forums. Patterns also evolve as the technology and specifications change.
- *Decomposition*: A platform should be implemented with components that provide modularity and independence in represented managed resources like applications and particular VEI elements. This will help to manage complexity because components can be reassembled quickly and flexibly into required services of SOI. Such architectural decomposition influences the system availability and scalability and provides a foundation for managed resources independence.
- *Managed resource independence*: As many virtualization and middleware technologies evolve or emerge it is impossible to provide the support in exposed functionality of SOI services. In spite of this, the platform developed should allow easy extension to support new resources and be independent of any underlying resource access interface.
- *Systemic qualities focus*: Software SOI services provided must be highly scalable, provide high levels of availability and offer the security expected by system administrators. It must also allow provisioning of application services, including virtualized resources and application middleware, leveraging existing physical resources and tools.

Adaptable architecture demands the design of a system whose components are structured and implemented flexibly to let them be *re-organized*, *replaced*, *re-configured*, and *extended* as needed. Such architecture can be attained by the use of comprehensive software techniques across all system's components, conforming to industry standards and required systemic qualities, like availability, scalability and security.

5.2 Layered architecture

The proposed scheme for the adaptable provisioning of SOI has architecture with nine layers (Figure 21), which arise from the previously described generic conceptual model and assumptions about the functional and non-functional requirements. The architecture Proposed is characterized by the openness of exposed interfaces and adaptability to manage any SOI Managed Elements, like VEI based on arbitrary lightweight virtualization technologies with any

running software product. The architecture supports interoperability between many vendors and scalability of SOI platforms with many Managed Resources. Exposed interfaces are standardized by leveraging and implementing well recognized and used protocols in the IT industry. The individual layers consist of specific modules of the platform, where each layer plays a key role in the process of adaptable provisioning of SOI. It is based on the characteristics used by the layer immediately adjacent and is decomposed with software components. These components, in their implementation, may use the technique of patterns to promote managed resource independence and installed within IMF environment, which ensures scalability and availability of the A-SOI-LV-PP.

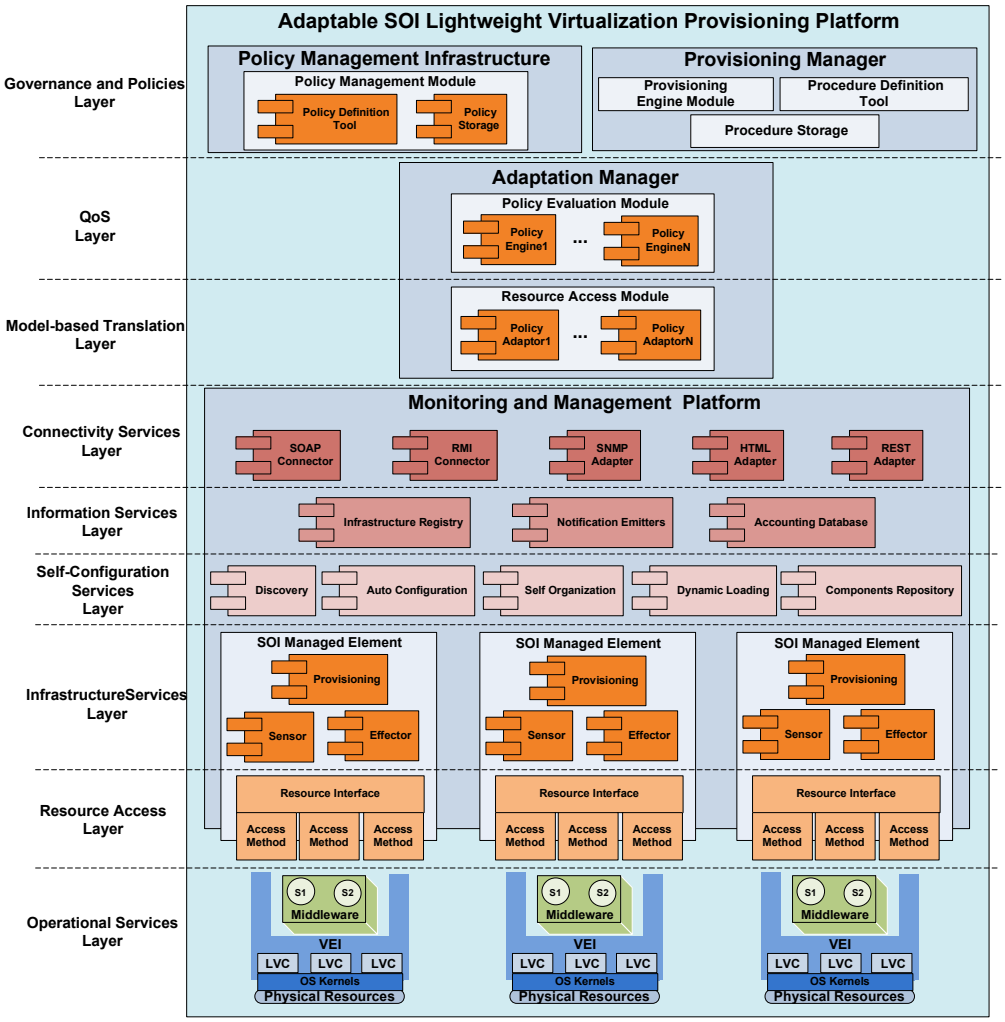


Figure 21. Layered architecture of the A-SOI-LV-PP.

5.2.1 Operational Services Layer

The lowest layer of the system is heterogeneous infrastructure resources defined within the OSL. They are supplied by the various elements used to deliver a complete runtime environment for application services that, besides physical hardware and OS Kernels, system libraries and services, includes specialized devices dedicated for the processing of business transactions like Layer-7 switches (load balancers) with integrated hardware SSL accelerators. Other examples are storage devices with equipment providing scalable access like InfiniBand

switches. Communication infrastructure is used to combine elements of physical infrastructure like router and switches based on hardware or software solutions. These elements are used to provision SOI using Lightweight Virtualization Containers based on specific hardware and software platforms, which constitute Runtime Infrastructure for application services.

5.2.2 Resource Access Layer

Many elements of the OSL expose the access interface with dedicated tools and software. Some of them are not accessed directly because of their category and their heterogeneity and, therefore, they were enclosed with an additional layer of abstraction to enable better provisioning and management. The Resource Access Layer (RAL) uses vendor-specific interfaces and different methods for communication with the infrastructure elements of the OSL. It is responsible for instrumentation of Managed Resources through encapsulation of low-level aspects allowing full efficiency and functionality of the system for SOI provisioning implemented with components (*MMP-NFR2*). The RAL, besides LVC, also enables seamless integration with any product in the VM virtualization technology, cloud platforms (OpenNebula, Nimbus, Amazon), middleware (J2EE application servers) and provisioning tools (N1SPS) ecosystems ,thus easing provisioning based on LVC-AA and their configuration using resource and service managers and other management tools.

5.2.3 Runtime Infrastructure as a Service Layer

The Runtime Infrastructure as a Service Layer (RIaaSL) exposes Infrastructure Services (HaaS, IaaS and PaaS - *MMP-FR1*, *MMP-FR2*, *MMP-FR3*) with the usage of *software components* responsible for exposition of particular vendor-specific interfaces into a common API, providing a set of interfaces used for provisioning, monitoring and management. Because all these interfaces are coherent and conform to particular resource specification, regardless of the access methods, they constitute a common and open resource interface used for constructing the SOI Managed Element. This layer provides a uniform access interface to Managed Resources of the OSL, which are instrumented and expose the sensor and effector interface, providing monitoring data to define the source of performance bottlenecks and tune the application services appropriately. Such data are also required for ensuring fault-detection functionality to determine if any element of the OSL is active. The provisioning components enable on-demand creation of VEI instances or addition of LVC based on LVC-AA. The RIaaSL provided services are orchestrated by the PM when realizing provisioning procedures and exploited by the AM when running adaptable steering services.

5.2.4 Self-Configuration Services Layer

The Self-Configuration Services Layer (SCSL) provides facilities that ensure adaptability and context-awareness of the MMP. The *Discovery* service components enable the localization of SOI Managed Elements where the range of the process may involve the entire infrastructure or a particular VEI instance. It can dynamically handle the lists of available physical servers managed by the OS Kernel, running LVC, application middleware and other devices. This self-

configuration capability implies dynamic reaction to failures and the capability of dynamic and automatic addition of SOI Managed Elements in that they are replaced. The *Discovery Responder* components, mandatory elements of each MMA, use discovery protocols based on multicast or other frameworks responding to the *Infrastructure Registry* services with the address of the MMA. *Dynamic loading* ensures automatic deployment of modules containing Manageability Endpoints, software components that expose services for provisioning, monitoring and management of SOI. Such an *auto-configuration* feature enables MMA to learn the characteristics (*MMP-FR4*) of the Runtime Infrastructure (configuration of physical hardware, version of OS Kernel and running services) and load proper modules with sensors and effectors depending on the discovered features of the environment where the agent is started (*MMP-FR6*). The process uses module descriptors dedicated for the specific hardware and OS platform and then the actual components are downloaded from the *Components Repository* (*MMP-FR8, AM-NFR1*).

5.2.5 Information Services Layer

The Information Services Layer (ISL) provide services to the storage and retrieval of data about the state of the Runtime Infrastructure (OSL) to determine the current number of VEI instances, number of LVCs in each VEI, including the physical server which is used by the particular OS Kernel, and running middleware (*MMP-FR5*). Such information is stored in the *Infrastructure Registry*, which registers for notifications from the Discovery Responder components running on particular instance of the MMA. These MMAs also provide monitoring data stored in the *Accounting Database* that aggregates information about concrete workloads and LVCs. The data allow predicting demand for resources that can be used by both the PM and AM, while deciding on the allocation of computing resources. *Notification emitters* send or redirect notifications to other interested parts like PM, AM or GUI management consoles (*MMP-FR7*). These notifications contain information about the state changes of particular SOI Managed Elements, which include increased resource usage (CPU, memory) and a lack of resources, like increased garbage collection activities within a specific JVM (Java Virtual Machine) instance. Notification services can also exploit aggregation of events from many SOI Managed Elements and correlate them to triggering actions to anticipate the condition of the Runtime Infrastructure and perform dynamic reconfiguration. This can be accomplished through CEP (Complex Event Processing) [145], which is exploited by some researchers [146].

5.2.6 Connectivity Services Layer

The Connectivity Services Layer (CSL) provides communication service components exposing common and open accessing methods of SOI Managed Elements due to its connector architecture, which allows plug-in of any type of protocol connector available, ranging from RMI, SOAP, and REST to many others (*MMP-NFR1, PM-NFR1*). It is very important, since the Adaptation and Provisioning Managers can be implemented with different technologies, hence access to the current state of the SOI Managed Elements and control functions should be available via an open interface, implemented with the use of industry standards. The SOAP connector can also act as a secure and stateful Web Service for communication with clients,

written in many popular programming languages (Perl, C/C++). Access to the SOI Managed Elements is possible through introduction of a single unified and adaptable interface, which makes strong use of reflection techniques[58]. It does not define particular management methods and attributes, but is general for the purposes of accessing any type of interface operations and attributes in existing and future elements of the OSL. Protocol adaptors can, in turn, be used as front-ends extending functionality for use with certain types of standard management applications such as adaptors for SNMP, WSDM and others. This fact makes the CSL coherent in all possible protocols and ready for future extensions.

5.2.7 Model-based Translation Layer

The Model-based Translation Layer (MbTL) is an intermediate layer, generic and not related to any particular technology or class of managed resources exposed with components and ensuring that the non-functional feature of provisioning is separated from the implementation of sensors and effectors (*AM-FR2*). It is also orthogonal to the management policy performed by the Adaptation Manager, making its integration with resources more straightforward through construction of an interface that supports interoperability with different policy engines. This layer defines the Resource Access Module (RAM) of the AM implemented with components responsible for integration with the MMP, exposing SOI Managed Elements. These components have well-defined resource representation allowing identifying resource, discovering (through the ISL) it management capabilities and providing resource sensors and effectors access[59] and are compatible with the particular Policy Engine specification. These are *Policy Adaptor* components, which provide full accesses to SOI Managed Elements managing interactions between the Policy Engine and Managed Resources.

5.2.8 Quality of Services Layer

The QoS Layer (QoSL) provides a Policy Evaluation Module (PEM) of the AM, whose key requirement is management of control loops (*AM-FR1*). There can be several reasoners installed and operating at the same time enabling flexibility through support for a wide choice of policy engines (*AM-NFR2*). The differences between reasoners are resolved by the particular *Policy Engine* components construction, which initialize a specific reasoner adaptor. Definition of policies are obtained from the PMI, next instantiated by the appropriate Policy Engine and evaluated. This activity uses the Policy Adaptor components from the RAM for sensing the Runtime Infrastructure attributes related to required QoS goals and then affect parameters, which influence particular metrics of QoS.

[58] Java Reflection API or Dynamic Skeleton Interfaces support creation of Dynamic Proxies, which encapsulates resource access with the interface defined dynamically during the runtime.

[59] These are Managability Endpoints required for the process of adaptable SOI provisioning as desribed in Step 2 – section 3.3.

5.2.9 Governance and Policies Layer

The Governance and Policies Layer (GaPL) covers all aspects of SOI provisioning and Runtime Infrastructure lifecycle (*PM-NFR2*). It provides a graphical interface for the system administrator available through GUI management consoles, allowing listing and editing system configuration, policies (*PMI-FR1*, *PMI-FR2*) and procedure definition related to SOI provisioning and management (*PM-FR1*, *PM-FR2*). The layer provides tools for initialization of the SOI provisioning process responsible for the initial allocation of resources required by middleware services used by application. This activity is an element of the specific provisioning procedure, which carries out initial verification of the availability of resources at the level required to ensure the QoS. The GaPL provides policies used by adaptation loops for managing service-level agreements, including capacity, performance and security (*PM-NFR2*). Graphical management consoles expose charts with monitoring metrics of the OSL elements and provide functionality for manual configuration of some elements.

5.3 Organization of the A-SOI-LV-PP's Modules for Runtime Infrastructure Provisioning

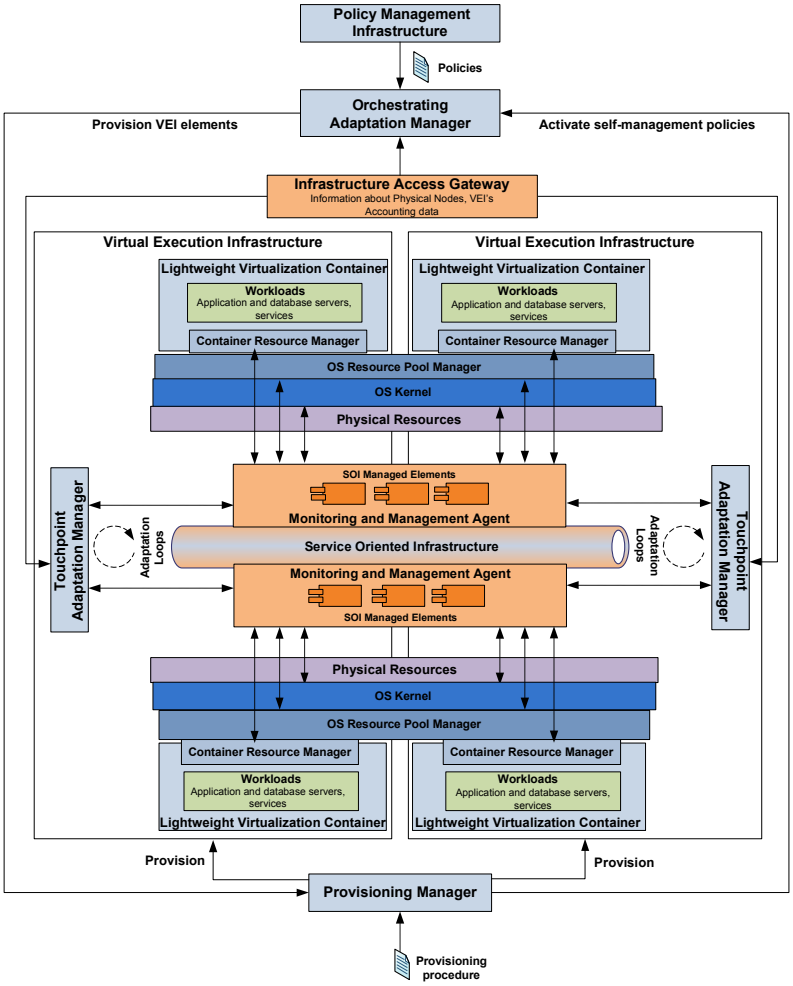


Figure 22. Structural organization of the OSL elements and theA-SOI-LV-PP's modules.

A frequently used method for construction of tools for infrastructure management in distributed environments is structural organization. In the context of SOI environments, such an approach is driven by requirements resulting from the strong distribution of SOI Managed Elements and assumes computational infrastructure partitioning between independent VEI instances provisioned on demand by the Provisioning Manager and controlled by the adaptation loops running within Adaptation Managers - Figure 22.

5.3.1 Operating System Kernel Resource Managers for Lightweight Virtualization Containers

Physical resources are virtualized and managed by the *OS Resource Pool Manager* implemented by a specific lightweight virtualization platform. The OS-RPM divides resources between LVC instances within a single instance of the OS-Kernel and is assigned to separate VEI domains. Such sharing means that a particular LVC contains workloads having diversified resource demands influencing resource control values, which define how virtualized resources are consumed by the lightweight container. The LVC Resource Manager is used by the LVC instance and, if supported by the OS Kernel, assigned computational resources can be further virtualized. This is the case for Solaris OS providing granular resource management framework with SRM and resource controls, where global zone acts as an OS-RPM for local zones in which LVC-RM's are used[60]. Deployment that is more complex, in the case of multilevel virtualization, can be based on VM's (Xen, VMware) or hardware domains (LDOM) hosting Solaris OS where in such a case there are two OS-RPM's (one that is VMM and second that is global zone) and LVC-RM used by local zones.

5.3.2 Provisioning Manager

The Provisioning Manager allows specification of compute resource configuration and provisioning procedures definition. Provisioning procedures can be defined through a workflows mechanism supported by many Business Process Management (BPM) [25] platforms, allowing orchestration of arbitrary infrastructure elements exposed through software components providing access to Infrastructure Services. Existing tools for BPM management, like JBoss jBPM[61], Intalio BPMS[62] and Oracle BPM[63], can be integrated with the MMP. They can orchestrate exposed SOI services, perform provisioning of VEI based on already-defined LVC-AA and perform other deployment tasks defined by a particular provisioning procedure. These platforms are also equipped with required tools for procedures definition with the usage

[60] In Solaris, resource partitioning can be performed beetwen projects and tasks running within LVC as described in Section 3.2.3.

[61] http://www.jboss.org/jbpm

[62] http://www.intalio.com/bpms

[63] http://www.oracle.com/us/technologies/bpm/index.html

of standardized languages like BPMN[64] or jPDL[65]; they are a well-integrated framework for realization of the Provisioning Manager.

5.3.3 Monitoring and Management Agents

The development process of provisioning and management of SOI involved in proper operation of adaptation and provisioning managers requires the Monitoring and Management Agent (MMA) running on each physical server. Each MMA is equipped with components, which enable dynamic partitioning of the physical server through provisioning of LVCs and monitoring their resource usage and changing resource controls attributes. Besides managing the virtualized environment, it is also able to monitor particular workloads and influence the internal configuration of application middleware like managing thread or database pool size, configuring required application resources. These are managed resources represented as SOI Managed Elements used by the PM, TAM and OEM realizing provisioning and resource management policies ensuring specified QoS goals for particular VEI domain.

An architecture overview of the MMA is presented in Figure 23, which is coherent with layered architecture already presented. Three groups of components are identified: *Infrastructure Services Components*, *Core Services Components* and *Connectivity Services Components*. The first group has been already described and is responsible for translation of particular-vendor specific interfaces into a common API used during provisioning. The group of Core Services components is responsible for auto-configuration, dynamic loading and discovery services. Native connector components enable communication between components exposing Infrastructure Services and other modules of the A-SOI-LV-PP.

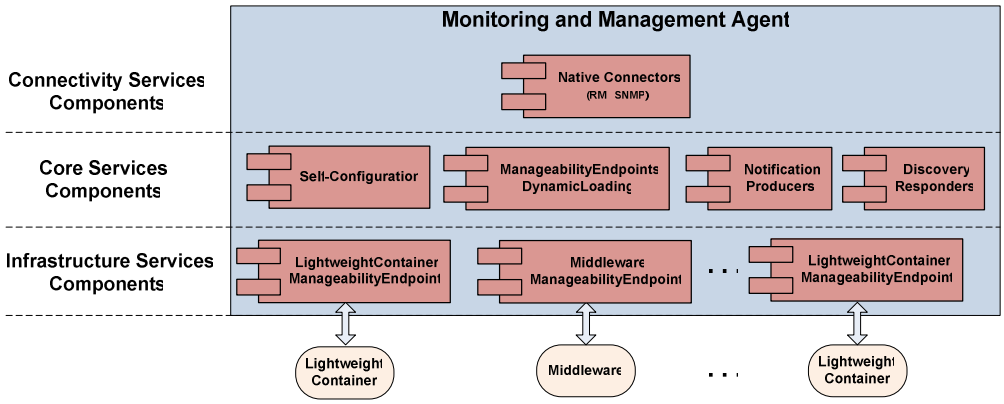


Figure 23. Component diagram of the Monitoring and Management Agent.

The types of loaded modules depend on the discovered features of the environment, where the agent is started and where the list of features can be passed as an argument to the MMA being started to manage a particular LVC instance. Such a method of agent start-up separates the phase of obtaining information about the environment and monitoring requirements from the

[64] http://www.bpmn.org

[65] http://docs.jboss.org/jbpm/v3/userguide/jpdl.html

phase of loading modules. This separation can be implemented using a shell script, which gathers information and passes it as flags of registry bits to the agent process. The only condition that should be met involves the uniformity of semantics of the flags set by the shell script and the ones read by the agent.

5.3.4 Infrastructure Access Gateway

The Infrastructure Services of the A-SOI-LV-PP are registered within an *Infrastructure Access Gateway* (IAG) running over the Access Node. This organization of SOI physical and Infrastructure Services allows avoiding problems related to complexity and scalability because the IAG stores all valuable information about the current state of particular SOI Managed Elements and can be used by other client application as an access point for the management of distributed instances of OS Kernels with provisioned Lightweight Virtualization Containers.

When the MMA discovers that it is installed on the Access Node, it automatically loads the IAG services implemented by the appropriate components (Figure 24). The Infrastructure Registry module is responsible for organizing MMA into a federation using discovery services and acts as a mediator between remote clients and SOI Managed Elements. Management of such a federation enables aggregation of useful information like monitoring data, retained and managed by the Accounting Database components, or managing events occurring in the particular SOI Managed Elements of the A-SOI-LV-PP. The IAG exposes the management functionality of the underlying agents proxying invocations to these agents in the form of Web Services based on SOAP or REST or other protocols like HTML supported by dedicated connector components.

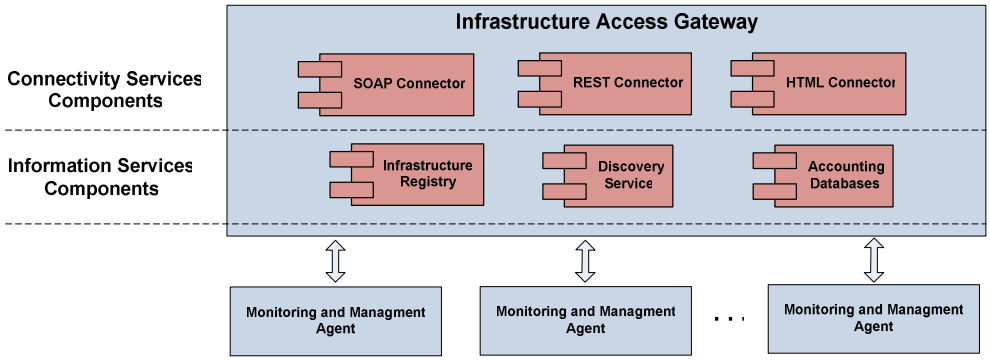


Figure 24. Component diagram of the Infrastructure Access Gateway.

5.3.5 Adaptation Managers

Adaptable provisioning of SOI allows a flexible modification of any VEI instance, depending on the dynamically changing requirements and the availability and level of virtualized resources. The proposed structure assumes that there can be more than one adaptation manager, each responsible for resource management (Adaptation Manager can be also resource) [45]:

- *Orchestrating Adaptation Manager* (OAM): Manage and coordinate activities performed by touchpoint autonomic managers. Such orchestration can be performed "*within a discipline*" - multiple touchpoint autonomic managers of the same type (one of

self-configuring, self-healing, self-optimizing or self-protecting), or "*across disciplines*" - touchpoint autonomic managers being different in type.

- *Touchpoint Adaptation Managers* (TAM): This type of AM manages one or more managed resources directly, using the touchpoint interface. Scope and range can be a *single resource* (network device, LVC, application server, services), *homogenous resources* (LVC's or OS Kernel) or *heterogeneous resources* (connected groups of physical servers, VEI's, groups of middleware or groups of storage devices).

Touchpoint Adaptation Manager

After provisioning of VEI, which uses shared physical infrastructure, its activity must be controlled. The TAM is responsible for managing adaptation loops, which realizes policies defining how virtualized resources dedicated to LVCs are partitioned to satisfy QoS goals. These policies are entered that specify rules on the use and sharing resources provided by an administrator during provisioning of SOI. Realization of adaptation strategies within the TAM is consistent with model proposed by the ACS model[66]. Logically, TAM is only responsible for managing computational resources that are isolated and assigned to particular LVC instances and workloads running within these LVCs. It collects monitoring data from the MMA, evaluates policies with the data and invokes management operations on the MMA, which are related to LVC-RMs and middleware configuration.

The TAM is subject to certain constraints that arise from aspects related to the potential use of resources by the LVC, which are a limitation of available CPU's, memory, networking throughput and storage capacity on a single physical server shared between many VEI's. An important aspect is also a security issue related to the OS-RPM, which cannot be accessible to TAMs. For instance, changing the number of assigned CPUs for Dynamic Resource Pool or memory limits associated with a specific LVC instance is managed by the GRM, which divides resource between many LVCs. Therefore, the OAM is responsible for the availability of resources for LVC in accordance with the policies between the rival instances of VEI.

An *interaction style* (Figure 25) defines how SOI Managed Element, exposed with sensor and effectors, interacts with the Adaptation Manager; *retrieve-state*/*receive-notification* that define how states through sensor components are delivered and *perform-operation*/*callout-request* that define ways to change the state through effector components [98]:

- *Retrieve-State*: The Adaptation Manager requests the SOI Managed Element to provide a particular value of a sensor's property. Such invocation can be synchronous or asynchronous and has a semantic of "request-response" flow providing some state information about a managed resource.

[66] Section 2.7

- *Receive-Notification*: The SOI Managed Element sends the Adaptation Manager an unsolicited message-based synchronous or asynchronous event in a single direction flow that does not return any state information except details about the success or failure of the flow.
- *Perform-Operation*: The Adaptation Manager initiates an action call of effectors of the SOI Managed Element with a single-direction flow without returning of any information except details about the success or failure of the flow.
- *Callout-Request*: An interaction style in which *a decision point* installed on the SOI Managed Element solicits a decision from the Adaptation Manager. The bidirectional "request-response" call can provide the state information and be synchronous or asynchronous.

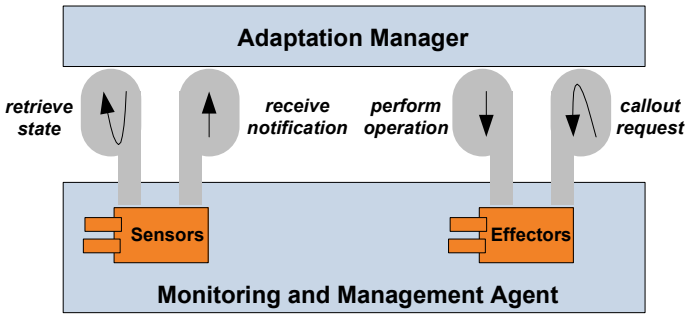


Figure 25. Interaction styles supported by the AM and MMA.

An important aspect is also how scenario policies are evaluated by the AMs. There are two known categories: *solicited* or *unsolicited*. Each solicited decision is a direct decision requested from a resource (SOI Managed Element) or any other external system. A solicited decision has an input and an output, which is a set of key-value mappings where "*key*" is not an empty string and "*value*" is an instance of Object reference. Unsolicited policies are evaluated periodically, with the evaluation property defining the time span between intervals, or are evaluated in reaction to a change of the system's state.

Orchestrating Adaptation Manager

The main task of the OAM component is control of multiple instances of TAM by activating adaptation loops driven by provided policies from the PMI. This control is ensured, on the one hand, by the required level of QoS for services running on the VEI instance and, on the other hand, by maximizing the use of available resources. Moreover, it communicates with the PM to provision extra capacity to specific VEI instance if resources currently used by the LVC cannot be satisfied by adjusting resource control values by OS-RPMs. The Provisioning procedure of SOI determines what and how much infrastructure is allocated and initiates adaptive management process supervised by the OAM. It ensures fair physical resource distribution between VEI based on monitored data acquired from the MMA and definition of policies for computational resources allocation according to control algorithms.

5.3.6 Policy Management Infrastructure

The general concept of the PMI with dependent components of the AM and MMP is presented in Figure 26. When a policy definition is deployed to the AM, it is translated into *Policy Object*, whose specification is defined by the specific policy language. Implementation of the Policy Storage might be based on a file system or database (typically used for such a purpose), which provides support for backups, remote access and multi-user access, including transaction behavior, thus the storage might be shared between many AM instances.

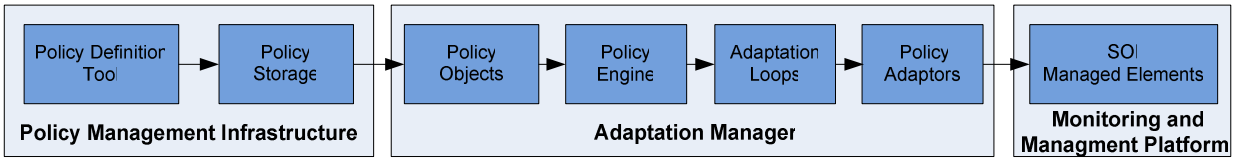


Figure 26. Policy Management Infrastructure with other related modules.

The approach taken by the author of the module design also stores information about the SLA[67], which is sewn into a given policy. It should be noted that support for any format implies that some policies can be defined through usage of dedicated tools supplied by providers. Therefore, these tools can be used independently of the Policy Definition Tool and then such policies are stored within the Policy Storage. During the provisioning process, PMI is utilized by the PM to activate a particular policy or set of policies as specified by the provisioning procedure.

5.4 Summary

The architecture of the A-SOI-LV-PP presented in this section is designed with the usage of trends useful for the construction of modern and adaptable platforms for provisioning and management of IT infrastructure. An important assumption is the modular design built using components that provide specific functionality. It enables graceful scaling of the whole environment and dynamic provisioning of SOI like Lightweight Virtualization Containers with required execution environments for application services. It is a software solution, which combines many components and ensures direct correlation between application specific QoS requirements and the adaptive management of LVC. The adapting services are able to proactively configure compute resources among application service workloads and assign consumption attributes of resources for particular VEI elements that host these workloads. They would adjust values of attributes that influence resource consumption of virtualized infrastructure running on physical hardware, queue or thread pool sizes and other resources[68] in middleware, ensuring QoS of application services and to route around bottlenecks. This will also ensure that such designed infrastructure will be able to identify its own problems automatically and activate appropriate procedures that deal with fixing errors.

[67] Step 1 of the process model is defined in the Section 3.4.1

[68] Application server middleware contains many application resources that are related to service scalability and availability, like JDBC resources (number of connections, statements cache size), JMS resources and others. Database middleware provides configuration with parameters such as buffers size for data, queries etc.

The proposed solution for implementation of the A-SOI-LV-PP supports a scalable and flexible architecture for easy expansion and implementation of new functionality by providing support for the abstraction of resources, dynamic configuration of components and detection and discovery of IT infrastructure elements.

6 Implementation of Adaptable SOI Lightweight Virtualization Provisioning Platform

After presenting the adaptable SOI provisioning process model and the requirements of the A-SOI-LV-PP, this chapter describes the reference implementation based on the proposed architecture. It motivates used API and technologies to implement software components exposing SOI Infrastructure Services. This chapter also deals with the problem of integration of Adaptation and Provisioning Managers with Monitoring and Management Platform.

This chapter presents the details of implementing the A-SOI-LV-PP. The following information concerns the general solutions and technologies used during the implementation process as well as provide details of the individual modules of the platform. Selected implementation details are discussed concerning discovery, self-configurability and adaptability, based on reflective middleware concepts. Construction of components exposing sensors, effectors and provisioning Infrastructure Services of the SOI running over Solaris Lightweight Virtualization platform is also presented.

6.1 Selected Technologies

From among many well-known and widely available management/monitoring tools and protocols, the author selected the Java language and the Java Managed Extensions (JMX) platform. Java is widely used in industry and minimizes requirements because it is supported on many hardware and OS platforms. JMX is a standardized management framework for managed resources' instrumentation, enabling a state's observation and operations' invocation. When applied to the implementation of software platforms for IT infrastructure management, it provides many advantages: *(i) software components called Managed Beans (MBeans) implemented in Java for managing heterogeneous resources with standardized interfaces, (ii) other management technologies can be integrated and exposed via MBean components, (iii) agents implemented with JMX can manage and easy scale to a large number of resources distributed over many physical devices.* JMX covers three levels of the presented environment: *Instrumentation level*, *Agent level* and *Distributed Services level* [93].

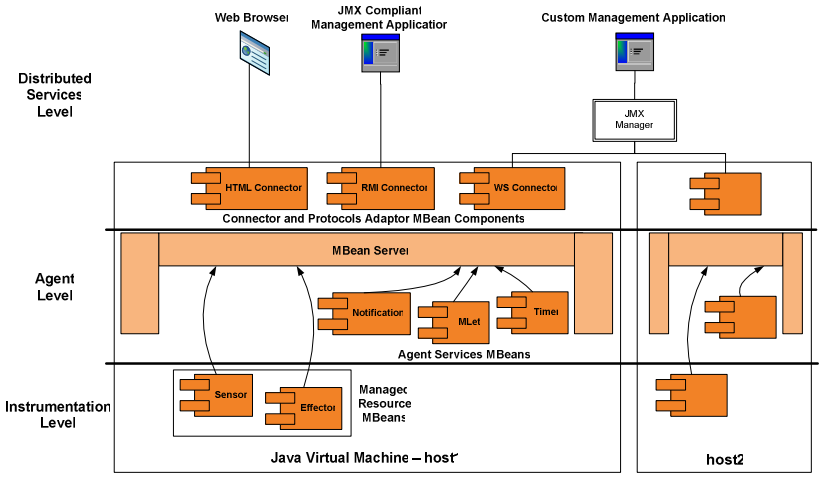


Figure 27. JMX Architecture.

The Instrumentation level focuses on object-oriented resource representation and exposure of a management interface to make a resource manageable. Object-oriented resource representation, as defined by JMX, is an MBean object conforming to Java Bean[69] (a reusable Java component) restrictions, with a well-defined management interface providing set of management operations, attributes and notifications. Access to the MBean interface uses *Reflection API*[70], allowing client applications to read and write managed attributes, as well as to call management methods based only on their names and signatures, which are defined at runtime. The second level, *the Agent level*, acts as a registry for all managed objects and other helper objects (such as monitors, timers, relation objects, objects facilitating dynamic loading of other MBeans from remote locations, connectors for RMI and SOAP, adaptors for SNMP and HTML). All MBean components are registered and managed by an *MBeanServer* to enable resource management and provide access to these components. This access is provided by specific connectors and adaptors (MBeans also) deployed in the agent layer constituting a remote API that can be used by remote clients. The third layer, *the Distributed Services level*, provides remote access and other functionality for the managed objects. It consists of remote MBean proxies that are used by remote clients for management purposes. Subsequently, adaptors deployed in the second layer allow remote clients to access the monitoring resources using JMX-compliant consoles[71], the SNMP management console and web browsers. JMX delivers a portable and scalable mechanism to manage different resources, instrumented with software components equipped with many services provided by the technology. Many vendors support it, for example, JVM and Java EE Application Servers are JMX compliant. Another example is MySQL database,

[69] JavaBeans are reusable software components conforming to the convention of a java object having nullary constructor, setters and getters and being serializable.

[70] It allows a Java application to examine, or "introspect", upon itself and manipulate internal properties of the program.

[71] JConsole is shipped with Java Development Kit (JDK)[71] or the console from the MC4J [148] package. These tools are equipped with rich GUI JMX client interfaces and can be used for connecting to any JMX agent.

which provides a MXJ Connector[72] package for deploying and managing the database. Dynamic MBean Loading Service (JMX-MLet) provides services for dynamic loading of MBean components into the MBeanServer. The JMX-MLet uses special descriptors named MLets which are pseudo XML files with special tags for describing required JAR files, component class and name in the MBeanServer. When combined with reflectivity techniques, it provides a powerful mechanism for loading only required components exposing appropriate Infrastructure Services.

The Policy Management Infrastructure and Adaptation Manager modules use rule engines for policy representation and adaptation loop processing and, through the requirements and architecture, require arbitrary policy engines support. The rule engines selected are JBoss Drools and Jess, both of which implement the RETE algorithm and provide Java integration through Rule Engine API[73]. Drools environment is a comprehensive open-sourced integrated platform for definition, management and execution of rules and event processing. Jess is a commercial platform and, in contrast to Drools, provides support for fuzzy logic.

The Provisioning Manager module requires a flexible BPM suite for visualization and execution of provisioning procedures, which orchestrate SOI. JBoss jBPM supports the entire life cycle of the business process (from authoring through execution to monitoring and management). It is an embeddable, lightweight Java process engine with process modeling in the Eclipse platform and the web (business users), which also supports tight, powerful integration with business rules and event processing.

As the technology for implementation of GUI system - Eclipse Rich Client Platform (RCP)[74], JBoss RHQ[75] and Google Web Toolkit (GWT)[76] were chosen. Eclipse RCP is a standardized platform for implementation of graphical tools, which incorporates Eclipse libraries. It should be used when more advanced GUI functionality not provided by currently existing Ajax[77] web frameworks is required and can be integrated with other extensions like jPBM Designer or Drools editor. JBoss RHQ delivers a core user interface used for monitoring and management of IT infrastructure and is designed with layered modules that provide a flexible architecture for deployment of plugins written according to abstraction interfaces that suit arbitrary SOI Managed Elements.

---

[72] http://dev.mysql.com/doc/refman/5.0/en/connector-mxj.html

[73] Java Rule Engine API provides a unified access interface to rule engines from Java applications. It standarizes the interface for such operations as registering and unregistering rules, parsing rules, inspecting rule metadata, execute rules, retrieve results and filter result.

[74] http://www.eclipse.org/rcp/

[75] http://rhq-project.org/

[76] http://code.google.com/webtoolkit/

[77] Ajax - Asynchronous JavaScript and XML are a group of interrelated web development methods used on the client-side to create interactive web applications.

Using the aforementioned software technologies and frameworks for implementation of the A-SOI-LV-PP will provide the IMF that conforms to the adaptable architecture requirement and realizes the defined SOI provisioning process. The remainder of this chapter elaborates implementation details of software components for SOI exposition within Monitoring and Management Platform and integration with Adaptation and Provisioning managers.

6.2 Monitoring and Management Platform

Having in mind the concept of object-oriented managed resource representation, an interface was implemented for adaptable provisioning of Solaris 10 virtualized infrastructure with the SOI paradigm considered. Such resources include network and hardware infrastructures, virtualized computational and storage resources, middleware, and applications. The proposed solution promotes a lightweight component approach implemented with JMX technology used for coupling between managed SOI elements, adaptation and provisioning managers. The Solaris components presented expose information about resource usage, configuration and management options and allow performing some management operations (including changes in configuration, creating new zones and projects, limiting resource usage, booting/halting/shutting-down zones).

6.2.1 JMX-Based Infrastructure Management System Platform

Since the primary goal of the MMP is monitoring and management, the author, for that purpose, uses the JMX-based Infrastructure Management System (JIMS) [139] platform. The main purpose of the JIMS is to provide an integrated software framework for managing elements of computational infrastructure and applications supporting a majority of Horn's requirements and supports the proposed architecture of the A-SOI-LV-PP:

- *The Instrumentation Layer* provides physical infrastructure management and monitoring information using module components with sensor and effectors installed in the JIMS Agents. They are able to discover the type of an environment it operates in (*context awareness*) and load proper modules with sensor, effector and provisioning software modules from the *Components Repository*. The JIMS automatically handles a list of SOI Managed Elements in each VEI (*self-configuration* and *re-configuration*) through appropriate discovery components, which also support reaction to node failures. *Heterogeneity* allows installation on almost any UNIX-like system, including all Solaris versions and many Linux distributions.
- *The Interoperability Layer* provides a common point of communication with computational nodes in clusters through dedicated Access Nodes. The JIMS *Gateway* (used as the *Infrastructure Gateway*) running in the interoperability layer discovers and organizes generic agents into a federation acting as a mediator between remote clients and these agents (*Infrastructure Registry services*). During start-up, the gateway discovers it is installed on an Access Node and automatically loads JMX Gateway module (*self-definition*). The JIMS Gateway provides access to

the SOI Managed Elements using a unified and adaptable interface (*openness*), which does not define particular management routines and is general for accessing any type of managed elements. *Reactiveness* is promoted through standard JMX notification services sent to interested parts when particular events occur in SOI Managed Elements. This layer provides all the required functionality of the Connectivity Services Layer of the MMP.

- *The Integration Layer* enables discovery of all accessible VEI instances and provides an overall view of the SOI environment. This layer consists of Global Discovery modules installed in JIMS Gateways running on Access Nodes. This service enables the election of Global Registry of JIMS Gateways, which can be used by GUI consoles and the Orchestrating Adaptation Manager to supervise Touchpoint Adaptation Managers installed on particular gateway instances. The election mechanism provided offers *self-healing* functionality, allowing operation in an unreliable environment.
- *The Client Application Layer* consists of applications connected to the JIMS system, which are consumers of the information produced by the system. Provisioning and Adaptation managers can be implemented with different technologies, hence access to the SOI Managed Elements is possible with the use of industry standards such as Web Services, RMI, HTTP or SNMP.

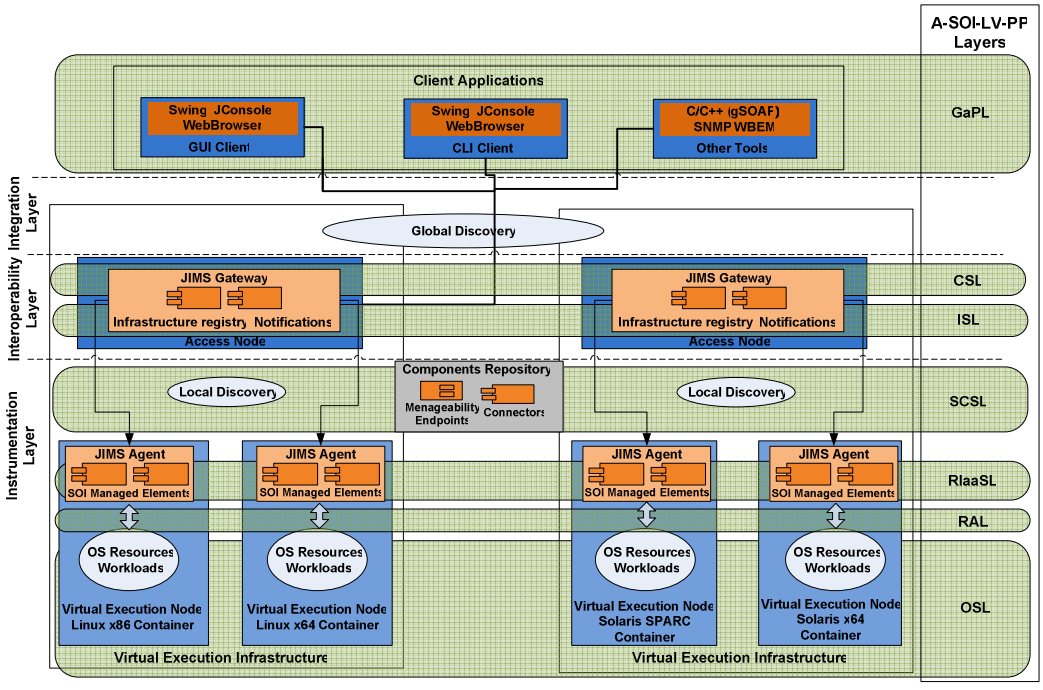


Figure 28. JIMS layered architecture.

The other Horn's characteristics, i.e. *Anticipatory, Self-Optimizing* and *Self-Protecting* are not covered by the JIMS system and are the main target of the Adaptation Manager. The internal architecture follows the *reflective middleware* foundation, providing a mechanism for delivering the host-Infrastructure Services for the OSL instrumentation, distribution of middleware with the various connectors interoperable with many software platforms and

discovery services and common middleware services like notifications or dynamic components loading.

The JIMS uses computational reflection for determination of the underlying operational environment and loads appropriate modules with JMX components through JMX-MLet service (Listing 1), exposing monitoring and control services for managing the underlying resources. Its features, such as adaptation to operating systems, kernels, and IP protocols, auto-configuration facility (VEI level auto-configuration) and dynamic deployment of proper monitoring sensors from one common modules repository, make it well suited to providing MMP as an element of the A-SOI-LV-PP. The JIMS can be extended to suit the abstractions of different heterogeneous elements of VEI used by the Adaptation and Provisioning Managers.

6.2.2 Resource Access Layer

The author found it useful to introduce a level of indirection when using diverse management interfaces for underlying resources and separates interface from implementation, thereby allowing uniform usage of all these methods by MBeans components and making it easy to switch implementations. It also allows *adaptation* when a new version of tools is released, which is accomplished through reflection and incorporation of *Command*[78] design pattern - Listing 2. Access methods for retrieval of configuration data use system files (wherever possible), command-line tools and Java Native Interface (JNI) through C-language Solaris API. Other management and provisioning operations, such as limiting resource usage with resource controls, modifying projects or managing zones lifecycle, are realized with proper command-line tool invocation based on the *Runtime.exec* method.

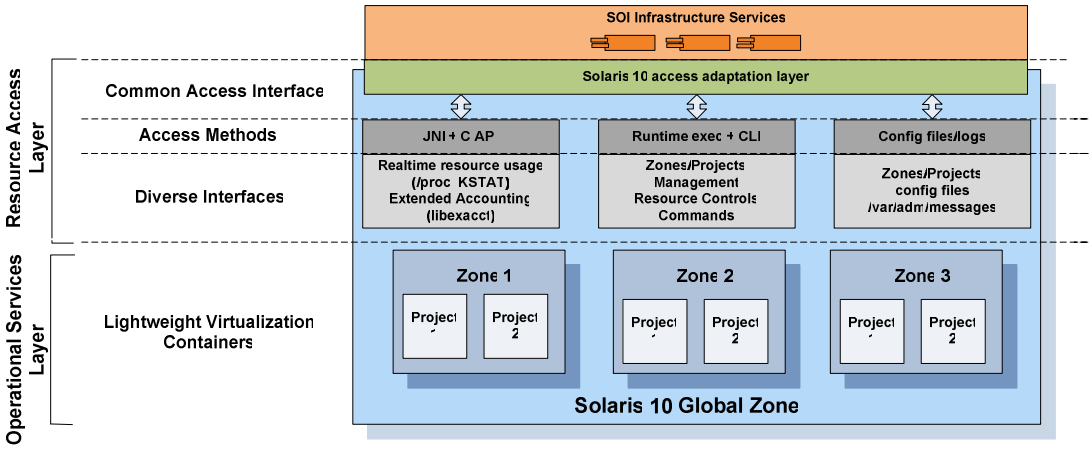


Figure 29. Integration of Resource Access Layer with Solaris 10 Lightweight Virtualization Containers.

Current consumption can be measured through acquiring a snapshot of resource usage from the */proc* Virtual File System (VFS), where each process in the zone has its own subdirectory (the global zone also contains entries for processes from local zones). The most important

[78] Separates the object that invokes the operation from the object that performs the operation. Simplifies adding new commands because existing classes remain unchanged [144].

information is contained in *psinfo, usage* and *status* files having a binary format that are processed. Because these entries have a native binary format and cannot be read directly by the Java application, it is necessary to use Solaris 10 API through the JNI. By summarizing information about resource usage of processes, it is possible to calculate resource usage of projects and zones.

The Extended Accounting information (precise overall resource usage) is collected from Extended Accounting logs on all LVC within VEI. These logs can be parsed using C or Perl API. While interoperability between Perl and Java is possible with Java-Perl Lingo (JPL), the author decided not to use it because it imposes constraints on Java code (like classes being stored in a default package), thus libexacct C API and JNI is exploited.

Solaris 10 does not offer any native mechanism for notifying about events, such as zone/project creation or removal, therefore such information must be read periodically from configuration files and MBeans should be added or removed when necessary. If changes (configuration, state or resource usage) in the managed resource occur, a JMX notification can be sent, so that higher layers, e.g. the PM or AM, can react to events instead of querying the state actively. This shows that interfaces with notification can be created even if the underlying system does not offer native notification mechanisms.

By implementing a discovery service, a mechanism for auto-discovery of Java applications is introduced to be automatically monitored and managed by the MMP. It relies on the instrumentation features of JVM, which allows the interception of Java applications running through the "*java-agent*" option enabled. Such an agent must provide a class with a *premain* method that is called before the *main* method of the application class. An implementation of such agent initializes the discovery mechanism, enabling automatic registration of an application within the running JVM (Listing 4). The application itself does not have to be modified; it's enough to add the *-javaagent:soima-javaagent.jar* option at startup. This will cause the application to be discovered by the Infrastructure Access Gateway and put the information in the Infrastructure Registry service. Such configuration of particular middleware can be done through a configuration template prepared within LVC-AA. After provisioning, such a discovery is automatically initiated during the start-up phase of middleware running within instrumented JVM.

Support for JMX is built into the Java Virtual Machine instrumented with MBeans and registered in a Platform MBeanServer with RMI connector configured by default. A lot of middleware based on the Java 2 Enterprise Edition platform takes full advantage of the JMX technology, simplifying management of such application runtimes through JSR-77[79]. They

[79] The JSR-77 specification defines a management information model called the J2EE Management Model for the J2EE platform. The J2EE Management Model is a specification of the attributes, operations and architecture of the managed objects required by compliant J2EE platform implementations. This model is inter-operable with a variety of industry standard management systems and protocols. It also provides standard mappings of the model to the Management Information Base (J2EE MIB) and to a Java API as a server resident EJB component [149].

provide MBeans components with API for configuration, management and deployment of applications with support for provisioning of new instances, including clustered configurations[80]. This allows use of such SOI Managed Elements with a simplified interface without detailed knowledge about JMX.

6.2.3 Runtime Infrastructure as a Service Layer

It was mentioned that JIMS uses MBeans as an abstraction for representing resources. Therefore, the following MBeans components shown in Figure 30 were implemented, which represent each zone and project defined in the system and are Manageability Endpoints required by the provisioning model. There are also other MBeans registered by the Solaris 10 extension module; however, they are omitted in this description, as the services they provide are not important for the module described.

- *Zone monitoring MBeans*: These MBeans provide information about zones' resource usage. For each zone in the system, there is one MBean registered.
- *Zone management MBeans*: These MBeans provide information about zones' configuration (i.e. zone name, resource consumption limits, configured projects, network interfaces) and state (i.e. running, halted). They also expose the interface for provisioning containers and changing configuration (i.e. creating or removing projects, setting resource consumption limits). There is one MBean registered for each zone.
- *Project monitoring MBeans*: These MBeans are counterparts of the zone monitoring MBeans for projects and provide information about projects' resource usage. For each project in each zone there is one MBean registered.
- *Project management MBeans*: These MBeans are project counterparts of zone management MBeans, provide information about projects' configuration (i.e. project name, belonging users and groups, resource consumption limits) and expose the interface for changing it. There is one MBean registered for each project.

JMX allows searching MBeans by the domain that they are registered in and by their properties (both domain and properties are specified when MBean is registered in the MBeanServer). Using this mechanism, it is possible to find MBeans representing all zones (or projects), specific zone (or project) or all projects belonging to given zone, etc. This way it is possible to learn about existing projects and zones.

The security aspect must take into account many things related to a broad variety of managed resources, which means that some operations, such as creating zones, modifying the project database etc., may be allowed only for restricted groups of users. The privileged position of operations performed in the global zone should be very carefully analyzed. It is necessary to

[80] Glassfish Application Server is equipped with Appserver Management Extensions (AMX) which are JMX components implementing API for platform management. It also provides services enabling self-management.

take into account situations when a provisioned VEI consists only of a cluster of local zones distributed over different global zones[81]. This is why two deployment options are provided: *Global* and *Restricted* Access.

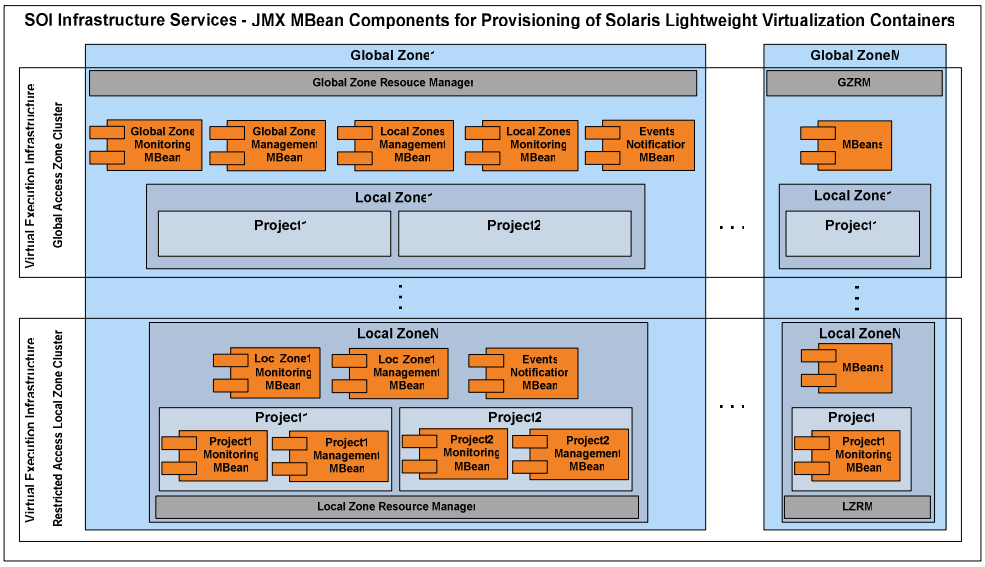


Figure 30. Manageability Endpoints components for Solaris 10 LVC provisioning and management.

Clients in the Global Access configuration connect only to the particular MMA installed in the global zone that has permission to perform operations on all local zones in the current Solaris 10 OS instance. Such a configuration option provides functionality for a wider spectrum of management operations like provisioning new zones/projects and resource controls configuration modification. In the Restricted Access configuration, each local zone contains the MMA which is able to perform only operations on the given zone, depending on the assigned privileges. Such configuration is required when dedicated VEI would be administered by external parts (for example private or public Clouds), and will be provided with SOI that is able to operate only within the "owned" infrastructure.

The aforementioned JMX MBean components provide services for provisioning of VEI over the Solaris lightweight virtualization platform. They are Manageability Endpoints installed within the MMP supporting the adaptable SOI provisioning process through Infrastructure Services exposition required by the Adaptation and Provisioning Managers. These JMX components can be deployed on demand to MMA to fulfill the requirements of a particular provisioning process implementing the sensor and effector interface required by adaptation loops and provisioning services. The JMX-MLet service can be used by the MMA for managing modules which contain components for Adaptation and Provisioning managers. It allows instantiation and registration of MBean components in the MBeanServer, which are downloaded from the Components Repository specified by a remote URL. Such a process requires an MLet descriptor (also specified by a URL) which defines the information on the

[81] OS Kernel instances running over physical servers.

MBeans to be obtained and makes it possible to create dynamically extensible agents. In the presented example (Table 4),***org.jims.modules.solaris.solaris10.mbeans.ZoneAgent*** MBean component uses computation reflection (Listing 2) to determine the current context for running MMA and adapts itself through compositional adaptation of other MBean components exposing the appropriate type of the Infrastructure dedicated to the OS Kernel of a regular LVC instance.

```
<mlet
  <!—- Main entry MBean component – exposes SOI Virtualized Services -->
  code="org.jims.modules.solaris.solaris10.mbeans.ZoneAgent"
  <!—- Libraries used by MBean components -->
  archive="jims-sunos-sparc-3.0.0.jar, jims-common-solaris-sparc-3.0.0.jar,
  jims-solaris-persistence-3.0.0.jar,
  commons-collections-3.1.jar, dom4j-1.6.1.jar, antlr-2.7.7.jar"
  <!—- Domain name in an MBeanServer -->
  name="Management:class=Solaris10Management" version="1.0">
</mlet>

<mlet code="org.jims.modules.solaris.mbeans.SystemInformation"
  archive="jims-sunos-sparc-3.0.0.jar,"
  name="Monitoring:class=SystemInformation" version="1.0">
</mlet>
```

Table 4. JMX-MLet descriptor for Solaris 10 Manageability Endpoints components.

Such construction enables on-demand deployment of any module of the A-SOI-LV-PP, for instance if there is a requirement for hierarchical structure of running AMs to realize complex policies, an appropriate JMX-MLet module can be deployed or undeployed on the target OS Kernel or LVC instances.

6.2.4 Information Services Layer

Both real-time resource usage and accounting data are stored in a database on the IAG. However, since a large quantity of real-time data might be generated, the module enables limitation of the amount of monitoring metrics about particular project or selected processes (information about zone resource usage may be derived from all participating projects). This is supported with MBean components allowing selection of running workloads by adding filters that use various criteria. By default, it is always possible to filter processes by their process identifiers - pids (for monitoring given applications), but it is also possible to create other kinds of filters, for example for selecting processes with high resource usage. While the system administrator may directly specify the process identifier, it also provides a mechanism that automates this task with previously discussed mechanism of instrumentation of Java

applications[82]. Monitoring data are stored in a relational database with the schema presented in Figure 31 and managed with Plain Old Java Objects (POJO) components persist with a Hibernate framework[83].

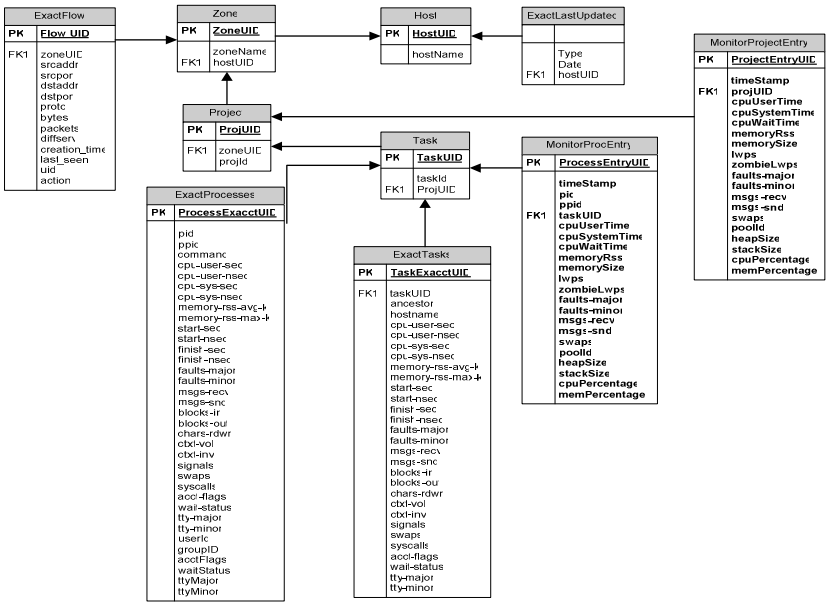


Figure 31. Database domain model of Solaris accounting services.

The notification services provide a way to learn about a newly created (or removed) project and zone without querying it through the JMX notification service. Exploitation of the service provides functionality for discovering Solaris 10 VEI elements like configured zones in the system and projects that particular zone contains. Two separate MBean components emit notifications defined with POJO (Figure 32) related to projects and zones lifecycle. Each component that wants to receive the information must register for notifications in following MBean components:

- *Zones Events Notification MBean*: This MBean sends notifications about events occurring in a particular zone. Discovering particular entities within a zone and handling call-backs related to zones' life-cycle is also supported:
  - A new zone is discovered in the system,
  - An existing zone is removed,
  - A particular zone changes state (i.e. zone is booted or halted).
- *Projects Events Notification MBean*: This MBean sends notifications about events regarding projects:
  - A new project is discovered in the system,

[82] Enabling persitency of monitirng data of JVM is accomplished via the "*persist"* option specification to *javaagent*.

[83] http://www.hibernate.org/

– An existing project is removed or modified.

It is also possibility to send notifications about attributes' threshold exposed by MBean components from the RIaaSL using built-in JMX monitors like *String*, *Gauge*, and *Counter* monitors for watching string, integer and numeric attributes appropriately. These monitors can be configured to periodically observe SOI Managed Elements and emit JMX notifications only if a specific attribute has changed beyond a specific threshold.

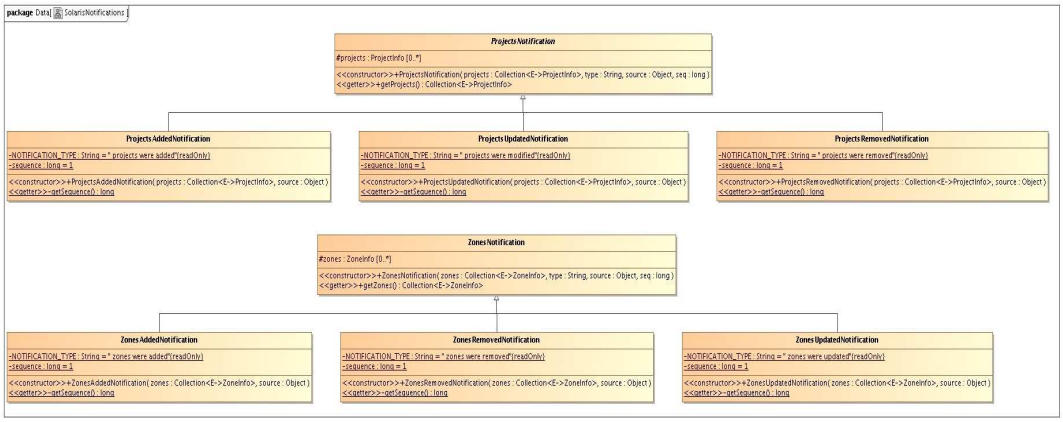


Figure 32. POJO components describing events related to Solaris containers lifecycle.

The IAG is able to handle events and modify the Infrastructure Registry with a list of available LVCs on a particular OS Kernel instance as well as events from JMX monitors. These events can be also sent to the PM or AM modules.

6.2.5 Governance and Policies Layer

The MMP also has a graphical console for centralized access to managed elements of the OSL. A specifically implemented plug-in to extend the system management interface of RHQ added functionality for provisioning, management and monitoring of VEI based on the Solaris platform. This extension allows viewing a list of physical nodes (instances of the OS Kernel) and the current configuration of lightweight containers (Figure 33) with possible modifications such as resource controls (Figure 34). The system administrator also has the ability to define graphs presenting the consumption of computing resources (Figure 35, Figure 36 and Figure 37) and defining alerts (Figure 38). This approach supports the management of many different operating systems and AEE platforms with a particular emphasis on solutions of JBoss solutions. Additionally, administrators can connect directly to the MMA instances using client application tools compatible with the JMX technology, enabling the usage of the instances of MBean components installed in a specific MBeanServer. In the drawings submitted, the *jconsole* tool is evaluated in which MBean components are presented for provisioning, management and monitoring of lightweight containers of the Solaris 10 platform. The downside of this solution is there is not an advanced user interface which, in order to perform a particular operation, requires in-depth knowledge of the precise semantics of the components providing appropriate SOI Infrastructure Services.

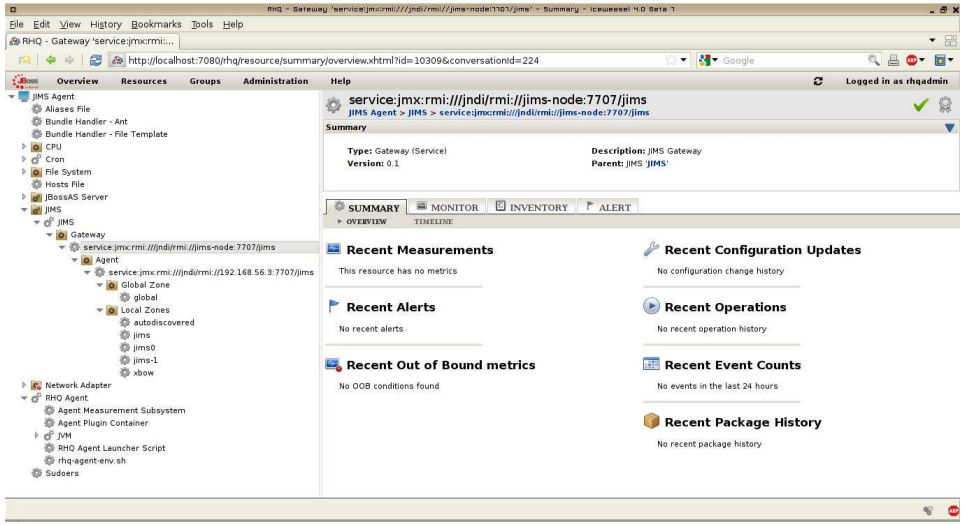


Figure 33. Solaris OS Kernel with list of configured Containers.

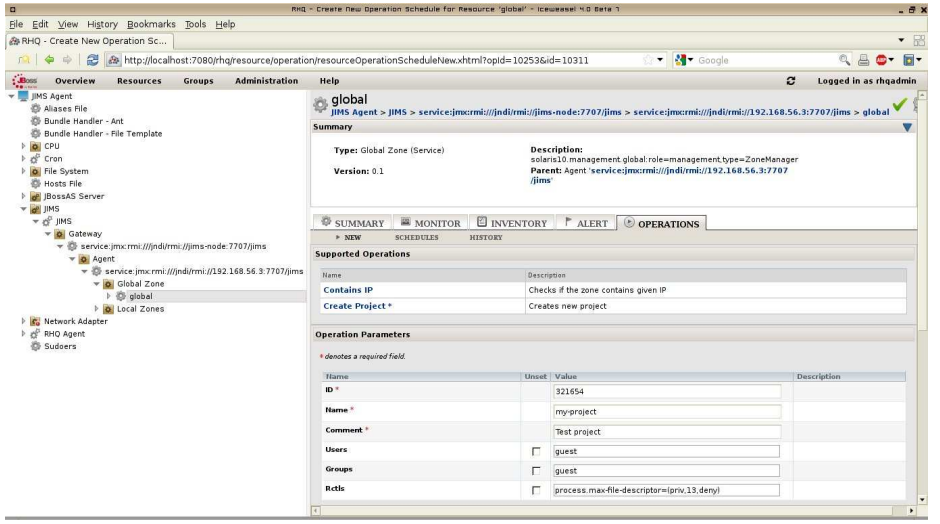


Figure 34. Modification of resource controls of the Solaris Container.

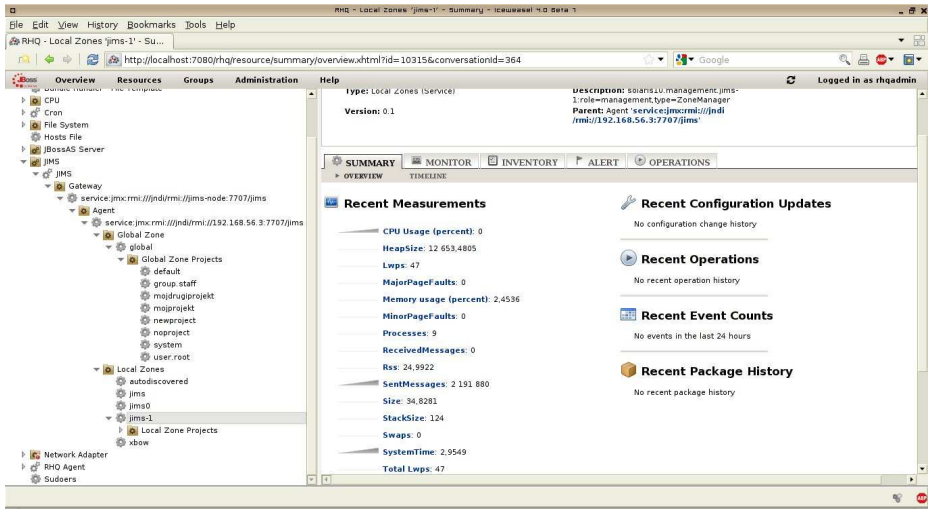


Figure 35. Solaris Container monitoring metrics.

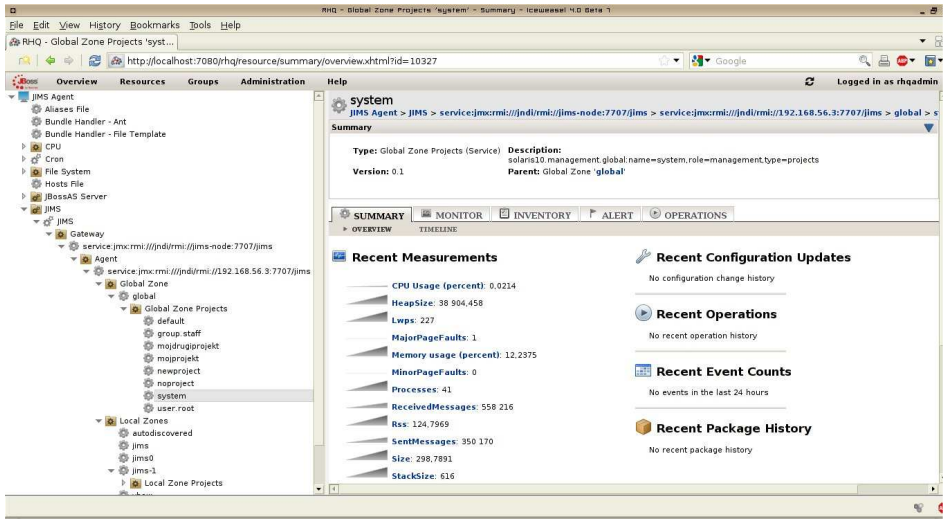


Figure 36. Solaris project monitoring metrics.

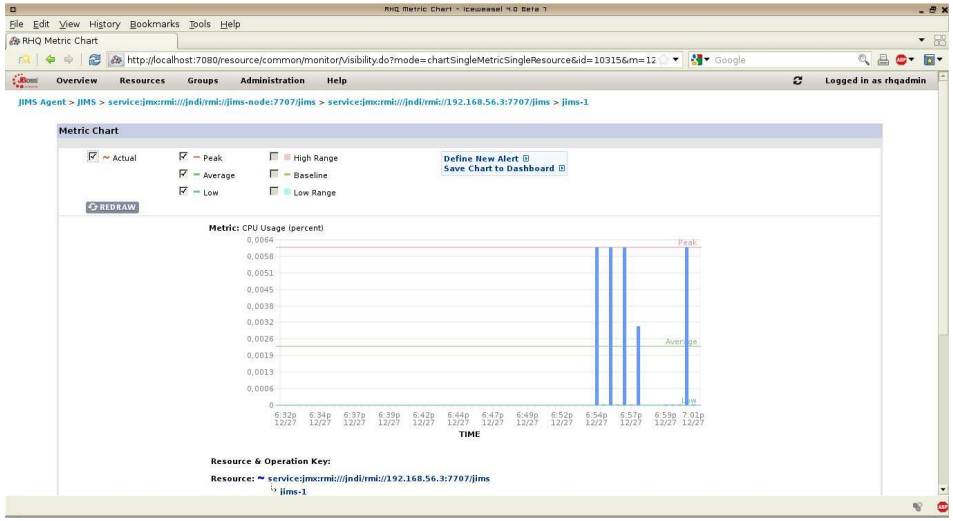


Figure 37. Chart with CPU metric of the Solaris Container.

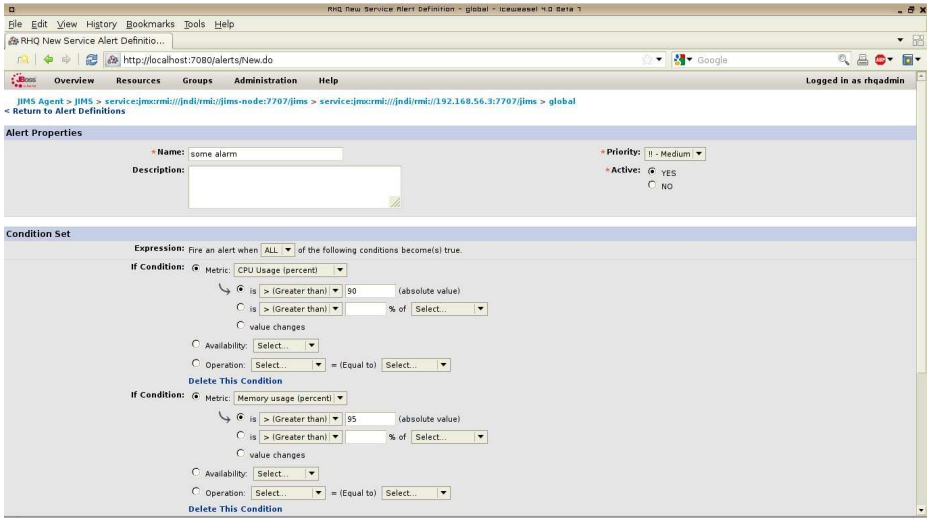


Figure 38. Alert definition for CPU metric of the Solaris Container.

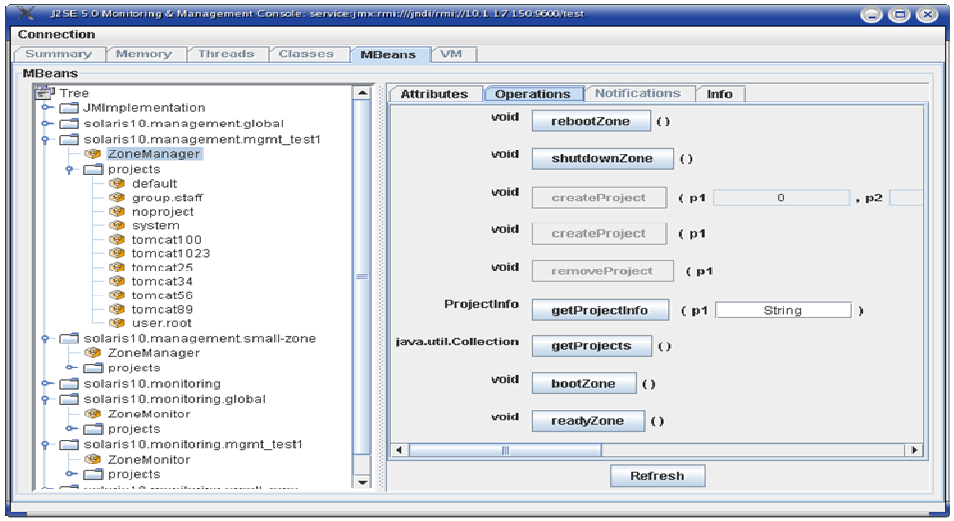


Figure 39. MBean components for Solaris Containers and projects management.

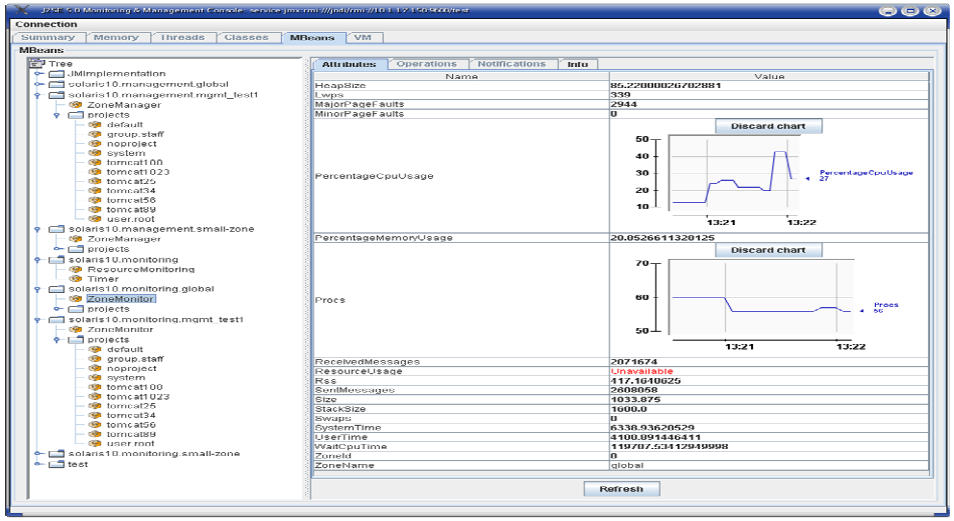


Figure 40. Monitoring metrics of CPU usage and number of processes.

6.2.6 Summary

In opinion of the author, the natural order of implementing the IMF for adaptable provisioning of SOI over lightweight virtualization is to first devote attention to SOI Managed Elements construction. This is justified by the fact that a management policy and an adaptation strategy of an adaptation system must be expressed in the context of capability provided by Managed Resources. The heterogeneity of native resource control interfaces offered by elements of the OSL creates the need for a uniform open interface. The presented implementation of the MMP based on the JIMS framework implementation is a solid background for the implementation of the Adaptation Manager and further orchestration of exposed Infrastructure Services by the Provisioning Manager.

6.3 Adaptation Manager

According to the proposed architecture, the AM contains modules responsible for adaptation loops management and integration with the PMI and MMP. It is implemented as a lightweight library that can be deployed on demand within the MMP using MLet service. Each modules implementation is based on the JMX technology with services implemented through MBean components. Policy evaluation requires coordination of activates (Figure 41) performed by the PMM (Governance and Policies Layer), PEM (QoS Layer) and RAM (Model-based Translation Layer) and is implemented with components realizing particular steps in the adaptable SOI provisioning process. Integration with SOI Managed Elements does not require any descriptors because reflective techniques are used and, as a result, appropriate policy object components (adaptors) required by policy engines are generated dynamically without a need to restart the runtime. The implementation of the AM supports, by default, three policy engines; Drools and Jess as rule engines and IBM's ACPL runtime. Nevertheless, the internal design and implementation also supports the possibility to add other custom policy engines, which can be attached and detached dynamically.

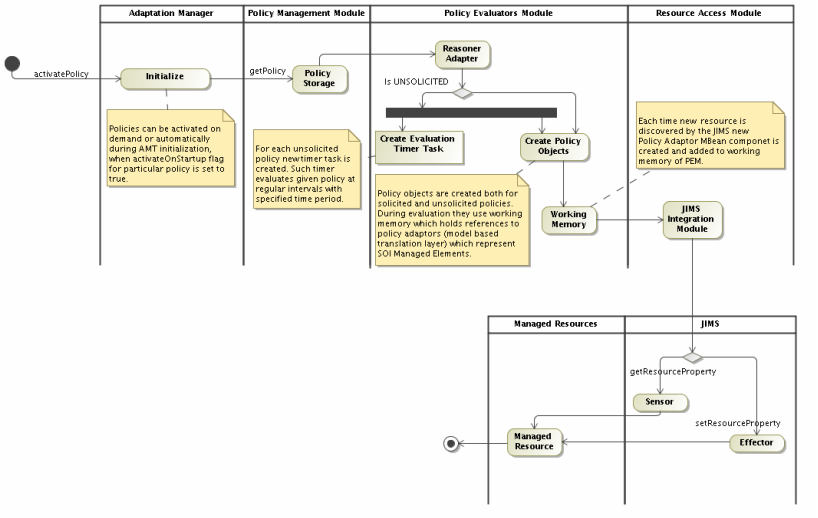


Figure 41. Activity diagram of policy's evaluation by particular modules of the A-SOI-LV-PP.

6.3.1 Resource Access Module

The representation of the *Policy Adaptor* components in the AM is specified with the *ManagedResource*[84] interface for representation of SOI Managed Elements. This interface exposes general-purpose methods for manipulation of properties of the underlying Managed Resources (Figure 42). To meet all these requirements, A *ResourceMBean* abstract class for resource representation is defined, which implements the ManagedResource interface. For each SOI Managed Element (also MBean component), the Policy Adaptor (MBean resource wrapper) class is generated with resource-specific action methods implemented that depend on MBean properties, methods and notifications. When an MBean attach request process is initiated, the RAM checks if suitable a resource wrapper class is already available. If there is no such class then a new wrapper generation process starts. Such an MBean wrapper is generated from a parameterized template (Listing 7), transformed into Java class source code using Apache Velocity[85] library, next compiled with Java Compiler API, loaded into JVM and registered in the MBeanServer of the AM.

The MBean resource wrapper encapsulates local or remote communication to a particular MMA's MBeanServer containing a component for managing a given SOI Managed Element, allowing it to read and write resource properties, execute actions on resources and subscribe to notifications about events related to resource.

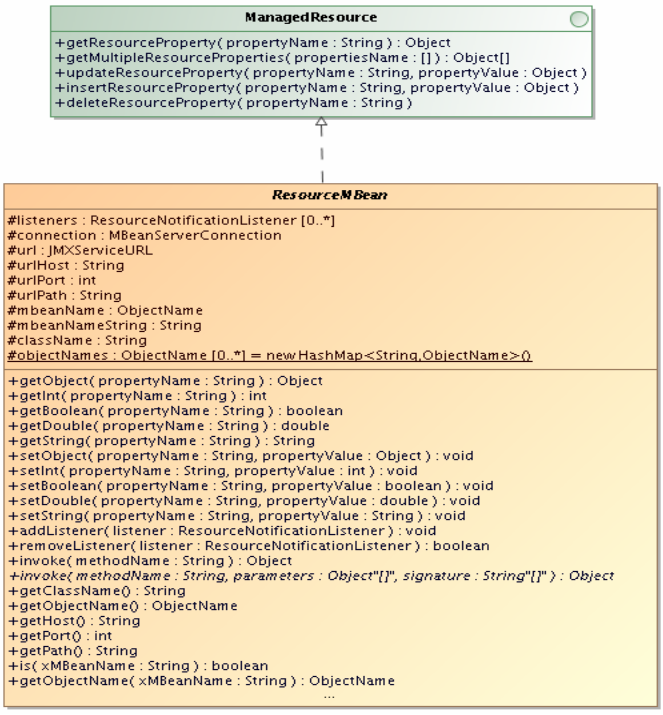


Figure 42. Interface specification of the Policy Adaptor component.

[84] This ensures compatibility with the ACPL.

[85] http://velocity.apache.org/

6.3.2 Policy Evaluation Module

The implementation of the AM system exploits the potential of the rule engine-based approach as attractive solution for the policy-driven SOI provisioning process. Such a solution brings mapping of a term policy to a production rule definition and could introduce some constraints in a policy expression, but it is not the case in most of the applications. The rule engine is a sophisticated software module that supports a scalable pattern matching algorithm, thus it might be used for a large number of facts and rules constituting a representation of knowledge. All rules are stored in *production memory*; in addition, facts that were matched against production rules can be found here (Figure 43).

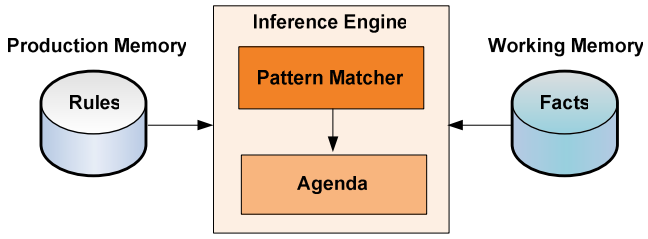


Figure 43. Rule engine structure based on Rete Networks.

The PEM's implementation supports many rule engines running in parallel, which might be attached and detached without restart and accessed with the Java Rule Engine API interface. In the PEM, SOI Managed Elements are represented as *facts* processed by the rule engine. Such exposition is performed during the AM's start-up phase through the Reasoner Adapter component. It is a class specific to a given reasoner acting as a bridge between the PEM and the reasoner and implements the interface described in Table 5. Its main responsibility is to provide access to selected resources and policies and to gather events from resources and provides them to the reasoner.

Table 5. Interface specification of Reasoner Adapter component.

Method Name	**Description**
attachResource	Attaches new a resource
detachResource	Detaches new a resource
addPolicy	Adds a policy for unsolicited or solicited decisions
removePolicy	Stops a policy evaluation and removes it from the reasoner
evaluateSolicitedDecision	Returns a solicited decision or null if the decision is not available
getInfo	Returns a reasoner description
shutdown	Performs a detach operation

Application of the rule engine influences the initialization procedure during start-up time of the PMI and AM (Figure 44). When the rule engine is started, it *activates a rule base* into

production memory for its use. The rule base contains all rules and class definitions that are to be evaluated against facts. The rule engine is driven by the arrival or change of the facts residing in the working memory. The changes of facts may be generated by external events because of rule engine activity or by the expiration of a timer that could be associated with previously received facts.

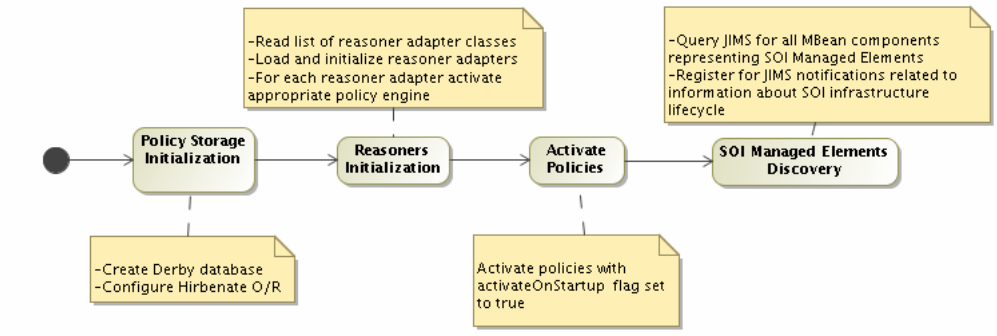


Figure 44. Bootstrap process of PEM.

The activity of the Adaptation Manager is performed in the following steps depicted in Figure 45:

1. The elements of the OSL (Managed Resources) are instantiated as MBeans within the MMA constituting SOI Managed Elements (Manageability Endpoints).
2. Policy Adaptors of SOI Managed Elements that play a role of Model-based Translation services are constructed automatically (2a) and put into Working Memory as facts (2b). The Reasoner Adapter interface is used in this step, which attaches particular SOI Managed Element representation to a specific Policy Engine.
3. Production Rules representing policies are loaded to Production Memory. At this point the Inference Engine is also started.
4. Pattern Matching algorithms are performed on all rules in production Memory and facts present in Working Memory.
5. All rules that are evaluated as true are put into Agenda to be performed.
6. Action is performed on the representation of Managed Resource in Working Memory.
7. The action is forwarded to SOI Managed Elements via Resource Wrapper and enforced with effectors.
8. Managed Resource parameters changes accessed by sensors are communicated to Policy Adaptors that triggers execution of step 4.

Steps 4 to 8 constitute a main execution loop of the Adaptation Manager. Since rules are declarative knowledge representation forms, they are not called like functions in a procedural language. Instead, they fire in response to changes in the facts available to the rules engine.

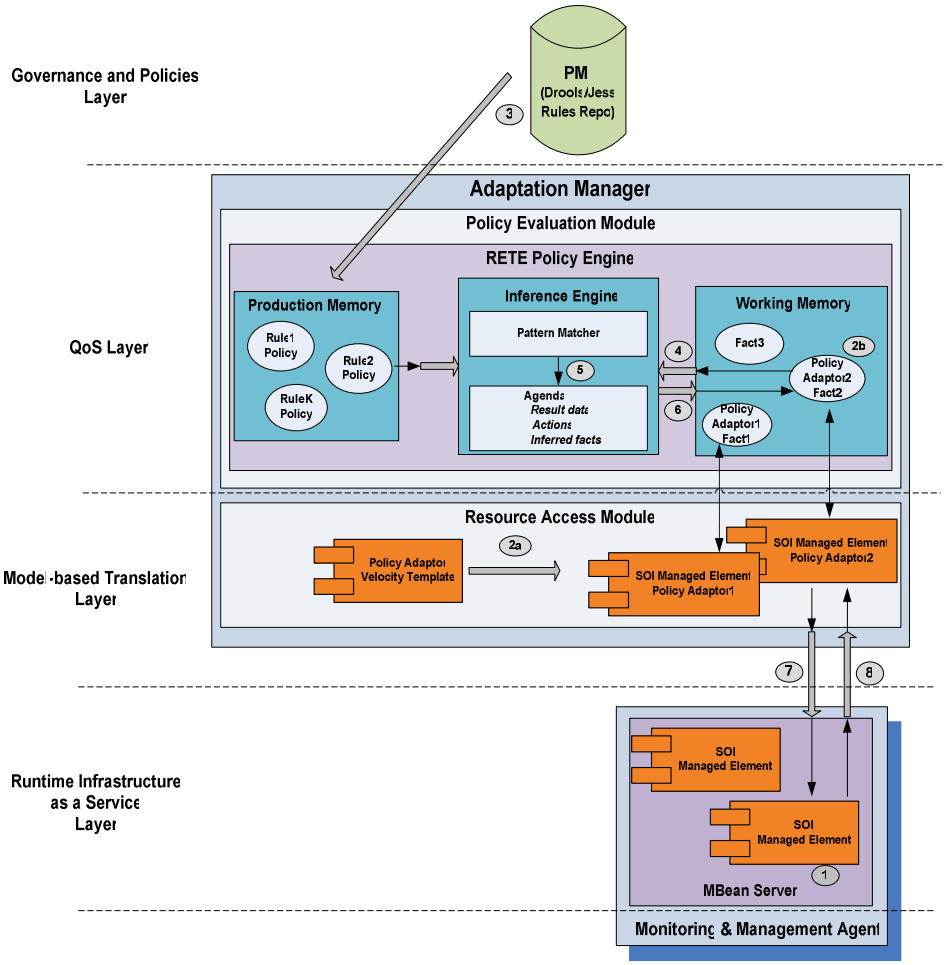


Figure 45. Adaptation loop processing performed by the PEM using rule engine.

6.4 Policy Management Infrastructure

The Policy Management Infrastructure provides services enabling policy definition and persistency support. The policies are managed manually and defined by the administrator using a GUI console that is also integrated with the AM. The implemented Policy Management Module delivers a graphical Policy Definition Tool console for policies' definition and integrates with the Adaptation Manager, enabling on-demand activation of policies.

6.4.1 Policy Definition Tool

The Policy Definition Tool is an integrated tool for policy definition, edition and activation. In the A-SOI-LV-PP, representation of policies allows handling any format having the following properties and is a container for policies expressed in arbitrary policy language:

- *Name*: Not an empty string, and must be unique within policy scope.
- *Scope*: Hierarchical structure of not a string used for denotation of the given policy applicability domain. An empty string is valid as policy scope and represents global scope; other scopes are non- global.
- *Type*: Policy can be of types solicited or unsolicited.

- *Evaluation in milliseconds*: Valid only for unsolicited policies, defines evaluation period in milliseconds.
- *Activated on start up*: If set to true, then policy evaluation is started immediately after initialization of the AM.
- *Policy Engine*: Name of the policy engine (reasoner) to be used by the policy.

Policies are identified by a name and scope for grouping (similarly, Java classes are grouped by packages) with a body provided as a regular string expressed to fit a particular policy engine. Defined policies are available for administrators to be activated within the AM through the "activate" option in the console or using exposed API, thus facilitating the integration of the PM.

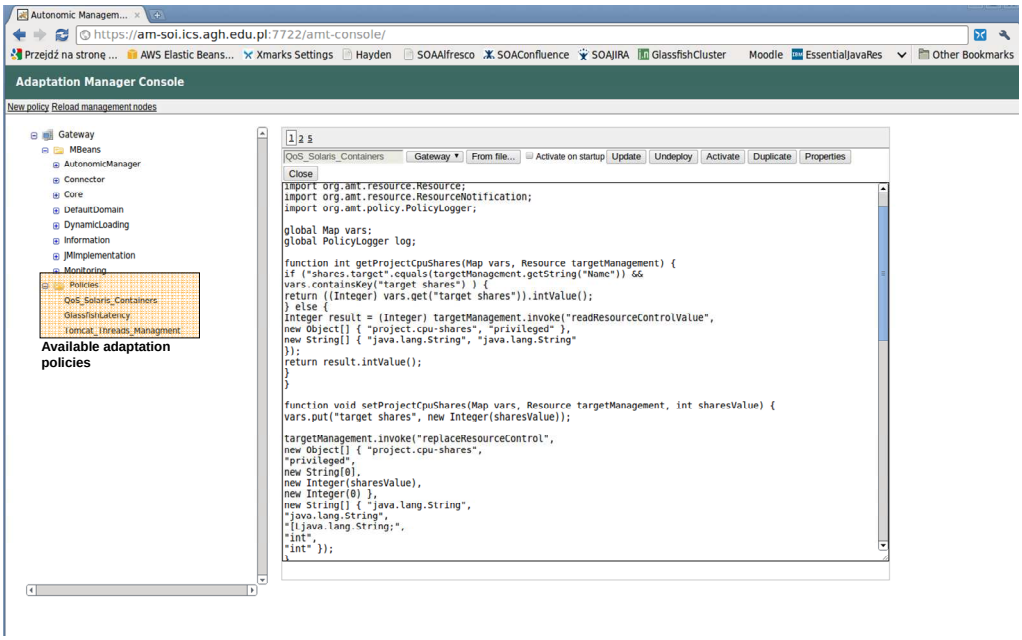


Figure 46. Web-console of the Policy Definition Tool.

6.4.2 Policy Storage

Before the policy is available for the A-SOI-LV-PP it is in a "*not exists*" state. If the policy must be made available for the provisioning process, it is deployed and persists in the storage, which results in a "*stored*" state. Such a policy is not evaluated until it is set to an "*active*" state and ready for evaluation on a specific LVC's node.

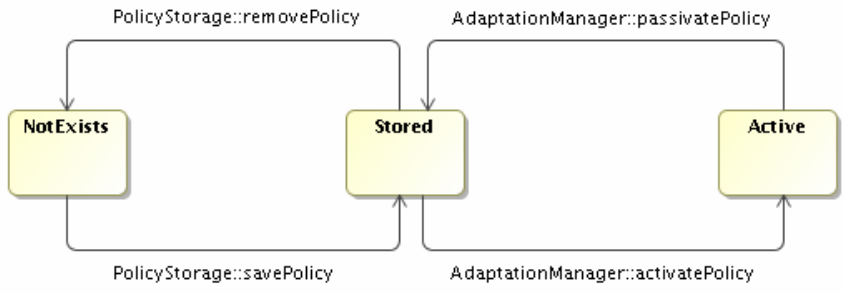


Figure 47. State-diagram of policy's life-cycle.

Policies are stored in persistent storage and managed by the Policy Storage component, which provides a standard interface for any repository presented in Table 6.

Table 6. Interface specification of Policy Storage component.

Method Name	Description
savePolicy	Adds a new policy
removePolicy	Removes a policy
getPolicy	Returns a policy or null if no suitable policy exists
listPolicies	Lists all available policies
clear	Removes all policies from policy storage
listStartupPolicies	Lists policies with a set of properties activated on start up
close	Closes policy storage

Exposed PMI services enable definition of any policy during the SOI provisioning process or activation of an already defined policy in the AM to oversee configured Runtime Infrastructure. Each policy activation causes booting of an adaptation loop managed by the AM.

6.5 Provisioning Manager

In accordance with previous assumptions, the provisioning procedure is defined and managed by the JBoss jBPM suite. The Procedure Definition Tool is based on jPDL designer, offering capabilities for specification of a set of activities for a particular adaptable SOI provisioning process. The example process shown in Figure 48 affects multiple elements of IT infrastructure and includes operations such as:

- VEI provisioning consisting of Solaris Lightweight Virtualization Containers installed on the basis of configuration template,
- Create clustered instances of an application server within VEI's execution nodes (LVC), including load balancer configuration and perform tuning according to assigned compute resources,
- Deploy application services including configuration of required services (JDBC data sources),
- Activate policies, which will supervise cluster's Runtime Infrastructure in accordance to QoS.

The individual steps that make up a provisioning procedure are called workflow transition, which is an operation on SOI Managed Elements through Manageability Endpoint components. Available "state" nodes allow definition of workflows used to handle provisioning operations initiated not only by administrator, but also in response to occurring events sent from the AM or MMP, such as “*application is overloaded - add a new server to the cluster*”.

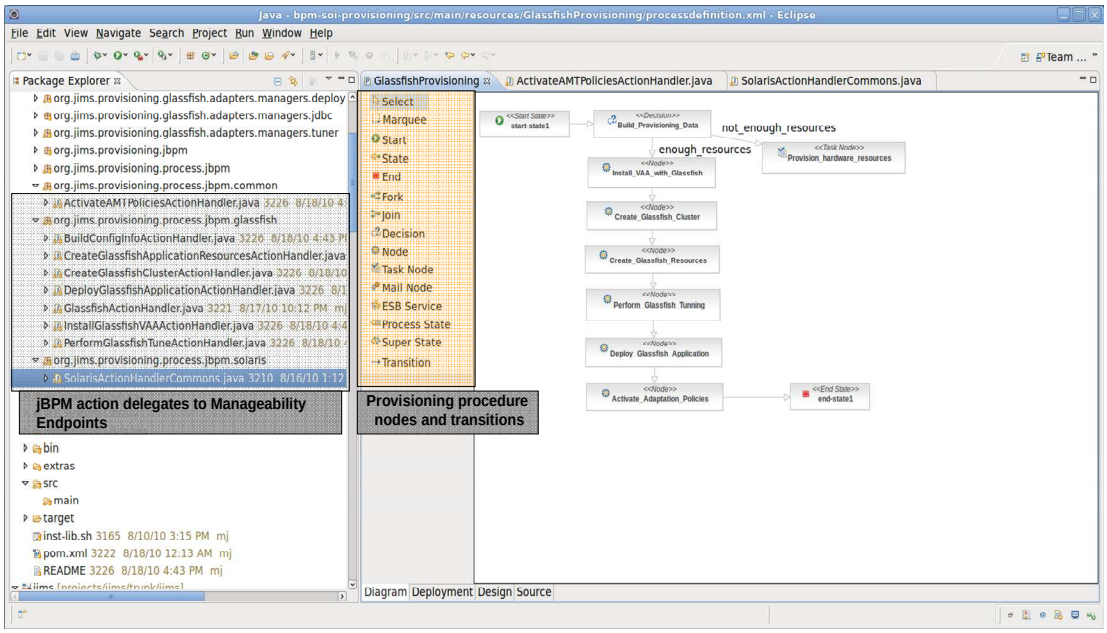


Figure 48. JBoss jPDL provisioning console

A defined provisioning procedure can be deployed to the jBPM Process Engine (Provisioning Engine) and initiated as a process instance with specified parameters such as target physical nodes, application components and LVC-AA names. The real advantage is the ability to define arbitrary provisioning procedures, create well-defined libraries and re-use them even in other procedures.

6.6 Construction of Lightweight Virtualization Containers Application Appliances

Solaris Containers provisioning is supported with regular administration commands (*zonecfg*, *zoneadm*) but, at the time of writing, the thesis is there is no support for LVC-AA[86]. Therefore, a ZFS snapshot mechanism is used to prepare LVC-AA on a reference server and place it in the repository that is made available to each physical server through Network File System (NFS) protocol. The prepared configuration templates contain SMF scripts (Listing 5) dedicated to handle specific runtime and are instrumented with the MMAs (Table 7) that contain only core functionality to initialize the components for exposition of SOI Managed Elements running within the provisioned Solaris container instance. Such construction ensures that, in case of modification of the Infrastructure Services, already defined LVC-AA configuration templates contain only MLet descriptors, which do not prescribe, and there is no necessity for an update process of these templates.

[86] Zone cloning is possible but only within local OS Kernel instance.

Particular middleware is instrumented with dedicated components that are initialized just after the Solaris Lightweight Container is installed on the target OS Kernel instance. This phase includes activities such as creation of administrative instances[87] and their initialization (Listing 6) is performed automatically without manual intervention of an administrator.

```
<mlet …
  name="Management:class=Solaris10Management" version="1.0">
</mlet>

<mlet code=
"org.jims.modules.provisioning.glassfish.mbeans.GlassfishNodeAgentManager"
  archive="jims-provisioning-glassfish-3.0.0.jar"
  name="Management:class=GlassfishNodeManagement" version="1.0">
</mlet>
```

Table 7. JMX MLet descriptor for Solaris Container LVC-AA with Glassfish configuration template.

6.7 Summary

The elaborated implementation of the Adaptable SOI Lightweight Virtualization Provisioning Platform provides Infrastructure Services for provisioning of Virtual Execution Infrastructures with lightweight virtualization platforms based on configuration templates with pre-configured Application Execution Environments for deploying application services. It supports monitoring (what the current resource consumption is by the Lightweight Virtualization Container and particular hosted workloads) and adaptive management (assign more compute resources like CPUs or memory portions). The implemented A-SOI-LV-PP is based on modular architecture and component design equipped with many advanced techniques of software engineering like design patterns, reflectivity and parameter and compositional adaptation. It also promotes flexibility and can be extended with components supporting arbitrary elements of the Operational Services Layer exposed as SOI Managed Elements which are provisioned and supervised by particular modules of the platform. The software components are implemented using JMX technology, which represent Solaris Lightweight Containers and application middleware providing sensors and effectors installed within the Monitoring and Management Platform (provide Manageability Endpoints required by the Step 2 of the process model). Communication connector components make it possible to couple JMX MBeans with the use any of the available RMI, HTTP or SNMP communication technologies. They are used by a decision subsystem that is an element of the Adaptation Manager responsible for realization of adaptation policies. The Policy Management Infrastructure provides GUI for adaptation policies definition (supporting Step 1 and Step 3). The Provisioning Manager allows creation of a

[87] Administrative instances are called node agents and control the life cycle of server instances.

hierarchy of self-management VEI instances based on LVC-AA, providing compute resources for middleware and application services according to the provisioning procedure (supporting Step 4 and Step 5).

The A-SOI-LV-PP fulfills the requirements, is coherent to the proposed architecture and is implemented with software components using low overhead instrumentation of Managed Resources of the OSL for gathering and delivering the monitoring and management services. A more exhaustive evaluation of the A-SOI-LV-PP calls for use cases related to SOI provisioning, proving that the constructed framework is operational and open to further enhancements. This is presented in the following chapter.

7 Experimental Studies

This chapter evaluates the developed Adaptable SOI Lightweight Virtualization Provisioning Platform using several complex use cases focusing on different aspects of provisioning and adaptive management of Service Oriented Infrastructure, but together forming one unified scenario. They present VEI and Glassfish middleware provisioning over Solaris Lightweight Virtualization Containers with adaptation loops implemented with control theory algorithms managing CPU availability for database workloads.

The Adaptable SOI Lightweight Virtualization Provisioning Platform provides a set of tools for provisioning and supervising the Runtime Infrastructure required by the application services. It exposes Infrastructure Services enabling realization of the adaptable SOI provisioning process model, which requires support for definition of arbitrary provisioning procedures and adaptation loops specification. This chapter presents a case study based on provisioning of J2EE Glassfish application server using Solaris LVC. The case study presents a provisioning process coherent with the model of adaptable SOI provisioning elaborated in Chapter 3 through the exploitation of the A-SOI-LV-PP.

7.1 Case Study Description

On behalf of the study, the author uses Java Pet Store 2.0[88], which is a sample application delivered by the Java BluePrint's program supervised by the former SUN Microsystem. The application uses Runtime Infrastructure provisioned according to the Isolated Tenancy model with a dedicated J2EE clustered Glassfish application server platform running over Solaris LVC provisioned with configuration templates (LVC-AA).

[88] The Java Pet Store 2.0 is the reference application for building Ajax web applications on Java Enterprise Edition 5 platform, using BluePrint's design patterns for programming Ajax with Java, JSF component libraries, Java Persistence API and others - http://java.sun.com/developer/releases/petstore/

7.1.1 Configuration of Testbed Hardware Infrastructure

The physical infrastructure is based on the Sun BP 1600 and Constellation Blade architectures, which provide resource pools equipped with Sun SPARC, AMD and Intel Xeon processors and running the Solaris 10 operating system (Figure 49). These physical servers are connected with 1GB Ethernet to assure high bandwidth for internal communication. In the case of Sun Blade 100 servers, they dual[89] Ethernet ports are used. The N1 Provisioning Server is installed on the Sun Fire v120 server, which also stores Solaris images[90] used when provisioning BP 1600 Blades. Sun Constellation provides servers with more powerful configuration when comparing to BP 1600 Blades (Table 8), thus, in the evaluation, the Constellation is used for hosting VEI provisioned over Solaris Lightweight Containers (Solaris Zones) and Sun Blade B100 servers hosting database workloads.

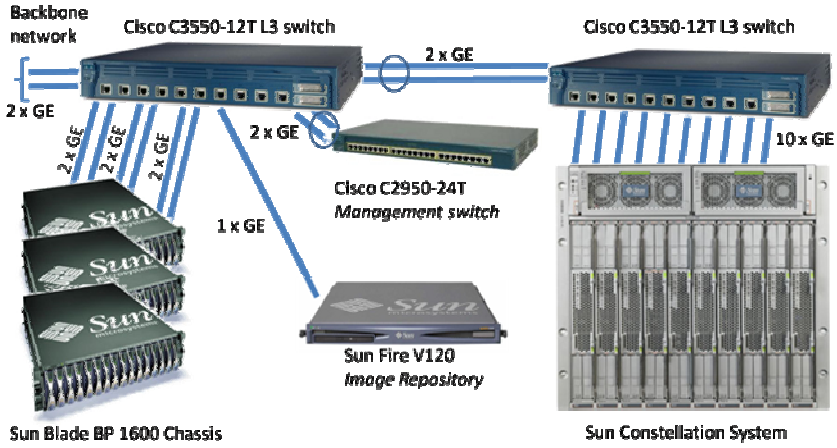


Figure 49. Testbed Hardware Infrastructure.

Table 8. Hardware specification.

Sun Blade BP 1600 Chassis	
5 x Sun Blade B100	1 GB RAM, 1 CPU SPARC II 650 MHz
Sun Constellation System	
2 x Sun Blade X6220 Server Module	8 GB RAM, 2 x Dual-Core AMD Opteron(tm) 2218 CPU
1 x Sun Blade X6250 Server	30 GB RAM, 2 x Intel(R) Xeon(R) E5345 @ 2.33GHz CPU
2 x Sun Blade X6250 Server	8 GB RAM, 2 x Intel(R) Xeon(R) 5160 @ 3.00GHz CPU

[89] Solaris IP Multipathing can be evaluated for providing links failover or extend available bandwith.

[90] These images are called *flash* archives, which are OS Kernel snapshots and can be replicated on any number of physical servers that have the same architecture as the reference server on which flash was obtained.

7.1.2 Provisioning Procedure Definition

The A-SOI-LV-PP enables definition of the provisioning procedure, which would provision *isolated tenant* Runtime Infrastructure based on Solaris LVC and Glassfish middleware. Preparing such infrastructure requires many complex activities, which besides placing appliance on target nodes, consists of many steps that should be performed in the correct order explained bellow. The provisioning process assumes that sufficient physical resources are available and there is an already configured database required by the application components.

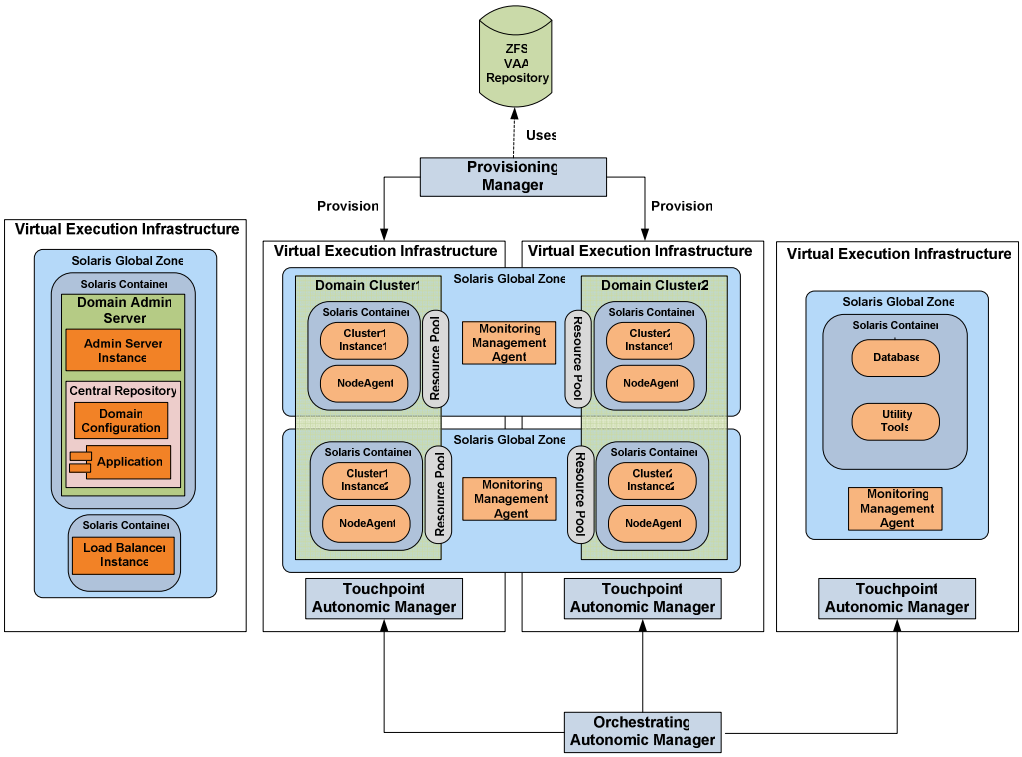


Figure 50. Runtime Infrastructure exploiting Solaris Containers and Glassfish middleware.

Provisioned VEI over Solaris Containers would deliver predictable levels of quality of service achieved due to the scheduling policies of Solaris Resource Manager. In the presented case study, CPU resource consumption within the OS Kernel (Solaris Global Zone) is managed with Dynamic Resource Pools with default Time Shared Scheduler (TSS) which are assigned to provisioned Container instances that manage the Glassfish environment. The TSS adjusts the priority of each thread based on the time a given thread consumes or spends waiting for CPU resource and time quantum, which is the limit of time for which a thread is assigned access to the CPU, depending on priority. When considering CPU resource consumption within a single Solaris Container like database and utility tools workloads, one must consider a situation when the utility tools monopolize available processor cycles and impact others workloads. This might lead to a situation where important database transactions suffer from insufficient CPU time to complete their work. It is desirable to have a scheduler which gives the ability to prioritize access to CPU resources based on the importance of the workload and can be fulfilled by the usage of the Fair Share Scheduler. The FSS provides precise control of CPU use, allowing optimal system CPU resource usage according to its share allocation, recent utilization

and CPU usage by the other threads within workloads. The only challenge is the control of the sufficient value of the *project.cpu-shares* resource control, which can be managed by the adaptation loop process.

Build Provisioning Input - Provisioning procedure instantiation requires specification of initial parameters. They define the physical nodes (OS Kernel instances) which are required by the provisioned LVC (Solaris Zone) instances hosting application server instances. Each provisioned LVC instance must be defined with the zone name, ZFS pool (zone file system), physical interfaces and Dynamic Resource Pools (number of CPUs) and OS schedulers (TSS used by LVC hosting Glassfish cluster and FSS by LVC with database workloads). Specified deployment components contain a bundle (Enterprise Application Archive) with application components and descriptors required by the Glassfish middleware. Configuration of a Glassfish cluster also requires localization and credentials of Domain Administration Server and Load Balancer.

Provision Solaris LVC Instances - Specified Solaris LVC-AA configuration templates containing pre-installed Glassfish middleware are used using during LVC provisioning with parameters specified in the previous step. The provisioned Solaris Container is booted and ready for other configuration activities related to the Glassfish environment. Procedures are invoked which configure Glassfish Node Agents (associate the agent instance with LVC configuration, prepare SMF script) and activate all services required by the DAS. Such a prepared set of LVC instances is Virtual Execution Infrastructure used by the Glassfish middleware, both constituting Runtime Infrastructure required by the application components.

Create Glassfish Cluster - Having already configured Glassfish infrastructure, there is the possibility to create a clustered domain, which requires definition of server instances and configuration of load-balancer through specification of the cluster name, member nodes (the server instances) and load-balancing policy.

Create Application Resources - Relational database schema creation performed by running an SQL script against the target database[91]. A created database must be configured as a JDBC resource in the application server middleware. If the application uses message brokers, like JMS for instance, configuration of destinations must also be defined [92].

Glassfish Tuning - Platform tuning is a very sophisticated task and demands expert knowledge about OS used, application server and database. This step includes setting many runtime configuration parameters that all depends on each other and, adjusted appropriately, increase the application performance. The following information is given as input such as the number of available processors or processor platform running on Sun Cool Threads, how much memory to be allocated for the JVM which is running a cluster's instance and a strategy for Garbage

[91] Database schema creation can be automatically performed when using persistence management frameworks.

[92] In the case of Point-To-Point and Publish-Subscribe scenarios, there are administrative objects called *Queues* and *Topics,* appropriately.

Collection. This step uses custom components implemented by the author, which interact with an internal tuner (Glassfish Performance Advisor - Figure 51) that exploits heuristics for finding the best to optimal configuration of particular cluster instances.

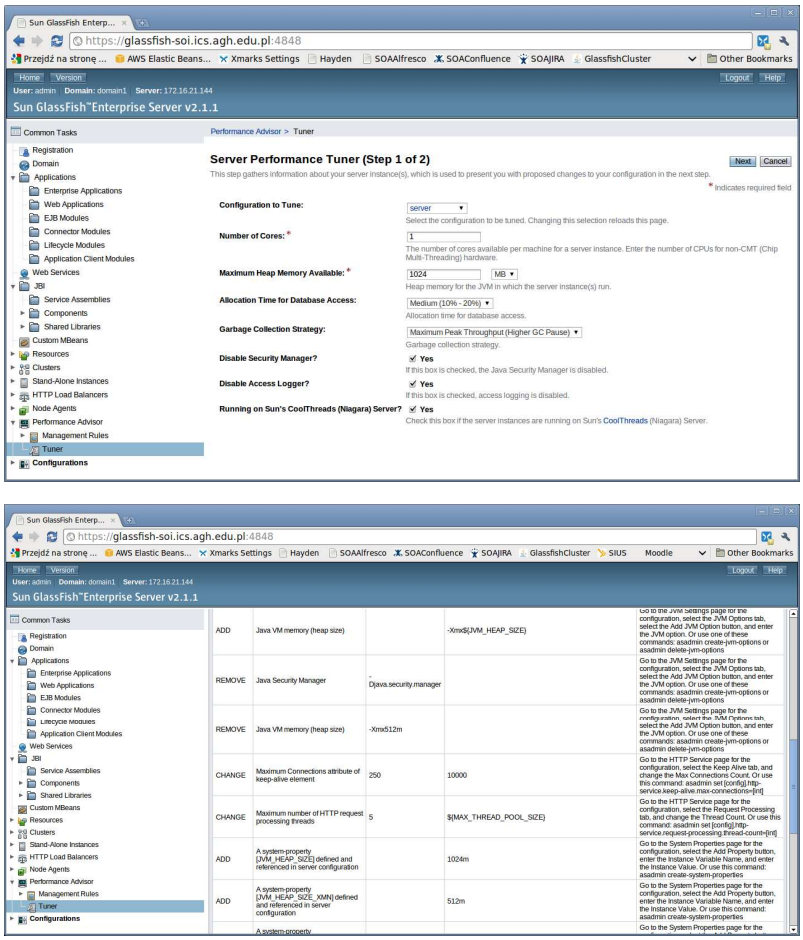


Figure 51. Glassfish Performance Advisor Console.

The drawback of using such an approach is the requirement for restarting each server instance when applying “tuned” configuration. Even in the case of clustered configuration, this can be problematic because the remaining instances must serve the increased number of requests, which can lead to instantaneous performance degradation.

Application Deployment - Application components installation within the clustered instances of the Glassfish. After the operation, application services are ready to serve a request, but the whole Runtime Infrastructure must be supervised to guarantee sufficient compute resources assigned to VEI and other infrastructure elements, like the database used by the application.

Activate Adaptation Policies - The control algorithm of the FSS scheduler for adaptive management of database workloads is expressed with Drools policy language and activated in the final phase of the discussed provisioning procedure. A constant monitoring process checks CPU consumption of database and utility tools workloads and influences the value of *project.cpu-shares* resource control value, ensuring the defined QoS goal.

7.2 Case Study Compliance with the Provisioning Process Model

7.2.1 Step 1 - *Self-Managing Requirement Specification*

The QoS goal of control is *assuring a constant allocation of the processor in conditions of variable and constant disturbance* i.e. database workload hosted by *project* within Solaris Container is guaranteed to use **70%** of CPU regardless of the number of other active projects' workload (disturbances).

7.2.2 Step 2 - *Providing Manageability Endpoints*

The result of the presented provisioning procedure is a very sophisticated platform designed according to the SOI paradigm running over Solaris Lightweight Virtualization Container with adaptation loops, ensuring guaranteed CPU reservation for database workloads.

The discussed provisioning activities require Infrastructure Services for managing the Glassfish and Solaris LVC platforms. Glassfish infrastructure provides a framework enabling centralized provisioning of a cluster of instances on multiple hosts and application deployments, support scalability, load-balancing, fail over protection and high availability (Figure 52).

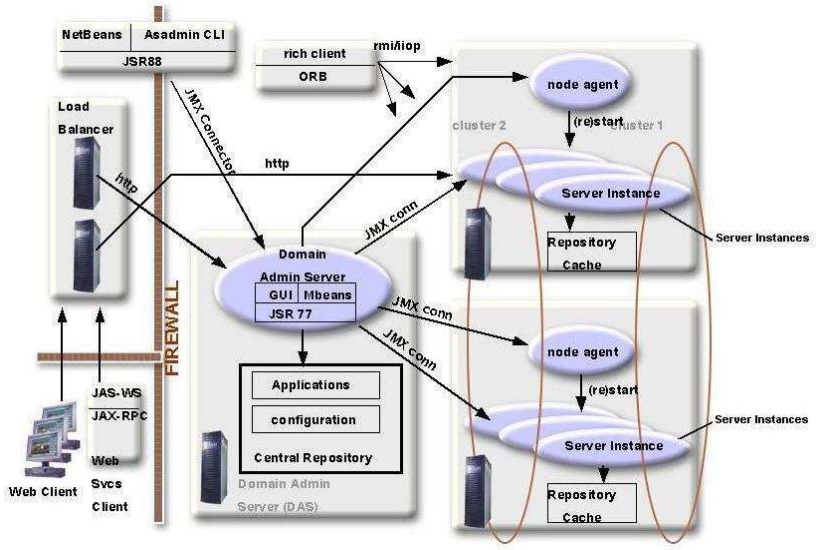


Figure 52. Glassfish architecture[93].

An administration console provides a web interface to manage and administer domains or replicate the enterprise environment on a separate domain running on separate machines, but the Glassfish environment is also coherent to the SOI paradigm, exposing Appserver Management Extensions (AMX) which are JMX components implementing API for the platform management.

[93] Source - http://wikis.sun.com/display/GlassFish/GlassFishV2Architecture

The A-SOI-LV-PP extensions for the Glassfish platform supports management of core elements like initial configuration node agents coherent with the provisioned LVC instance and automatic tuning process. The required Virtualized Services for Solaris LVC provisioning provided by the A-SOI-LV-PP[94] also support integration with the Glassfish environment like discovery of LVC with running instances and monitoring workloads, thus enabling orchestration of Solaris and Glassfish Runtime Infrastructure in a uniform way.

7.2.3 Step 3 - *Adaptation Policies Definition*

Adaptation policy must use some control algorithm expressed in the policy language (Figure 53), which is supported by the Policy Evaluation Module of the Adaptation Manager. As described earlier in the thesis, control algorithms can be used from disciplines such as fuzzy logic, control theory or heuristics[95]. Exploitation of a particular method mainly depends on the complexity of the adaptation process and, very often, hybrid controllers can be used, which use discussed techniques together for implementation of the adaptation loop. Because computer systems are characterized by their high complexity, using hybrid controllers ensures stable operation and, at the same time, allows the adaptation process to ensure effective defined QoS.

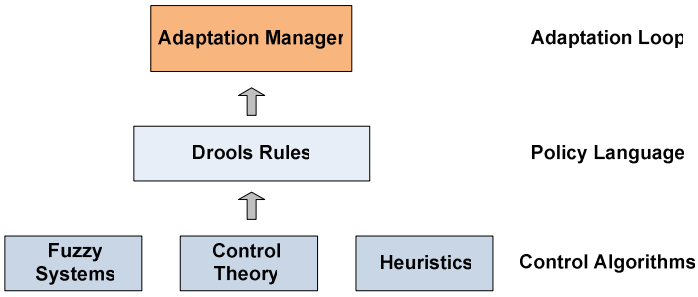


Figure 53. Overview of control algorithms for adaptation loops definition.

Control Theory Exploitation for Adaptive Management of Solaris LVC

In order to realize the adaptive management process the author proposes the use of control algorithms known from classical control theory i.e. *closed-loop* (Figure 54) and *open-loop* (Figure 55) controllers. When designing such controllers, properties like *stability* (managed computer system must be stable), *accuracy* (much the measured output i.e. goal differs from the reference input) and *settling time* (how quickly a system converges to its steady state value). Problems which arise during the usage of the feedback controller appeals to activities related to online measurements and design to ensure that a system is stable, accurate and has short settling times. The idea behind the open-loop controller is to avoid measuring output for adjusting the control input and using reference input (and sometimes the disturbance input) to calculate the value of control input to achieve the desired goal. However, a serious challenge behind the construction of such a controller is to determine an accurate model of the target system which,

[94] Elaborated in the Section 6.2.3

[95] These mechanims are described in more detail in Section 2.8

for instance, might be a result of experiments. The controller uses the model when determining control input value, thus it must be robust to react to changes occurring (e.g. disturbances such as execution of CPU intensive tasks) in its operating environment. Both controllers have similar high-level architecture, with some differences [72]:

- *Control error*: is the difference between the reference input and the measured output.
- *Control input*: is a parameter that affects the behavior of the target system and can be adjusted dynamically.
- *Controller*: determines the setting of the control input needed to achieve the reference input. The controller computes values of the control input based on current and past values of control error.
- *Disturbance*: is any change that affects the way in which the control input influences the measured output.
- *Measured output*: is a measurable characteristic of the target system, such as CPU utilization and response time.
- *Reference input*: is the desired value of the measured outputs (or transformations of them), such as CPU utilization, which should be 55%. Sometimes, the reference input is referred to as desired output or the setpoint.
- *Target system*: is the computing system to be controlled.

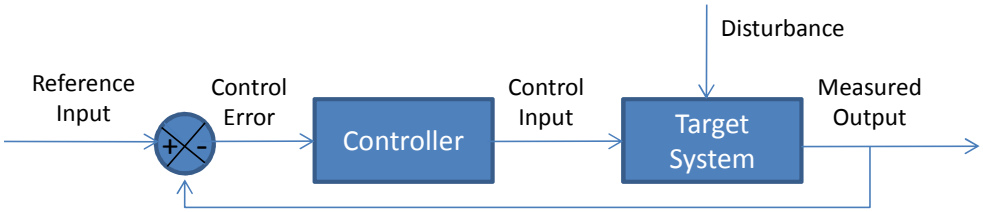


Figure 54. Architecture of closed-loop controller.

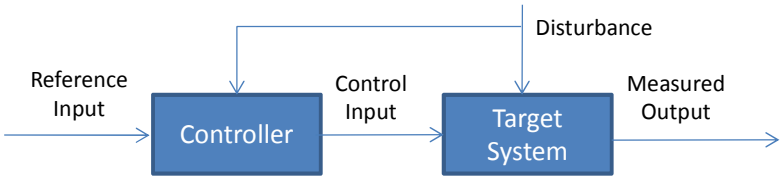


Figure 55. Architecture of open-loop controller.

The Adaptation Manager can use these algorithms for adaptive workload management of Solaris Containers. Both algorithms use the same mechanism for controlling the resource allocation managed by the FSS to achieve desired CPU allocation to the workload, but differ in share the calculation procedure. It is assumed that the manager performs a control algorithm at discrete time $t = 1, 2,\ldots$ and one tick of this clock is equal to T_O, which is the interval at which control loops are running.

Closed-loop workload manager (Figure 56) does not use the FSS model and the whole system is treated as a black box using the closed-loop controller. The controller uses current CPU usage

value to calculate the control error and adjusts the reference input (requested CPU usage) by changing shares (control signals). In this case, the CPU consumption U_w^t at time *t* is measured and used by the controller for calculation of a desired share's value.

Open-loop workload manager (Figure 57) exploits the FSS model. The number of active containers or projects with their current share (S_i) assignment is monitored and the model (equation 4) is used for calculation of relative entitlement E_w of workload W and suitable S_i adjustment.

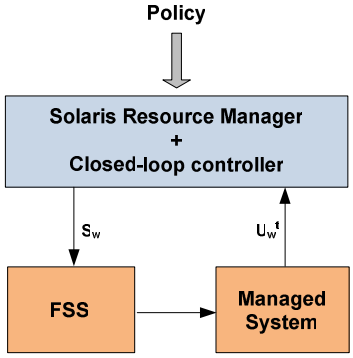


Figure 56. Solaris Resource Manager with closed-loop controller.

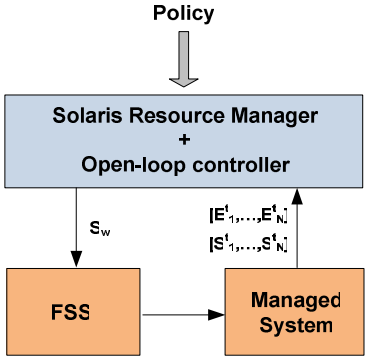


Figure 57. Solaris Resource Manager with open-loop controller.

Control Algorithm of the Closed-loop Controller

A sample controller algorithm could use the *Proportional (P) or Proportional-Integral (PI)* regulators expressed by equations (1) and (2) respectively, where: (i) U_w - required usage of CPU by workload W_w, (ii) U_w^t - usage observed at time t by workload W_w, (iii) S_w^t - number of shares set at time t for workload W_w, (iiii) K_p - proportional coefficient, K_i - integral coefficient.

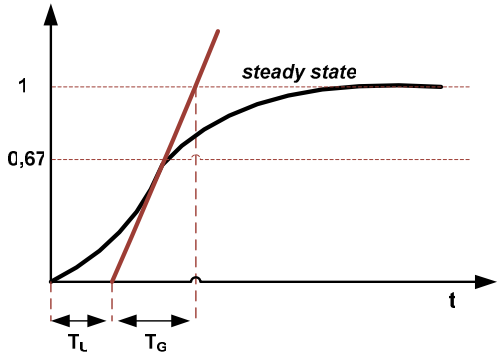
$$\mathbf{S_w^{t+1}} = \mathbf{S_w^t} + K_p * \mathbf{e(t)}, \text{ where } \mathbf{e(t)} = U_w^t - U_w \qquad (1)$$

$$\mathbf{S_w^{t+1}} = \mathbf{S_w^t} + K_p * \mathbf{e(t)} + K_i \sum_i^t \mathbf{e(t)} \qquad (2)$$

PI controllers allow the elimination of extremely low frequency disturbance, which translates into a zero offset, determined hard to achieve in the P-type controllers. Coefficients of the *Proportional* regulator can be specified manually. In the case of the *Proportional-Integral*, it can be calculated using analysis of the step-response (Ziegler-Nichols) method, which consists of the time evolution of its outputs when its control inputs are signaled at a specified time and stays switched on indefinitely (Table 9) [78]. The figure shows the engineering method of the selection of control parameters in the neighborhood of the point of work. It is assumed that the target system (Solaris Resource Manager) has the characteristics of an inertial object for which reaching a steady state is accomplished after some delay. In fact, when switching shares for the FSS, the target CPU consumption is achieved after 60 seconds.

Table 9. Identification of coefficients for P and PI regulators using the Ziegler-Nichols method.

	K_P	K_I
P	$\frac{T_G}{T_U + T_O}$	-
PI	$\frac{0.9T_G}{T_U + T_O/2} - \frac{0.135T_OT_G}{(T_U + T_O/2)\text{^}2}$	$\frac{0.27T_OT_G}{Kp(T_U + T_O/2)\text{^}2}$



T_G - Time constant after which the system reaches a certain percentage of the steady state (this follows from specific approximation, in this case it is equal to 67%).

T_U - Delay, the time that is needed to transport the signal, the analyzed scenario, it can be assumed zero.

Selection of the appropriate controller settings for the coefficients K_i and K_p can be related to carrying out manual testing on the system, where finding best values is done based on the algorithm that calculates the sum of squared error controls, as presented in the equation (3).

$$\mathbf{I_e} = \sum_{k=0}^{M} \mathbf{e(kT_O)}^2 \qquad (3)$$

Control Algorithm of the Open-loop Controller with the FSS Model

The system administrator expresses the importance of each workload by the number of shares, which are not the same as CPU percentages; shares define the relative importance of a given active workload to other active workloads. If (i) S_w – shares assigned to workload W, (ii) N – number of active workloads, (iii) S_i – shares assigned to active workload i={1,..,N}, then the relative entitlement E_w of workload W can be expressed with the following equation:

$$\mathbf{E_w} = S_w / \sum_i^N S_i \qquad (4)$$

The open-loop regulator is based on the FSS model and takes into account the fact that the scheduler considers only active workloads and, if a given workload is not CPU-bound, then the

remaining CPU portions might be consumed by other workloads. Let' us assume that: (i) number N_w of workloads ≥ 2, (ii) number of active workload is changing at time t according to activity state vector $A^t = [A^t_1, \ldots, A^t_{Nw}]$ where $A^t_i = 0$ if W_i is not active and $A^t_i = 1$ if W_i is active, i = 1,...,N_w, (iii) each workload is CPU bound and has allocated shares S_i. Following the mathematical transformations of equation (4) we obtain equation (5) for shares S_w to be set for workload W_w:

$$\mathbf{S^t_w} = (U_w \sum^{Nw}_{i \neq s} S_i * A^t_i) / (1-U_w) \qquad (5)$$

Where, $\sum^{Nw}_{i \neq s} S_i * A^t_i$ is the disturbance monitored by the manager and is equal to the sum of all active workload shares, excluding workload W_w at time interval t.

The above control algorithms can be defined with policies using the PMI based on Drools language (Listing 8) and ready for evaluation for adaptive management of database workload according to a specified QoS goal.

7.2.4 Step 4 – *Provisioning Procedure Initiation*

The required input information for the proper implementation of the procedure has already been described (*Build Provisioning Input* activity) and, in accordance with previously agreed objectives served, the following parameters are specified: the number of instances equal to 2 for the newly created cluster nodes along with the location of physical servers where the LVC instances are provisioned. The required parameters are also the names of the Resource Pools used by the LVC with already assigned CPUs, path to the LVC-AA containing Glassfish configuration template, the Petstore application distribution and the adaptation policy.

7.2.5 Step 5 – *Provisioning Procedure Activation*

Once all the required information is specified, the provisioning procedure described in Section 7.1.2 can be successfully invoked, resulting in the provisioning of Runtime Infrastructure presented in Figure 50. The Petstore application services use database running within the LVC that must be supervised by the Adaptation Manager to ensure QoS metric which uses Infrastructure Services exposed by the Monitoring and Management Platform. The whole process of adaptive management is described in the Drools policies and evaluated in detail in the next Section.

7.3 Experiment Design

There are two kinds of disturbances: *variable* and *constant* (Figure 58). The *variable* disturbance is generated with a period of 90 seconds, whereas *constant* disturbance was activated after the controlled workload reached a steady state (considering the fact it was the only CPU-bound workload, it reached close to 100% CPU usage). The *nspin* application, provided by the Solaris Resource Manager Benchmark, tools was used.

Experiment 1 - *Closed Loop controller for managing workload under constant disturbance*

Experiment 2 - *Closed Loop controller for managing workload under variable disturbance*

Experiment 3 - *Open Loop controller for managing workload under constant disturbance*

Experiment 4 - *Open Loop controller for managing workload under variable disturbance*

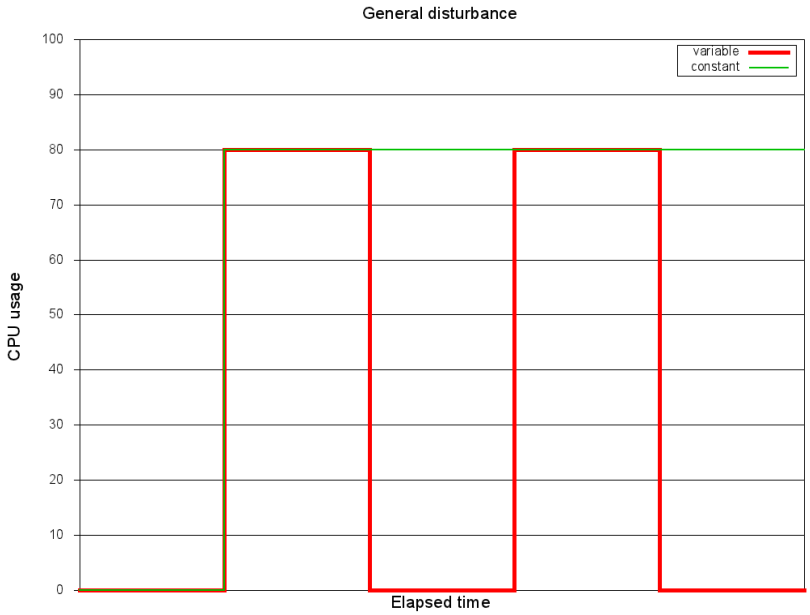


Figure 58. Disturbance generated during tests.

Each experiment was conducted in an identical scenario. In Phase I, database workload is activated to simulate transactions. In Phase II, after a certain time when it reaches a certain CPU usage, a disturbance is introduced which causes a decrease in the consumption of the CPU below the established goal (in our case 70%). Then in the Phase III the regulator responsible is activated for the adaptive management of settings of the FSS managing the availability of CPU resources for a particular database workload according to the specified goal.

7.4 Experiment Results

The following tests were conducted on the BP 1600 blade platform. During the tests, the author encountered many challenges related to the characteristics of the FSS and the Solaris OS that are not included in Section 7.2.3. Therefore, at the outset assumptions are presented which ensure that activities related to the adaptive management process maintain a system in a stable state.

7.4.1 Preliminary Assumptions

The selection of an appropriate period that takes into account the inertion of the FSS is crucial and affects the controller interval T_O. This is especially important he in case of variable disturbances when FSS must adapt itself to satisfy CPU consumption according to assigned shares. In the scenarios analyzed, the controller interval was equal to T_O=20 seconds, but thread responsible for acquiring monitoring data was activated each ΔT = 5 seconds and stored metrics in vector, whose values were then averaged for the control error calculation. In addition, the initial setup of the FSS within Solaris means that during the evaluation of control algorithms, the CPU shares value are calculated as fractional numbers and causes instability of regulation

by precision errors and, therefore, is made more accurate by multiplying the value by the constant 100.

It is important to emphasize that FSS only limits the CPU usage if there is competition for the CPU; otherwise CPU shares are never wasted and a given active workload and can use 100% of CPU resources. When regulating CPU shares for the project, we should check if the OS is fully utilized, because only then does FSS schedule the threads according to assigned CPU shares to specific projects; otherwise, CPU resources that are not used are assigned to other workloads. This fact influences the implementation of the controller and it is desirable to add a simple heuristic that checks if the controlled project contains CPU-bound workloads (Listing 9).

Figure 59 depicts the case when a single CPU-bound process is started. Unfortunately, as can be seen, the output from *prstat* increases gradually over a period of more than one minute because *prstat* shows incremental CPU usage for the workload and the measured value does not reach 100 %. It is necessary to point out than *prstat* is the only utility which reports CPU utilization per project. Such inconsistency leads to instability of the regulator and requires a solution to check immediate, total CPU utilization. For that reason, implementation of the rule that checks if the system is fully utilized uses SOI Virtualized Services, which exploit the *kernel statistics* (KSTAT) monitoring interface.

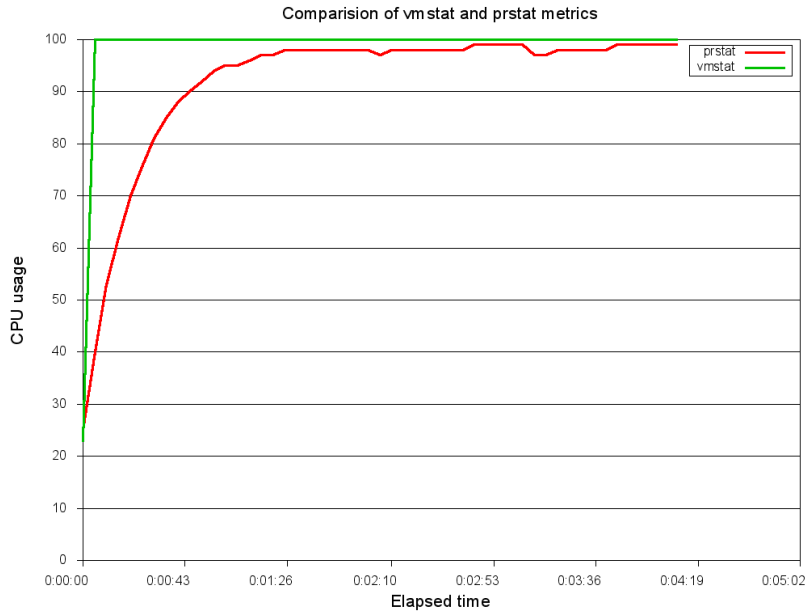


Figure 59. Comparison of vmstat and prstat monitoring tools.

It may happen that under a fully utilized system some application threads are not properly pre-empted. Such a scenario occurred in the test depicted in Figure 60. In this case, the OS did not schedule the MMA monitoring thread responsible for acquiring monitoring data properly and, if the data are not effectively acquired, the closed-loop controller calculates the shares as zero. Such a situation might be observed when the OS is saturated, i.e. the running queue length is greater than the number of CPUs. As a workaround, a simple rule is used, which checks if the size of the vector which stores data about CPU usage of controlled project and should be equal to the number of samples that is provided by $N = T_O/\Delta T$. This rule is evaluated at each

regulator interval and, if the result is negative, that operation is cancelled, but only for the current interval.

The consumption of the CPU due to monitoring and control activity performed by the MMA depicted in Figure 61 shows that this overhead is not substantial and does not adversely affect the performance of specific components of the A-SOI-LV-PP responsible for carrying out activities relating to the adaptive management process.

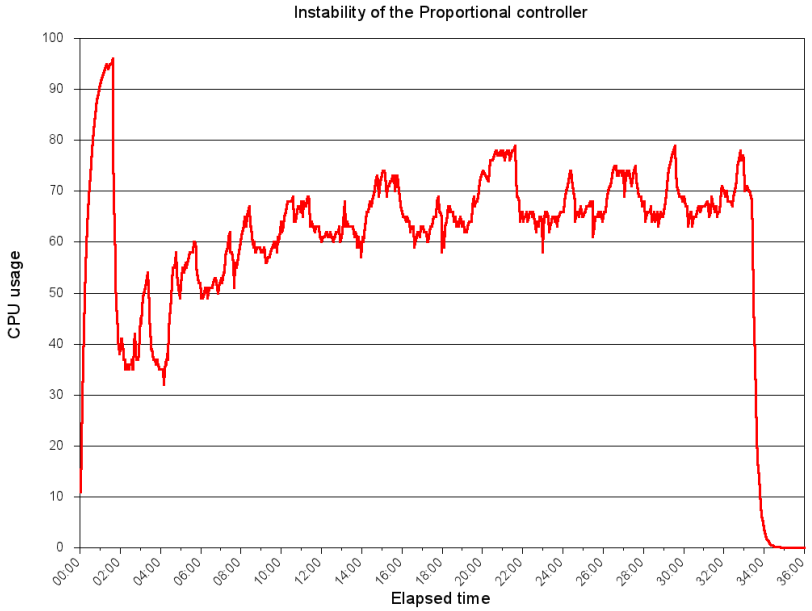


Figure 60. Instability of the Proportional regulator caused by irregular scheduling of the monitoring thread.

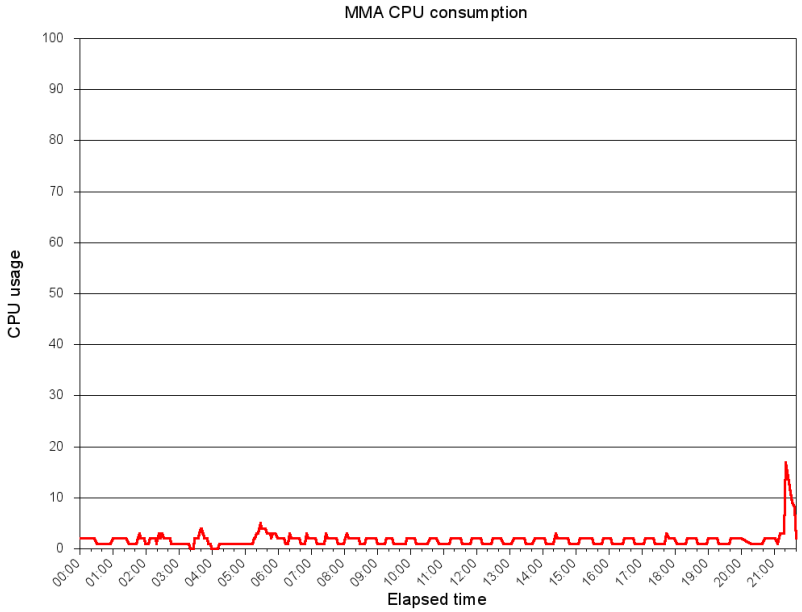


Figure 61. CPU usage of the Monitoring and Management agent.

The following Sections report results of adaptive management of database workloads running over Solaris Lightweight Virtualization Containers exploiting the closed and open loop controllers equipped with the described heuristics.

7.4.2 Experiment 1

Figure 62 presents the case where only one database CPU-bound workload is started in the selected project, at the beginning. After a few seconds, when CPU usage increases to 95%, two other CPU-bound disturbance workloads are started in other projects, which results in a drop of CPU usage of the selected database workload project. Then, after several seconds, the P controller is turned on and changes the share allocation to the controlled project, stabilizing CPU usage at 70%. The experiment was repeated for different values of the K_p coefficient and as shown in Figure 63. The best results were achieved for K_p=7, that also means that the goal was reached during first control loop iteration (Table 10).

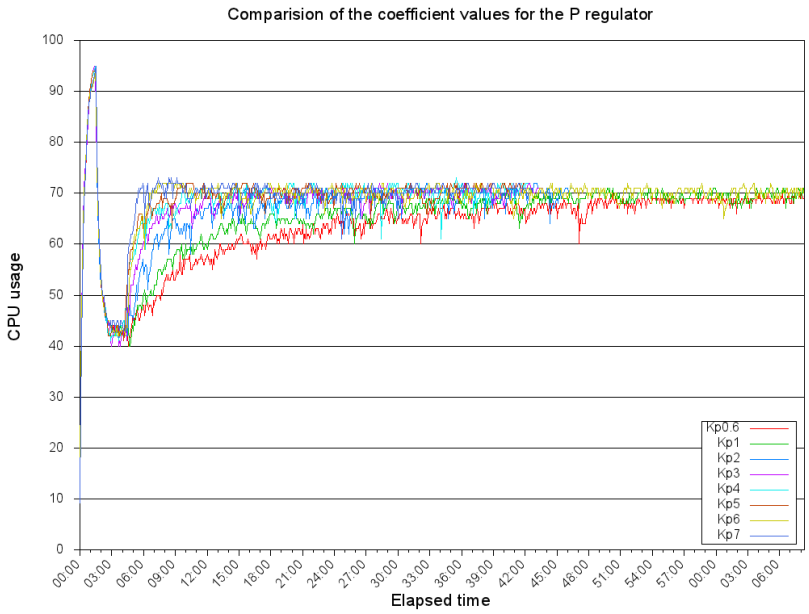


Figure 62. Proportional regulator with constant disturbance.

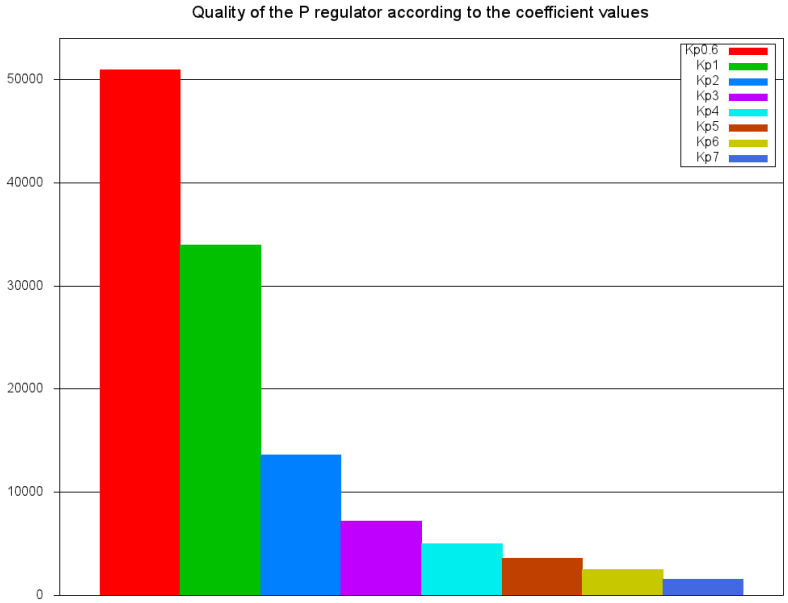


Figure 63. Quality of the Proportional regulator with constant disturbance.

Table 10. Comparison of settling times of the Proportional regulator.

Proportional Coefficient	0.6	1	2	3	4	5	6	7
Settling time [min]	53,4	40	15,3	9,2	6,7	4,8	3,7	3

7.4.3 Experiment 2

When considering the case with a *variable* disturbance, we should take into account many important factors that influence definition of control policy that, besides the described Proportional or Proportional-Integral regulators, must be extended with some heuristics. In a situation where there is only one CPU-bound workload, and the whole CPU is assigned to that workload, it makes no sense to run the control loop. The implemented heuristic based on the FSS model, which assigns CPU according to share value, considering other active workloads (Listing 9) returns a list of workloads and the controller may check whether the list contains a specific workload. Another factor in the case of a variable disturbance is a situation when the CPU share for a controlled workload is not properly calculated. The explanation is very simple; namely when monitoring data are stored in a vector, some of them are acquired at a time when only controlled database workload is active and assigned nearly the entire CPU. Such data dilute the mean calculated value and, if the value is bigger than the goal, shares are decreased instead of increased, leading to instability of the managed workload. The solution to the problem is to use a decaying average, similar to the Jacobson [108] algorithm used in the TCP/IP Protocol to smooth measured values (equation 6), where φ – Jacobson coefficient. Applying these rules provides satisfactory results as to the quality of applied policies (Figure 64) and strictly depends on the value of the Jacobson coefficient (Figure 65).

$$\mathbf{NU^{t}_{w}} = \varphi * \mathbf{U^{t-1}_{w}} + (1- \varphi) * \mathbf{U^{t}_{w}} \qquad (6)$$

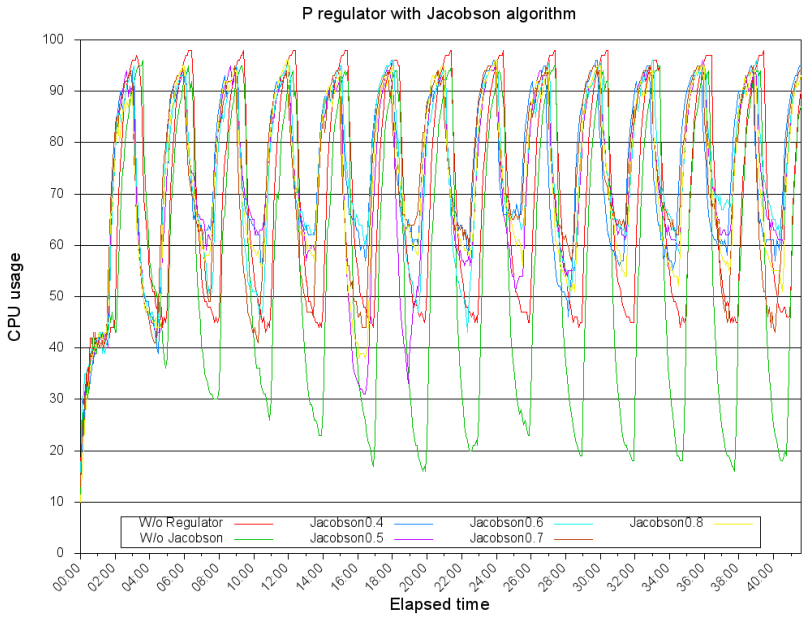


Figure 64. Proportional regulator with variable disturbance.

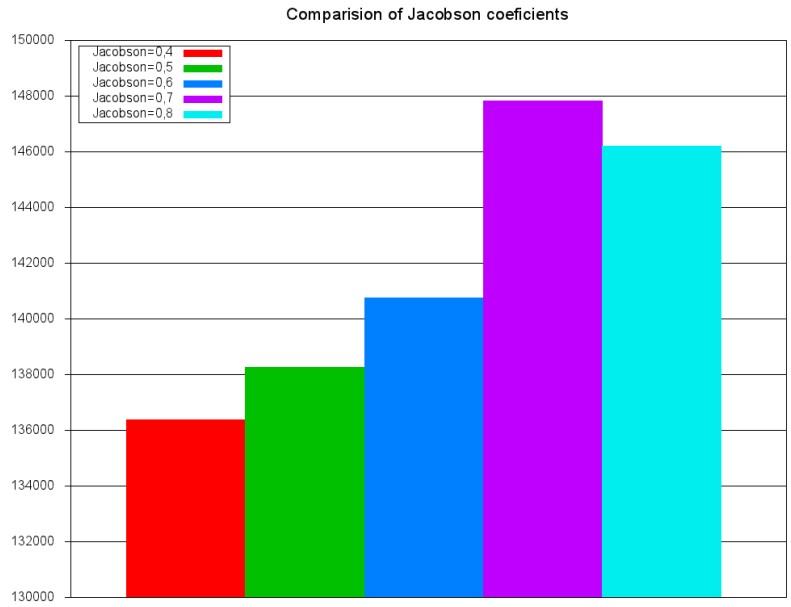


Figure 65. Quality of the Proportional regulator with variable disturbance.

Figure 66 shows the engineering method of selection of control parameters in the neighborhood of the point of work (70% of CPU Usage). In *Phase I*, we are dealing with the arrival of a steady state open-loop system, where system is stabilizes by the consumption of the processor by the workload as specified by the QoS goal. In *Phase II*, we introduce the disturbance resulting in the dropping of the CPU usage to a new value established. In the *third phase*, identification of the system dynamics is performed by increasing the share's value as specified by the FSS model to reach the goal. Using acquired data, an input matrix is constructed and, using the Matlab toolbox (Figure 67), we can calculate the controller settings based on the equations in Table 9 (T_G=21).

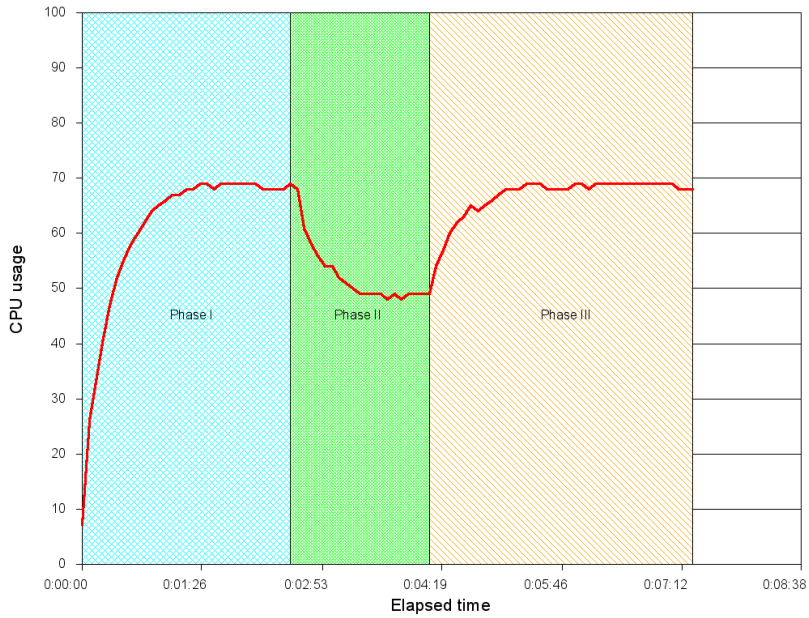


Figure 66. Experiment using open-loop controller performed by a system administrator.

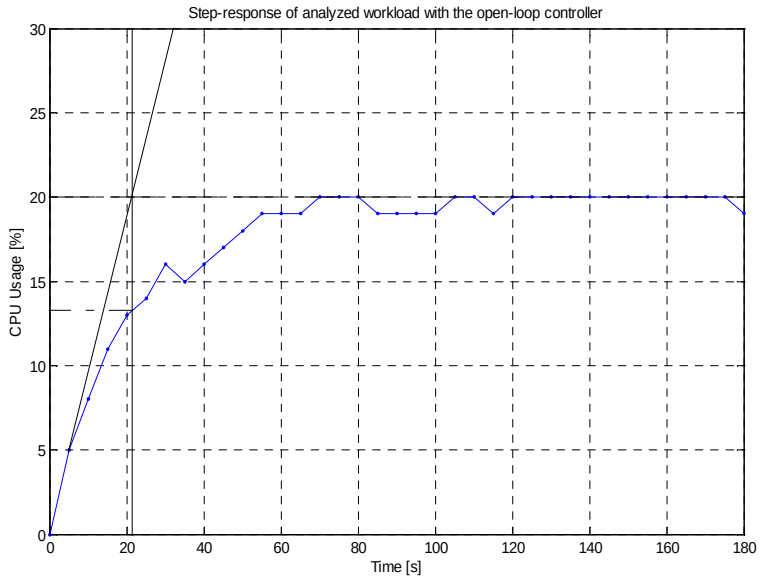


Figure 67. Matlab toolbox chart - identification of system based on analysis of step-response method.

These results justify the application of P and PI regulators for CPU usage control of the selected project, under variable disturbances as presented in Figure 68. It is evident that the smoothing operation performed by Jacobson algorithms significantly improves control quality and stabilizes the system. The P and PI regulators provide a similar quality of control (Figure 69), as can be noticed after analyzing of the chart presenting the quality of the controllers, though it seems that the behavior of the PI is as expected according to the elimination of effects of arising disturbances in a more efficient manner.

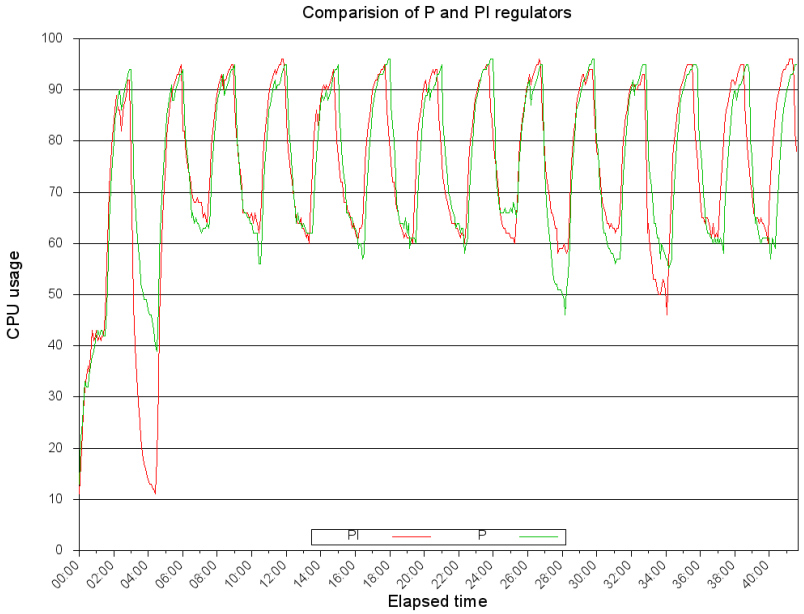


Figure 68. Proportional and Proportional-Integral regulators with variable disturbance.

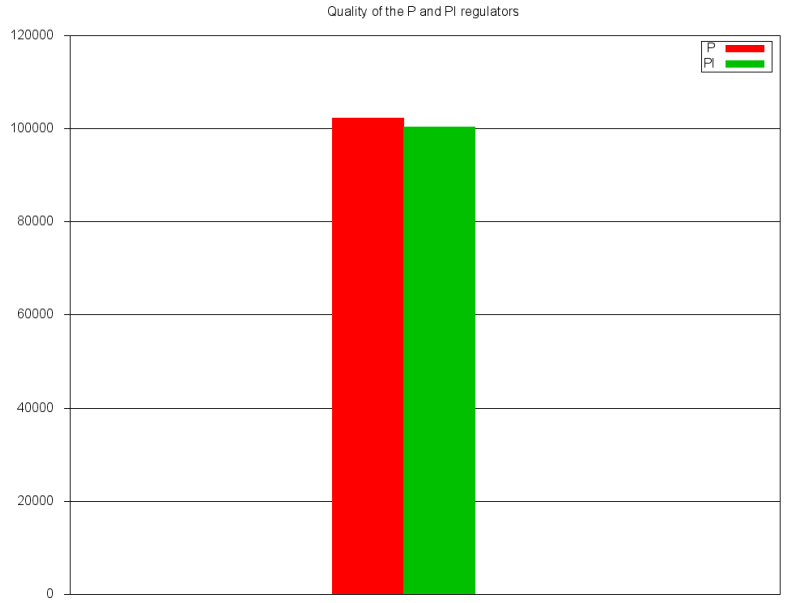


Figure 69. Quality of Proportional and Proportional-Integral regulators with variable disturbance.

7.4.4 Experiment 3

Adaptive management exploiting open system control using a model of the FSS enables the achievement of the QoS goal already during the first iteration. This approach does not require the control parameters calculation, as was the case for proportional control, maintains adequate quality control, although the consumption of CPU resources is greater than expected at the point of view disturbance workloads, which might be inappropriate (Figure 70).

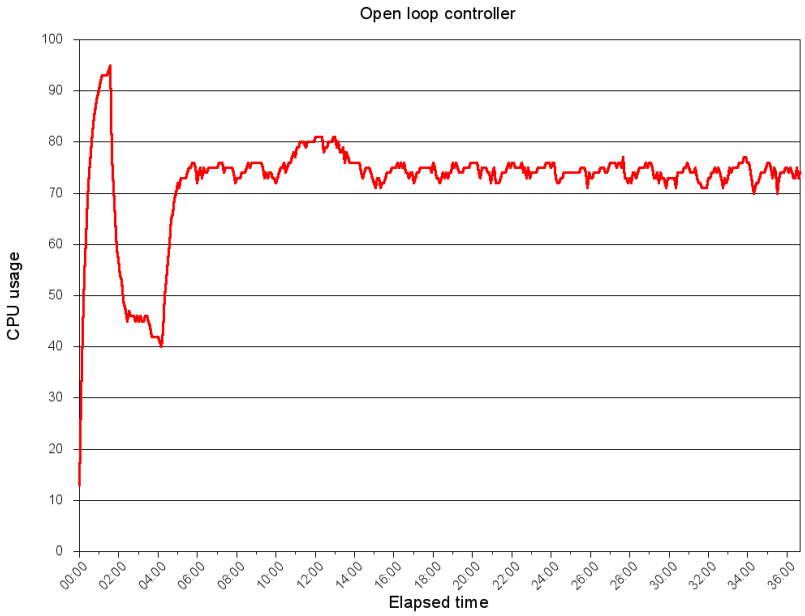


Figure 70. Open loop controller with constant disturbance.

7.4.5 Experiment 4

Like the previous scenario, the use of control in an open system under conditions of variable interference ensures the maintenance of the system at a preset level. As in Experiment 2, the controller must be fed with previously reported heuristics, including exploitation of the Jacobson algorithm.

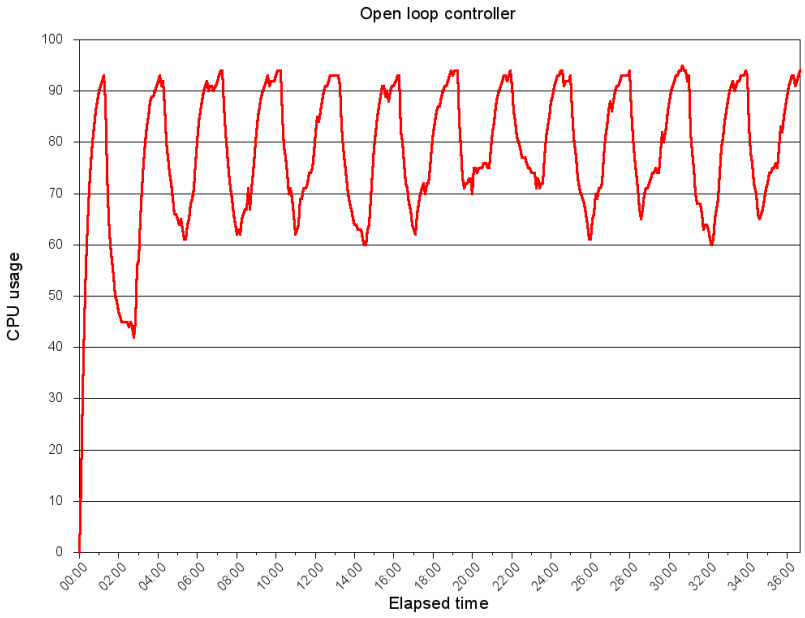


Figure 71. Open loop controller with variable disturbance.

7.4.6 Summary

The presented concept of exploitation of the traditional control theory for the realization of control algorithms proved to be an effective solution for ensuring the defined QoS goal. However, defining adaptation policies, in addition to the algorithm used, also include a set of heuristics to ensure the stability of the managed system. Heuristics are the result of the complexity of a managed infrastructure based on lightweight containers and must be exploited to ensure that the proposed adaptation process is an efficient solution for maintaining required QoS.

7.5 Summary

The chapter presented the practical exploitation of the model of adaptable SOI provisioning with LVC proposed in the thesis in the example of the use of the Solaris LVC and Glassfish platform, with particular attention to guaranteeing the availability of computing resources for the container in which the database is running.

In the course of realization of the provisioning procedure, individual A-SOI-LV-PP's modules were used. The procedure is modeled using a graphical tool that orchestrates various Infrastructure Services for the provisioning of Solaris 10 LVC with a configured Glassfish clustered platform delivering required Runtime Infrastructure by the Petstore application services.

An important element of the provisioning process was to provide a specified level of the CPU availability for the LVC running the database used by clustered Glassfish server instances. The adaptive management process used hybrid controller based on Proportional and Proportional-Integral algorithms enriched with heuristics, thereby ensuring stable operation of the Fair Share Scheduler that governs the availability of CPU resources. These algorithms were described using a set of policies defined at a high level of generality in a user-friendly way. To this end, the author suggested the use of the Drools rule engine which is supported by the Adaptation Manager module.

Of course, during the evaluation, some shortcomings were also encountered that are discussed in the next chapter, along with some ideas that may find support in the new version of the platform.

8 Conclusions

This chapter summarizes the dissertation, verifies the thesis statement and analyses the implementation of the A-SOI-LV-PP with a focus on limitations and possible improvements which could be considered in future work on the platform created.

The dissertation presented elaborates an approach to provide Runtime Infrastructure required to application services using the lightweight virtualization technology and exposed as Service Oriented Infrastructure. The proposed and implemented architecture of the Adaptable SOI Lightweight Virtualization Provisioning Platform provides an original solution to promoting a component approach enhanced with advanced software engineering techniques. The work presented in the dissertation confirms the validity of the assumptions adopted at the outset of the effectiveness of adaptable provisioning of SOI with lightweight virtualization based on a flexible platform supporting definition and creation of the Runtime Infrastructure (Virtual Execution Infrastructure and Application Execution Environments) required by application services and its continuous adaptive management.

8.1 Thesis Verification and Confirmation

The implemented A-SOI-LV-PP fully supports the model of adaptable SOI provisioning with lightweight virtualization. The model assumes the existence of specific components that provide SOI Infrastructure Services, enabling Runtime Infrastructure provisioning using LVC-AA configuration templates and then monitoring and management using policies defined by the input of a particular provisioning process.

The proposed approach fully exploits the potential of the contemporary lightweight virtualization technology exposed with software components implemented with the Java platform based on JMX technology. The exploited Solaris lightweight virtualization platform enables flexible resource management with access to a pool of resources rather than control of fixed resource as in heavyweight virtualization technologies. The Solaris Containers reduce the need for the system administrator to manage multiple operating system images with running OS instances. The compute resources assigned to LVC have the ability to access the pool of CPU or

memory resources, which can be resized dynamically during adaptive management activities. Moreover, these pools can be shared between many LVC instances where idle resources can be consumed by these LVCs which have such resource demands.

A significant element of this research is a methodology of exposition of infrastructure resources in the form of services and their management in order to realize adaptation policies. An important element is also the methods for defining adaptation policies consisting of set of rules defined with a means that are friendly to the administrator. In particular, there is a proposal for using rules engines such as Drools or Jess, which support scalable and advanced processing with the usage of the Rete algorithm. Special emphasis is put on analysis, practical usage and evaluation of classic control theory using steering in open-systems, based on the system model, and closed-systems considering *Proportional* (P) and *Proportional-Integral* (PI) regulators. Particular aspects connected with proper selection of regulators' coefficients (analysis of *step-response* method and *Jacobson* algorithm) are also evaluated.

The implemented environment supports the adaptive computing requirements, is general in nature and can be applied to different levels of a distributed computing system, starting from physical resources, through virtualized operating systems, up to application servers and software components. The architecture of the A-SOI-LV-PP is based on a modular structure promoting flexibility and the ability to be expanded to various implementations of lightweight virtualization in the future. The usability of the implemented environment is proven by the experiments performed on unique SUN BP 1600 and Constellation blades hardware architecture.

In conclusion, the author proved the thesis statement by using the concept of the defined model of adaptable SOI provisioning with lightweight virtualization and specific functional and non-functional requirements. The implementation process proved that it is possible to develop the A-SOI-LV-PP using the techniques of component-based design, enriched with compositional adaptation and computational reflection that deliver the Infrastructure Services used by provisioning procedures and adaptation loops realizing particular QoS enforcements.

8.2 Novelty of the Thesis

The proposed solution for the visibility of computing resources over the lightweight virtualization technologies using a Service Oriented Infrastructure introduces a new approach to the concept of managing IT infrastructure. The architecture's model of the Infrastructure Management Framework defined by the SOI Reference Model has been mapped in the proposed solution by the author based on reflective middleware and adaptation techniques used during the implementation of the components that make up the specific modules of the A-SOI-LV-PP.

The concept of the system proposed in this thesis was to complement opportunities offered by the use of lightweight virtualization functionality of provisioning the Runtime Infrastructure, together with the automation of tasks related to the initial tuning of its specific elements and the configuration steps and procedures of adaptable steering. An important feature is the possibility

to define any procedure for creating a VEI by using lightweight containers and configuration templates and policy based-management with resource allocation algorithms that use control theory with heuristics to ensure the specified QoS.

The existing practical applications of virtualization technology are dominated by the heavy-weight virtualization techniques. However, it is the author's opinion that this situation will change in the direction of solutions to lightweight virtualization because of the possibility of solving the numerous problems associated with the scalability of the existing arrangements for allocating resources. Analyzing the existing arrangements set out in Section 2.3, the solution proposed in the thesis is one of the most interesting and most useful examples of possible applications of lightweight virtualization.

8.3 Critical Evaluation

It is certainly possible with the implemented system to perform work related to optimization and streamlining of its operations. In the elaborated work, not enough emphasis has been placed on ensuring an adequate level of security. It is therefore necessary to complement the funcionality related to user authentication mechanisms and authorization of their actions.

The Provisioning Module does not have support for an early validation and simulation that the input is correct and whether the initial setup of the physical resources allows a smooth launch of the procedure. Currently, the administrator is obliged to clearly define the target for physical resources and so it is desirable that automatic calculation of the resources from the pool should be used. The solution built is also known for single points of failure, for example, immobilization of Orchestrating or Touchpoints Adaptation Managers will not coordinate in the performance of adaptive strategies.

8.4 Opportunities for Further Development

The system’s capabilities evaluation proposed by the test scenarios performed used a highly advanced stage of hardware platform. Although those techniques of adaptable steering exploited the control theory, one can imagine more advanced scenarios in which an appropriate group of management policies uses a hybrid controller based on fuzzy logic or queuing networks. Although the A-SOI-LV-PP for accounting services has not been exploited in the evaluation scenarios, they are a good base for the application of forecasting methods after which long-term analysis of the use of computing resources could result in better decision-making related to the supervision of the use of resources by computing applications.

The implemented solution has a flexible architecture and can be adapted in the future to other solutions such as OpenVZ or heavyweight virtualization technologies. According to the proposed concept in Section 3.3, the use of multilevel mechanisms for lightweight virtualization inside an instance of OS managed by the VM allows more efficient resource management. This approach can be used in Grid environments designed for scientific computing where provisioning of VM instances for a particular *job* can be a very inefficient solution.

The Solaris platform discussed offers very rich functionality that may have support in the A-SOI-LV-PP. An interesting solution is the expansion of functionality to support network virtualization delivered by the Crossbow framework to enable monitoring of network resources and their management with a special emphasis on the limits of the band assigned to a lightweight container and running workloads. Very valuable DTrace services can be used for comprehensive monitoring of AEE, delivering low-level monitoring metrics with a particular emphasis on aspects relating to the use of computing resources, which allows for more in-depth analysis and precise selection of adaptive strategies.

The use of advanced processing tasks using Complex Event Processing [145] technologies is a very interesting solution, enabling the identification of the complex symptoms of many parallel streams of messages from specific SOI Managed Elements that are often together in relationships that could then be represented as a condition for inference. Correlation of monitoring metrics from, for example VEI, with concrete LVC instances with running applications allows reflecting the character of events and could be helpful in identifying anomalies which have occurred. It can serve as a rationale for the decision to modify the management policy.

The author is convinced the implemented A-SOI-LV-PP exposing Infrastructure Services for SOI provisioning with lightweight virtualization is a very solid base for expanding support for other applications to create a universal solution.

9 Terminology

This supplementary chapter contains selected abbreviations, definitions as well as an index of more relevant terms used in the thesis.

9.1 Abbreviations

AC	Autonomic Computing
ACS	Autonomic Computing System
ACPL	Autonomic Computing Policy Language
A-SOI-LV-PP	Adaptable SOI Lightweight Virtualization Provisioning Platform
AE	Autonomic Element
AEE	Application Execution Environment
AM	Adaptation Manager
AOP	Aspect Oriented Programming
BPM	Business Process Management
BPMN	Business Process Modeling Notation
CCM	Corba Components Model
CIM	Common Information Model
CIMOM	CIM Object Manager
CIM-SPL	Simple Policy Language for CIM
CSL	Connectivity Services Layer
DMTF	Distributed Management Task Force
DRP	Dynamic Resource Pool
DSD	Dynamic System Domain
EIS	Enterprise Information Systems
EJB	Enterprise Java Bean
EMA	Enterprise Management Associates
ESB	Enterprise Service Bus
GaPL	Governance and Policies Layer
GGF	Global Grid Forum

GMA	Grid Monitoring Architecture
GRM	Global Resource Manager
HaaS	Hardware as a Service
IaaS	Infrastructure as a Service
IFEAD	Institute for Enterprise Architecture Developments
IMF	Infrastructure Management Framework
IAG	Infrastructure Access Gateway
ISL	Information Services Layer
ITIL	Information Technology Infrastructure Library
ITMF	Information Technology Management Framework
IVT	Intel Virtualization Technology
JMX	Java Managed Extensions
jBPM	java Business Process Management
JIMS	JMX-based Infrastructure Management System
jPDL	java Process Definition Language
LWP	Lightweight Process
LVC	Lightweight Virtualization Container
LVC-AA	Lightweight Virtualization Container Application Appliance
LVC-RM	Lightweight Virtualization Container Resource Manager
MbTL	Model-based Translation Layer
MC	Manageability Components
MDF	Multiple Domain Facility
ME	Managed Element
MOM	Message Oriented Middleware
MMA	Monitoring and Management Agent
MMP	Monitoring and Management Platform
MR	Managed Resource
NIC	Network Interface Card
N1PP	N1 Provisioning Platform
N1SPS	N1 Service Provisioning System
OAM	Orchestrating Adaptation Managers
OCCI	Open Cloud Computing Interface
OMCM	Operations Management Capabilities Model
OS Kernel	Operating System Kernel
OS-RPM	Operating System Resource Pool Manager

OSL	Operational Services Layer
TAM	Touchpoint Adaptation Manager
TSS	Time Shared Scheduler
PaaS	Platform as a Service
PCIM	Policy Core Information Model
PEM	Policy Evaluation Module
PMAC	Policy Management for Autonomic Computing
PM	Provisioning Manager
PMI	Policy Management Infrastructure
PMM	Policy Management Module
POJO	Plain Old Java Objects
PDL	Policy Description Language
RAL	Resource Access Layer
RAM	Resource Access Module
REST	Representational State Transfer
RIaaSL	Runtime Infrastructure as a Service Layer
RMI	Remote Method Invocation
SCSL	Self-Configuration Services Layer
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SOI	Service Oriented Infrastructure
SFM	Solaris Fault Management
SLA	Service Level Agreement
SLO	Service Level Objective
SMF	Solaris Management Facilities
SNMP	Simple Network Management Protocol
SRM	Solaris Resource Manager
SSO	Single Sign On
UOGC	U.K. Office of Government Commerce
WS	Web Services
WSDL	Web Services Description Language
WSDM	Web Services Distributed Management
ZFS	Zeta File System
QoS	Quality of Service
QoSL	Quality of Service Layer

VEI	Virtual Execution Infrastructure
VEE	Virtual Execution Environment
VM	Virtual Machine
VMM	Virtual Machine Monitor
VNIC	Virtual Network Interface Card
VPS	Virtual Private Servers
XACML	eXtensible Access Control Markup Language

References

1. Gassman B., *Provisioning IT Services*, Gartner Group, October 2002
2. Cała J., *Adaptive Deployment of Component-based Applications in Distributed Systems*, A dissertation for the degree of Doctor of Philosophy, Faculty of Electrical Engineering, Automatics, Computer Science and Electronics, Institute of Computer Science, 2010
3. Gokhale A., Balasubramanian K., Balasubramanian J., *Modeldriven middleware: A new paradigm for deploying and provisioning distributed real-time and embedded applications*, Elsevier Journal of Science of Computer Programming: Special Issue on Model Driven Architecture, 2004
4. *N1 Grid Service Provisioning System - Application Awareness: Automating the Data Center*, SUN Microsystems Whitepaper, 2004
5. Zielinski K., Jarzab M., Kosinski J., *Role of N1 Technology in the Next Generation Grids Middleware*, Advances in Grid Computing - EGC 2005, Lecture Notes in Computer Science, 2005, Volume 3470/2005, 942-951
6. Zielinski K., Jarzab M., Kosinski J., *Configuration Management of Massively Scalable Systems*, Proceeding of the 2005 conference on Software Engineering: Evolution and Emerging Technologies, ISBN: 1-58603-559-2, pp: 172-183, IOS Press, Amsterdam, The Netherlands, 2005
7. *Adaptable Architecture: Best Practices for Meeting Dynamic Requirements*, Sun Microsystems, 2002
8. The Open Group, SOA Working Group, *Service Oriented Infrastructure Project Description "From Spaghetti to Lasagne"*
9. The Open Group, SOA Working Group, *Service Oriented Infrastructure Reference Framework*, Draft Technical Standard 2008
10. Chang M., He J., Castro-Leon E., Service-*Orientation in the Computing Infrastructure*, Proceedings of the Second IEEE International Symposium on Service-Oriented System Engineering (SOSE'06)
11. Ann Bednarz, Network World, *SOA, meet SOI*, Network World, 22/X/2007
12. A White Paper from BEA and VMware, *SOA and Virtualization: How Do They Fit Together?*
13. Doran R. W., *Amdahl Multiple-Domain Architecture*, Computer, Volume 21, Issue 10, Oct 1988 pp:20 – 28, Digital Object Identifier 10.1109/2.7054
14. Mc Dougall R., Cockcroft A., Hoogendoorn E., Vargas E., Bialaski T., *Resource Management - Sun Blueprints*, Prentice Hall PTR, ISBN: 0130258555
15. Heppolette V., Hoogeveen J.P., *Sun Enterprise™ 3000-6500 Dynamic Reconfiguration Best Practices*, Sun Microsystems Technical White Paper, 2000
16. SUN Microsystems, *J2ME Building Blocks for Mobile Devices - White Papper on KVM and the Connected, Limited Device Configuration (CLDC)*, 2000
17. Czajkowski G., Sun Microsystems Laboratories, *Application Isolation in the Java™ Virtual Machine*, ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications, October 15-19, 2000
18. Jordan M., Daynes L., Czajkowski G., Jarzab M., Brice C., *Scaling J2EE Application Server with the Multi-Tasking Virtual Machine*, Software Practice & Experience, Volume 36 , Issue 6 (May 2006), pp: 557 - 580, ISSN:0038-0644, 2006
19. Jordan M.,Czajkowski G.,Kouklinski K.,Skinner G., *Extending a J2EE Server with Dynamic and Flexible Resource Management*, ACM/USENIX 5th International Middleware Conference, Toronto, October 2004
20. Arsanjani A., Liang-Jie Zhang, Ellis M., Allam A., Channabasavaiah K., *S3: A Service-Oriented Reference Architecture*, IT Professional , pp: 10-17, Volume: 9, Issue: 3, May-June 2007
21. Tierney B., Aydt R., Gunter D., Smith W., Swany M., Taylor V., Wolski R., *A Grid Monitoring Architecture*, 2002
22. Darmawan B., Chen G.,Varkonyi L., *End-To-End Availability and Performance Management using IBM Tivoli Solutions*, An IBM Redguide publication, IBM Form Number:REDP-4518-00, 2009
23. Powell B., *IT Infrastructure Library (ITIL) V3: Overview and Impact*, Global Technology Services, ITS Strategy and Architecture
24. Wustenhoff E., Moore M.J., Avery D.H, *Operations Management Capabilities Model*, SUN Microsystems Blueprints, Part No. 819-1693-10 ,Revision 1.0, 1/14/05 Edition: February 2005
25. Elzinga D.J., Horak, T., Chung-Yee Lee, Bruner, C., *Business process management: survey and methodology*, IEEE Transactions on Engineering Management, pp: 119 - 128, Issue Date: May 1995, Volume: 42 Issue:2
26. Kou X., *GlassFish Administration*, Packt Publishing, ISBN-13: 978-1-847196-50-7, December 15, 2009
27. Lageman M., *Solaris Containers – What They Are and How to Use Them*, Sun Microsystems, http://www.sun.com/blueprints/0505/819-2679.pdf, 2005
28. Haff G., *Solaris Rises*, Illuminata Research Note report, http://www.sun.com/software/solaris/solaris_rises_report.pdf , 2004

29. Elling R., *Designing Enterprise Solutions with Sun Cluster 3.0*, Prentice Hall PTR, December 5, 2001, ISBN-10: 013008458
30. Gunther J.N., *The Virtualization spectrum from hyperthreads to grids*, Performance Dynamics Company, Castro Valley, California, USA, www.perfdynamics.com/Papers/njgCMG06.pdf
31. Bozman J.S., *Addressing Virtualization and High Availability Needs with SUN Solaris Cluster*, IDC White Paper - Sponsored by SUN Microsystems, October 2009
32. SWsoft Corporation, *OpenVZ User's Guide* Version 2.7.0-8, 2005
33. Savit J., *Energy Efficiency Strategies Sun Server Virtualization Technology*, Sun BluePrints, Part No 820-3023-10, Revision 1.0, 8/14/07, Edition: August 2007
34. Harris N., Jeyakumar L.R., Mann S., Sumarga Y., Wei W., *Logical Partitions on System i5: A Guide to Planning and Configuring LPAR with HMC on System i*, IBM Redbook, ISBN 0738496251, IBM Form Number SG24-8000-01
35. Cassier P., Defendi A., Fischer D., Hutchinson J., Maneville A., Membrini G., Ong C., Rowley A.,Bari P., *System Programmer's Guide to: Workload Manager*, IBM Redbook, ISBN 073848993X, IBM Form Number SG24-6472-03
36. An Enterprise Management Associates (EMA), *5 Key Virtualization Management Challenges – and How to Overcome Them*, Prepared for Hewlett-Packard, March 2009
37. Groß-Hohnacker B., Diplomarbeit, *Servervirtualisierung unter Linux Analyse und Bewertung*, Fachhochschule Köln University of Applied Sciences Cologne Campus Gummersbach, 2006, http://download.openvz.org/contrib/doc/xen-vs-openvz/Diplomarbeit Servervirtualisierung.pdf, (English version - http://download.openvz.org/contrib/doc/xen-vs-openvz/Diplomarbeit Servervirtualisierung English (part).pdf)
38. M. Bolte, M. Sievers, G. Birkenheuer, O. Niehorster, A. Brinkmann, *Non-intrusive virtualization management using libvirt*, Design, Automation & Test in Europe Conference & Exhibition (DATE), 2010
39. Jarząb M., Kosiński J., Zieliński K., *Virtualization of Grid Networking Resources for Computation Mobility Support*, Computational Methods in Science and Technology - Special Issue 2010, pp. 35-44, November 2010
40. Kosinski J., *Zarządzanie zasobami gridowymi z użyciem parawirtualizacji*, Rozprawa doktorska, Wydział Elektroniki, Automatyki, Informatyki i Elektroniki, Akademia Górniczo-Hutnicza, Kraków 2008
41. Horn P., *Autonomic Computing: IBM's Perspective on the State of Information Technology*, October 15, 2001, http://researchweb.watson.ibm.com/autonomic
42. Murch R., *Autonomic Computing*, pp. 0–20:25–40, ISBN: 013144025, IBM Press (April 3, 2004)
43. Ganek A.G., Corbi T.A., *The dawning of the autonomic computing era*, IBM Systems Journal, VOL 42, NO 1, 2003
44. *Autonomic computing concepts*, IBM corporation E-business white paper, G325-6904-00, 2001
45. *An architectural blueprint for Autonomic Computing*, IBM Corporation white paper - Third Edition, June 2005, http://www-03.ibm.com/autonomic/pdfs/AC%20Blueprint%20White%20Paper%20V7.pdf
46. Nami R. M., Bertels K.,*A Survey of Autonomic Computing Systems*, Computer Engineering Laboratory, Delft University of Technology
47. Mazeiar S., Ladan T., *Autonomic Computing: Emerging Trends and Open Problems*, In International ICSE Workshop on Design and Evolution of Autonomic Application Software (DEAS), pages 82{88, May 2005.
48. Lobo J., Bhatia R., Naqvi S., *A policy description language*, AAAI '99/IAAI '99 Proceedings of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference innovative applications of artificial intelligence
49. Agrawal D., Lee K., Lobo J., *Policy-based Management of Networked Computing Systems*, IEEE Communications Magazine, Vol. 43, No. 10, pp. 69 – 75, October 2005
50. Sloman M., *Policy Driven Management for Distributed Systems*, Journal of Network and Systems Management, Vol. 2 (1994)
51. OASIS, *Extensible Access Control Markup Language*, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml
52. Damianou N., Dulay N., Lupu E., Sloman M., *The Ponder Policy Specification Language*, LNCS,2001,pp: 18--38, Springer-Verlag
53. Stone G.N., Lundy B., Xie G.G., *Network policy languages: a survey and a new approach*, Network, IEEE, Volume: 15 Issue:1, Jan/Feb 2001, pp: 10 - 21, ISSN: 0890-8044
54. Agrawal D., Calo S., Lee K., Lobo J., Verma D., *Policy Technologies for Self-Managing Systems*, 2008, ISBN:0132213079 9780132213073
55. Strassner J.C., *Policy-based Network Management*, Morgan Kaufman, ISBN 1558608591, 2004
56. DMTF, *Common Information Model (CIM) Standards*, http://www.dmtf.org/standards/cim,2008
57. Moore B., Ellesson E., Strassner J.C., Westerinen A., *Policy Framework Core Information Model*, RFC 3060, February 2001

58. Kephart J.O., Walsh W.E., *An artificial intelligence perspective on autonomic computing policies*, Policies for Distributed Systems and Networks, 2004. POLICY 2004. Proceedings. Fifth IEEE International Workshop on 7-9 June 2004, pp. 3-12, ISBN: 0-7695-2141-X
59. C. Efstratiou, A. Friday, N. Davies, and K. Cheverst, *Utilising the event calculus for policy driven adaptation on mobile systems*, in 3rd International Workshop on Policies for Distributed Systems and Networks, 2002, pp. 13–24
60. H. Lutfiyya, G. Molenkamp, M. Katchabaw, and M. Bauer, *Issues in managing soft QoS requirements in distributed systems using a policy-based framework*, in 2nd International Workshop on Policies for Distributed Systems and Networks, 2001
61. L. Lymberopoulos, E. Lupu, and M. Sloman, *An adaptive policy based management framework for differentiated services networks*, in 3rd International Workshop on Policies for Distributed Systems and Networks, 2002, pp. 147–158
62. S. Wang, D. Xuan, R. Bettati, and W. Zhao, *Providing absolute differentiated services for real-time applications in static-priority scheduling networks*, in IEEE Infocom, 2001
63. J. Yoon and R. Bettati, *A three-pass establishment protocol for real-time multiparty communication*, Texas A&M University, Tech. Rep. 97-006, 1997
64. A. Chandra, W. Gong, and P. Shenoy, *Dynamic resource allocation for shared data centers using online measurements*, International Workshop on Quality of Service, 2003, pp. 381–400
65. T. Kelly, *Utility-directed allocation*, Workshop on Algorithms and Architectures for Self-Managing Systems, 2003
66. R. Rajkumar, C. Lee, J. P. Lehoczky, and D. P. Siewiorek, *Practical solutions for QoS-based resource allocation problems*, in IEEE Real-Time Systems Symposium, 1998, pp. 296–306
67. Bennani N., M., *Autonomic Computing through analytic performance models*, PhD Dissertation, George Mason University, 2006
68. W. E. Walsh, G. Tesauro, J. O. Kephart, and R. Das, *Utility functions in autonomic systems*, in International Conference on Autonomic Computing, 2004
69. Team Quest Model, *Capacity Planning Software for modeling, what-if analysis, and performance prediction*, http://www.teamquest.com/pdfs/products/model.pdf
70. Gunther N.J., *Analyzing Computer System Performance with Perl::PDQ*, Springer Berlin Heidelberg New York, ISBN 3-540-20865-8
71. Gunther N.J., *Guerrilla Capacity Planning, A Tactical Approach to Planning for Highly Scalable Applications and Services*, Springer-Verlag GmbH, Heidelberg, Germany, ISBN: 3540261389
72. Hellerstein J. L., Diao Y., Parekh S., Tilbury D. M., *Feedback Control of Computing Systems*, Wiley-IEEE Press, August 24, 2004, ISBN-13: 978-0471266372
73. Diao Y., Hellerstein J.L., Kaiser G., Parekh S., Phung D., *Self-Managing Systems: A Control Theory Foundation*, IBM Research Report, RC23374 (W0410-080) October 13, 2004 Computer Science
74. Pradeep P., Kang G. Shin, Xiaoyun Z., Uysal M., Wang Z., Singhal S., Merchant A., Salem K., *Adaptive control of virtualized resources in utility computing environments*, EuroSys '07 Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007
75. Diao Y., Hellerstein J.L., Parekh, S., *Using fuzzy control to maximize profits in service level management*, IBM Systems Journal, Volume: 41 Issue:3, pp: 403 - 420, 2002
76. Gmach D., Krompass S., Scholz A., Wimmer M., Kemper A., *Adaptive quality of service management for enterprise services*, ACM Transactions Web, Volume 2 Issue 1, February 2008
77. Bubnicki Z., *Teoria i algorytmy sterowania*, Wydawnictwo Naukowe PWN, Warszawa 2005, ISBN 83-01-14414
78. Grega W., *Metody i algorytmy sterowania cyfrowego w układach scentralizowanych i rozproszonych*, Uczelnianie wydawnictwo Naukowo-Dydaktyczne Akademii Górniczo-Hutniczej w Krakowie, 2004 ISBN 83-89388-78-2
79. Pearl J., *Heuristics: Intelligent Search Strategies for Computer Problem Solving*, New York, Addison-Wesley, 1983
80. Vlad N.G., Benatier V., Lehn R., Henri B., *Ontology-Based Autonomic Computing for Decision Support Systems Management: Shared Resources Allocation between Groups of Data Warehouses*, 2010 Third International Conference on Communication Theory, Reliability, and Quality of Service (CTRQ), 13-19 June 2010
81. *Java™ Management Extensions Instrumentation and Agent Specification*, Sun Microsystems, Maintenance Release v1.2, October 2002
82. Schulte R. W., Natis Y. V., *Service Oriented Architecture*, Gartner Report, available at: http://www.gartner.com, document id.: SPA-00-7425, 1996, verified: Dec. 2006
83. Zimmermann O., Tomlinson R. M., Peuser S.,*Perspectives on Web Services: Applying SOAP, WSDL and UDDI to Real-World Projects*, Springer Berlin Heidelberg New York, ISBN 3-540-00914-0

84. Topcuoglu H., Hariri S., Kim D., Kim Y., Bing X., Ye B., Ra I., Valente J. *The Design and Evaluation of a Virtual Distributed Computing Environment*, The Journal of Networks, Software Tools and Applications (Cluster Computing), 1998
85. Zielinski K., Jarzab M., Wieczorek D., Balos K., *Open Interface for Autonomic Management of Virtualized Resources in Complex Systems - Construction Methodology*, Future Generation Computer Systems Volume 24, Issue 5, May 2008, pp. 390-401
86. Padee A., Nawrocki K. et al., *Postprocessing Monitoring System, Report on the Results of the WP3 2nd and 3rd Prototype*, ed. Mayer N., CrossGrid Deliverable D3.5, available at: http://www.eu-crossgrid.org/Deliverables/M24pdf/CG3.0-D3.5-v1.2-PSNC010-Proto2Status.pdf, ALGO, CSIC, CYFRONET, DATAMAT, ICM, PSNC, TCD, UAB, UCY, Poznań, 2004, pp. 148–167
87. Massie M. L., Chun B. N., Culler D. E., *The ganglia distributed monitoring system: design, implementation, and experience*, Parallel Computing, Vol. 30/2004, Elsevier B.V., Holland, 2004, pp. 817–840
88. Andreozzi S., Bortoli N., Fantinel S., Ghiselli A., Tortone G., Vistoli C., *GridICE: a monitoring service for the grid*, CGW'03 Workshop Proceedings, ACC Cyfronet AGH, Kraków, pp. 220–226
89. Bałos K., Zieliński K., *JIMS -The Uniform Approach to Grid Infrastructure and Application Monitoring*, CGW '04 –Workshop Proceedings, 2005, pp. 160–167
90. Bałos K., Bizoń L., Rozenau M., Zieliński K., *Interoperability Architecture for Grid Monitoring Systems*, Proceedings of the CGW '03 (Cracow Grid Workshop 2003), Oct. 27 – 29 2003, pp. 245–253
91. Bałos K., Zieliński K., *WS-based Discovery Service for Grid Computing Elements*, European Grid Conference, February 14 -16 2005, Springer LNCS 3470, pp. 711–720
92. Wojtas K., Wasilewski L., Bałos K., Zieliński K., *Discovery Service for JMX-enabled Monitoring System - JIMS Case Study*, CGW'05 Proceedings
93. Sun Microsystems: *Java™ Management Extensions (JMX™) Remote API 1.0 Specification (JSR160*), Final Release, October 2003
94. MC4J Management Console, http://mc4j.sourceforge.net/
95. Oppermann R., *Adaptive user support: ergonomic design of manually and automatically adaptable software*, German National Research Center for Computer Science, Germany, 1994, ISBN:0-8058-1655-0
96. Freeman, T., D. LaBissoniere, P. Marshall, J. Bresnahan, and K. Keahey, *Nimbus Elastic Scaling in the Clouds*, http://www.nimbusproject.org/files/epu_poster4.pdf, 2010
97. Dong X., Hariri S., Xue L., Chen H., Zhang M., Pavuluri S., Rao S., *Autonomia: an autonomic computing environment, Performanc*e, Computing, and Communication Conference, 2003. Conference Proceedings of the 2003 IEEE International, IEEE, pp. 61–68
98. *Policy Management for Autonomic Computing*, Tivoli, version 1.2.1 http://dl.alphaworks.ibm.com/technologies/pmac/PMDevGuide121.pdf
99. Dong X., Hariri S., Xue L., Chen H., Zhang M., Pavuluri S., Rao S., *Autonomia: an autonomic computing environment, Performance, Computing, and Communication Conference*, 2003. Conference Proceedings of the 2003 IEEE International, IEEE, pp. 61–68
100. Topcuoglu, H., Hariri, S., Kim, D., Kim, Y., Bing, X., Ye, B., Ra, I., Valente, J. *The Design and Evaluation of a Virtual Distributed Computing Environment*, The Journal of Networks, Software Tools and Applications(Cluster Computing), 1998
101. R. Asadollahi, StarMX: *A Framework for Developing Self-Managing Software Systems*, A thesis presented to the University of Waterloo in fulfillment of the thesis requirement for the degree of Master of Applied Science in Electrical and Computer Engineering, Waterloo, Ontario, Canada, 2009
102. Liu Y., Gorton I., Trivedi N., *An Extensible, Lightweight Architecture for Adaptive J2EE Applications*, accepted by International Workshop of Software Engineering and Middleware (SEM), Nov 2006
103. *Enabling Adaptation of J2EE Applications Using Components, WebServices and Aspects*, National ICT Australia, *http://www.cse.unsw.edu.au/~yliu/asf-demo/index.html*
104. Liu Y., Gorton, I., *Implementing Adaptive Performance Management in Server Applications*, ICSE 2007 workshop on Software Engineering for Adaptive and Self-managing Systems (SEAMS), 2007
105. Kephart, Jeffrey O., *Research challenges of autonomic computing*, ICSE '05: Proceedings of the 27th international conference on Software engineering, ACM Press, New York, NY, USA, 2005, pp. 15–22
106. McDogall R., Mauro J., *Solaris Internals – Solaris 10 and OpenSolaris Kernel Architecture*, Sun Microsystems Inc., Santa Clara, USA, August 2006, ISBN 0-13-148209-2
107. B. A. Tate, J. Gehtland, *Better, Faster, Lighter Java*, O'Reilly Media, 2005, ISBN: 0596006764,
108. Jacobson V., Congestion Avoidance and Control, SIGCOMM, 1988, Stanford, California
109. Zielinski K., Jarzab M., *Database Access with EJB Application Servers Performance Study*, published in SUN One Application Server Performance White Paper, SUN Microsystems, July 2003
110. Zielinski K., Kosinski J., Jarzab M., *The improvement techniques of scalability and performance of J2EE based applications*, TASK Quarterly 4 (2004)
111. Zadrozny P., Aston P., Osborne T., *J2EE Performance Testing with BEA WebLogic Server*, Peer Information, 1st edition (April 2002), ISBN 1904284000

112. Bennani M.N., *Autonomic Computing Through Analytic Performance Models*, A Dissertation Submitted to the Graduate Faculty of George Mason University in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy Computer Science, George Mason University Fairfax, 2006
113. Schantz E. R., Schmidt D. C., *Middleware for Distributed Systems - Evolving the Common Structure for Network-centric Applications*, Encyclopedia of Software Engineering, Wiley and Sons, 2001
114. Coulson G. , *What is Reflective Middleware*?, IEEE Distributed Systems Online, http://dsonline.computer.org/middleware/RMarticlel.html, 2002
115. Capra L., Emmerich W., Mascolo C., *Reflective middleware solutions for context-aware applications*, In Proceedings of Reflection 2001, Lecture Notes in Computer Science, (Kyoto, Japan), Springer Verlag, 2001
116. Geihs K., *Middleware Challenges Ahead*, IEEE Computer, 2001, 34(6) , pp. 24-31
117. Portillo A.R., Walker S., Kirby G., Dearie A. (2003), *A Reflective Approach to Providing Flexibility in Application Distribution*, In proceedings of the 2nd Workshop on Reflective and Adaptive Middleware, Middleware 2003, Rio de Janeiro, Brazil; Springer-Verlag, Heidelberg, Germany
118. Blair G.S., Costa F.M., Coulson G., Duran H.A., Parlavantzas N., Delpiano F., Dumant B., Horn F., Stefani J.B., *The Design of a Resource-Aware Reflective Middleware Architecture*, In proceedings of the Second International Conference on Meta-Level Architectures and Reflection (Reflection '99), Springer, St. Malo, France, pp. 115-134
119. McKinley P.K., Seyed Masoud Sadjadi, Kasten E.P., Cheng Betty H. C., *A Taxonomy of Compositional Adaptation*, Software Engineering and Network Systems Laboratory, Department of Computer Science and Engineering, Michigan State University, Technical Report MSU-CSE-04-17, May 2004
120. Mirko M., Loris P., Anna P., *Towards Goal-Oriented Development of Self-Adaptive Systems*, In ICSE International Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), pages 9-16, May 2008.
121. R. Reichle, Khan M.U., Geihs K., *How to Combine Parameter and Compositional Adaptation in the Modeling of Self-Adaptive Applications*, PIK - Praxis der Informationsverarbeitung und Kommunikation. Volume 31, Issue 1, Pages 34–38, ISSN (Print) 0930-5157, DOI: 10.1515/piko.2008.007, Januar-März 2008
122. Maes P. ,*Concepts and experiments in computation reflection*, ACM SIGPLAN Notices, 22(12):147-155, December 1987
123. Smith B. C., *Reflection and Semantics in a Procedural Language*, PhD Thesis, 1985
124. Kiczales G., J. des Rivi`eres, Bobrow D. G., *The Art of Metaobject Protocols*, MIT Press, 1991
125. Information Sciences Institute - University of Southern California, *RFC 793 - Protocol Specification of Transmission Control Protocol*, Darpa Internet Program September 1981, http://tools.ietf.org/html/rfc793
126. Sousa J. P., D. Garlan, *Aura: an architectural framework for user mobility in ubiquitous computing environments*, In Proceedings of the third Working IEEE/IFIP Conference on Software Architecture, pp. 29–43, 2002
127. Liang-Jie Z., Zhou Q., *CCOA: Cloud Computing Open Architecture*, Web Services, 2009. ICWS 2009. IEEE International Conference on 6-10 July 2009
128. *Oracle Cloud Resource Model API*, Document Number: TBD ,Date: 2010-06-28, Version 1.0, http://www.oracle.com/technetwork/topics/cloud/oracle-cloud-resource-model-api-154279.pdf
129. Open Community Leading Cloud Standards, *Open Cloud Computing Interface*, http://occi-wg.org
130. The Cloud Computing Use Cases Discussion Group, *Cloud Computing Use Cases White Paper*, Version 4.0, 2010
131. Amazon WS, http://aws.amazon.com
132. Kiczales G., Lamping J., Mendhekar A., Maeda C., Videira Lopes C., Loingtier J. M., Irwin J., *Aspect-oriented programming*, Proceedings of the European Conference on Object-Oriented Programming (ECOOP), Springer-Verlag LNCS 1241, June 1997
133. Microsoft Corporation, *DCOM Technical Articles - DCOM Technical Overview*, http://msdn.microsoft.com/en-us/library/ms809340.aspx, November 1996
134. Programming with Application Domains and Assemblies, http://msdn.microsoft.com/en-us/library/dah4cwez.aspx
135. Microsoft Corporation, *NET Framework Developer's Guide - .NET Framework Conceptual Overview*, http://msdn.microsoft.com/en-us/library/zw4w595w.aspx
136. Object Management Group, *CORBA Component Model Specification* - OMG Available Specification, Version 4.0 , 2006
137. Sun Microsystems, DeMichiel L. Michael K., *Enterprise JavaBeansTM,Version 3.0 - EJB Core Contracts and Requirements*, 2006
138. Carolan J., Radeztsky S., Strong P., Turner E.,*N1 Grid Architecture Realized: Measurable Requirements*, SUN Microsystems Blueprints, Part No. 819-1742-10, Revision A, 2/28/05, Edition: March 2005

139. Balos K., *Self-Configurable Service with Elements of Adaptability for Monitoring of Infrastructure and Applications in a Grid Environment*, PhD Dissertation, AGH – University of Science and Technology Kraków, Poland, 2006
140. Chung L., Nixon B.A., Yu E., Mylopoulos J., *Non-Functional Requirements in Software Engineering, Series: International Series in Software Engineering*, Vol. 5, 1999, 476 p., Hardcover, ISBN: 978-0-7923-8666-7
141. Gamma E., Helm R., Johnson R., Vlissides J.M., Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Professional; 1 edition (November 10, 1994), ISBN-10: 0201633612
142. Schmidt D., Stal M., Rohnert H., *Pattern-Oriented Software Architecture Series Volume 1-4*, John Wiley & Sons
143. Fowler M, *Analysis Patterns: Reusable Object Models*, Addison-Wesley, ISBN 0201895420.
144. Allen P., Bambara J., *SUN Certified Enterprise Architect for J2EE Study Guide*, McGraw-Hill Osborne Media; 1 edition , March 27, 2003, ISBN-10: 0072226870
145. D. C. Luckham, *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001
146. Zmuda D., Psiuk M., Zielinski K., *Dynamic monitoring framework for the SOA execution environment*, Procedia Computer Science, Volume 1, Issue 1, ICCS 2010, May 2010, Pages 125-133, ISSN 1877-0509
147. Gunther N.J., *Solaris System Resource Manager: All I Ever Wanted Was My Unfair Advantage (And Why You Can't Get It!*), December 5-10 1999, Computer Measurement Group Conference, Reno, NV
148. Kay J., Lauder P., *A Fair Share Scheduler*, Communication of the ACM, Volume 31, Issue 1, January 1988, ISSN:0001-0782, pp. 44 – 55
149. Hrasna H., *JavaTM 2 Platform, Enterprise Edition Management Specification JSR-77*, Proposed Final Draft 2, Sun Microsystems, Inc., http://jcp.org/jsr/detail/77.jsp

Appendix A – Listings

```
if (checkRegistryFlag(JIMSAgentConstants.JIMS_CONST_MON, JIMSRegistry)) {
       String osModule = getOSModuleNameForDownload("mbeanOSInfo",osProfile);

       loadModuleRemote(ceHost, osModule);
}

/**
 * Calculate module name to be downloaded by the MLet service.
 * Final name depends on baseName, OS platform (reflection is used - os.name
 *  + os.arch) and specified profile.
 */
protected String getOSModuleNameForDownload(String baseName, String profile) {
       StringBuilder name = new StringBuilder();

       name.append(baseName).append(System.getProperty("os.name")).
       append("-").append(System.getProperty("os.arch"));

       if ( profile != null && profile.compareToIgnoreCase("")  != 0  ) {
              name.append("-").append(profile);
       }

       name.append(".mlet");

       return name.toString();
}
```

Listing 1. Computational reflection used by the MMA for dynamic module acquisition from the MLet service.

```
if ( zoneManager.isGlobalZone() ) {
       extractHelpers();
       zoneManagement = new GlobalZoneManagement();
       beanName = SolarisManagementCommons.createZoneManagementObjectName("global");
       mbs.registerMBean( zoneManagement, beanName);
} else {
       String zoneName = zoneManager.getZoneName();
       zoneManagement = new LocalZoneManagement();
       beanName = SolarisManagementCommons.createZoneManagementObjectName( zoneName );
                        mbs.registerMBean( zoneManagement, beanName);
}
```

Listing 2. Computational reflection used by the ZoneAgent MBean component for dynamic composition of SOI Virtualized Services for Solaris 10 management and monitoring.

```
public abstract class SolarisCommandFactory  {
       public static final int SOLARIS10 = 10;
       public static final int SOLARIS11 = 11;

       public static SolarisCommandFactory getFactory(int factoryId) {
              SolarisCommandFactory factory = null;

              if ( factoryId == SOLARIS10) {
                     factory = new Solaris10CommandFactory();
              }
              else if (factoryId == SOLARIS11) {
                     factory = new Solaris11CommandFactory();
              }
              else {
                     throw new IllegalArgumentException("Solaris factory ID is not
                     valid!");
              }

              return factory;
       }

       public abstract CreateProjectCommand getCreateProjectCommand()
         throws CommandException;
       public abstract ModifyProjectCommand getModifyProjectCommand()
         throws CommandException;
       public abstract RemoveProjectCommand getRemoveProjectCommand()
         throws CommandException;
       public abstract GetProjectCommand getProjectCommand() throws CommandException;
       public abstract GetProjectsCommand getProjectsCommand()
         throws CommandException;

       public abstract CreateZoneCommand getCreateZoneCommand()
         throws CommandException;
       public abstract CreateZoneFromSnapshotCommand
         getCreateZoneFromSnapshotCommand() throws CommandException;
       public abstract ModifyZoneCommand getModifyZoneCommand()
         throws CommandException;
       public abstract RemoveZoneCommand getRemoveZoneCommand()
         throws CommandException;
       public abstract GetZoneCommand getZoneCommand() throws CommandException;
       public abstract GetZonesCommand getZonesCommand() throws CommandException;
       public abstract GetZoneNameCommand getZoneNameCommand()
         throws CommandException;

       public abstract SwitchExacctCommand getSwitchExacctCommand()
         throws CommandException;

       public abstract FsCommand getFsCommand() throws CommandException;

       public abstract GetSolarisResourceControlsCommand
         getSolarisResourceControlsCommand()
              throws CommandException;
}
```

Listing 3. Exploitation of Command design pattern by Solaris factory class.

```
public static void premain(String agentArgs,  Instrumentation inst) {
   try {
      props.load(Agent.class.getResourceAsStream("/discovery.properties"));

      String mcastAddr = null;
      String persistMonitorData = null;
      String jmxRemotePort = null;
               boolean registerDiscoveryMBean = true;
               String[] credentials = null;
               if ( agentArgs != null) {
                 String[] args = agentArgs.split(",");

       mcastAddr = getParamValue("mcastaddr", args);
       persistMonitorData = getParamValue("persistmonitordata", args);
       jmxRemotePort = getParamValue("jmxremoteport", args);
       String register = getParamValue("registerdiscovery", args);
       if ( register != null)
          registerDiscoveryMBean = Boolean.parseBoolean(register);
          String tmp = getParamValue("credentials", args);
              if ( tmp != null) credentials = tmp.split("\\\\");
       }

       if ( jmxRemotePort == null) {
          jmxRemotePort = System.getProperty("com.sun.management.jmxremote.port");
          if ( jmxRemotePort  == null) {
             throw new IllegalStateException("JIMS premainclass has not specified
                         JMX port!");
          }
        }
        if ( mcastAddr == null ) {
           mcastAddr = props.getProperty("jims.mbs.discovery.group") +  "/" +
                props.getProperty("jims.mbs.discovery.port");
        }
        String jmxAddress = "service:jmx:rmi:///jndi/rmi://" +
           InetAddress.getLocalHost().getHostAddress()
                +  ":" + jmxRemotePort + "/jmxrmi";

        DiscoveryUserData userData = new DiscoveryUserData(jmxAddress,
         ProcUtil.getPID() );
        if ( persistMonitorData != null )
           userData.setPersistMonitorData(Boolean.valueOf(persistMonitorData));
           if (credentials != null) {
               userData.setCredentials( credentials );
                String[] operatingSystemInformation = {System.getProperty("os.name"),
                  System.getProperty("os.version"), System.getProperty("os.arch")};
               userData.setOSInfo(operatingSystemInformation);

               DiscoveryResponderMBean discoveryMBean = new DiscoveryResponder();
               if ( registerDiscoveryMBean ) {
                  MBeanServer mbs = ManagementFactory.getPlatformMBeanServer();
                  ObjectName discoveryObjName =  new
                      ObjectName("Core:name=DiscoveryResponder");
                mbs.registerMBean(discoveryMBean, discoveryObjName);
               }
               discoveryMBean.setUserData( userData );
               discoveryMBean.setMulticastAddress(mcastAddr);
               discoveryMBean.start();

               System.out.println("JIMS pre-main auto-discovery agent is started." );
        }
        catch(Throwable t) {
           System.err.println("Failed to initialiaze JIMS pre-main-agent:" +
t.getMessage());
        }
}
```

Listing 4. Implementation of Java premain agent.

```
<service_bundle type="manifest"
                name="JIMS infrastructure management agent">
  <service name="application/JIMS/jims-node-agent"
           version="1"
           type="service">
    <single_instance/>
    <exec_method type="method"
                 name="start"
                 exec="/opt2/servers/jims/jims-agent/bin/jims-agent.sh start"
                 timeout_seconds="0">
      <method_context working_directory=":default">
        <method_credential user="jims"
                           group=":default"/>
                           <!-- privileges="basic" -->
      </method_context>
      <stability value="Evolving"/>
    </exec_method>
    <exec_method type="method"
                 name="stop"
                 exec="/opt2/servers/jims/jims-agent/bin/jims-agent.sh stop"
                 timeout_seconds="0">
      <method_context working_directory=":default">
        <method_credential user="jims"
                           group=":default"/>
      </method_context>
      <stability value="Evolving"/>
    </exec_method>
  </service>
</service>
```

Listing 5. Configuration of Service Management Facility for Monitoring and Management agents.

```
# Create node agent instance
function node_agent_create {
       echo "Create node agent ..."
       $GLASSFISH_HOME/bin/asadmin create-node-agent --host $DAS_HOST_NAME --port
         $DAS_PORT_NUMBER --user $USER_NAME --passwordfile
         ${GLASSFISH_HOME}/.glassfish_pwd $NODE_AGENT_NAME
       return_code=$?;
       echo "Node agent created!"
}

# Start node agent using SMF service
function start_smf_node_agent {
       echo "Starting node agent with SMF ..."
       /usr/sbin/svcadm enable application/SUNWappserver/glassfish-node-agent
       echo "Node agent started!"
}

# SMF descriptor for node agent
function smf_node_agent_create {
       echo "Create SMF descriptor for node agent ..."
       $GLASSFISH_HOME/bin/asadmin create-service --name glassfish-node-agent
          --type node-agent --passwordfile ${GLASSFISH_HOME}/.glassfish_pwd
        ${GLASSFISH_HOME}/nodeagents/$NODE_AGENT_NAME
       return_code=$?;
       echo "SMF descriptor created!"
}
```

Listing 6. Shell script for Glassfish provisioning.

```
package ${mbeanPackage};

/**
 * Wrapper class designated for use with Autonomic Management Toolkit compliant
policy based management system.
 * Wrapped class: ${mbeanInfo.ClassName}Mbean
 *
 * Generation date: ${generationDate}
 */
public class ${mbeanClassSimpleName} extends org.amt.server.resource.Resource
implements java.io.Serializable {

        /**
         * Logger handler.
         */
        private static final org.apache.log4j.Logger logger =
           org.apache.log4j.Logger.getLogger(${mbeanClassSimpleName}.class);

        /**
         * Creates wrapper instance.
         *
         * @param connection remote wrapped object Mbean server connection.
         * @param mbeanName remote wrapped object Mbean name.
         * @param isNotificationBroadcaster mbean supports notifications.
         * @param name adaptation manager internal resource name.
         */
        public ${mbeanClassSimpleName}(
           javax.management.MbeanServerConnection connection,
           javax.management.remote.JMXServiceURL url,
           java.lang.String className, javax.management.ObjectName mbeanName,
           boolean isNotificationBroadcaster)
         throws javax.management.InstanceNotFoundException, java.io.IOException {
                super(connection, url, className, mbeanName);

                this.connection = connection;
                if (isNotificationBroadcaster)
                  connection.addNotificationListener(mbeanName, this, null, null);
        }
        public void deleteResourceProperty(String propertyName)
           throws com.ibm.autonomic.policy.am.JavaMRException {
           logger.error("Unsupported operation deleteResourceProperty is has been
                  invoked.");
           throw new UnsupportedOperationException(
                  "Operation deleteResourceProperty is not supported.");
        }
        public void insertResourceProperty(String propertyName, Object propertyValue)
           throws com.ibm.autonomic.policy.am.JavaMRException {
           logger.error("Unsupported operation insertResourcePropertyhas been
                        invoked.");
           throw new UnsupportedOperationException(
            "Operation insertResourceProperty is not supported.");
        }
        public Object[] getMultipleResourceProperties(String[] propertyNames)
                        throws com.ibm.autonomic.policy.am.JavaMRException {
           try {
               javax.management.AttributeList attributes =
                           connection.getAttributes(mbeanName, propertyNames) ;
               Object[] result = new Object[propertyNames.length];
               for (int I = 0; I < propertyNames.length; i++)
                    result[i] = ((javax.management.Attribute)
                                        attributes.get(i)).getValue();
               return result;
           } catch (Exception e) {
                logger.error("Mbean " + mbeanName
```

```
                        + “: cannot get multiple resource properties.”, e);
         throw new org.amt.server.resource.ResourceException(
                        “Cannot get multiple resource properties.”, e);
   }
}
public Object getResourceProperty(String propertyName)
                throws com.ibm.autonomic.policy.am.JavaMRException {
         try {
                return connection.getAttribute(mbeanName, propertyName);
   } catch (Exception e) {
      logger.error("MBean " + mbeanName
          + ": cannot get resource property.", e);
      throw new org.amt.server.resource.ResourceException( "Cannot get
          resource property.", e);
   }
}
public void updateResourceProperty(String propertyName, Object propertyValue)
                throws com.ibm.autonomic.policy.am.JavaMRException {
   try {
       connection.setAttribute(mbeanName,new javax.management.Attribute(
                        propertyName, propertyValue));
   } catch (Exception e) {
      logger.error("MBean " + mbeanName + ": cannot update resource property"
                        + propertyName + ".");
      throw new org.amt.server.resource.ResourceException(
                        "Cannot update resource property.", e);
   }
}
public Object invoke(String methodName, Object[] parameters,
    String[] signature)
    throws java.lang.NoSuchMethodException, java.lang.IllegalAccessException,
    java.lang.reflect.InvocationTargetException {
    try {
        return connection.invoke(mbeanName, methodName, parameters,
              signature);
    } catch (Exception e) {
        logger.error("MBean " + mbeanName + ": cannot invoke "
                        + "resource operation " + methodName + "().", e);
        throw new org.amt.server.resource.ResourceException(
                        "Cannot invoke resource operation.", e);
    }
}
/**
 * If wrapped object has notifications wrapper is listener for these
 * notifications.
 */
public void handleNotification(javax.management.Notification notification,
           Object handback) {
   for (org.amt.server.reasoner.ResourceNotificationListener i : listeners)
                i.handleNotification(new
      org.amt.server.resource.ResourceNotification(this, notification,
                   handback));
   }

   #foreach($i in ${mbeanInfo.Operations})
      public ${i.ReturnType} ${i.Name}(#set($addComa = false)#foreach($j in
       ${i.Signature})#if($addComa), #end${j.Type} ${j.Name}#set($addComa =
       true)#end) {
         try {
           #if(${i.ReturnType} != "void")
            return (${i.ReturnType})#end
              connection.invoke(mbeanName, "${i.Name}",
           new Object[] { #set($addComa = false)#foreach($j in
           ${i.Signature})#if($addComa), #end${j.Name}#
               set($addComa = true)#end },
           new String[] { #set($addComa = false)#foreach($j in
           ${i.Signature})#if($addComa), #end"${j.Type}"#
            set($addComa = true)#end });
```

```
                    } catch (Exception e) {
                      logger.error("MBean " + mbeanName + ": cannot invoke " + "resouce
                        operation ${i.Name}().", e);
                       throw new org.amt.server.resource.ResourceException(
                                  "Cannot invoke resource operation.", e);
                    }
          }
#end
}
```

Listing 7. Velocity template for dynamic definition of Resource Adapter's MBean component.

```
package org.adaptive.soi.provisioning.containers.shares;

import java.util.Map;
import org.amt.resource.Resource;
import org.amt.resource.ResourceNotification;
import org.amt.policy.PolicyLogger;

global Map vars;
global PolicyLogger log;

function int getProjectCpuShares(Map vars, Resource targetManagement) {
        if ("shares.target".equals(targetManagement.getString("Name")) &&
             vars.containsKey("target shares") ) {
             return ((Integer) vars.get("target shares")).intValue();
        } else {
             Integer result = (Integer)
             targetManagement.invoke("readResourceControlValue", new
                 Object[] { "project.cpu- shares", "privileged" }, new
              String[] { "java.lang.String", "java.lang.String" });
             return result.intValue();
        }
}

function void setProjectCpuShares(Map vars, Resource targetManagement,
        int sharesValue) {
        vars.put("target shares", new Integer(sharesValue));
        targetManagement.invoke("replaceResourceControl", new Object[] {
              "project.cpu-shares", "privileged", new String[0],
              new Integer(sharesValue), new Integer(0) }, new String[] {
             "java.lang.String", "java.lang.String",
             "[Ljava.lang.String;", "int", "int" });
}

rule "Initialize shares sum"
salience 3
    when
        eval(true)
    then
        vars.put("SharesSum", new Integer(0));
end

rule "Get project shares"
salience 2
   when
     globalZoneProjectManagement : Resource()
     eval("org.jims.mbeans.solaris10.ProjectManagement".
        equals(globalZoneProjectManagement.getClassName()))
     eval("global".equals(globalZoneProjectManagement.getString("ZoneName")))

     globalZoneProjectMonitoring : Resource()
  eval("org.jims.mbeans.solaris10.ProjectResourceUsageMonitor".
     equals(globalZoneProjectMonitoring.getClassName()))
  eval("global".equals(globalZoneProjectMonitoring.getString("ZoneName")))
  eval(globalZoneProjectManagement.getString("Name").equals(
      globalZoneProjectMonitoring.getString("ProjectName")));
```

```
  eval(globalZoneProjectMonitoring.getInt("Procs") > 0)
  then
    vars.put("SharesSum", new Integer(((Integer)
      vars.get("SharesSum")).intValue() + getProjectCpuShares(vars,
  globalZoneProjectManagement)));
end

rule "Set shares"
salience 1
    when
       target : Resource()
       eval(target.is("solaris10.monitoring.global:type=projects,
         name=shares.target,role=monitoring"))
       eval(target.getDouble("PercentageCpuUsage") != 70.0)
       targetManagement : Resource()
       eval(targetManagement.is("solaris10.management.global:type=projects,
       name=shares.target,role=management"))

      cpu : Resource()
       eval(cpu.is("solaris10.monitoring.cpu:id=cpu0,class=Kstat"))
       eval(cpu.getInt("CpuUsage") == 100)
      then
       setProjectCpuShares(vars, targetManagement, (int)
             (getProjectCpuShares(vars, targetManagement) +
               3.0 * (70.0 - target.getDouble("PercentageCpuUsage"))));
end
```

Listing 8. Drools rule implementing Proportional regulator for Solaris Container adaptive management.

```
List getCPUBoundWorkloads () {
    List workloadsList = new ArrayList();

    /** Get active projects for which number of processes
    and CPU usage is greater than zero */
    List activeProjects = getActiveProjects();

    /** Calculate the sum of shares of these projects */
    int sumOfShares = 0;
    foreach (projectId:activeProjects) {
       sumOfShares += getProjectShares(projectId);
    }
    /** Project is to be considered CPU-bound if the current CPU usage is
        greater or equal then its CPU  entitlement according
        to assigned  shares */
    float entitlement; float cpuUsage;
    foreach (projectId:activeProjects) {
      entitlement = getProjectShares(projectId)/
                        sumOfShares;
      cpuUsage = getCpuUsage(projected);
      if ( entitlement <= cpuUsage ) {
        workloadsList.add(projected);
      }
    }
  return workloadList;
}
```

Listing 9. Heuristic for determination of Solaris CPU bound workloads.