Computing and Informatics, Vol. 34, 2015, 1309–1339

View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Computing and Informatics (E-Journal - Institute of Informatics,...

ADAPTABLE SERVICE ORIENTED INFRASTRUCTURE PROVISIONING WITH LIGHTWEIGHT CONTAINERS VIRTUALIZATION TECHNOLOGY

Marcin Jarząb

Department of Computer Science, AGH University of Science and Technology
Al. Mickiewicza 30, 30-059 Kraków, Poland
&
Smart Cloud Solutions, Platform Architecture and Security
Samsung Research Poland
Al. Bora-Komorowskiego 25, 31-476 Kraków, Poland
e-mail: mj@agh.edu.pl

Krzysztof Zieliński

Department of Computer Science, AGH University of Science and Technology
Al. Mickiewicza 30, 30-059 Kraków, Poland
e-mail: kz@agh.edu.pl

Abstract. Modern computing infrastructures should enable realization of converged provisioning and governance operations on virtualized computing, storage and network resources used on behalf of users' workloads. These workloads must have ensured sufficient access to the resources to satisfy required QoS. This requires flexible platforms providing functionality for construction, activation and governance of Runtime Infrastructure which can be realized according to Service Oriented Infrastructure (SOI) paradigm. Implementation of the SOI management framework requires definition of flexible architecture and utilization of advanced software engineering and policy-based techniques. The paper presents an Adaptable SOI Provisioning Platform which supports adaptable SOI provisioning with lightweight virtualization, compliant with the structured process model suitable for construction, activation and governance of IT environments. The requirements, architecture and implementation of the platform are all discussed. Practical usage of

the platform is presented on the basis of a complex case study for provisioning JEE middleware on top of the Solaris 10 lightweight virtualization platform.

Keywords: Service oriented infrastructure, converged infrastructure, provisioning, lightweight virtualization, adaptability

1 INTRODUCTION

Modern computing infrastructures for applications should be based on architectures that provide flexible platforms and operational capabilities for provisioning *Runtime Infrastructure* defined as application containers within application middleware running on an operating system instance. These include activities required to prepare, deploy and activate a particular application service, and ensure required resources including: computing, storage, network, supporting middleware and applications. These are classical provisioning procedures which are repeatable and can be defined in a structured process [1]. Adaptable provisioning and management of Runtime Infrastructure must provide means to effective usage of resources to guarantee that a given *application service* is available for *consumers* with a specified Quality of Service (QoS) that might apply to many aspects associated with a service running, for instance *performance* or *reliability*.

The Service Oriented Infrastructure (SOI) represents a shift from dedicated infrastructures for specific applications to a generic architecture in which IT resources and system management tools are exposed as services and allocated on demand. The SOI provides foundational support for a Service Oriented Architecture (SOA) or other application architecture [2]. The goal of the paper is to propose a suitable methodology for constructing an Adaptable SOI Provisioning Platform (A-SOI-PP) enabling flexible and adaptive provisioning of lightweight virtualization environments which ensure the required resources for user applications according to specified QoS goals. Although the platform is oriented to lightweight virtualization, it is generic and can be used with other virtualization technologies. It represents an integrated view on infrastructure and applications layers enabling the realization of adaptable provisioning process including specification of QoS requirements and constant governance. In our work the particular attention is devoted to lightweight virtualization offered by Solaris 10 OS, workloads consolidation and QoS enforcements using policy-driven adaptive management. Subsequently, software architectures and construction techniques of adaptation loops for SOI provisioning and coherent to Autonomic Computing Systems (ACS) are described. Principles of controller implementation are proposed and practically (conditions of applicability) verified.

The paper is organized as follows. Section 2 presents the state of the art. Section 3 describes requirements and a model for adaptable SOI provisioning. The architecture and implementation of A-SOI-PP are presented in Sections 4 and 5 re-

spectively. In Section 6 a case study is presented. The paper ends with conclusions and a list of possible improvements which could be considered a part of the future work.

2 STATE OF THE ART

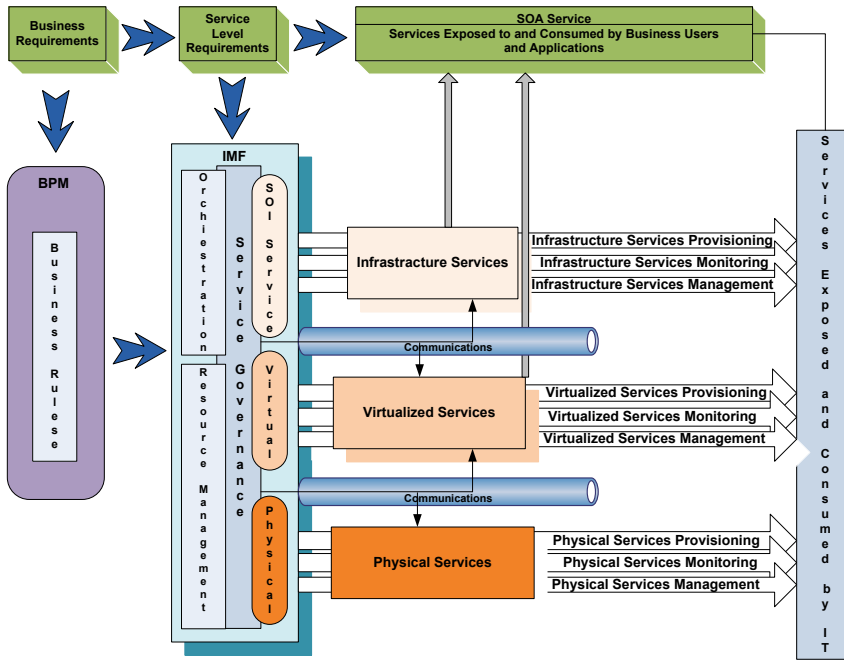

Figure 1. SOI reference model

The SOI reference model defined by *The Open Group* (Figure 1 – [2]) merges Infrastructure Services with applications and consumers, thus capturing all the activities related to IT infrastructure provisioning and governance. The model defines the conceptual building blocks enabling arbitrary services (or other elements) to be used in SOI once such architecture is implemented. An Infrastructure Management Framework (IMF) provides a collection of software tools that enable provisioning and management of IT infrastructures in an adaptable way, driven by business rules. An Infrastructure Services expose interfaces for management of infrastructure resources. These services are implemented as combinations of virtualized (specific hardware virtualization solutions) and physical (hardware, storage and operating system resources) services. The IMF, a platform for adaptable SOI provisioning, must be flexible, give the system administrator flexibility and increase the 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. The implementation of the framework must be reconfigurable, built from modularized components that can be re-organized and extended with new functionality to meet new business requirements. Successful implementation of the platform for SOI provisioning involves fusion of technologies, business models and infrastructure management techniques. Such approach would also enable realization of converged infrastructure [3] concept where computing, network and storage resources are combined into a single framework thus providing the optimized computing environment.

Virtualization is a basic mechanism required by SOI which increases IT agility and flexibility because virtualized environments can be provisioned on demand, allowing deployment of application services (so-called deployable entities – DE). Another useful technique is clustering, which takes multiple services that can be running over virtualized services and using appropriate software technologies, can be exposed as a single service available to consumers. It enables dynamic scaling across virtualized services through agility to add or remove these services with running DEs. When used, these two techniques can be extremely powerful in providing better performance and reliability.

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 (Figure 2). Lightweight virtualization [4] is an attractive approach to SOI provisioning, providing granular resource partitioning: a single OS instance can host many instances of Lightweight Virtualization Containers (LVC). In comparison to other heavyweight virtualization solutions such as Virtual Machines (VM) or Hardware Domains (HD), LVCs are provisioned more easily and consume fewer resources [5]. Although the number of DEs remains the same in case of each technology, the management complexity expressed by the number of Physical Services in terms of servers, hypervisors and OS is the smallest in case of lightweight virtualization. In addition to mechanisms which enable resource partitioning between LVCs, it is also capable of apportioning resources within a particular LVC instance to specific processes organized into groups called workloads. It can be used in conjunction with VM or HD providing support for multilevel virtualization. Such aproach is utilized by many Platform as a Service engines such as CloudFoundry, OpenShift [6] where application containers dedicated for particular users are running over cluster of VMs and isolated within these VMs with lightweight containers. LVC has negligible impact on performance; however it also introduces an important limitation related to the fact that applications are often certified for a specific OS and its version. Some well-known implementations of LVC include Solaris Containers, Linux Containers and OpenVZ. In opinion of the authors, Solaris provides the most advanced and mature implementation of LVC because of integrated compute, storage and network virtualization stack combined with many services like high-availabilty and clustering frameworks [7].

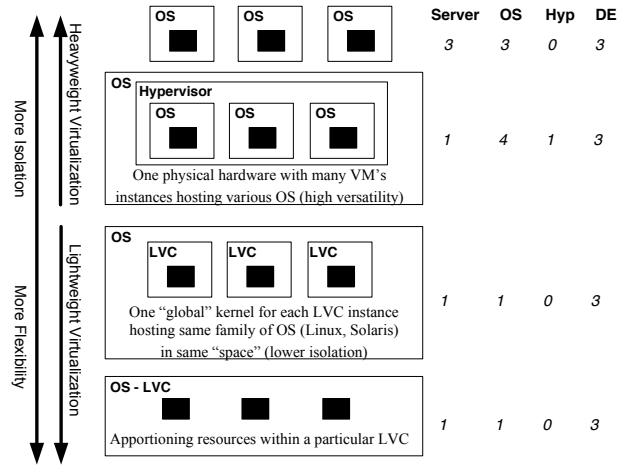

Figure 2. Management complexity of heavyweight and lightweight virtualization

When analyzing implementation of the IMF, lightweight virtualization offers more unified access interfaces required by the management and monitoring activities. The Infrastructure Services can only instrument the global kernel OS interfaces which can operate directly on each LVC instance and running workloads inside. In case of VM or HD technologies, appropriate agents must be running inside the hypervisor and guest OS. Same occurs in case of security functionality where policies defined within global OS can be easily replicated to hosted LVCs. Available LVC implementations do not provide effective tools for QoS management and complex life-cycle management, including converged view on virtualized resources and running applications. Administrators may only specify computing resource limitations for LVC instances in the scope of well-known configurations (number of entities with assigned resource limits).

Currently there are many platforms which support provisioning of virtualized infrastructures based on VM and lightweight virtualization technologies. OpenNebula, OpenStack and Eucalyptus [8] orchestrate compute, storage and networking resources to provide virtualized environment to end users. They enable dynamic deployment of multi-tier services (groups of interconnected VMs) on distributed infrastructures in accordance with allocation policies. Nimbus [9] provides a toolset for infrastructure provisioning with emphasis on the needs of science, although many non-scientific use cases are also supported. It allows clients to lease remote resources by deploying VMs on such resources and configuring them to represent a user-defined environment (so-called "Virtual Workspace Service"). The presented frameworks facilitate control and management, permitting clients to deploy and configure the required appliance software and hardware resources. They lack management features exploiting Monitor-Analyze-Plan-Execute (MAPE) [26] pattern of

hosted workloads, required to maintain the desired level of QoS for applications as well as system services. In addition, they do not constitute an integrated framework for provisioning, monitoring and management of Runtime Infrastructures.

The PMAC [10] framework represents the outcome of IBM's research into autonomous systems. Although it provides the ability to manage various resources, in the authors' opinion, its core services are very complex, heavyweight and inefficient. Autonomia [11] provides users with all the tools required to specify the appropriate control and management schemes, as well as with suitable services to configure and deploy the required software and network resources and then manage their operation to meet the overall system requirements.

Interesting functionality is provided by OPTIMIS toolkit [12] which enables cloud services optimization covering cloud lifecycle related to construction, deployment and operation. Based on architectural framework, developed programming model, integrated development environment and deployment tools it enables the service providers and consumers automate the management of infrastructure, and aims to improve the resource utilization efficiency. Providers and consumers are enabled to make intelligent deployment decisions based on preferences regarding trust, risk, eco-efficiency and cost. Though the toolkit is equipped with features supporting the all required activities for the Runtime Infrastructure provisioning it lacks the services for the constant governance. Another tool supporting comprehensive Runtime Infrastructure provisioning and governance is SCALR [13] which is cloud management system designed to aid in the scalability, organization and management of public or private cloud infrastructures. It provides integrated framework with rich funcionality for configuration management (Puppet, Chef) [14], compute farms composition and monitoring including auto-scaling capabilities based on some pre-defined policies. The SCALR supports many cloud stacks and provides SDK which can be utilized by developers to implement new extensions according to well defined API. However, the architecture is dense and non-flexible, the policy based management services are limited to horizontal auto-scaling aspects based on several metrics (CPU, Mem utilization) and do not support business rules regarding e.g. the infrastructure leasing cost.

Therefore a motivation behind our research is to construct an appropriate IMF for SOI provisioning with LVC, providing the flexibility typically required in the context of SOI provisioning and its individual phases: construction, activation and governance with following features:

Flexible Architecture aligned with SOI paradigm which enables integration with any elements of Runtime Infrastructure to support required activities within the phases to extend classical provisioning with mechanisms enabling flexible construction of any DEs on behalf of complex applications. This architecture would also enable the integration with other infrastructure tools which can be utilized for particular purpose. The example tools can be those already elaborated (e.g. OpenStack, Puppet, PMAC) or even public clouds providers to support on-demand leasing of resources.

Converged Exposition of Runtime Infrastructure elements through services characterized with open and interoperable interface which enables the efficient orchestration and adaptive management. The proposed construction can provide the unified and standardised access to computing, storage and network resources on behalf of virtualization and provisioning purposes.

Declarative Description of above elaborated aspects in terms of composition of software components exposing Infrastructure Services, provisioning procedures which orchestrate these Services and adaptation policies exploiting MAPE pattern related to many aspects such as horizontal/vertical scaling or runtime optimization regarding defined QoS.

Lightweight Virtualization exploitation that provides better dynamic scaling, workload management within the server, dynamic resizing supporting horizontal and vertical scaling. However the framework can be also exploited on behalf of heavyweight virtualization as well.

The proposed architecture model for the IMF would enable the realisation of adaptable SOI provisioning and is open for future extensions.

3 CONCEPT OF ADAPTABLE SOI PROVISIONING PROCESS

The aim of adaptable SOI provisioning is a flexible deployment of application components and management of the Runtime Infrastructure through shifting of compute, storage and network resources according to management policies defining the whole adaptive management process. These help react to changing business requirements and allocate computer resources in such a way in order to efficiently fulfill QoS requirements. These requirements are common to various virtualization technologies and are deployable entities which are running over SOI virtualized services.

3.1 Required Functionality

The IMF must be designed with ease of administration in mind, 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 predefined SLA contracts. To support individual phases of the adaptable SOI provisioning process, several modules should be distinguished within the IMF. They must provide an abstraction layer for the computing infrastructure, providing control over resource allocation and configuration to automate scaling by adding or removing physical resources, detecting current network topology configuration and querying resource assignments (list of available CPUs, memory, number of NICs) which can be partitioned between LVC instances. Many other complex infrastructure-related tasks, like discovery of physical servers and LVC instances, must also be supported without the need for manual intervention. Tenancy support requires each tenant to be provisioned with a dedicated Virtual

Execution Infrastructure (VEI) instance for application deployment, and an LVC configuration template equipped with all middleware elements. In the context of support for adaptable systems we need to deal with the so-called Horn's requirements, discussed in detail in [26]: they are expressed as a list of characteristics, which includes Self-Defining, Context-Aware, Self-Configuring and Re-configuring, Self-Healing, Open, Self-Optimizing, Self-Protecting, and Anticipatory. The requirements are also precisely described and analyzed by authors in [33]. Such functionality requires the Infrastructure Services to be well-structured and organized into categories describing specific aspect of the physical, virtualized and application infrastructure.

3.2 Taxonomy of Infrastructure Services

The SOI is founded on the concept of the SOA, along with the notions of service providers and service consumers for virtualized computing, networking and storage resources. When the application's demand for a specific resource changes (e.g. additional CPU, RAM, network or I/O bandwidth are needed) a SOA-style request is dispatched to the IMF. Computational infrastructures built according to the SOI paradigm are used for hosting a wide spectrum of applications. Thus, the key challenge is the identification of managed resources within SOI, including the relationships between provisioned and managed elements to provide a converged view on the whole infrastructure.

The S3 SOA reference model [15, 16] specifies some elements, like Operational Systems (OSL), QoS and Governance and Policies that can be mapped to the SOI conceptual model (Figure 3). The administrator of a SOI-compliant IT infrastructure monitors and manages the configuration of specific elements within the S3 layers through specification of QoS goals (*QoS layer*) for provisioned elements, and definition of provisioning procedures and adaptation policies (*Governance and Policies layer*) stored in a centralized repository. The IMF solution for adaptable SOI provisioning must enable the provisioning of Runtime Infrastructure elements within the OSL as a Virtual Execution Infrastructure (VEI) with Application Execution Environments (AEE) for application services.

A reference model of SOI delivers virtualized and physical services that can be categorized as Platform as a Service (PaaS), Infrastructure as a Service (IaaS) and Hardware as a Service (HaaS) infrastructures. Such categorization, in turn, enables the structured orchestration of Infrastructure Services to allow the provision of the Runtime Infrastructure and latter governance according to management policies. Though it also requires an open mechanism for instrumentation of managed resources which are asked for the provisioning, monitoring and management. The presented SOI provisioning scheme enables the automation of management activities for specific QoS contracts and, in case of problems, it attempts to reconfigure the system to fulfill the contracts and enable realization of converged infrastructure framework.

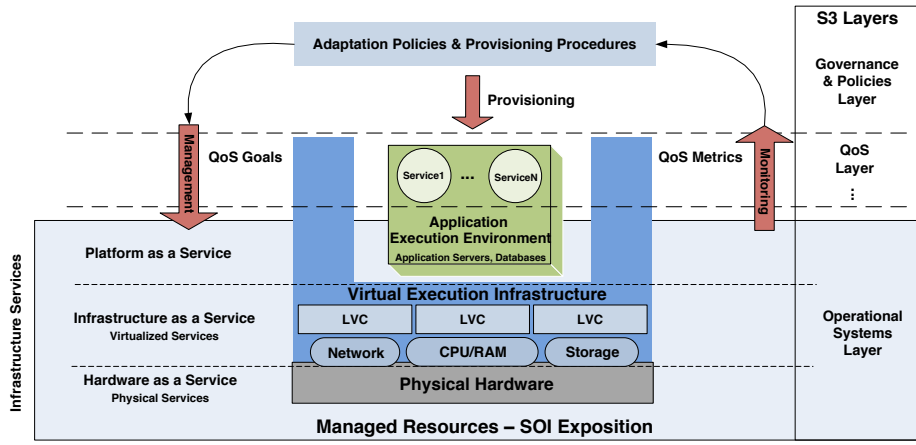

Figure 3. Taxonomy of SOI infrastructure services for adaptable provisioning

3.3 Integration of Cloud Computing and SOI

The concept of exposition of computational infrastructure in the form of services is also exploited by *cloud computing* [17] 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. Cloud platforms are proprietary solutions and not standardized, thus some initiatives like Cloud Resource Model API [18], Open Cloud Computing Interface (OCCI) [19] and Amazon WS [20] define the common elements of a cloud implementation by specification of the relevant machines, storage volumes and networks. Unfortunately, the specifications are not relevant to other aspects like adaptive management of virtualized resources and application middleware (PaaS). The cloud platform for the end user does not provide functionality to allow flexible allocation of resources for already existing VEIs[1].

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. For example Docker [21] framework provides automation services to run any AEE's and applications as a lightweight, portable, self-sufficient container running consistently on and between virtually any server.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. Implementation of such

[1] Amazon EC2, for example, does not provide services for dynamic compute resource reallocation for already running VM instances; instead auto-scaling services scale provisioned VEI with a number of VM instances.

deployments using the SOI paradigm provides a framework for provisioning arbitrary *application service models*[2], *universal access*[3] and *scalable services*[4] thus enabling cloud-bursting or federation of services [22].

3.4 Model of the SOI Provisioning Process

Provisioned application services are multilayered, often clustered, spanning many LVC instances (elements of a particular VEI instance) and involving many infrastructural components, and have complex relationships with each other. As already stated in Section 2, the proposed model extends the concept of classical provisioning providing more flexibility, adaptability and a converged view to complex IT systems by introducing phases:

1. ***Construction***, which determines the required infrastructure elements for user applications with QoS metrics;
2. ***Activation***, that is automatic configuration of each provisioned infrastructure element; and
3. ***Governance***, auditing particular elements in terms of their compliance with QoS metrics, driven by the adaptive management process.

The process facilitates the orchestration of Infrastructure Services and defines the generic IMF architecture to support adaptable provisioning of arbitrary elements of the OSL. Although the model is related to LVC, in our opinion it is generic and can be also used in context of other well known virtualization technologies. The proposed process also defines governance phase according to MAPE (Monitor, Analyze, Plan, and Execute) pattern used by the ACS specific systems as defined by IBM [24]. Provisioning procedures (workflows) define actions to be performed on specific physical resources that are virtualized and exposed via corresponding SOI services, orchestrated for particular activities when triggered by a specific action on the target host. A workflow can define an entire provisioning process, affecting multiple servers, or a single step in a broader reconfiguration process including clustered configurations where many Runtime Infrastructure elements needs to be installed and configured in a correct order using operations on Infrastructure Services. SOI provisioning with LVC can be carried out using LVC-Application Appliances (LVC-AA), which are pre-engineered and validated IT infrastructure units that conform to standards and best practices related to security, availability and performance.

Step 1 – Self-Managing Requirements Specification: Specification of SLA documents which contain QoS metrics for provisioned elements of the OSL running within LVC.

[2] Multi-tenancy, isolated tenancy and mega-tenancy.

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

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

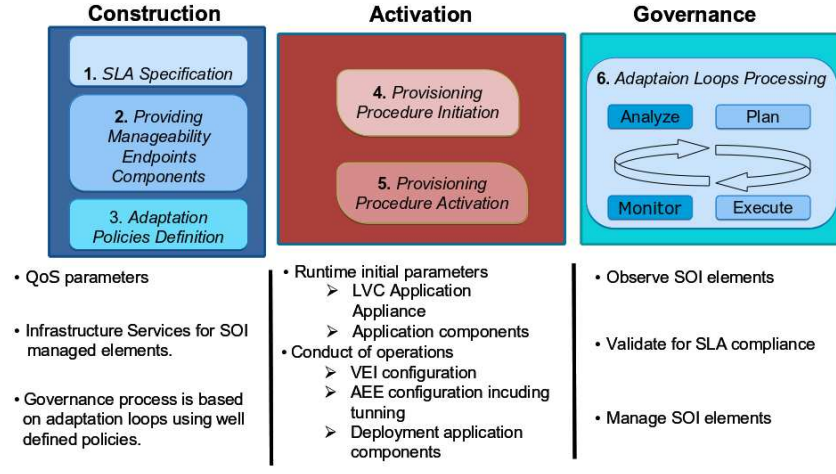

Figure 4. Adaptable SOI provisioning process model

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 carry out the adaptation process. The step requires ensuring that particular infrastructure endpoints, i.e. computing, storage and network resources are properly instrumented and exposed as converged infrastructure.

Step 3 – Adaptation Policy Definition: Once the computational infrastructure is provisioned, continuous management to fulfill the SLA becomes necessary. Each adaptation loop is driven by a specific control algorithm [27] expressed through policies using a particular policy language [23] – Figure 5. The step might require system identification and analysis through practical tests which consider many aspects related to particular applications and working conditions. Control algorithms can rely upon such disciplines as fuzzy logic, control theory or heuristics [28]. The choice of method depends mainly on the complexity of the adaptation process. As computer systems are typically characterized by a high complexity, using hybrid controllers ensures a stable operation and, at the same time, allows the adaptation process to ensure effective QoS [29, 30, 31, 32].

The outcome is a management policy using specific adaptation strategies and parameters which ensure that the managed system will be stable and operate with the required QoS.

Step 4 – Provisioning Procedure Initiation: Prior to activation of the provisioning procedure, the administrator must specify initial runtime parameters required to initiate that procedure. In case of complex applications this might

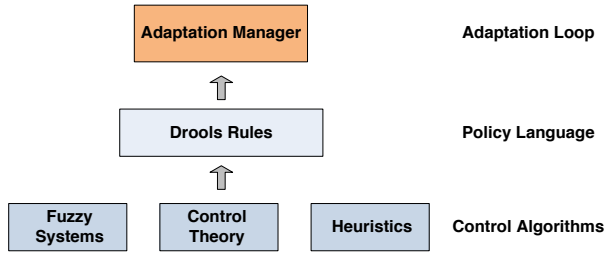

Figure 5. Overview of control algorithms for adaptation loops definition

include "load balancer address" and "number nodes" in the clustered configuration and SOI access points.

Step 5 – Provisioning Procedure Activation: The provisioning procedure is activated and performs activities for LVC installation and contextualization with the required configuration of application resources and component deployment, taking into account the initial tuning phase of AEEs which utilizes performance heurisitcs. The final task is to activate governance activities with adaptation loops.

Step 6 – Adaptation Loops Processing: These include *monitoring*, *analyzing*, *planning* and *execution* activities that provide the *control-loop* functionality according to predefined policies in Step 3.

The adaptable provisioning process for SOI can be realized by introducing an architectural framework equipped with best practices (design patterns and methodologies ITIL [25]). There must be emphasized that the *Construction* phase is performed manually by the system administrator and results i.e. configuration templates, management policies can be reused by the *Activation* and *Governance* which are fully automated. In the next sections authors present A-SOI-PP which is an implementation of the IMF that meets the presented business goals, objectives and enables realization of converged infrastructure concept governed by the adaptable management activities.

4 ARCHITECTURE OF A-SOI-PP

The essence of A-SOI-PP lies in its architecture, which is expected to respond to constantly changing business requirements and ensure flexibility and agility, both crucial for the adoption of new technologies. To support various elements of the adaptable SOI provisioning process, modules that will provide certain services, should be distinguished within the A-SOI-PP.

4.1 Generic Building Blocks

The A-SOI-PP conceptual model is translated into an architecture which comprises several major modules, presented in Figure 6. The architecture is consistent with the SOI Reference Model and fulfills the requirements specified in Section 3.1. Individual modules are integrated by the common model of understanding the infrastructure components that describes the relation between VEI and AEE supporting the realization of the process of adaptable SOI provisioning with LVC or other virtualization technologies.

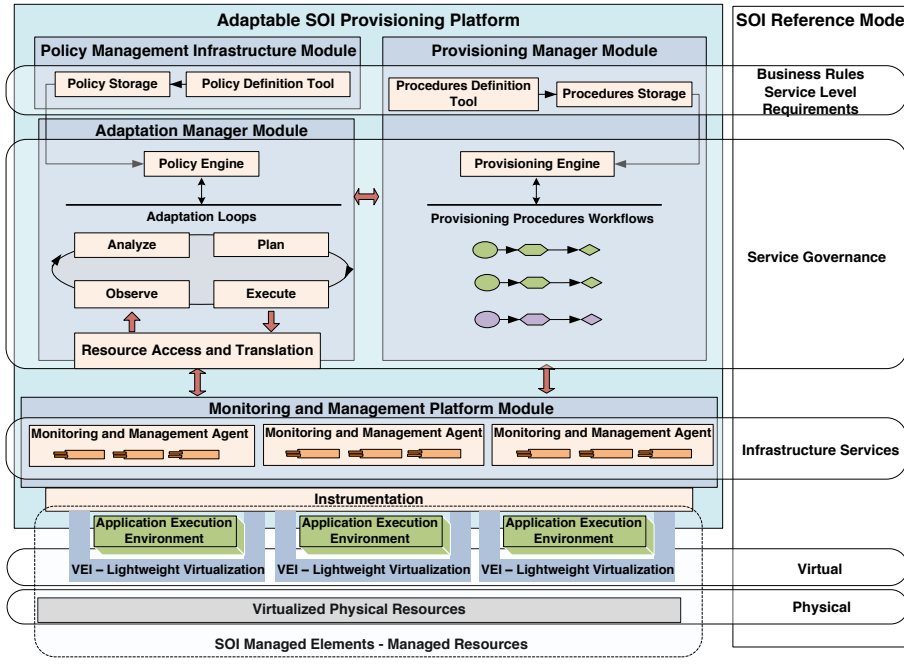

Figure 6. A-SOI-PP software modules

The *Monitoring and Management Platform* (MMP) module exposes a unified interface to manage or to provide SOI Managed Elements. The interface provides converged access to Virtualized or Physical Infrastructure Services, exposing infrastructure elements for the purposes of observation and control. The platform can be modular in the sense that monitoring and management agents (MMA) are equipped with software components delivering sensor and effector interfaces. Sensors enable access to the state of SOI Managed Elements while effectors provide the functionality to alter their state.

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 computing resources, as specified

by the system administrator. The Provisioning Engine can manage many procedure instances concurrently. Upon provisioning, the infrastructure must be constantly monitored to track SLA contract fulfillment; this process is supervised by the *Adaptation Manager*.

The *Adaptation Manager* (AM) is a module responsible for controlling the SOI Managed Elements sensing the performance degradation and determining corrective actions to be taken. The self-adapting process is driven by adaptation loops, which are defined on the basis of a specific adaptation policy. The architecture of adaptation loops is based on ACS and consists of monitoring, analysing, planning and execution phases, each of which requires comprehensive support from IT infrastructure, tools and management systems. The module is also equipped with a Translation Layer internal service, representing a uniform abstraction formalism, providing access to 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. Each policy specifies adaptation loop activities to be undertaken by the provisioning procedure and defines control algorithms suitable for a particular self-adapting process. Policies are stored in a persistent storage to be reused by provisioning procedures when activating a specific adaptation loop.

4.2 Layered Architecture

The presented conceptual model of A-SOI-PP promotes modular design and exposes Infrastructure Services for flexible provisioning and adaptive management, supporting the adaptable SOI provisioning process model. The proposed layered architecture consists of nine layers (Figure 7), which correspond to the previously described generic building blocks. The architecture is characterized by the openness of exposed interfaces and the ability to manage any SOI Managed Elements VEIs based on arbitrary LVC technologies or AEE with any running software product. It supports cross-vendor interoperability and scalability, allowing many managed elements to be handled simultaneously. Exposed interfaces are standardized by leveraging and implementing industry-standard IT protocols. The individual layers consist of specific modules, where each layer plays a key role in the process of adaptable SOI provisioning and is based on the characteristics of adjacent layers. Each layer is decomposed into software components whose implementation relies on appropriate architectural design patterns to promote managed element independence. Each of these components is, in turn, installed within the A-SOI-PP environment.

The *Operational Services Layer* (OSL) is the bottom layer of the system, with heterogeneous infrastructure resources to deliver a full-featured Runtime Infrastructure environment for application services including physical hardware, virtualization platforms and middleware. Because it must be accesible with common access interface following industry, the next layers provide appropriate abstractions.

In the *Resource Access Layer* (RAL) OSL elements expose access interfaces with dedicated tools and software which is not accessed directly (due to its specificity or heterogeneity) and must therefore be covered by an additional layer of abstraction to enable better provisioning and governance. The RAL uses vendor-specific interfaces and different methods of communication with the infrastructure elements of the OSL, enabling instrumentation. In addition to LVC, RAL also enables seamless integration with any product in the VM virtualization technology, cloud platforms (OpenNebula, Nimbus, Amazon AWS), middleware (J2EE application servers) and dedicated provisioning tools (N1 [36]), thus facilitating provisioning (based on LVC-AA) and configuration (via resource and service managers, and other management tools). This layer is equipped with software components which for integration with underlying elements of the OSL can use native access interfaces like shell scripts, APIs (EC2 or Java Open Cloud API) or native libraries.

The *Runtime Infrastructure as a Service Layer* (RIaaSL) provides Infrastructure Services through the usage of composite *software components* responsible for exposing particular vendor-specific interfaces as parts of a common API, providing a set of interfaces used for provisioning, monitoring and management of HaaS, IaaS and PaaS elements. These interfaces are coherent and conform to specific resource specification thus they constitute a common and open resource interface for constructing SOI Managed Elements according to converged infrastructure concept.

The *Self-Configuration Services Layer* (SCSL) provides facilities that ensure adaptability and context-awareness of MMP. *Discovery* service components enable localization of SOI Managed Elements and can dynamically handle lists of available virtualized physical servers, running LVC, application middleware and other devices. This self-configuration capability implies dynamic reaction to failures and the capability to dynamically and automatically modify SOI Managed Elements. *Discovery Responder* components use discovery protocols based on multicast or other frameworks corresponding to *Infrastructure Registry* services with MMA addresses. *Dynamic loading* ensures automatic deployment of modules containing Manageability Endpoints as well as software components that expose services for provisioning, monitoring and management of SOI. Such *auto-configuration* features enable MMA to learn the characteristics of the Runtime Infrastructure and load proper modules with sensors and effectors, depending on the discovered features of the environment where a given agent is started. The process uses module descriptors dedicated for specific hardware and OS platforms, following which of the actual components can be downloaded from the *Components Repository* service.

The *Information Services Layer* (ISL) provides 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. Such information is stored in the *Infrastructure Registry*, which registers for notifications from the *Discovery Responder* components running on particular instance of the MMA. Notification emitters send or redirect notifications to other interested parts like PM, AM or GUI management consoles. These notifications contain in-

formation about the state changes of particular SOI Managed Elements, which e.g. include increased resource usage (CPU, memory) and a lack of resources, like increased garbage collection activities within a specific JVM (Java Virtual Machine) instance.

The *Connectivity Services Layer* (CSL) provides communication service components exposing common and open access methods for SOI Managed Elements. Due to its connector-based architecture, the layer allows any type of protocol connector to be plugged in, including RMI, SOAP, REST, etc. This feature is very important, since the Adaptation and Provisioning Managers can be implemented in 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.

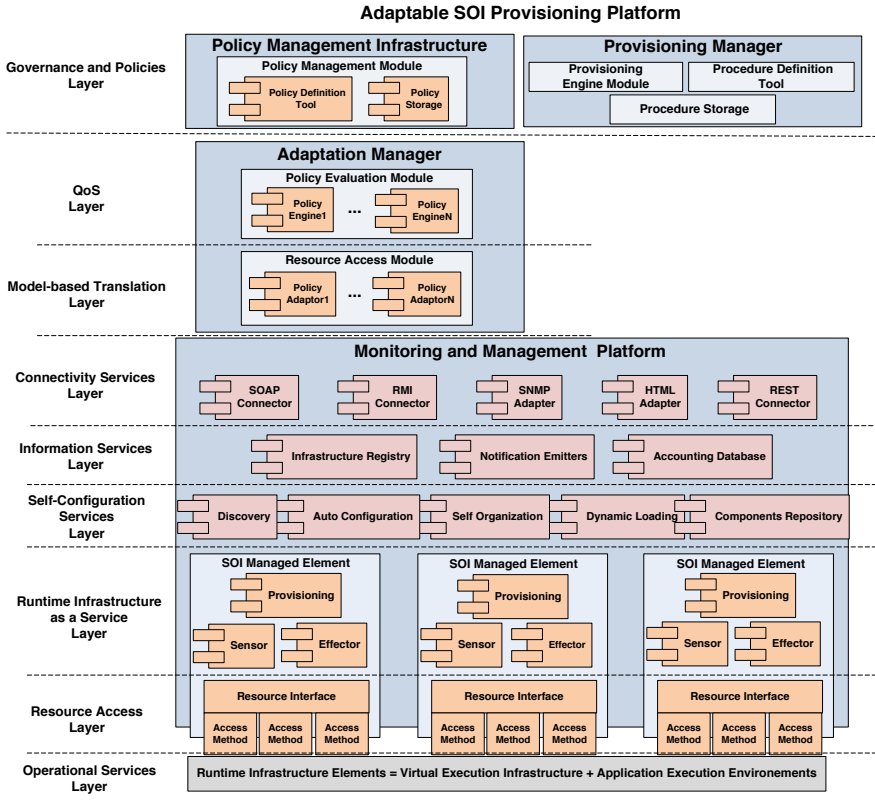

Figure 7. Layered architecture of the A-SOI-PP

The *Model-based Translation Layer* (MbTL) is an intermediate layer, generic and not related to any particular technology or class of managed resources. It is orthogonal to the management policy performed by the AM, 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.

The *QoS Layer* (QoSL) provides a *Policy Evaluation Module* (PEM) for the AM, whose key requirement is management of control loops. Several reasoners can be installed and operate at the same time, ensuring flexibility through support for a wide range of policy engines. Differences between reasoners are resolved by the *Policy Engine* components which initialize specific reasoner adaptors. Policy definitions are obtained from the PMI, instantiated by the appropriate Policy Engine and evaluated.

Finally, the *Governance and Policies Layer* (GaPL) provides a graphical interface for the system administrator available through GUI management consoles, enabling system configurations, policies and procedure definitions related to SOI provisioning and management to be listed and edited. This layer also provides tools for the initialization of the SOI provisioning process responsible for initial allocation of resources required by middleware services consumed by applications. This activity is an element of the specific provisioning procedure which includes initial verification of the availability of resources at the level required to ensure the requested QoS. The GaPL provides policies used by adaptation loops for management of SLA, including capacity, performance and security. Graphical management consoles expose charts with monitoring metrics for OSL elements and support manual configuration of some elements.

5 IMPLEMENTATION

From among many well-known and widely available management/monitoring tools and protocols, the authors have selected the Java language and the Java Managed Extensions (JMX) [37] platform for the implementation of the MMP [33]. JMX is an open and standardized management framework widely used throughout the industry to instrument managed resources, enabling state observation and invocation of operations through notion of so-called MBean components. It provides service 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 thus supporting open and flexible architecture. These JMX components can be deployed on demand to MMA to fulfill the requirements of a particular provisioning process 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. Such a process requires a MLet descriptor which defines the information on the MBeans to be obtained and makes it possible to create dynamically extensible agents.

```
<mlet
  code="org.jims.modules.solaris.solaris10.mbeans.ZoneAgent"
  archive="sunos-sparc-3.0.0.jar,dependencies.jar"
  name="Management:class=Solaris10Management" version="1.0">
</mlet>
```

The PM module uses JBoss jBPM [40] supporting the entire lifecycle of the provisioning process (authoring, execution, monitoring and management) and provides tight, powerful integration with business rules and event processing – Figure 8.

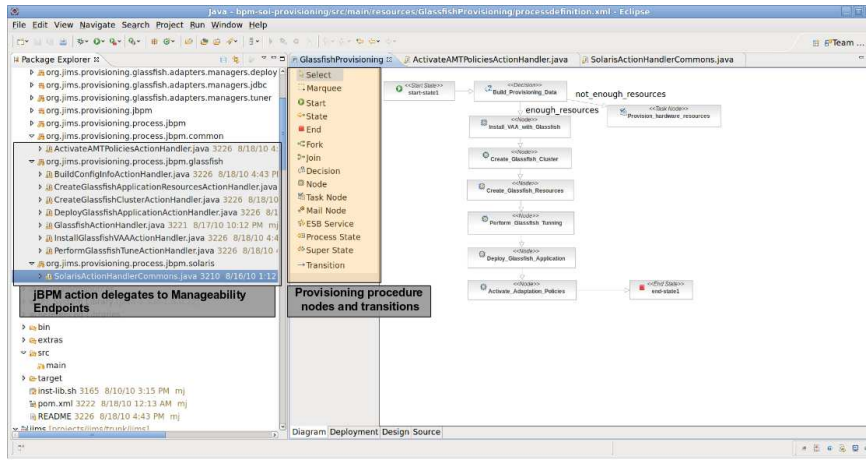

Figure 8. Provisioning console

As the technology for implementation of the GUI system, JBoss RHQ [39] and Google Web Toolkit (GWT) [38] have been chosen. JBoss RHQ is the open platform that delivers a core user interface for monitoring and management of IT infrastructures. The RHQ is designed with layered modules that provide a flexible architecture for deployment of plug-ins which implement abstraction interfaces suitable for arbitrary SOI Managed Elements.

The AM contains modules responsible for adaptation loop management and integration with the PMI and MMP – Figure 11. It is implemented as a lightweight library that can be deployed on demand within the MMP and is based on the JMX technology. The implementation of the AM system exploits the potential of the rule engine-based [41] approach as attractive solution for the policy-driven SOI provisioning process. 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 9). The AM supports reasoners based on Drools, Jess and even PMAC policy languages [27].

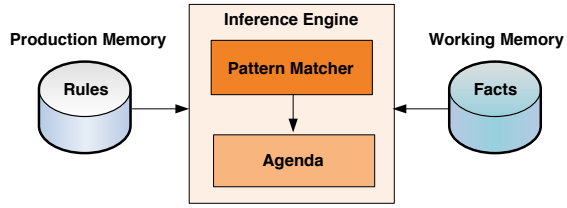

Figure 9. Rule engine structure based on rete networks

In the PEM, SOI Managed Elements are represented as facts processed by the rule engine – Figure 10. Such exposition is performed during the AM's start-up phase through the Policy Adaptor component. Its main responsibility is to provide access to selected resources and policies and to gather events from resources and provides them to the reasoner. 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.

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 Policy Adaptor interface is used in this step, which attaches the 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.

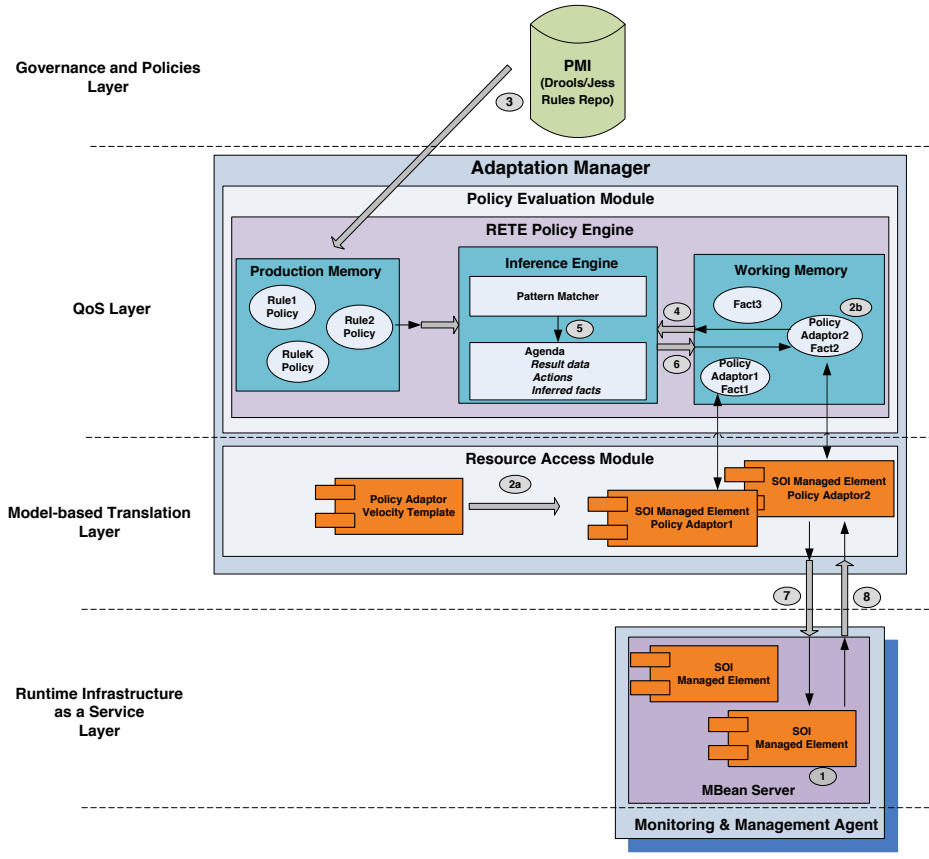

Figure 10. Adaptation loop processing performed by the PEM using rule engine

Applying the aforementioned software technologies and frameworks for implementation of A-SOI-PP yields an IMF that conforms to the adaptable architecture requirement and realizes the defined SOI provisioning process. The implemented A-SOI-PP is based on modular architecture and component design equipped with many advanced techniques of software engineering like design patterns, reflectivity [34] and compositional adaptation [35]. Utilized technologies are based on the industry standards thus ensuring interoperability requirements. Such implementation promotes flexibility and can be extended with components supporting arbitrary elements of the Operational Services Layer exposed as SOI Managed Elements which are provided and supervised by particular modules of the platform.

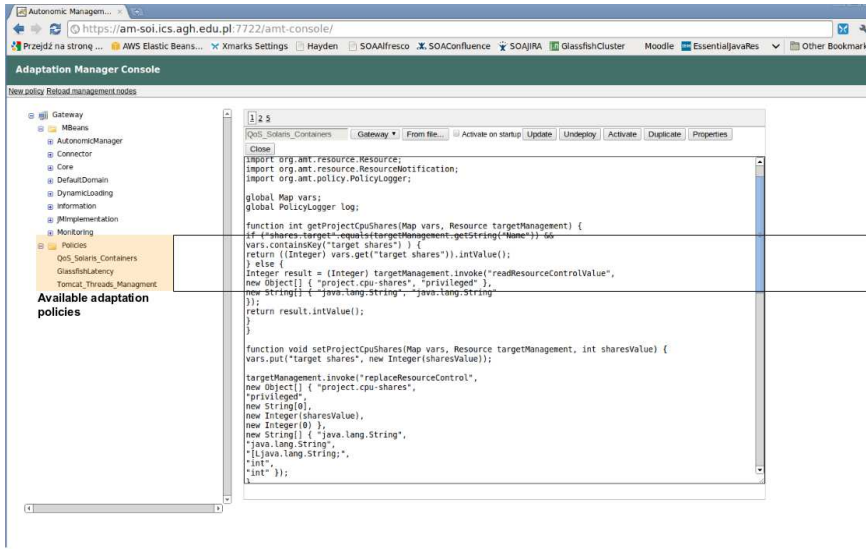

Figure 11. Web console of the Policy Definition Tool and adaptation loop management

6 CASE STUDY

For demonstration purposes the authors have selected Java Pet Store 2.0 [42], which is a multi-layered application delivered by the Java BluePrint program. The application utilizes the Runtime Infrastructure, provisioned according to the *Isolated Tenancy* model with dedicated JEE clustered Glassfish (PaaS layer) application server and MySQL database running over Solaris LVC (IaaS layer). This setup is depicted in Figure 12. Preparing such an infrastructure requires many complex actions which must be performed in the correct order, as explained below. The actions must include some technological aspects related to virtualization technology, scheduling algorithms and available management interfaces. They are supported and automated by the platform supporting the self-configuration properties. The only responsibility of the system administrator is a definition of LVC-AA templates with pre-configured middleware and manageability endpoints which are automatically configured, activated and discovered during provisioning process.

VEI provisioned over Solaris Containers [43] would deliver predictable levels of QoS, owing to scheduling algorithms enforced by the Solaris Resource Manager. CPU resource consumption within the Solaris Global Zone (Solaris OS Kernel that hosts LVCs) is managed through Dynamic Compute Resource Pools (DRP) with default Time Shared Scheduler (TSS) which is assigned to provisioned Container instances that manage the Glassfish environment. When considering CPU resource consumption within a LVC with running database, we must consider a situation where the tools monopolize available processor cycles and impact the database per-

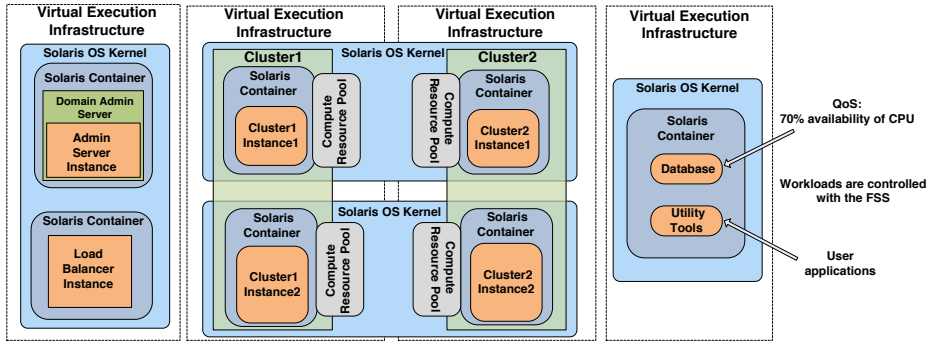

Figure 12. Runtime Infrastructure exploiting Solaris LVC, Glassfish and MySQL db

formance. This might lead to a situation where important database transactions suffer from insufficient CPU time to complete their work. The Fair Share Scheduler (FSS) provides a precise control of CPU consumption, allowing the optimal system CPU resource usage as determined by CPU-share resource control [44] settings. The only challenge is controlling the quantity of the appropriate resource control value (*project.cpu-shares* [43] in the use-case), which can be managed by the adaptation loop.

Platform tuning is a sophisticated task demanding expert knowledge about the used OS, application server and database. It requires setting many runtime configuration parameters that all depend on each other and, adjusted appropriately, increase the application performance. In the presented case study (in the context of a Glassfish application server) information, such as the number of available processors (including their architecture), memory to be allocated for the JVM operating on a cluster instance and a garbage collection strategy, needs to be specified. Glassfish is eqipued with tuner service (Glassfish Performance Advisor – Figure 13) which utilizes internal heuristics to determine the optimal configuration for particular server instances.

The tuning exploits custom implemented components, which interact with the tuner during 'Step 4' of the process, automating the whole functionality. The drawback of using such an approach is the requirement to restart each server instance whenever the 'tuned' configuration is to be applied. Even in the case of clustered configuration, this can become a problem, as the remaining instances are required to process an increased number of requests, which can lead to instantaneous performance degradation. The following operations are required when provisioning such a runtime infrastructure to end users, pursuant to the process described in Section 3.4.

Step 1 – Self-Managing Requirement Specification: The QoS goal of control is ensuring constant allocation of the processor in conditions of variable and constant disturbance, i.e. database workload within the Solaris Container is

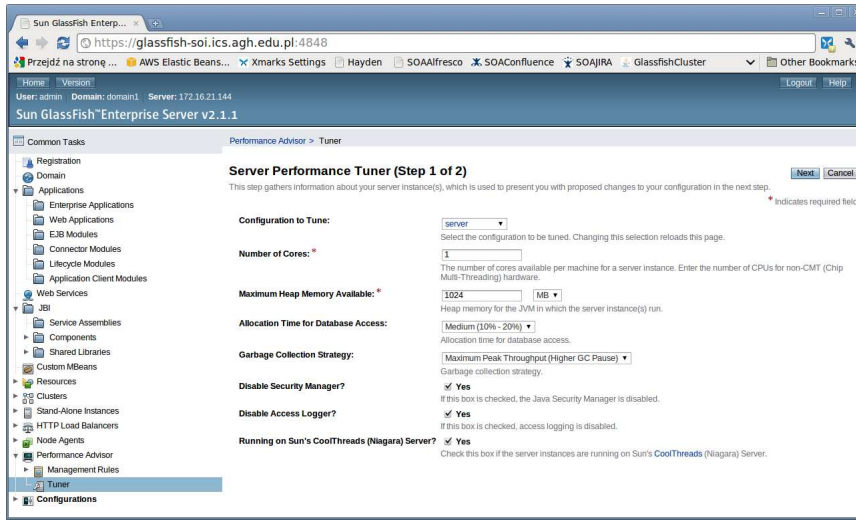

Figure 13. Glassfish performance advisor console

guaranteed to use **70 %** of CPU regardless of the number of other active projects (disturbances).

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 LVC, with adaptation loops, ensuring guaranteed CPU reservation for database workloads. The discussed provisioning activities require Infrastructure Services for managing Glassfish and Solaris LVC platforms. Glassfish infrastructure provides a framework (Appserver Management Extensions – AMX) enabling centralized provisioning of a cluster of instances on multiple hosts and application deployments, support scalability, load balancing, failover protection and high availability. The A-SOI-PP extensions for the Glassfish platform use AMX and support management of core elements, such as initial configurations of node agents running within the provisioned LVC instance and automatic tuning processes.

Step 3 – Adaptation Policies Definition: For this particular case we have proposed an open-loop regulator using the FSS model extended with appropriate heuristics. The system administrator expresses the importance of each workload as the number of shares (which are not the same as CPU percentages – rather, shares define the relative importance of a given active workload in relation to other active workloads). If

1. S_w – shares assigned to workload W;
2. N – number of active workloads;
3. S_i – shares assigned to active workload, $i = \{1, \ldots, N\}$,

then the relative entitlement E_w of workload W can be expressed by the following equation:

$$\mathbf{E}_w = \frac{\mathrm{S}_w}{\sum_i^N S_i}. \tag{1}$$

The open-loop regulator takes into account the fact that the scheduler considers only active workloads and, if a given workload is not CPU-bound (consumes all available resources), then the remaining CPU portions might be consumed by other workloads. Let us assume that:

1. number of workloads $N_w \geq 2$;
2. number of active workload changes at time t according to activity state vector $A^t = [A_1^t, \ldots, A_{N_w}^t]$ where $A_i^t = 0$ if W_i is not active and $A_i^t = 1$ if W_i is active; $i = 1, \ldots, N_w$,
3. each workload is CPU-bound and has S_i shares allocated,
4. U_w – required usage of CPU by workload W_w.

Following the mathematical transformations prescribed by Equation (1), we obtain Equation (2) specifying the number shares S_w to be assigned to workload W_w:

$$\mathbf{S}_w^t = \frac{\left(U_w \sum_{i \neq s}^{N_w} S_i * A_i^t\right)}{(1 - U_w)}, \tag{2}$$

where $\sum_{i \neq s}^{N_w} S_i * A_i^t$ is the disturbance monitored by the manager, equal to the sum of all active workload shares, excluding workload W_w at interval Δt. To provide the stability of the managed system there were also utilized heuristics which result from the complexity of the managed LVC infrastructure and must be exploited in order to ensure that the proposed adaptation process is an efficient solution for maintaining the required QoS [29]. For instance, if we deal with the variable disturbances many important factors influence definition of control policy. 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 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 the controlled database workload (db) 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 rolling average, similar to the Jacobson algorithm [45] used in the TCP/IP protocol to smooth measured values (Equation (3)), where φ – Jacobson coefficient. Applying these rules provides satisfactory results as

to the quality of applied control algorithm and strictly depends on the value of the Jacobson coefficient.

$$NU_w^t = \varphi * U_w^{t-1} + (1 - \varphi) * U_w^t \tag{3}$$

Selection of the appropriate controller settings for the Jacobson coefficient is 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 (4).

$$I_e = \sum_{k=0}^{M} e(kT_O)^2. \tag{4}$$

The control algorithm based on the FSS model and heuristics is defined with policies using the PMI based on Drools policy language.

Step 4 – Provisioning Procedure Initiation: Each provisioned LVC instance must be defined in terms of virtualized storage pool containing the LVC filesystem, physical interfaces (virtualized network [7]), Dynamic Resource Pools (number of CPUs) and OS schedulers (TSS used by LVC hosting Glassfish cluster and FSS by LVC with database workloads). Two instances are specified for newly created cluster nodes and the location of physical servers (LVC placement, i.e. where the LVC instances are provisioned) is calculated by applying a simple round-robin strategy.

Step 5 – Provisioning Procedure Activation: Once all the required information is specified, the provisioning procedure can be successfully invoked. LVCs are installed and the Glassfish clustered configuration is automatically tuned. Subsequently, application components are deployed. The application services use a database running within the LVC that must be supervised by the Adaptation Manager to ensure appropriate CPU availability (according to QoS metrics).

Step 6 – Adaptation Loop Processing: The use of an open-loop controller under conditions of disturbance (defined by square wave function [46]) ensures that the system may continue to operate at a preset level – Figure 14. However, in addition to the algorithm used, adaptation policies also involved a set of heuristics already described in Step 3 to maintain the stability of the managed system. If comparing to case when there is no control loop (green-line), the quality of the system is ensured (red-line) to reach the 70 % of the CPU. The selection of an appropriate period that takes the inertia of the FSS into account is crucial and also affects the controller interval T_O. This is especially important in case of variable disturbances when FSS must adapt itself to satisfy CPU consumption according to assigned shares (internal tests showed that this process requires 60 seconds). In the analyzed scenarios, the controller interval was equal to $T_O = 120$ seconds, but thread responsible for acquiring monitoring data was activated each $\Delta T = 5$ seconds and stored metrics in vector, whose values were then averaged for the control error calculation. The Jacobson's coefficient was

set to value $\varphi = 0.4$. 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.

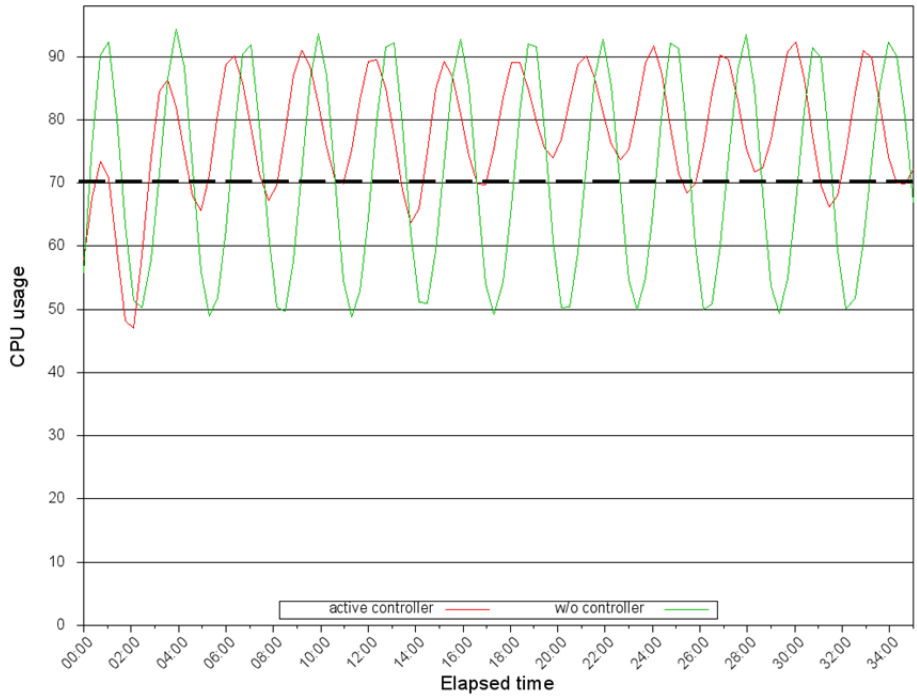

Figure 14. Adaptive management of database workload

The presented case study exploits the model of adaptable SOI provisioning with LVC utilizing Solaris, Glassfish and MySQL platforms, with a particular attention to guaranteeing the availability of computing resources for the container in which the database is running. The provisioning process was fully automated, organized according to a defined model and supported by the A-SOI-PP. Though the JEE and Solaris OS were used the presented methodology can be aplied to other systems as well. The only step which is platform specific is 'Step 2' which provides specific manageability endpoints exposing sensors and effectors for particular elements of the Runtime Infrastructure.

7 SUMMARY

The proposed and implemented architecture of A-SOI-PP 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 and governance utilizing policies defined by the input of a particular provisioning process. The platform supports adaptable provisioning based on a flexible architecture integrating IaaS and PaaS layers according to converged computing architectures requirements. Moreover, it supports definition, deployment and management based on MAPE pattern of arbitrary Runtime Infrastructure topologies, as required by the application services. The architecture is based on modular structure and implemented with components utilizing JMX technology which were verified for Solaris and JEE technologies, and in opinion of authors, it would very efficient solution for other platforms as well. The A-SOI-PP is open, general in its nature and can be applied to different levels of a distributed computing system to be provisioned and managed, starting from physical resources, through virtualized operating systems, up to application servers and software components. The existing practical applications of virtualization technology are dominated by the heavy-weight virtualization techniques. However, it is the authors' 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 resources allocation.

The A-SOI-PP can be adapted in the future to comply other solutions such as OpenVZ or heavyweight virtualization technologies. An important feature is the possibility to define any procedure for creating a VEI by using lightweight containers, configuration templates and policy based management with hybrid controllers that use control theory and heuristics to ensure the specified QoS.

In the proposed model the activities of the Construction phase are not automated which is the element of the potential improvement. As it was presented in the case study this can especially affect the steps necessary to choice the adaptation strategy which is a very complex process and demands an expert and semantic knowledge in the domain problem, and it can include the other algorithms based e.g. on queuing theory or other self-learning techniques. The use of advanced processing tasks using Complex Event Processing [47] 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.

The authors are convinced that the implemented A-SOI-PP is a very solid base for expanding support for other applications to create a universal solution.

Acknowledgment

The presented research was supported by funding from the European Regional Development Fund No. POIG.01.03.01-00-008/08 and MNSiW No. 15.11.230.267.

REFERENCES

[1] Gassman, B.: Provisioning IT Services. Gartner Group, October 2002.

[2] The Open Group, SOA Working Group: SOI Reference Framework. 2008.

[3] Garber, L.: Converged Infrastructure: Addressing the Efficiency Challenge. Computer, Vol. 45, 2012, No. 8, pp. 17–20.

[4] Vaughan-Nichols, S. J.: New Approach to Virtualization Is a Lightweight. Computer, Vol. 39, 2006, No. 11, pp. 12–14.

[5] Gross-Hohnacker, B.: Server Virtualization on Linux – Analysis and Evaluation. 2006.

[6] Carlson, L.: Programming for PaaS – A Practical Guide to Coding for Platform-as-a-Service. O'Reilly Media, 2013, ISBN: 978-1-4493-3490-1.

[7] Jarząb, M.—Kosinski, J.—Zieliński K.: Virtualization of Grid Networking Resources for Computation Mobility Support. Computational Methods in Science and Technology, Special Issue, 2010, pp. 35–44.

[8] Delgado, V.: Exploring The Limits of Cloud Computing. M.Sc. Thesis, KTH Information and Communication Technology, 2010, `http://upcommons.upc.edu/pfc/bitstream/2099.1/13421/1/VDelgadothesis.pdf`.

[9] Nimbus Platform: `http://www.nimbusproject.org`.

[10] Chruz, C.—Petty, C.: PMAC Dev Guide. IBM Redbook Guide, 2006.

[11] Dong, X.—Hariri, S.—Xue, L.—Chen, H.—Zhang, M.—Pavuluri, S.—Rao, S.: Autonomia: An Autonomic Computing Environment. Proceedings of IEEE International Performance, Computing, and Communications Conference 2003, pp. 61–68.

[12] Ferrer, A. J.—Hernández, F.—Tordsson, J.—Elmroth, E.—Ali-Eldin, A.—Zsigri, C.—Sirvent, R.—Guitart, J.—Badia, R. M.—Djemame, K.—Ziegler, W.—Dimitrakos, T.—Nair, S. K.—Kousiouris, G.—Konstanteli, K.—Varvarigou, T.—Hudzia, B.—Kipp, A.—Wesner, S.—Corrales, M.—Forgó, N.—Sharif, T.—Sheridan, C.: Optimis: A Holistic Approach to Cloud Service Provisioning. Future Generation Computer Systems, Vol. 28, 2012, No. 1, pp. 66–77.

[13] Scalr White Paper, Architecting the Right Stack for Your Enterprise Cloud. `http://www.scalr.com/white-paper-architecting-the-right-stack`.

[14] Spinellis, D.: Don't Install Software by Hand. Software, IEEE, Vol. 29, 2012, No. 4, pp. 86–87.

[15] Zieliński, K.—Szydlo, T.—Szymacha, R.—Kosinski, J.—Kosinska, J.—Jarzab, M.: Adaptive SOA Solution Stack. IEEE Transactions on Services Computing, Vol. 5, 2012, No. 2, pp. 149–163.

[16] Jarzab, M.—Hamiga, M.: An Analysis of Methods for Sharing an Electronic Platform of Public Administration Services Using Cloud Computing and Service Oriented Architecture. Computer Science, AGH Press, Vol. 13, 2012, No. 4, pp. 115–132.

[17] Liang-Jie, Z.—Zhou, Q.: CCOA: Cloud Computing Open Architecture. Proceedings of the IEEE International Conference on Web Services (ICWS 2009), 2009, pp. 607–616.

[18] 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.

[19] Open Community Leading Cloud Standards: Open Cloud Computing Interface. http://occi-wg.org.

[20] Amazon WS: http://aws.amazon.com.

[21] Docker Engine: http://www.docker.io.

[22] Rajkumar, B.—Rajiv, R.—Rodrigo, N.: InterCloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services. Proceedings of the 10th International Conference on Algorithms and Architectures for Parallel Processing, 2010, Part I, pp. 13–31.

[23] de Leusse, P.—Kwolek, B.—Zieliński, K.: A Common Interface for Multi-Rule-Engine Distributed Systems. Proceedings of the 4th International Web Rule Symposium (RuleML-2010), 2010, pp. 21–23.

[24] Nami, R. M.—Bertels, K.: A Survey of Autonomic Computing Systems. Proceedings of the Third International Conference on Autonomic and Autonomous Systems, 2007.

[25] Powell, B.: IT Infrastructure Library (ITIL) V3: Overview and Impact, Global Technology Services, ITS Strategy and Architecture.

[26] Horn P.: Autonomic Computing: IBM's Perspective on the State of Information Technology. VSA Partners, Inc., USA, 2001.

[27] Adamczyk, J.—Chojnacki, R.—Jarząb, M.—Zieliński, K.: Rule Engine Based Lightweight Framework for Adaptive and Autonomic Computing. Proceedings of the 8th International Conference on Computational Science, Part I, LNCS, Vol. 5101, 2008, pp. 355–364.

[28] Hellerstein, J. L.—Diao, Y.—Parekh, S.—Tilbury, D. M.: Feedback Control of Computing Systems. Wiley-IEEE Press, August 24, 2004, ISBN: 978-0-471-26637-2.

[29] Jarzab, M.—Zieliński, K.: Framework for Consolidated Workload Adaptive Management. Proceedings of the IFIP CEE-SET 2007, Software Engineering in Progress, 2007, pp. 17–30.

[30] Slota, R.—Nikolow, D.—Skalkowski, K.—Kitowski, J.: Management of Data Access with Quality of Service in PL-Grid Enironment. Computing and Informatics, Vol. 31, 2012, No. 2, pp. 463–479.

[31] Nikolow, D.—Slota, R.—Lakovic, D.—Winiarczyk, P.—Pogoda, M.—Kitowski, J.: Management Methods in SLA-Aware Distributed Storage Systems. Computer Science, AGH Press, Vol. 13, 2012, No. 3, pp. 35–44.

[32] Gmach, D.—Krompass, S.—Scholz, A.—Wimmer, M.—Kemper, A.: Adaptive Quality of Service Management for Enterprise Services. ACM Transactions on the Web, Vol. 2, 2008, No. 1, Art. No. 8.

[33] Bałos, K.—Jarząb, M.—Wieczorek, D.—Zieliński, K.: Open Interface for Autonomic Management of Virtualized Resources in Complex Systems – Construction Methodology. Future Generation Computer System, Vol. 24, Vol. 5, 2008, pp. 390–401.

[34] 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. Proceedings of the 2nd International Conference on Meta-Level Architectures and Reflection 99, Springer, 1999, pp. 115–134.

[35] McKinley, P. K.—Sadjadi, S. M.—Kasten, E. P.—Cheng, B. H. C.: A Taxonomy of Compositional Adaptation. Software Engineering and Network Systems Laboratory, Michigan State University, TR, MSU-CSE-04-17, May 2004.

[36] Jarząb, M.—Kosiński, J.—Zieliński, K.: Role of N1 Technology in the Next Generation Grids Middleware. Proceedings of the European Grid Conference 2005, LNCS, Vol. 3470, 2005, pp. 942–951.

[37] Java Management Extensions Specification – Version 1.4, 2006, `http://docs.oracle.com/javase/7/docs/technotes/guides/jmx/JMX_1_4_specification.pdf`.

[38] Tacy, A.—Hanson, R.—Essington, J.—Tokke, A.: GWT in Action. Manning Publications, 2013, ISBN-10: 1935182846.

[39] JBoss RHQ Platform: `http://rhq-project.org/`.

[40] JBoss jBPM Suite: `http://www.jboss.org/jbpm`.

[41] Forgy, Ch. L.: Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. Book Expert Systems, IEEE Computer Society Press, ISBN: 0-8186-8904-8, 1990, pp. 324–341.

[42] Java Petstore Reference Application, Oracle Corporation, `http://www.oracle.com/technetwork/java/index-136650.html`.

[43] Lageman, M.: Solaris Containers – What They Are and How to Use Them. SUN Microsystems Blueprint, 2005.

[44] Kay, J.—Lauder, P.: A Fair Share Scheduler. Communication of the ACM, Vol. 31, 1988, No. 1, pp. 44–55.

[45] Jacobson, V.: Congestion Avoidance and Control. Proceedings of the Symposium on Communications Architectures and Protocols (SIGCOMM '88), 1988, pp. 314–329.

[46] Abramowitz, Stegun: Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables. National Bureau of Standards Applied Mathematics Series, Vol. 55, 1964, pp. 1020.

[47] Balis, B.—Kowalewski, B.—Bubak, M.: Real-Time Grid Monitoring Based on Complex Event Processing. Future Generation Computer System, Vol. 27, 2011, No. 8, pp. 1103–1112.

Marcin Jarząb received his Ph.D. degree in computer science from the University of Science and Technology (AGH – UST) in Kraków, Poland, in 2012. Currently he works in Samsung Reserach Poland, beeing responsible for architecting Smart Cloud Solutions. He is also Head of Platform Architecture and Security Department. He worked as a software consultant at Consol Solutions and Software from 2000–2002, participating in many projects for Telco companies. He was an intern at Sun Labs in the latter half of 2003, investigating the application of the multi-tasking Java Virtual Machine to the JEE environment. His research interests include tuning and performance evaluation of distributed systems, design patterns, frameworks, lightweight virtualization technologies, and architectures of autonomic computing environments.

Krzysztof Zieliński is Full Professor and Head of Institute of Computer Science at AGH – UST. His interests focus on networking, mobile and wireless systems, distributed computing, and service-oriented distributed systems engineering. He is an author of over 200 papers in this area. He has been Project/Task Leader of numerous EU-funded projects, like e.g. PRO-ACCESS, 6WINIT, Ambient Networks. He served as an expert with Ministry of Science and Education. His research interest concerns adaptive SOA solution stack, services composition, service delivery platforms and methodology. He is a member of IEEE, ACM and Polish Academy of Sciences, Computer Science Chapter.