Towards the Future Internet
G. Tselentis et al. (Eds.)
IOS Press, 2009

doi:10.3233/978-1-60750-007-0-227

Design for Future Internet Service Infrastructures

B. Rochwerger[1], A. Galis[2], D. Breitgand[1], E. Levy[3], J. A. Cáceres[4], I. M. Llorente[5], Y. Wolfsthal[1], M. Wusthoff[3], S. Clayman[2], C. Chapman[2], W. Emmerich[2], E. Elmroth[6], R. S. Montero[5]

[1]IBM Haifa Research Labs - Israel {rochwer, davidbr, wolfstal}@ il.ibm.com
[2]University College London, U.K. {a.galis, s.clayman}@ee.ucl.ac.uk; {c.chapman, w.emmerich}@cs.ucl.ac.uk
[3]SAP Research – Israel and U.K. {eliezer.levy, mark.wusthoff}@sap.com
[4]Telefónica I+D - Spain {caceres@tid.es}
[5]Universidad Complutense de Madrid - Spain {llorente, rubensm}@dacya.ucm.es
[6]Umeå University- Sweden {elmroth@cs.umu.se}

Abstract. This paper presents current research in the design and integration of advance systems, service and management technologies into a new generation of Service Infrastructure for Future Internet of Services, which includes Service Clouds Computing. These developments are part of the FP7 RESERVOIR project and represent a creative mixture of service and network virtualisation, service computing, network and service management techniques.

Keywords: Service Computing, Service and Network Management, Virtualisation, Service Infrastructure

1. Background and Motivation

At present a number of fundamental concepts and systems, including: grid and service computing, virtualisation, networking, service and network management are being developed separately. This paper argues for the integration of such systems into a new type of Service Infrastructure for Future Internet of Services.

Virtualisation has re-emerged as a gripping method for reducing service lifecycle costs and for increasing physical resource utilization. The main idea of all virtualisation techniques is the introduction of a logical structure between the physical resources and the computational processes. Virtualisation itself takes many forms. The most commonly known form of virtualization is "System virtualisation", also referred to as server virtualisation, is the ability to run multiple heterogeneous operating systems on the same physical server. With server virtualisation a control program (commonly known as "hypervisor" or "virtual machine monitor") is run on a given hardware platform and provides an environment for one or more other computer environments (virtual machines). Each of these virtual machines, in turn, runs its respective "guest" software, typically an operating system, which runs just as if it were installed on the stand-alone hardware platform. Additional forms of virtualisation include "storage virtualisation" and "network virtualisation", which give the ability to present a logical view of the storage and network resources respectively, which is different than the underlying physical resources.

Several research efforts have investigated the use of virtualisation with grid environments, and these can be largely classified as either virtual machine management on grid or grid-like virtual machine management. When combined with grids, virtualisation technologies suffer from several shortcomings. These shortcomings include: limitations on where and when a virtual machine can run (e.g., a Xen virtual machine cannot run on a VMware hypervisor); when, where, and how a virtual machine can be relocated (e.g., relocation can take place within an IP subnet and between hosts with shared storage); limitations on the performance of the virtual machine; overly-complex administrative interfaces; lack of mechanisms to meet pre-defined SLAs; and lack of adequate security.

Grid computing [1-4] is a powerful paradigm for running ever-larger workloads and services; in grids, many heterogeneous computing, network and storage resources are connected across administrative boundaries, and service providers share and exploit the infrastructure across nodes to run their services.

The Service Cloud Computing paradigm for hosting web-based services (i.e. Amazon Elastic Compute Cloud (EC2) [5] or Google's App Engine [6]) aims to facilitate the creation of innovative internet scale services without worrying about the computational infrastructure needed to support these services. However, these new "cloud computing infrastructure providers" have a scalability problem of their own. That is, what warranties can a single hosting company give to ensure that resources will always be available? In fact, no single hosting company can create a seemingly infinite infrastructure capable of serving increasing number of online-services, each having massive amounts of users and access at all times from all locations. It is only by partnering with each other that infrastructure providers can achieve the economies of scale needed to provide a seemingly endless compute utility.

What makes cloud computing [7-10] different is that recent developments in IT such as fast adoption of virtualisation technology servers [11-13] , as well as adoption of Software as a Service [14-16]as an alternative method for delivering functionality to both individuals and companies, has finally created an opportunity for a global service computing utility. The provision of Software as a Service, requires businesses to monitor the behaviour of these services.

Business Service Management (BSM) is the application of service management principles to manage the Service Levels for a business function. The purpose of BSM is to ensure that the ICT infrastructure will support the business functions by meeting requirements that are set in Service Level Agreements (SLAs). A Service Level Agreement is an agreement, or contract, between a service provider and a service consumer, in which expectations are set for the level of service to be provided by the infrastructure. The SLA is specified with respect to availability, performance, and other measurable objectives. Some of the key challenges in BSM involve SLA management.

The combination of these concepts and systems into a new computing paradigm called Service-Oriented Computing [17-19] will foster new and advanced services presented as software components exposed through network-accessible, platform and language independent interfaces. This will enable the composition of complex distributed applications from loosely coupled components. Service-Oriented Computing (SOC) carries the visionary promise of reducing software complexity, reducing costs, expediting time-to-market, improving reliability, and enhancing accessibility of consumers to both government and business services.

The paper is structured as follows: section 1 presents the motivation for a new generation of Service Infrastructure, section 2 provides the new model Service Oriented Infrastructure (SOI), section 3 describes the main functions and requirements envisaged by the SOI and section 4 concludes the paper and gives some further work.

2. Model for Service Oriented Infrastructure

RESERVOIR has a new and unique approach to Service Oriented Computing. In the RESERVOIR model, there is a clear separation between service providers and infrastructure providers. Service providers are the entities that understand the needs of particular business and offer service applications to address those needs. Service providers do not need to own the computational resources needed by these service applications, instead, they lease resources from an infrastructure provider. The infrastructure provider owns and leases a computing cloud, which provides the service provider with a seemingly infinite pool of computational resources. The cloud is capable of giving resources to many service providers.

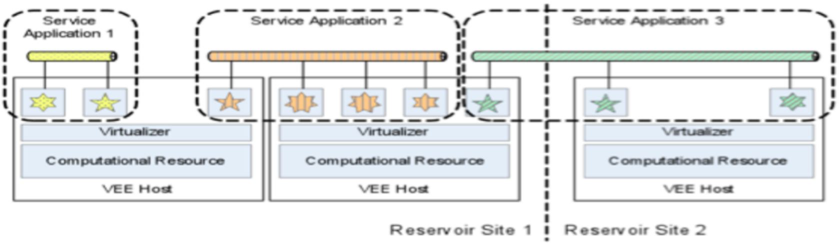

Fig. 1 - Service applications are executed by a set of VEEs

This computing cloud is made up of cooperating Reservoir Sites, which own and manage the physical infrastructure on which service applications execute. To optimise resource utilisation, the computational resources within a site are partitioned by a virtualisation layer into virtual execution environments (VEEs). The VEEs are fully isolated runtime environments that abstract away the physical characteristics of the resource and enable sharing of the physical hardware. The virtualized computational resources, alongside with the virtualisation layer and all the management enablement components, are referred to as the VEE Host.

A service is a set of software components, which work collectively to achieve a common goal. Each component of a service application executes in a dedicated VEE. The running VEEs are placed on the different VEE Hosts within the site. In some cases, it can be possible to migrate VEEs to different sites, according to automated placement policies that govern the site (see Fig. 1). The VEEs are represented by squares in this figure. The VEEs can be seen distributed across the Virtual Execution Environment Hosts (VEEHs), which make up the computing cloud. The VEEs for a particular service application may all be collocated in the same VEEH (i.e. as in service application 1), or may spread across VEEHs within the same site (i.e. as in service application 2), or may even spread across sites (i.e. as in service application 3). As long as SLA is maintained, the service is unaware of the actual location of its VEEs.

Service providers deploy applications in the computing cloud by passing a service definition manifest to a single infrastructure provider. This manifest includes: i) a list of the functionally distinct component types that make up the application; ii) The functional requirements for each component type, characterized by a reference to a master image, which is a self contained software stack (OS, middleware, applications, data and configuration) that fully captures the functionality of that component type; iii) Component grouping instructions, which are the requirements and constraints referring to a group of heterogeneous components so that they are treated as a single allocation entity; iv) Component topology instructions, that is how the different components types are related to each other and what are their inter-dependencies; v) Capacity Requirements, that is how much memory or cpus are needed; vi) Elasticity Rules, which are set of rules that express how the application scales. These specify the capacity (resource requirements) of each application component instance, as well as the number of instances of a particular component type, and how they can be dynamically adapted to properly satisfy the requirements of the application, while at the same time minimize cost; vii) Service Level Objectives (SLOs), that is the rules to ensure the service levels for business function are maintained viii) a Monitoring Specification, which specifies which elements of the application can send data for the elasticity rules and the SLOs.

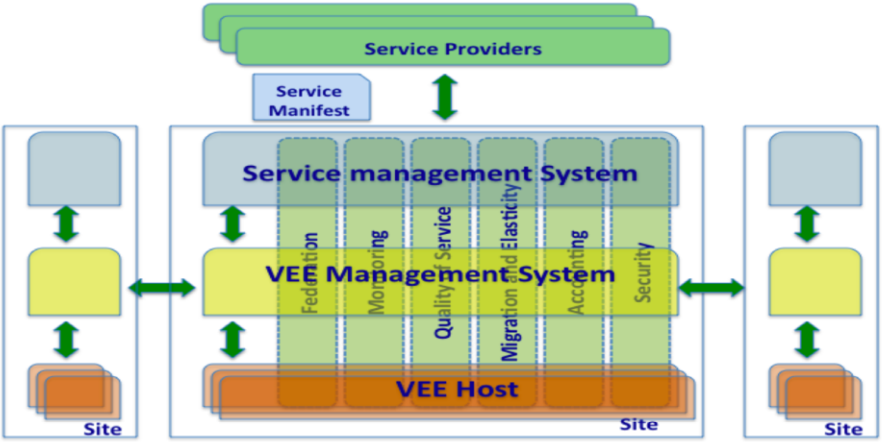

Fig. 2 – Reservoir Site Management Stack & Cross Layer Functionality

The layers of the architecture for a RESERVOIR system are presented next.

The **Service Manager** is responsible for the instantiation of the service application by requesting the creation and configuration of VEEs for each service component in the manifest. In addition, the Service Manager is responsible for ensuring SLA compliance by monitoring the execution of the service applications and executing the elasticity rules. That is, adjusting the application capacity either by adding or removing service components and/or changing the resource requirements of a particular component according to the load and measurable application behaviour. Within each Reservoir Site, the resource utilization is monitored and the placement of VEEs is constantly updated to achieve optimal utilization. Similarly, the execution of the service applications is monitored and the capacity is constantly adjusted to meet the requirements specified in the manifest. These on-going optimizations are done without human intervention by the Reservoir site management stack (see).

The **Virtual Execution Environment Manager (VEEM)** is responsible for the placement of VEEs into VEE hosts. It receives requests from the Service Manager to both create and resize VEEs, and it also finds the best placement for these VEEs in order to satisfy a given set of constraints (set by the Service Manager). The VEEM optimizes a site total utility function, i.e., VEEM is free to place and move the VEEs anywhere, even on remote sites, as long as the placement is done within the constraints such as VEE affinity and anti-affinity, security (never place VEEs from competitors together), and cost. In addition to serving local requests, the VEEM is the component in the system that is responsible for the federation of remote sites.

The **Virtual Execution Environment Host (VEEH)** represents a virtualized resource that can host a certain type of VEEs. This abstraction is needed to enable the separation of the logical algorithmic processing of the system from the actual plumbing, i.e., VEEM issues generic commands to manage the lifecycle of VEEs, and VEEHs are responsible for translating these commands into commands specific to the virtualisation platform abstracted by each VEEH. For example one type of a VEEH can be a physical machine with the Xen hypervisor controlling it, whereas another type can be a machine with the necessary software to host Virtual Java Service Containers (VJSC). In addition to translation functionality VEEHs are responsible for adding to the virtualisation platform they encapsulate the necessary hooks to meet the advanced requirements of different use cases.

3. Future Internet Service Infrastructure Functions

This section presents an overview of the main functions of the Reservoir Infrastructure that are important for Future Internet Service provision. The functions were determined through the analysis of many use-cases and from the general research direction set for Service Oriented Computing, for Cloud Computing, and for Future Internet of Services. These functions, together as a set, but not necessarily per individual requirement, define what distinguishes Reservoir from earlier virtualisation technologies and what the Reservoir project brings to the Future Internet efforts.

The main functions for a Future Internet Service Infrastructure are defined as:

Separation between Infrastructure and Services: Reservoir as a virtualisation infrastructure for services, enforces a clear separation between service providers and infrastructure providers. A Reservoir infrastructure provider will own and manage the computational, networking, and storage resources necessary to host arbitrary service applications. The infrastructure will provide functions and management facilities, which allow dynamic mapping of service components to the physical computational, networking and storage resources. In particular, service components should be opaque to the infrastructure providers and the service provider can deliver components that could contain virtually arbitrary software stacks.

Extensibility: At all layers Reservoir would support a minimum set of capabilities, yet provide for the extensibility of capabilities using a Plug-and-Pay and Unplug-and-Pay fashion. For each layer the capabilities are presented.

- **Service Manager (SM)** capabilities include: (i) request VEEs for a service; (ii) SLA repositories and management; (iii) Monitor SLA commitments at all levels; (iv) Monitor Context changes at all levels; (v) maintenance of Service related metrics; (vi) trigger and manage migration/ configuration/ contextualisation of service components as function of changes in context and/or SLA; same or multiple domains; (vii) assurance management; (viii) accounting and billing management; (ix) service life-cycle management; (x) performance management; (xi) open interfaces to service portals; (xii) open business policies framework for service and infrastructure providers relationship management (i.e. pay-per-use business model management).
- **Virtual Execution Environment Management (VEEM)** capabilities include: (i) abstraction of execution environments; (ii) supports the execution of services; (iii) Virtual Machines; (iv) service containers; (v) dynamic provisioning, supervision and re-allocation of VEEs; (vi) service-driven policy engine; (vii) open interfaces to control and monitor VEEs
- **Virtual Execution Environment Host (VEEH)** capabilities include: (i) interface with different virtualisation technologies. Reservoir would provide an abstract interface that is agnostic to the virtualisation technology; (ii) partition and management of physical resources in VEEs; (iii) open interfaces to VEEMs.

Multi-Site Operation: Reservoir provides the capability of sharing resources for the execution of a service application across multiple sites and multiple administrative domains (that are operated and managed by the same or different authorities). A Reservoir system is actually a federation of sites that cooperate for the optimal execution of service applications.

Service Orientation: Reservoir is about efficient provisioning and optimal management of service applications. We have determined that:

- a service application may be a complex entity and therefore its provisioning should be automated and streamlined.
- a service application is an infinite computation that is characterized by a workload that changes (sometimes dramatically) over time. This is in contrast to job scheduling in High-Performance Computing where resources are allocated to jobs with a finite duration.

Resources, therefore, should be allocated in reaction and in proportion to the changing workload.

Virtualisation Technology Independence: Reservoir should support different VEEH virtualisation technologies. Namely, Reservoir would provide an abstract interface that is agnostic to the virtualisation technology.

Security: Reservoir should provide a seamless, comprehensive, and flexible security scheme that operates consistently across dynamically changing layers. This embedded security shall be characterized by: (i) trustworthy operation; (ii) robustness and resilience under attack and mishap; (iii) protection and privacy of user and service information and assets; (iv) protection and privacy of identity and location and accountability; (v) all relevant service information should stay on some specific trusted domains; (vi) confidentiality and privacy for services and data (using ciphers) will be maintained; (vii) when two or more sites are cooperating, a trust-relationship is created. This includes especially data for authentication, authorization and accounting.

Accountability: The mechanisms that enable accountability of the service components are encapsulated in the Utility Computing Cost Model. In particular, this should provide an efficient, reliable, and secure way to collect and manage accounting data to support the different business cases and "pay-per-use" schemes. It should also support accounting across multiple sites and manage accounting for migrating components. Accountability denotes the need for the various interfaces, that Reservoir supports, to be capable of conveying or collecting accounting information. For the purpose of accounting and billing, Reservoir provides metering functions of the use of the virtual and physical resources. The granularity of the metering must at least reflect the information need for producing bills, in accordance with the accountability requirements.

3.1 Virtualisation Technology Assumptions

The assumptions regarding the VEEH underlying the virtualisation technologies are made explicit so that they can be treated as requirements for providers of virtualisation technologies.

Virtualisation Overhead: The underlying assumption is that virtualisation adds an overhead of at most 10% to the end-to-end performance of the application. Otherwise, the use cases make no sense to begin with. The most stringent consequence is that the throughput of a virtualized setup of the application does not degrade by more than 10% compared to a native setup running on the same hardware, and all this without compromising the end user experience.

Migration Performance: Live migration degrades the performance of the migrated application as it consumes CPU and I/O resources while preparing for the migration (especially for stateful applications). In addition, live migration requires an actual downtime of the application during the actual migration. The application relies on complex stack that is sensitive to timeouts at different levels (e.g., network, database). The performance degradation and the downtime should not have adverse affect in terms of too many aborted user transactions.

3.2 VEEM & VEEH Infrastructure Requirements

This section presents the requirements for the VEE Hosts, the VEE Host Interface, VEEM and the VMI. These components are largely unaware of the service semantics associated with the components they execute and manage.

Adaptive Resource Allocation: Reservoir should enable dynamic changes of the physical resources allocated to a VEE in a case the VEE requires additional or different capacity (e.g. CPU capacity, Memory capacity, I/O bandwidth etc) provided that the underlying physical system is able to serve the requirements for more or different capacity. If the resource consumption in a site is oversized, automatic downsize of the consumed resources in order to limit costs should be initiated. These dynamic changes should be transparent to the service consumer.

Elastic Arrays of VEEs: Reservoir should support dynamic control of the number of identical VEEs for the purpose of adapting this number according to the load, for example. The relevant service application manifest must indicate this potential elasticity. In particular, all VEEs in the array share the same master image as specified in the manifest. The dynamically launched VEE should transparently join its already running siblings in the sense of serving some of the workload. It should be possible to stop a running VEE that was defined as part of an array without disrupting the service of the overall service application. It is the responsibility of the Service Provider to implement correctly a service component that is array-enabled.

Warm Images: The execution of a service component may require complex preparation steps (e.g. retrieving data from the backend). Rather than doing the preparation separately for each dynamically launched component, it should be possible to create an image that creates a warm VEE with all its context (e.g., warm caches, established connections), configuration and state.

Migration: Reservoir supports live migration of a virtual system to another pool of physical resources (i.e. computational, networking and storage resources). The live migration is performed of service components while maintaining state (of the components itself and also any impending data exchanges). Migration could also be performed on a suspend/resume mode with minimal service disruption. The migration capability is performed transparently to the service applications, which run on the virtual system. Reservoir should support the commonly practiced migration scenarios. Namely:

- Migration of groups of VEEs in order to optimise the utilization of physical resources in order to save power and to manage power in any period.
- Migration of groups of VEEs in order to facilitate massive lifecycle and maintenance scenarios (e.g. install patch, physical resource /driver upgrades, etc).
- Completion of migration of certain components within a specific time frame.
- Request response time as observed by the end-user must not exceed a limit.
- Seamless migration, without downtime, of groups of virtual systems when physical resources or the hypervisor would require maintenance activities to be performed (e.g. install patch, physical resource upgrade, driver upgrades, etc).

The Reservoir-specific dimensions for migration are:

- Migration can be across sites.
- The grouping of migrated VEEs should reflect their membership in and the structure of the relevant service applications.
- Migration of groups of virtual systems in order to reduce/optimise the number of physical resources or save power or to better manage power in any period.

3.3 Migration and Elasticity Transparency

The following requirements specify that behaviour (as observed externally and between its components) of a service application does not change as a result of elastic starting and stopping and migration its components. First, the following assumptions should be stated explicitly: (1) In most cases, it can be assumed that storage can be (logically) shared between the origin and destination hosts of the migration; (2) The service application has a built-in elasticity capability. That is, the application is capable of dynamically adding and removing components while running. Under these assumptions, the relevant requirements are:

- The networking and storage views of VEEs are kept intact when VEEs are migrated. That is, if a VEE accessed particular storage device and communicated with particular network entity, the same view is preserved in spite of migrations.
- A VEE that is cloned in a VEE array inherits the storage and networking view of the other array members.
- The public end-points of the service application are kept intact in spite of VEE migration and cloning.

Cost-Based Optimization: The resource allocation optimization should be driven by a configurable cost-model. The cost-model should approximate the relative anticipated latencies associated with the different allocations options. The cost model, for example, should factor the cost of cross-site migrations.

Autonomous Local Optimizations: A VEEM should be able to autonomously improve and optimize the utilization of the site local resources regardless of the service-level monitoring and optimization. That is the VEEM should be able to take advantage of opportunities of free local resources on its own right as longs as SLAs are not violated.

Management Interface for Standardization: Reservoir should expose a management API that hides the details of the virtualisation technology. The goal is to standardize this API. It should be possible to compose API primitives in scripts for the streamlined automation of more complex tasks in the data centre.

Plug-Ins: The VEEM should expose a plug-in architecture, such that various implementations of the API can be created. Moreover, a VEEM should be able to manage hosts of different virtualisation technology using the API.

3.4 Service Management Requirements: Service Definition Manifest

A service application may be composed of inter-related components. It is required to completely specify the application components and setup using a declarative language in the form of a manifest. The manifest should specify, for example, all the images, the storage configuration, the database content and the relevant applicative configuration. Moreover, the application should be provisioned as a single logical unit. The manifest should support the encapsulation of various service components as images for the rapid initial provisioning as well as the rapid dynamic adaptation of service applications. Moreover, the manifest should enable the automation of provisioning and management of the service application.

Template-Based Provisioning: It should be possible to use the service manifest as a template for easily provisioning instances of the application. The template must allow for instance-specific components due to instance-specific configuration and customization. This requirement is important for multi-tenant scenarios where the template should be parameterized by tenant.

Flexible Virtualisation Configurations: A virtualisation configuration maps the components specified in the service manifest to physical hosts. The manifest should support expressing flexible virtualisation configurations of the applications in order to satisfy various performance and TCO requirements. In particular, it should be possible to specify dependencies and starting order among components. It should be possible to specify sharing of a component by more than a single service application. For example, a DBMS service component may be shared by multiple service applications.

Resource Consumption, Management and Enforcement: Reservoir provides facilities for monitoring, management and enforcement of physical resource consumption. Through isolation the degree of resource consumption can be controlled including control of greedy services.

Conflicts Resolution and Avoidance: Service components may require certain resources from the system to be allocated statically, e.g. a certain port number. To resolve conflicts between different services the service components may be executed isolated in virtual systems while sharing physical resources.

4. Conclusions

RESERVOIR's research on virtualisation, service computing, network and service management both enables and unifies some of the emerging trends in Service Oriented Computing and the Future Internet. This paper presents work in progress (Reservoir project started work in February 2008) for the definition and integration of such systems into a new generation of Service Infrastructure. Full design, realisation, and evaluation of the RESERVOIR Infrastructure will be completed in the next 2 years.

Acknowledgments

This work was undertaken in the context of the Reservoir FP7 project [20], which is partially financed by the EU. The Reservoir consortium consists of International Business Machines Haifa Research Lab (HRL), Telefónica Investigación y Desarrollo (TID), Centre d'excellence en technologies de l'Information et de la communication (CETIC), University College of London (UCL), Universidad Complutense de Madrid (UCM), Elsag Datamat (ED), Sun Microsystems (Sun), Thales, Università della Svizzera Italiana (University of Lugano) (USI), Umeå University (Umeå), SAP Research, University of Messina (UniMe), OGF.eeig

References

[1] "The Information Factories" - Wired Magazine, Issue 14.10, October 2006 http://www.wired.com/wired/archive/14.10/cloudware.html?pg=1&topic=cloudware&topic_set=

[2] "Reflections on Cloud Computing" - Irving Wladawsky-Berger's Blog, March 2008, http://blog.irvingwb.com/blog/2008/03/reflections-on.html

[3] "Understanding Cloud Computing" - Wallis Paul, Keystones and Rivets, February 2008, http://www.keystonesandrivets.com/kar/2008/02/cloud-computing.html

[4] "The Big Switch – Rewiring the World from Edisson To Google"- Nicholas Carr, published by W. W. Norton, January 2008

[5] "Amazon Elastic Compute Cloud" -Amazon EC2 web site,

http://www.amazon.com/gp/browse.html?node=201590011
[6] "What is Google App Engine" http://code.google.com/appengine/docs/whatisgoogleappengine.html
[7] "The MAC system : The computer utility approach" - R. M. Fano, in IEEE Spectrum, vol. 2, pp. 5644, January 1965.
[8] "What is the Grid? A Three Point Checklist" - Ian Foster, 2002, http://www-fp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf
[9] "The Globus Toolkit" - http://www.globus.org/toolkit/
[10] "The Open Grid Services Architecture"- http://www.globus.org/ogsa/
[11] "Server Virtualisation: Doing More with Less"- Leon Erlanger, Inforworld Report, Sept 06, http://www.infoworld.com/article/06/09/11/37FEvirtcaseserv_1.html
[12] "Xen and the Art of Virtualisation" - P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt and A. Warfield, in Proceedings of the 19th ACM Symposium on Operating Systems Principles, Bolton Landing, NY, USA
[13] "The VMware" Web Site www.vmware.org
[14] "Turning Software into a Service" - Mark Turner, David Budgen, Pearl Brereton, Computer, vol. 36, no. 10, pp. 38-44, Oct., 2003
[15] "The Different Faces of IT as Service" I. Foster, S. Tuecke - www.ggf.org/documents/Diff_Faces_foster.pdf
[16] "The salesforce" web site http://www.salesforce.com/
[17] "Service Oriented Computing" –A Research Roadmap, M. Papazoglou, P. Traverso, D. Schahram, F. Leymann, B. Kraemer, Dagstuhl seminar 2006 http://drops.dagstuhl.de/opus/volltexte/2006/524/
[18] "Service-Oriented Computing: State of the Art and Research Challenges" -Mike P. Papazoglou, Paolo Traverso, Schahram Dustdar, Frank Leymann: IEEE Computer 40(11): 38-45 (2007)
[19] "Service-Oriented Computing: a Research Roadmap" - Mike P. Papazoglou, Paolo Traverso, Schahram Dustdar, Frank Leymann: Int. J. Cooperative Inf. Syst. 17(2): 223-255 (2008)
[20] "Reservoir project" www.reservoir-fp7.eu