This is the ACCEPTED VERSION of the article:

J. C. Dueñas, J. L. Ruiz, F. Cuadrado, B. García and H. A. Parada G., "System Virtualization Tools for Software Development," in *IEEE Internet Computing*, vol. 13, no. 5, pp. 52-59, Sept.-Oct. 2009, doi: 10.1109/MIC.2009.115.

DOI: 10.1109/MIC.2009.115.

https://ieeexplore.ieee.org/document/5233611

© Copyright Owner: IEEE

© 2022 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

System Virtualization Tools for Software Development

Juan C. Dueñas, Félix Cuadrado, Boni García, and Hugo A. Parada G.
Universidad Politécnica de Madrid

José L. Ruiz
Indra

The configuration complexity of preproduction sites coupled with access-control mechanisms often impede the software development life cycle. Virtualization is a cost-effective way to remove such barriers and provide a test environment similar to the production site, reducing the burden on IT administrators. An Eclipse-based virtualization tool framework can offer developers a personal runtime environment for launching and testing their applications. The authors have followed a model-driven architecture approach that integrates best-of-breed virtualization technologies, such as Xen and VDE.

The increasing complexity of IT ecosystems has exacerbated the need for specialized roles.1 In most organizations, system administrators control deployment, whereas the software development team handles all previous development tasks. (We use the phrase development team loosely to describe the many roles involved in the development process, such as requirements engineers, software architects, quality-assurance staff, and testing engineers.) System administrators are primarily responsible for operating and maintaining production sites, and service availability and performance are their main concerns. Consequently, administrators have neither the resources nor technical skills to help the development team test and deploy immature software components. Developers' productivity and, as a result, time to market are negatively affected by this role mismatch. The situation is even worse when organizations apply agile software development models that dictate rapid iterations over the basic waterfall life cycle. Frequent test execution is the cornerstone of these methods, so the need for a suitable deployment environment is even more pressing. This organizational gap is a showstopper for the software development process.

Many organizations set up a preproduction site to cope with this situation. Unfortunately, this further burdens both teams because administrators must install, configure, and maintain a new set of servers that developers without the proper expertise will have difficulty manipulating and configuring for their tests. In addition, concurrent manipulation of preproduction servers is prone to errors. This dilemma is too often solved by establishing access-control mechanisms to the preproduction site, under administrators' supervision. Software deployment to a preproduction site comprises activities such as managing databases and application servers, loading test data, reconfiguring application components, running scripts, and issuing formal requests to coordinate the work between teams.

Effective testing requires that the system under test run on an environment as close as possible to the production site. From the software development viewpoint, each developer would ideally enjoy a fully independent, always-on runtime environment to launch and test applications. With real servers and networks, however, this quickly leads to machine sprawl.

Fortunately, virtualization tools make it possible to partition physical server resources to match each consumer's specific needs.2 With this technology, we can embed a preproduction site into each developer's machine3 and reduce IT costs. We created a platform-independent model (PIM) for describing a virtualized distributed system that supports a range of virtualization technologies. To conceal these technologies' underlying complexity from developers, we designed an Eclipse-based virtualization tool to integrate it into productive software development processes. Our goal is to provide a developer-friendly environment; adopting technologies developers are already familiar with flattens the learning curve and thus improves productivity.

Virtualization Technologies

The purest approach to virtualization is emulation. In an emulator, software replicates the complete functionality of a hardware processor. A program can be run on different platforms, regardless of the processor architecture or operating system (OS). Because this technique imposes a high-performance penalty, we discarded it for our solution.

Native virtualization is the alternative to full-blown emulation. Rather than replicating a hardware platform, native virtualization provides an adaptation layer to guest machines.4 In this scenario, the three main elements are the host OS, the guest OS, and the virtual machine monitor (VMM) or hypervisor. The VMM handles privileged instructions on behalf of the virtualized OS. This technique has been recently refined in favor of lighter-weight approaches that improve performance by working with higher-level abstractions. Paravirtualization and OS-level virtualization are outstanding representatives of these technologies.5 In paravirtualization, the guest OS is changed to cooperate with the VMM and the host, issuing system calls to either real devices or the VMM. OS-level virtualization consists of partitioning the existing OS layer. Its main limitation is that the host and guests must have the same OS.

A common factor among virtualization technologies is a performance penalty over x86 architectures when handling privileged processor instructions issued by the guest OS because they must be dynamically translated by the VMM or controlled by the host OS. Processor manufacturers have addressed this problem,6 and now there is hardware support that accelerates VMM call management.

Table 1 compares some of the most relevant commercial and open source software (OSS) technologies for server virtualization, showing the standard trade-off between a product's performance and flexibility. Our aim is to support a range of production scenarios; thus, we can't select a unique virtualization technology beforehand. Because many of the analyzed technologies share a conceptual basis (such as guest systems that are distributed as image files), we can support them with a generic approach.

So far, we've focused on creating virtual nodes, but to emulate a distributed production scenario, we also need to create a virtual network. Communication among the nodes is a must, as is connectivity with external systems. Node virtualization technologies usually provide their own networking options. As an alternative to interconnect different solutions, we opted for virtual distributed Ethernet (VDE),7 which provides a virtual switch capable of creating a virtual network compatible with multiple node virtualization technologies (Xen, QEMU, and UML).

System Design

Our solution consists of a tool framework aimed at easing distributed applications' execution and testing. It provides developers with a personal system to deploy, start, and test their applications.

Product	Type	License	Highlights	Guest performance
Bochs	Emulator	Open source	Allows debugging the guest OS	Very slow
QEMU	Emulator/native virtualization	Open source	Supports a wide range of hardware architectures	Slow (10% of host)
VMware	Native virtualization	Commercial	Provides a mature product family to manage virtual infrastructures	Close to native
VirtualBox	Native virtualization	Dual license	Remote desktop protocol support in commercial version	Close to native
User Mode Linux (UML)	Paravirtualization	Open source	Stable support for Linux systems	Close to native
Xen	Paravirtualization	Open source	Supports virtual machine migration on the fly	Native
OpenVZ	OS-level virtualization	Open source	Efficient resource partitioning	Native

Table 1. Virtualization technology comparison.

Use-Case Analysis

We performed a use-case analysis to determine which functionalities the tool system should provide. The target scenario is a complex distributed system with an Internet connection. Potentially, the nodes included in the system are powerful servers running a plethora of different software components. As a reference, in the context of the ITECBAN project,8 we're using a distributed runtime environment consisting of four Linux boxes provisioned with certain middleware components that provide an advanced services-oriented architecture (SOA) and business process management (BPM) execution environment (see the right side of Figure 1).

Our study identified the developer and system administrator as the two main actors. The developer codes, runs, and tests applications in the distributed system. The administrator manages the basic IT infrastructure (OS, installed components, databases, and so on) for the virtualized elements included in the system. Under these assumptions, we identified the following use cases:

• Administrators create virtual node configurations. The tool system must provide editors and storage facilities for these configurations, including the virtual images. Node configurations can eventually be used to build complete virtual distributed systems; the data included in these node configurations should be as close as possible to the production environment.

• Developers (testers) or administrators define the deployment scenario, modeled in resemblance to the physical distributed system.

Tools must enable a rapid definition of virtualized scenarios (including network, nodes, and software layers) with minimal effort. Scenarios can be reused and shared.

• Developers launch any virtual model. Applications can be deployed to the virtual runtime environment. Tests over the production site can therefore be executed in the developers' machines.

• Developers continuously monitor the virtual system's state while carrying out some operations controlling the host and guest execution.

Although the actors interact with the system in different use cases, the virtualized system models are a shared concern for both of them.

Distributed System Metamodel

One of our tool system's primary goals is to support a range of virtualization technologies such as Xen, QEMU, VDE, and UML, while retaining vendor independence. We've followed a model-driven architecture (MDA) approach to design our tool framework. PIM instances (see Figure 2) are transformed and enriched to platform-specific models (PSMs) by framework extensions, which contribute the specifics of particular technologies.

Of course, distributed system modeling isn't a novel field, and we didn't design our PIM from scratch. We found valuable inputs for our model in the Object Management Group's Deployment and Configuration for Distributed Systems (D&C) specification9 and network simulation technologies such as VNUML,10 a domain specific language (DSL) for easily defining complex network configurations. Because we also intended to deploy software over runtime virtualized environments, we thought the OMG D&C would be a good basis for our model. However, the specification assumes the deployment target is already in place, whereas we need to create the virtual environment on the fly. Network simulators have already addressed this situation, but they're strongly tied to the underlying technology — for example, UML in VNUML. We've designed the PIM to handle as many technologies as possible. This problem vanishes once the virtual environment is launched because the means to describe it are compatible with the D&C model. We've defined the PIM (as Figure 2 shows) with these factors in mind.

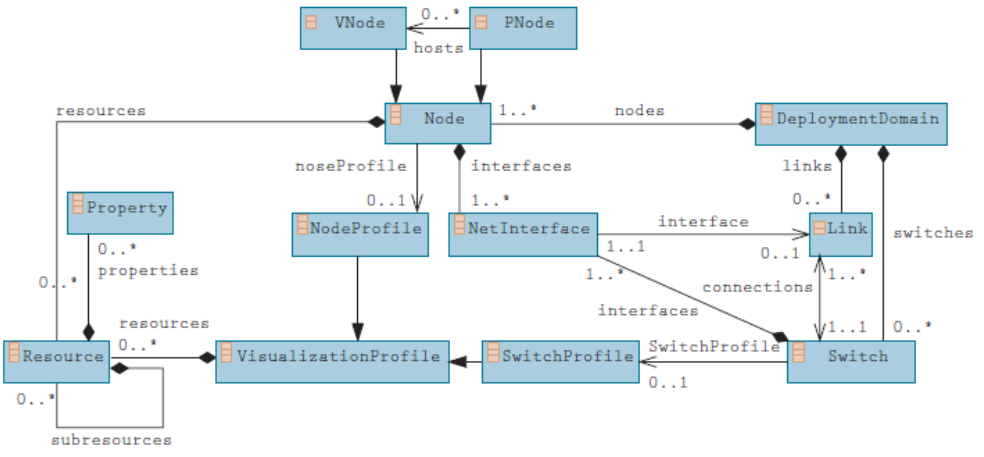

Figure 1. Tool architecture.The virtualization framework is composed by a set of Eclipse plug-ins, integrated into the development environment. From this environment, users can launch one of the defined virtualization scenarios, which will be instantiated in their machines.

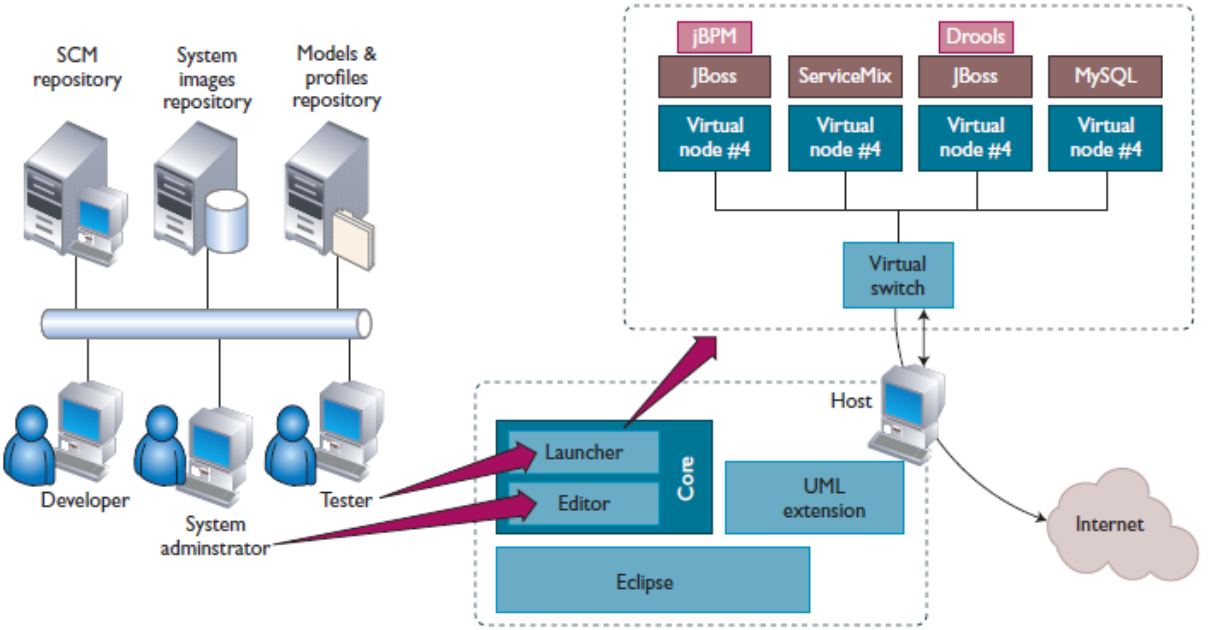

Figure 2. Distributed system platform-independent model (PIM). The model represents the base network topology and the computing nodes of a virtualization. Nodes can be either physical (hosts) or virtualized, and their specific characteristics are encapsulated in profiles.

As in the OMG D&C model, the central entity is the deployment domain. A deployment domain aggregates switches and nodes, which can be qualified with resources modeling software, hardware, and communication capabilities. The model includes virtual nodes (VNodes) and physical nodes (PNodes). A PNode can be included in a PIM instance either because it hosts VNodes or because it's a target for software deployment. Our PIM considers multihosted simulations; the association between PNode and VNode allows expressing host relationships.

The Switch element represents virtual networks. Nodes connect to a network through the NetInterface element, and we modeled the resulting connection as a Link. We included configurable network parameters in the NetInterface and Switch entities.

To promote model reuse, the PIM includes profiles, which are collections of parameters that completely define a virtual node or switch and are linked to a specific virtualization technology (such as VDE). In addition, profiles are stored and can be shared among developers using the profile repository (see the left side of Figure 1). Development teams will share complete model instances or reference profiles — for example, an application server profile (including the OS, network services, and server applications). Our system supports change control in profile definitions with version attributes in the PIM. So, a development team could test applications against different versions of the application server profile.

Tool Framework

Figure 1 shows the architecture of our tooling environment. Basically, each development station is leveraged with our virtualization tools. Networked repositories support distributed development. Typically, an organization controls its software development with a software configuration management

(SCM) repository. Our tools also use a system images repository (SIR) and model and profiles repository (MPR).

System administrators create virtual system images and store them at the SIR. Both the OS and middleware (without applications) are stored as virtual images. Administrators create, configure, and add virtual images to the SIR. The extended IDE retrieves the necessary elements from the SIR to launch a virtualization.

We selected Eclipse as the base platform to integrate our tool because of its wide adoption and extensible architecture.11 Eclipse's architecture is leveraged by an OSGi kernel,12 which provides a service-oriented programming model to the Eclipse platform. Through this model, we've divided the system's architecture into two main blocks: a core framework and technology extensions. This architecture facilitates the integration between virtualization technologies (strongly tied to the OS level) and the IDE. Our framework currently supports the definition and execution of distributed system virtualizations with VNUML, UML, VDE, QEMU, and XEN. The core framework builds on the distributed systems metamodel, adding three fundamental aspects: profile definition, model creation, and virtualization launch.

First, the core provides the profile manager. This component lets administrators create, edit, save, and share profiles through the MPR. Profiles are defined in the PIM as the necessary configuration parameters to fully characterize a virtual node or switch. Profiles allow uniform handling of element instances. However, profile parameters depend on the specific technology and thus must be provided by technology extensions. Additional configuration parameters, such as the native technologies' installation location and correctness checking, can also be configured through preference pages.

Second, the framework includes a distributed systems editor. This component provides a rich, graphical editor for creating model instances. An administrator can drag and drop nodes, networks, and connections and quickly define distributed system models. We minimize manual configuration by using default values and the abstraction provided by profiles. This abstraction lets the editor be technology agnostic. Thus, the editor supports any technology contributed through the extension points without modifications.

Finally, the core can launch and control virtualization scenarios. Initiating a launch requires no additional arguments or configuration and is invoked with just a mouse click. Internally, each technology extension creates a virtualized element instance from the profile parameters. For instance, when we launch a distributed virtualization scenario with a VDE switch, the VDE delegate analyzes the range of IP addresses for both the nodes connected to its network and the switch configuration parameters. This extension uses DNSMasq to provide DNS/DHCP services to the guest nodes. Also, the plug-in modifies the host network configuration to provide external Internet access to the guests, as well as virtual consoles. The tool framework controls networking processes to provide a clean virtualization stop.

For example, the VNUML plug-in contributes extensions for configuring the native technologies as well as defining and storing VNUML profiles. Besides, the plug-in registers a launcher for VNUML systems that can receive the PIM model as input. It can also perform a model transformation, converting the generic model into the VNUML PSM model, which can be directly processed by the VNUML Perl parser. The tool system also supports extensions for QEMU, UML, and XEN virtualization technologies. Conceptually, each extension implementation is similar to this one.

Case Study

The right-hand side of Figure 1 depicts a representation of a distributed enterprise runtime environment. The virtual runtime uses OSS servers to leverage a SOA/BPM architecture. The entire configuration, which administrators create beforehand, consists of four UML/Linux virtual machines interconnected by a VDE switch. The environment contains the following installed services:

• Virtual node 1. The JEE application server (JBoss) — extended with JBPM, a runtime for long-term processes — manages the processes life cycle by interacting with the database installed and configured in virtual node 4.

• Virtual node 2. The enterprise service bus (ESB) platform (ServiceMix) provides a service-oriented integration layer for the virtual environment.

• Virtual node 3. A JBoss server is extended with Drools, an enterprise framework that provides a business rules management system (BRMS) for business rules edition, change, and management.

• Virtual node 4. A database management system (MySQL) is the server where the services' application data can be persisted in relational form.

Once created, we can seamlessly integrate this complex enterprise runtime in the developer IDE (in this case, Eclipse) so developers don't need to know the distributed system's internal details. This approach lets developers focus on developing, deploying, and testing their software components, regardless of the details of the host, virtual nodes, and services.

Figure 3 is a snapshot of our virtualization tool in action. The left-hand panel shows a deployment scenario consisting of four nodes connected to the network switch. The deployment model can be launched with no further configuration required. The right-hand side of the image shows the terminals of the virtual nodes at runtime. Through them, users can open FTP connections to the external SCM repository to get, install, configure, and launch the versions of the software to be tested. Because there's a connection from the development host to the virtual system, the developer can interact with the system using a common Web browser.

Our future work will especially focus on integration with a software deployment and testing infrastructure that automates component distribution, resolution of component dependencies (keeping version constraints), and adaptation of the process to the virtual network resources. Additionally, the tool system (see Figure 1) includes an SIR, managed by IT administrators, that hosts a collection of virtual nodes ready to be used in virtualizations. Currently, each variant of a node image must be stored as a complete file. This approach is inefficient because the repository is rapidly filled with multiple versions of the same base image. We can address this limitation by modularizing image persistence to assemble images on the fly. However, this modification is technically challenging because the repository logic would have to cope with a range of operating systems and software deployment mechanisms. Each supported deployment model needs a specific weaver for generating images. A repository based on this principle will have all the advantages of current SCM systems, combining versioning capabilities and optimizing storage resources with differential storage.

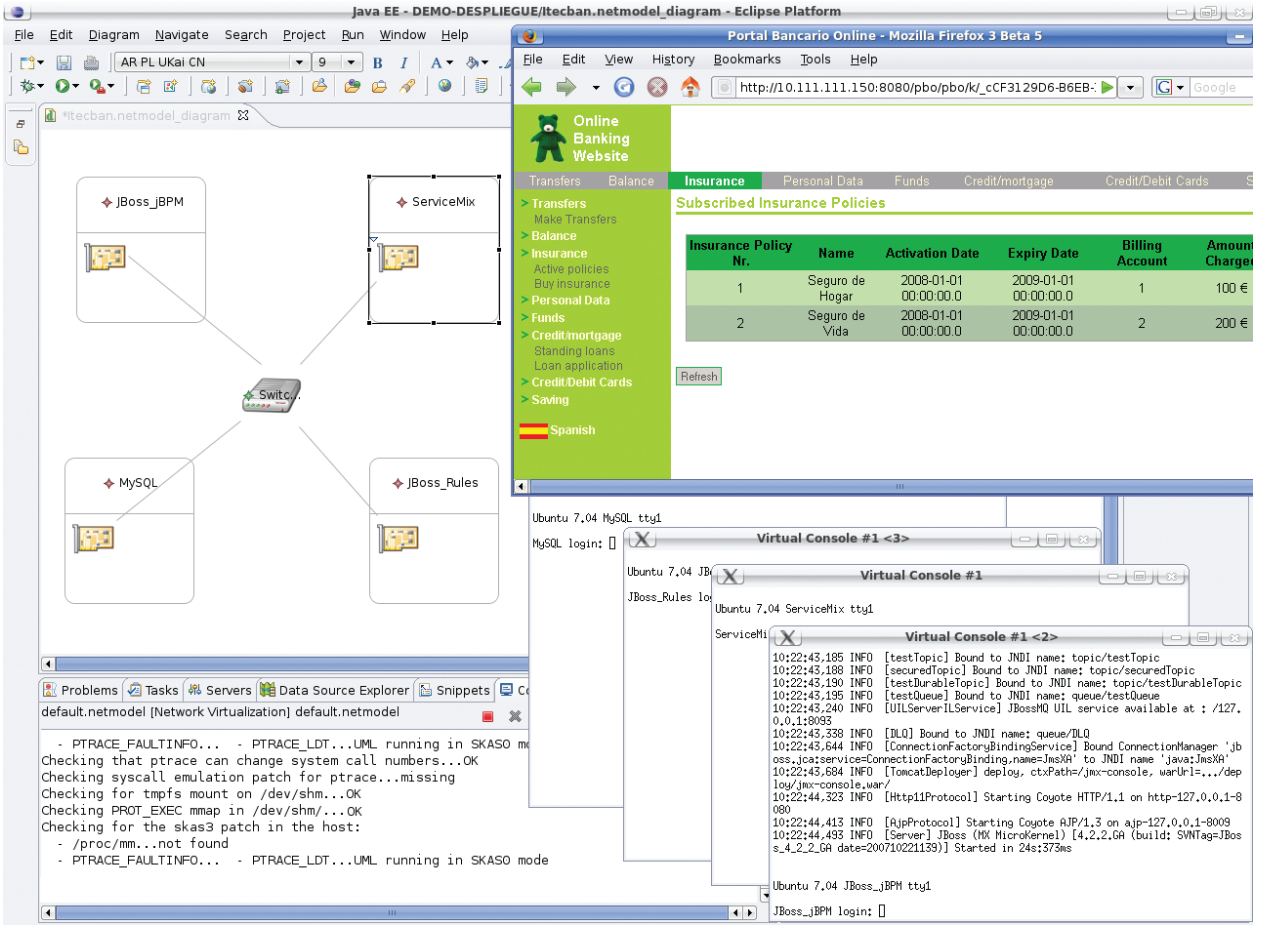

Figure 3. Virtualization tool in action. The background shows the virtualization editor opened with a running scenario. For each one of the virtual nodes, a control terminal has been created for additional configuration. On top of that, the developer can access the developed application through a Web browser connected to the virtual server.

Also, although our tool has proven successful for functional testing, it's clearly out of scope for stress, performance, and reliability testing because the behavior of guests differs from actual systems. Developers' machines should be resourceful enough to hold several production machines without interference, but this is uncommon. To overcome this problem, we're working on the capability to launch remote virtualizations on a dedicated cluster of hosts, with enough raw computing power to faithfully mirror actual production systems.

Another ongoing line of work aims at monitoring the virtual system through the tools environment. We're developing a resource-gathering infrastructure that will let us detect anomalous behavior in the system and create a model of a mirror virtualized system from information we extracted from the real one.

Acknowledgments

ITECBAN is an IT innovation project partially funded by CENIT (a Spanish public R&D program). We're grateful to MITYC (Ministerio de Industria, Turismo y Comercio) and CDTI (Centro para el Desarrollo Tecnológico e Industrial) for supporting ITECBAN through CENIT.

References

1. J. Lentz and T. Bleizeffer, “IT Ecosystems: Evolved Complexity and Unintelligent Design,” Proc. 2007 Symp. Computer-Human Interaction for the Management of Information Technology, ACM Press, 2007, article no. 6.
2. A. Mendoza, Utility Computing: Technologies, Standards, and Strategies, Artech House, 2007.
3. S. Seetharamanm and K. Murthy, “Test Optimization using Software Virtualization,” IEEE Software, vol. 23, no. 5, 2006, pp. 66–69.
4. M. Rosenblum and T. Garfinkel, “Virtual Machine Monitors: Current Technology and Future Trends,” Computer, vol. 38, no. 5, 2005, pp. 39–47.
5. S. Vaughan-Nichols, “New Approach to Virtualization Is a Lightweight,” Computer, vol. 39, no. 11, 2006, pp. 12–14.
6. R. Uhlig et al., “Intel Virtualization Technology,” Computer, vol. 38, no. 5, 2005, pp. 48–56.
7. R. Davoli, “VDE: Virtual Distributed Ethernet,” Proc. Testbeds and Research Infrastructures for the Development of Networks and Communities Conf. (TridentCom 05), IEEE Press, 2005, pp. 213–220.
8. J.L. Ruiz, J.C. Dueñas, and F. Cuadrado, “A Service Component Deployment Architecture for E-Banking,”
9. Deployment and Configuration of Component-Based Distributed Applications Specification, v. 4.0, OMG formal specification, 2002.
10. F. Galán et al., “Use of Virtualization Tools in Computer Network Laboratories,” Proc. Information Technology Based Higher Education and Training (ITHET 04), IEEE Press, 2004, pp. 209–214.
11. E. Gamma and K. Beck, Contributing to Eclipse: Principles, Patterns, and Plugins, Addison-Wesley, 2003.
12. O. Gruber et al., “The Eclipse 3.0 Platform: Adopting OSGi Technology,” IBM Systems J., vol. 44, no. 2, 2005, pp. 289–300.