Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 11 (2013) 346 – 351

www.elsevier.com/locate/procedia

2nd International Through-life Engineering Services Conference

Integration of virtualized environments in PDM systems for embedded software product development

Michael Hopf *, Jivka Ovtcharova

Karlsruhe Institute of Technology (KIT), Institute for Information Management in Engineering, Zirkel 2, 76131 Karlsruhe, Germany

* Corresponding author. Tel.: +49-163-7320509; *E-mail address:* michael.hopf@partner.kit.edu (M. Hopf)

Abstract

The number of products with embedded software increases across all application areas continuously. Thus, the complexity between the hardware and software is steadily increasing. This leads to an increment of software defects. Therefore, new approaches are needed to ensure the product quality. In the context of PLM, virtualization can support crucial stages of the product development and test automation by providing virtual environments. This paper shows an architectural approach, and how to perform an integration of virtualization software in PDM systems.

Selection and peer-review under responsibility of the International Scientific Committee of the "2nd International Through-life Engineering Services Conference" and the Programme Chair – Ashutosh Tiwari

Keywords: Lifecycle Management; PDM; PLM; Virtualization; Virtual Machine; Embedded System; Process Definition; Workflow; Product Development

1. Introduction

During the last two decades, the proportion of software as a component in products has increased steadily. Thus, the worldwide market for embedded systems was around 60 billion euros with an annual growth rate of 14 percent and up to 20 percent in certain domains [1, 2]. Nowadays, software is used in various fields of applications and forms. Embedded software systems are significantly more complex in an environment embossed by hardware than software systems without the interaction with the real world. While embedded software systems interacting with the real environment, they have to cope with a variety of constraints such as real-time, security, energy management as well as other resources such as memory management and communication [3]. Safety-critical embedded systems are used in almost half of the stationary and mobile phones, network control and monitoring systems, transmission technologies, medical analysis and treatment devices as well as infotainment terminals [4].

Embedded systems can perform several background tasks such as control and monitoring as well as direct interactions with the user perform. Due to the increasing complexity of the software, a comprehensive testing of embedded systems is necessary. Thus, the test automation represents the most important part of the development of embedded systems with the risk of delays in the delivery, which can quickly exceed the cost of a product caused by software errors. Companies must elicit, what can be done in the design phase, and have to seek for opportunities to automate the various phases [5].

This issue is evidenced by a variety of recalls in the automotive industry because of software errors. Pontiac had to initiate a recall in 2004 because the software did not understand leap years. In 2005, Toyota recalled 75,000 vehicles due to a software defect [6]. This subject matter refers not only to the automotive industry, but also to other industries. The test automation and software development of embedded systems can be supported by virtualizer through cloning, snapshots and more flexible disaster recovery of test environments (TEs).

Selection and peer-review under responsibility of the International Scientific Committee of the "2nd International Through-life Engineering Services Conference" and the Programme Chair – Ashutosh Tiwari
doi:10.1016/j.procir.2013.07.014

This paper shows the benefits of the lifecycle management integration for virtual machines (VMs) in Product Data Management (PDM) systems. A modeling of processes for virtual infrastructures is shown as well as an architecture approach is presented, which allows the lifecycle management and the deployment of virtual environments (VEs) for embedded systems development. The paper is organized as follows: section two analyzes currently used virtualization approaches in the field of software testing. Section three describes different virtualization techniques and virtualizer. Section four presents the modeling of the lifecycle management of VMs in PDM system. Section five shows the architecture and its limitations. Finally, in section six the conclusion is formulated and suggested further research.

2. Related work

The integration of VEs in PDM system has not been investigated. Meanwhile, the integration of Software Configuration Management (SCM) in PDM systems has been studied. Muhammad et al. [7] discussed the integration of Product Lifecycle Management (PLM) and SCM systems and the role of these systems to be applied during the development and maintenance. Do and Chae [8] propose an architecture that supports the extension of the data model version items for SCM systems. The application supports the functionalities of PDM and SCM as well as the integration of hardware and software parts for product configurations and engineering change management. Strong, Hayka and Langenberg [9] describe an approach and methods dealing with engineering analysis problems for product development and manufacturing in the context of computer-aided engineering (CAE) applications, grid infrastructure and how they can be solved.

3. Virtualization techniques

Virtualization has brought significant benefits to the operation of the IT infrastructure in companies. The key technical benefits are compatible with all standards x86 operating systems, the isolation of VMs, the encapsulation of software in a container, portability, rollbacks, high availability and resource sharing. Entrepreneurial advantages include cost savings by consolidating server hardware on a central instance, greater flexibility in the management, mobility and robustness of the infrastructure as well as a higher environmental impact by reducing the power consumption [10]. Especially the last point must be considered in terms of utilization, since physical environments rarely take a consistently high CPU power and therefore having increased power consumption by providing the power and cooling of hardware. A variety of factors acts on a VE (see Fig. 1).

Virtualization is divided into two main areas, hardware virtualization and software virtualization. The hardware virtualization refers to the creation of virtual instance of the entire system or individual physical hardware components whereas the software virtualization emulates the operating system or only single application(s) through the virtualizer.

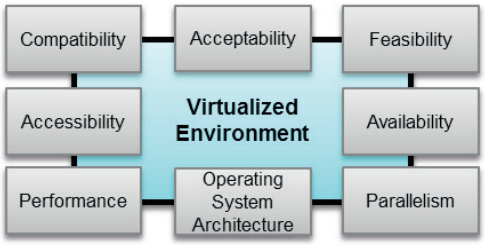

Fig. 1. Relevance factors for virtualized environments

A summary of current virtualization techniques was discussed by Rodríguez-Haro [11] as well as a comparison between software and hardware virtualization for x86 architectures was conducted by Adams and Ole [12]. Schlosser et al. examine the effects of network throughput of virtualized systems [13]. In this work, the lifecycle management for PDM systems is studied for the software virtualization of the entire operating environment, including applications. The system virtualization is enabled by a virtualizer that provides a runtime environment within an enclosed container, the so-called virtual machine VM. A VM is an implemented software abstraction of the real hardware, which is presented to the operating system (OS). The OS within the VM cannot directly access the system interfaces on the host system and interfaces need to be emulated. The emulation is necessary for virtualizers such as *Oracle VirtualBox*, *QEMU* or *VMware ESXi*. The Virtual Machine Monitor (VMM) or so-called hypervisor is responsible for the allocation and management of hardware resources, so that the OS within the VM can use all resources on request. This requires that all defined hardware components for the VM are emulated for the guest OS. Test scenarios can be performed in an unadulterated virtualized run-time environment to minimize subsequent problems in physical production environments. The disadvantage of this approach results in the emulation of the hardware resources that may lead to performance reduction. The disadvantages that may arise through virtualization must be individually analyzed and taken into account for the particular technical context of the embedded system.

4. Modeling in PDM systems

In order to perform the modeling of lifecycles for VMs, the characteristics of software products must be analyzed. Thus, there are products that can fulfill its function completely autonomously while other products cannot operate without prior configuration. The majority of embedded systems is involved in a technical context and depends on it in order to carry the application purpose. For example, the software inside time recording terminals with fingerprint authentication cannot operate without a fingerprint sensor, a navigation device software without a GPS receiver cannot calculate a route, and Point of Sale (POS) terminal software cannot perform cashless payments without a card reader. Such examples show that certain requirements must be placed for virtualized environments, in order to perform software

development and software test automation. Thus, certain functionality could be emulated by software inside the VM, if this seems possible. Smartphones already use emulators for the development of mobile applications (apps) that imitate a fully functional mobile device. Google provides for the Android mobile operating system the so-called *Android emulator* whereas Apple provides *iOS Simulator* and Microsoft the *Windows Phone Emulator*. More software emulation in other fields of embedded product would support the approach of VEs and requires a closer collaboration of hardware and software development teams.

4.1. Determination of the VM states

Products such as *ZENworks Orchestrator* by Novell or the open source virtualization toolkit *libvirt* have already defined rudimentary lifecycles for VMs. The purpose of such software is primarily related to the remote management of infrastructure components. The integration of virtualization into an existing process landscape for product engineering in PLM is not considered. An integration of such products can only be achieved by customization and through the development of additional interfaces and components. In order to represent systems lifecycle management for VEs in PDM systems, a division into the following three areas must be made:

- Lifecycle in the context of product development
- Lifecycle in the context of product testing
- Lifecycle of VMs

4.1.1. Lifecycle in the context of product development

The product lifecycle with embedded software is complex through partitioning and co-design of hardware and software (see Fig. 2). The architecture and modeling for software development and hardware development run through different phases. However, different development stages are passed through and while the engagement with each other and coordination between these two developments has an enormous importance. The success of the subsequent integration as well as the testing and validation depends largely on the collaboration of these two development phases. The creation of prototypes is made in the integration phase through components of the software development which are linked with those of the hardware design into a functional prototype. Designers can make changes on the prototype in this phase, before the final product for the customer will be built and deployed. The specification of the prototype defines the risk factors for the product to be published. The integration of VEs can be made for software development throughout the entire development phases (see Fig. 3). After the partitioning of hardware and software, the PDM systems could provide an appropriate VE for software engineers, which include all the tools needed for development. Such software can be CASE tools, integrated development environments (IDEs) and other tools used for the development that software developers need in the field of embedded software systems to perform their tasks.

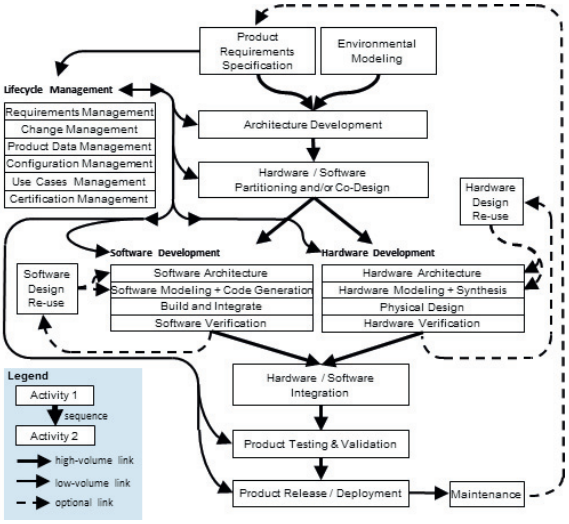

Fig. 2. Product lifecycle view for embedded systems [14]

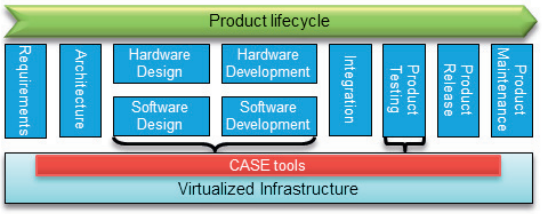

Fig. 3. Virtualization for embedded systems

4.1.2. Lifecycle in the context of software testing

Considering the development lifecycles with dependence on software, it can be seen that the complexity rapidly increases for a variety of hardware devices with different hardware versions and individual software versions. Software features, enhancements, bug fixes as well as specific hardware devices and versions must be tested. Test results for completed test cases must be documented and considered for further quality measures. Different Bill of Materials (BOMs) helps to consolidate information about the allowable combinations of compatible hardware and software versions of a product. The question is formulated by the configuration management perspective which hardware and software combinations represent valid configurations. This question can only be answered by development and testing teams. Virtualization could assist these teams to provide VEs for the test case execution. The test case results would be transferred back from the VE to the PDM system to generate a matrix of different hardware and software combinations.

4.1.3. Lifecycle of the VM

A lifecycle must be defined for the VE provision to manage the different VM states in the context of PDM (See Fig. 4). It is not meant to manage the operational states such as "*VM powerOn*" or "*VM powerOff*".

Fig. 4. Generic lifecycle for the provision of a VM

Rather, the relevant test conditions and boundary conditions should be considered in advance during the VM creation or selection to re-play a specific test scenario within a VE. Thereby factors such as the definition of VM hardware properties, guest OS type, virtual hardware interfaces and specific modifications within the guest OS represent important parameters in test scenarios. The VE provision lifecycle should manage tasks of an administrative level. These include the VM selection and identification from a pool, the configuration of selected VEs based on a specified profile and the deployment, archiving and disposal of VEs.

4.1.4. Process definition

The process definitions are one of the core functions of each PDM system. Therefore, the workflow management is the most pronounced component that uses visualization tools for the description of parallel and sequential processes [15]. Thus the basic functions of such tools only differ in the implementation by the respective manufacturers. For the definition of processes, tools such the *Business Modeler* application in *ENOVIA V6* and the *Workflow Designer* in *Teamcenter* are used. For the process modeling, activities can be defined that allow branching within a process. Likewise, processes can include sub-processes which are nested with each other. Tasks may be defined properly that the user intervention is necessary or that they are performed automatically. Actions automated tasks can be defined to call external applications.

5. Architecture

The architecture describes the following six parts: process definitions for software development, software testing and VM management; action handlers to trigger VM tasks; virtualization engine for the provision of VMs; VM management application for VM operations; VM hardware interfaces as well as the limitations of this architecture.

5.1. Process definitions

Firstly, the processes for the three areas Software Development, Software testing and VM Management must be defined. Secondly, the workflow instances are generated from the pre-defined processes which are used for the respective context.

5.1.1. Process definition for software development

The process definition for the provision of software development environments was determined as follows: firstly, an existing software development environment is selected from a pool of existing VEs. If no appropriate VE is available, it must be requested. Such a request results in a new task that is created and assigned to a person who is responsible for the generation of a virtual software development environment. This person adds the created VE in form of a template to the pool of existing VE templates and registers this template against the PDM system. Once this newly created VE template is available, the workflow owner will be informed. Secondly, the hardware requirements for the VE are selected. Therefore, the workflow owner can set parameters such as memory, disk space and hardware interfaces applying to the VE that has been previously selected. Finally, the automated tasks are processed in order to enable the provision of the VE. The tasks include the selection of the virtualization system, the deployment process, the compilation of information as well as the notification to the developer getting access to the provided VE. Once the VE is no longer required by the developer, it can be transformed in the state of archiving or disposal. It should be noted that one developer can have one or more VE instances. Moreover, a single instance could be used by several developers. The number of VE instances can be flexible adapted to the developers' needs.

5.1.2. The deployment process definition of software testing

The deployment of VEs for software testing differs only slightly to software development. Each deployment subject has its own pool of VE templates. A VE is assigned to a specific pool based on safety reasons. In a more comprehensive process definition it should be possible to choose a VE template which is exclusively provided to departments or groups. Based on the test scenario, the user is able to modify VE parameters more flexible to meet the hardware requirements.

5.1.3. Process definition for VM management

The VM process definition was designed simply. It contains only the basic steps (see Fig. 5). This means that no process branches or vendor specific steps were made in the definition to ensure the independence of the virtualization vendor. Thereby, vendor specific logic has been placed into the integration module. A complex modeling of the VM process depends on the depth of the virtualization integration into the PDM system. The advanced process steps include, for example, the physical location of VEs (required for site-related deployment), the deployment process (e.g. copy, modify and register the VM) and the power management (e.g. power on, power off, suspend, resume and pause).

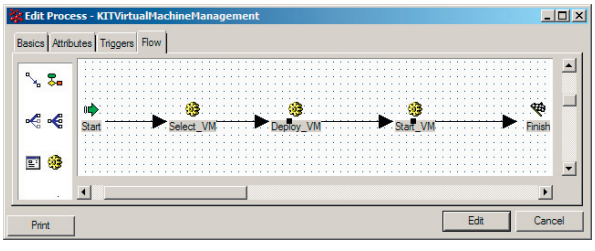

Fig. 5. VM process definition in ENOVIA Business Modeler

5.2. Action handlers

PDM systems offer a variety of options to recognize triggered process events. For the occurrence of expected events, predefined actions can be performed. This functionality is represented in *ENOVIA V6* as *triggers* and in *Teamcenter* by *Action handlers*. An action can perform a wide range of activities, for example, send notifications to process involved persons, call external applications and scripts or execute custom applications running within the PDM system. To define actions for events that occur, the current workflow state must be known. A workflow has defined several states, for example, *begin*, *finish*, *stop*, *pause*, *resume*, *apply*, *undo* or *skip*. The keyword for the corresponding status depends on the particular PDM system. In order to integrate the workflow process directly into an application, PDM systems offer an appropriate integration. The workflow component of PDM system directly calls the API when events occur. Therefore, applications must implement the integration of the workflow component. After the process definitions were completed and the associated action handler was defined by the individual process steps, external tools can be called by trigger applications. Therefore, the API interface of the virtualization software is used by the external application which was previously defined as trigger application in the PDM system.

5.3. Virtualization engine

As virtualizer, the *VMware ESXi* was used. The underlying hardware of the virtualization software is dictated by the respective manufacturers through compatibility lists, which previously had to pass a certification process. In recent years, blade servers have prevailed as a suitable hardware in companies for virtualization purposes. Blade servers are an assembly of independent hardware units, which have a compact size and allow a space-saving installation inside a blade chassis. While standard rack-mount server can independently work with power cord and network cable, blade server cannot operate without the blade chassis [16]. The *ESXi* software was developed as bare metal embedded hypervisors which is installed without an underlying OS. VMware provides as administration tools the *VMware vSphere Client* and the associated SDKs. The SOAP API allows a complete server administration systems and is suitable for the integration into existing automation systems, ERP systems and PDM systems.

5.4. VM management application

This component within the architecture represents the link between the PDM system with its lifecycle and processes as well as the virtualization software. The objective is to process incoming requests that were triggered by occurred events in the workflow and to return an appropriate response. The integration must ensure the dialogue with the PDM system and the virtualization software. For example, such tasks contain the physical access management to VM repositories, to process configurations modification of VMs, to initiate and manage deployment processes for virtualization server as well as to monitor and track the appropriate results to the PDM system. Further tasks include the VM power management as well as the archiving and disposal of VM instances which are no longer required. To allow the interaction between these two worlds, proprietary APIs of the respective manufacturers must be provided and implemented.

5.5. VM – hardware interfaces

An essential component within the virtualization software provides the I/O virtualization. Within the VE, the access to required hardware interfaces is not comparable as it is for the access of a real environment. Therefore, the I/O virtualization provides an extensive range of functions that address different topics such as security, I/O sharing, isolation and consolidation. The guest OS must have installed tools which provide a set of virtual drivers for graphics, storage, memory, networking, and multimedia. The performance through I/O virtualization by software is different compared to physical systems that can directly access I/O interfaces. For this reason, three primary approaches exist [17]: the direct I/O virtualization through the hypervisor, Pass-Through I/O, and Para-virtualized devices.

- Direct I/O virtualization: a virtual device is emulated by a device driver. The I/O stack has the task to translate the I/O requests from the guest OS to the host system. Therefore, the I/O stack manages the internal communication between the physical and virtual device. The performance of this approach is limited by the CPU of the host system. However, offers a complete independence from the hardware.
- Pass-Through I/O: this approach is designed to use the hardware of the host system directly for the guest OS. Therefore, an enhanced performance compared to the I/O virtualization is achieved by a low CPU utilization [18]. However, an independence of the complete hardware is no longer guaranteed. Thus, this approach is only advantageous for performance-critical applications.
- Para-virtualized devices: the guest OS interacts directly with the API of the host system's hardware. The API is provided by the virtualization layer. The device drivers for the guest OS are similar to those of the host system. Para-Virtualized devices have the advantage that there are more appropriate because they have a lower latency and higher throughput for I/O intensive applications. The complexity of the VMM is reduced, thereby specific drivers must be provided for different guest OS.

5.6. Limitations

Based on the architecture, three restrictions have been identified: performance, interface availability and offline availability.

- The performance of a VE is an important and critical element which has an enormous impact on the user experience. This concerns in particular VEs that require interactions with users. Users, who have to accept long

delays because of insufficient CPU power, will be quickly discouraged. Therefore, a virtual infrastructure must hold sufficient hardware resources to cover peaks in rush hours and prevent major performance bottlenecks. During the operation off-peak hours, enough hardware resources would be available in order to assign VEs more dynamic resources such as CPU or memory.

- Interface availability addresses the limitation of physical available interfaces of the host system. For example, two serial ports allow that only two serial devices can be connected. The physical expansion interfaces is particularly difficult and expensive for server hardware. Therefore, the use of emulators within the guest OS is advantageous as it is found already today for mobile OS like Android and iOS.
- Offline availability describes the use of VEs without an active connection to the corporate network. For safety reasons, this option should be avoided. This means that an active connection must always be available to the corporate network to access hosted VEs by users. With respect to safety, this limitation is advantageous because a higher protection can be established for product developments as it is the case for disconnected VEs.

6. Conclusions and future works

This paper has presented the architecture for the VE integration that provides significant advantages in the product development of embedded software systems. It is important to capture requirements for VEs and to keep enough hardware resources available. The limitation in accessing physical interfaces can be eliminated by emulators within the VM, but this could result in a reduction of performance. The advantages of the presented architecture through virtualization allow IT departments to manage resources more cost-effective through dynamic hardware allocation, reduce the time delays in provision and disaster recovery of VMs, and increase the availability of VM through hardware independence and more efficient data backup strategies. Further research is needed to access physical interfaces from another host system over a network protocol. Therefore, the implementation of emulators could be avoided and the limitation of hardware interactions through pure emulation could be repealed. For extensive product developments with embedded software, an integration of queuing systems in the process definition would be more beneficial. A third research field has been identified in the use and integration of cloud-based solutions for the deployment of VEs that enables independent software testing automation for embedded systems.

References

[1] ARTEMIS Joint Undertaking. The public private partnership for R & D Embedded Systems, http://www.artemis-ju.eu/, Accessed April 4, 2013.

[2] Ebert C., Jones C., Embedded Software: Facts, Figures, and Future. In: *Computer*, vol. 42, no. 4, pp. 42-52, April 2009.

[3] CTB Embedded Systems - Embedded Systems Guide, 2012: Application Software Development, http://www.embedded-systems-portal.com/CTB/Application_Software_Development,1001.html, Accessed April 11, 2013.

[4] Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e. V. (BITKOM), 2008. Studie zur Bedeutung des Sektors Embedded-Systeme in Deutschland. http://www.bitkom.org/files/documents/embedded_systeme_mit_grusswort_kleiner.pdf, Accessed April 16, 2013.

[5] Fabbre J., Entwicklungs-Tools: Wie sich der Entwicklungsprozess für Embedded-Software optimieren lässt, 2010, http://www.elektroniknet.de/embedded/entwicklungstools/artikel/30075/, Accessed April 19, 2013.

[6] Prell M.J., 2006, Underdogma: How America's enemies use our love for the underdog to trash American power, Dallas, TX, 2011.

[7] Muhammad A., Esque S., Aha L., Mattila J., Siuko M., Vilenius M., Järvenpää J., Irving M., Damiani C., Semeraro L., 2009, Combined application of Product Lifecycle and Software Configuration Management systems for ITER remote handling. In: *Fusion Engineering and Design*, vol. 84, no. 7-11, pp. 1367-1371, 2009.

[8] Do N., Chae G., 2011. A Product Data Management architecture for integrating hardware and software development. In: *Computers in Industry*, vol. 62, no. 8-9, pp. 854-863, 2011.

[9] Stark R., Hayka H., Langenberg D., 2009, New potentials for virtual product creation by utilizing grid technology. In: *CIRP Annals - Manufacturing Technology*, vol. 58, no. 1, pp. 143-146, 2009.

[10] Jin S., VMware VI and vSphere SDK. Managing the VMware Infrastructure and vSphere, Prentice Hall, 2009.

[11] Rodríguez-Haro F., Freitag F., Navarro L., Hernánchez-sánchez E., Farías-Mendoza N., Guerrero-Ibáñez J.A., González-Potes A., 2012. A summary of virtualization techniques. In: *Procedia Technology*, vol. 3, pp. 267-272, 2012.

[12] Adams K., Agesen O., A comparison of software and hardware techniques for x86 virtualization. In: *Proceedings of the 12th international conference on Architectural support for programming languages and operating systems* (ASPLOS XII). ACM, New York, NY, USA, pp. 2-13, 2006.

[13] Schlosser D., Duelli M., Goll S., 2011, Performance Comparison of Hardware Virtualization Platforms. In: *NETWORKING 2011* (Lecture Notes in Computer Science), [Domingo-Pascual J., Manzoni P., Palazzo S., Pont A., Scoglio C.], editors, vol. 6640, pp. 393-405, Berlin/Heidelberg, Springer, 2011

[14] CTB Embedded Systems - Embedded Systems Guide, 2012: Product Lifecycle View, http://www.embedded-systems-portal.com/CTB/Product_Lifecycle_View,2.html, Accessed April 11, 2013.

[15] Eigner M.; Stelzer R., Product Lifecycle Management. Ein Leitfaden für Product Development und Life Cycle Management. Berlin/Heidelberg, Springer, 2009.

[16] Sobotta A. T., Sobotta I. N., Gøtze J., Greening IT. How a greening IT can form a solid base for a low-carbon society, United States, Greening IT Initiative, 2010.

[17] Hu K., Zhou W., Jin H., Shao Z., Chen H., High-Quality Sound Device Virtualization in Xen Para-Virtualized Environment. In: *Lecture Notes in Electrical Engineering*, [Park J. J., Jin H., Liao X., Zheng R.], editors, pp. 529–537, Netherlands, 2011.

[18] Ueno H., Hasegawa S., Hasegawa T., Virtage: Server Virtualization with Hardware Transparency. In: *Euro-Par 2009 – Parallel Processing Workshops*, [Lin H., Alexander M., Forsell M., Knüpfer A., Prodan R., Sousa L., Streit A.], editors, vol. 6043, pp. 404-413, Berlin/Heidelberg, 2010.