

Adaptation of CCM Applications Based on Lightweight OS Virtualization

J. Cała, Ł. Kubica, W. Wiśniowski, K. Zieliński

Department of Computer Science
AGH – University of Science and Technology
Krakow, Poland
{jc, kz}@agh.edu.pl

Abstract—This paper presents the concept of applying OS lightweight virtualization to adaptation of CCM component-based systems. The presented approach, called virtual redeployment, is complementary to standard CCM deployment planning which also takes into account resource utilization of the execution environment. The adaptation loop presented in this work consists of sensors measuring the application performance, an adaptation strategy module that plans suitable control actions, and effectors implementing the specified plan. As a technology providing mechanisms for adaptation of an execution environment, Solaris 10 OS Container and Projects management has been chosen. The paper briefly describes the OS mechanisms used and shows their integration into a standard CCM deployment architecture. Furthermore, the proposed solution for adaptation of CCM applications is validated in the experimental study. The object of the study is a simple CCM application whose performance is controlled to satisfy possible user QoS requirements. The obtained results prove the effectiveness and usefulness of the proposed solution.

Adaptation, lightweight virtualization, CORBA component model

I. INTRODUCTION

The implementation of an adaptation process for computer systems requires construction of a feedback control loop typically built of sensors monitoring the current execution performance, a control system which implements an adaptation strategy, and effectors providing mechanisms for implementation of the control activity [14]. This simple model could be further refined with four steps: monitoring, analysis, planning and executing as proposed in the autonomic computing element model [10][21]. These steps are used for knowledge collection and discovery, applied to adaptation decision-making.

The concept of adaptive software environments is not new and there is a lot of successful applications done in this area [3][5][12][13] and works such as [16] provide an overview of this broad domain. However, progress in component-based software technologies and virtualization techniques of modern operating systems allows applying previously elaborated patterns in new domains such as adaptive component deployment.

From a wide spectrum of available virtualization technologies this study focuses on OS Containers which are

classified as a lightweight virtualization technique [18]. This technology was selected due to its scalability and accessibility in advanced operating systems such as Solaris 10. As a component model for distributed applications, the CORBA Component Model (CCM) [23] was chosen – one of the most general middleware models available.

A CCM application consists of a set of interconnected components which are deployed over a collection of component servers. Taking into account the existing lightweight virtualization technology, each CCM component server may be started in a separate OS container or project. The computer resources such as processor time, memory, number of threads etc. may be allocated to each container or project dynamically [25]. This creates the chance to control conditions of the application execution via modifications of resource allocation.

In the implementation of the feedback control loop two approaches can be distinguished. The first one assumes that monitoring functionality and notification mechanisms are integrated into a CCM application. In this case the application informs decision module about the observed performance, hence this solution may be described as adaptability-aware. The second approach has the monitoring infrastructure embedded into a CCM component server. Therefore, applications which are not adaptability-aware may also take advantage from the adaptable execution environment. This solution is, however, more complex, requiring implementation of ancillary mechanisms such as Container Portable Interceptors [24].

The goal of this paper is to present that lightweight OS virtualization in connection with feedback control loop provides effective means to supplement existing redeployment techniques [3][7] with so called *virtual redeployment*. The initial deployment of CCM components over component servers distributed in a computer environment may be considered static. However, the resources assigned to each node – represented by OS Containers – are adjusted according to the provided requirements. This approach is complementary to the adaptation performed in the application layer e.g. via a component migration mechanism [6].

The structure of the paper is as follows. Section 2 presents the problem of deployment of component-based applications in distributed environments. Three possible approached are

introduced to make the process adaptive. In Sect. 3 the OS Container virtualization technology is briefly described. Section 4 presents the solution which uses mechanisms offered by the operating system in order to adapt the component application execution. The verification of the proposed solution is described in Sect. 5 showing a strategy which is able to satisfy QoS requirement of the sample application. The paper is ended with conclusions.

II. POSSIBLE ADAPTIVE DEPLOYMENT STRATEGIES

In execution of component-based applications or, more generally, distributed systems consisting of many interconnected components a pivotal role is played by the application deployment [9]. This is because in order to make use of system modularization its components need to be distributed over many hosting nodes of the execution environment and deployment is means of facilitating this distribution. Generally speaking, deployment of component-based applications is a process of installing and configuring the application components in an execution environment according to a provided scheme. As a result, the deployed application is ready to be executed. Frequently, deployment tools offer also mechanisms to run the deployed application providing a complete solution for running complex systems.

According to the D&C specification [22], the deployment procedure consists of five steps: (1) installation in a software repository, (2) configuration of default properties, (3) planning how and where the software will run, (4) preparation the target domain to execute the application, and finally, (5) launching the software in the domain which includes instantiation, configuration and interconnection of the application components.

Two important notions related to the process of deployment are: target domain and application description. The former describes an execution environment expressing its capabilities, structure and attributes whereas the latter provides a structure, attributes as well as resource and QoS resources requirements of the interconnected application components.

The crucial part of a successful deployment procedure is a planning phase which involves searching for the best matching of the application description for the target domain. In order to perform reasonable matching, detailed and accurate descriptions of both the target domain and application components are required. Otherwise, the resulting plan, if ever found, might be ambiguous, impossible to implement or simply unsuitable. Unfortunately, providing the description is not a straightforward task. In the case of the target domain there is the problem of constantly changing capacities such as available CPU processing power, memory size, or network bandwidth. This issue is even more complicated today, when computer systems are very powerful and consolidation of many applications on a single machine is a common practice.

Similarly, describing application requirements is not an easy job. Rarely is it possible to affirm that an application has the same, stable requirements during its whole runtime. In real-life cases its needs change depending on factors such as the number of users, the state of the application, availability of resources etc. This leads to the conclusion that some automatic and

adaptive measures have to be provided to deploy component-based applications.

The possible approaches proposed thus far may be divided into two categories: static and dynamic. Static adaptive deployment, such as ones presented in [2][11], assumes that the deployment plan is prepared before distribution of components, taking into account the most recent information about the state of the target domain. Once components are assigned to nodes in the domain they remain there until the application is finished. Despite the fact that this approach allows us to achieve some gains, its planning phase is a computationally complex task and for more complicated applications and target domains becomes unreachable [15]. Moreover, when using the static deployment it is hardly possible to describe temporal dependencies of application requirements. For example, when a component is used by one user it may require 1 MB of memory but if used by two users its requirements may rise to 1.5 MB. Obviously, such dependencies may be even more complicated making the component description cumbersome or impossible to provide [7].

A more sophisticated approach involves dynamic, adaptive deployment. It assumes that distribution of components is not a single-step action but may evolve during application runtime. In consequence, it allows better adaptation and may follow changing usage patterns of the application. Moreover, through dynamic deployment it becomes possible to make use of this more relaxed component-to-node assignment and provide a simple plan at the beginning which will be further refined with consecutive adaptation steps. This makes dynamic deployment a highly attractive approach. The application and target domain descriptions are used only at the beginning of the process and may contain only suggestions about where to place the application components. Hence, the component developer and the application packager are both freed from providing exact requirements of the application. Similarly, the target description needs not to be very accurate and this two facts make the first stage of planning easier to perform.

Dynamic deployment adapts application execution by rearrangement of components in the target domain. It is worth to note, however, that this adaptation has two complementary forms. More specifically, it may influence the location of a component in the domain by moving it from one hosting node to another such as in [3][7]. Conversely, if the deployment process is performed over virtualized target domain elements, it is possible to adjust the elements themselves instead of changing the deployment plan. For example, increasing memory resources of a virtualized node hosting a component has the same result as moving the component to a node with a larger amount of memory.

This kind of adaptation – which we call *virtual redeployment* – may be performed in a transparent way from the application's point of view and seems to be far less complicated than other techniques e.g. a migration of live components among execution nodes. Moreover, detailed analysis has shown that by using modern virtualization technologies some QoS application requirements may be easily guaranteed, making virtual redeployment both an interesting and useful solution.

III. LIGHTWEIGHT OS VIRTUALIZATION

The selected technique of dynamic resource control is one possible approach from a set of different software adaptation methods [1]. However, modern advanced operating system environments have this mechanism built-in to better satisfy performance requirements of running applications. Lightweight OS virtualization, as it is called [18], has primary two distinguishable forms:

- ❑ Container-based, which involves software that virtualizes an operating system environment. There is only one underlying operating system kernel, which the containers enhance by providing distinct borders offering increased isolation between groups of processes. Containers do not emulate any of the underlying hardware. Instead, the virtualized OS or application communicates with the host OS to share resource usage, which then makes the appropriate calls to real hardware.
- ❑ Paravirtualization, which virtualizes part of an operating system environment but also selectively emulates the hardware devices that a virtualized OS requires. Paravirtualization provides both a virtual machine and access to the native hardware and thereby lets users run many instances of different OSes.

In the presented study the first kind of virtualization technology will be explored, as used in the Solaris 10 operating system. This technology is called lightweight because many thousands of containers may be created without degradation of system performance. Containers can be used to isolate and manage multiple applications on the same server reducing costs and complexity of the infrastructure. Furthermore, Solaris 10 supports even lighter virtualization mechanisms offering resource control called projects. The project serves as an administrative tag used to group related work in a manner deemed useful by resource management without providing the isolation level supported by containers. Consolidation of application instances into separate containers or projects on the same system enables them to run in a flexible space of resource allocation which may change as performance needs evolve.

In order to efficiently consolidate applications on a single system, the system must be able to flexibly respond to the varying workloads that are generated by the applications. When the resource management features of Solaris OS are not used the system responds to workload demands by providing equal access to the resources. However, the resource management instruments of Solaris OS offer the ability to tread workloads individually enabling administrators to:

- ❑ restrict access to a specific resource,
- ❑ offer resources to workloads on a preferential basis,
- ❑ isolate workloads from each other,
- ❑ deny resources or prefer one application over another for a larger set of allocations than otherwise permitted,
- ❑ prevent an application from consuming resources indiscriminately,

- ❑ change an application's priority based on external events,
- ❑ balance resource guarantees to a set of applications against the goal of maximizing system utilization.

These features enable Solaris containers and projects to deliver predictable levels of quality of service. Some of them are achieved due to scheduling which is a resource sharing mechanism that refers to making a sequence of resource allocation decisions at specific intervals. An application that does not need its current allocation leaves or is detached from the resource which is then made available for another application's use.

The fair share scheduler (FSS) is an example of a scheduling mechanism that manages application access to CPU resources in a controlled manner. The mechanism allocates CPU resources using CPU shares and basing on the number of the shares allocated ensures that the resources are fairly distributed among active containers and projects. Therefore, more important workloads should be allocated more CPU shares which define the portion of the CPU resources available to a project in a resource pool. It is important to note that CPU shares are not the same as CPU percentages. Shares define the relative importance of projects with respect to other projects.

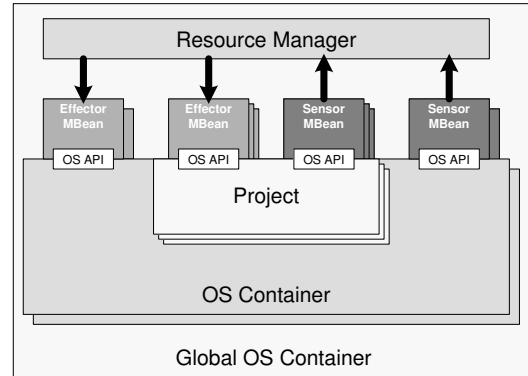


Figure 1. JIMS modules for Solaris 10 management.

Implementation of the concept of adaptive virtualized target domain using the Solaris OS Containers mechanism requires providing sensors and effectors for resource observation and management respectively. The most convenient way is to implement sensors and effectors using JMX technology and present them to a decision subsystem as Java managed beans (MBeans). This solution is described in more details in [4][20] and is roughly illustrated in Fig. 1. Each container and project has a separate instance of Effector MBean and Sensor MBean. The MBeans wrap up the native OS mechanisms of projects and containers resource management and expose them as JMX connectors. The connectors make it possible to couple MBeans with the decision subsystem using any of the RMI, HTTP or SNMP communication technologies.

IV. ARCHITECTURE OF OS CONTAINER-BASED ADAPTABLE CCM EXECUTION ENVIRONMENT

Basing on the OS resource management techniques available in the form of sensors and effectors it becomes

possible to extend DnC deployment model for CCM applications with adaptive mechanisms. From the adaptation point of view the proposed solution consists of three key elements (Fig. 2):

- ❑ a manageable target domain,
- ❑ a manageable application infrastructure¹, and
- ❑ an adaptive manager controlling both the application and target domain.

The adaptability is activated by two factors: (1) changes in the target domain e.g. CPU load, and (2) changes in the application state e.g. progress in data processing. The variations sensed are transformed by the manager into control signals put into action using one of the effectors. This allows the manager to take full advantage of the information provided from sensors and influence the application execution in a way limited only by an implementation of an adaptive strategy.

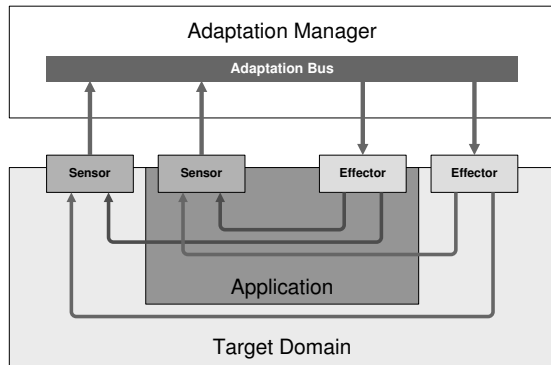


Figure 2. Possible adaptation loops in the presented solution

The presented general structure of adaptation mechanisms was integrated with the existing infrastructure of component deployment based on DnC specification. Figure 3 shows the architecture of the proposed extension.

The components in light gray are standard DnC management model elements and actors. The Planner provides a first-step static plan using a domain description and application structure, the Executor implements the plan distributing application components over a target domain whereas Node Managers manage execution of the application components on the node level. The standard DnC infrastructure has been extended with three additional elements (components in dark gray): *Target Domain Manager*, *Application Monitor*, and *Adaptive Planner*.

Target Domain Manager extends the standard DnC *Target Manager* with the ability to monitor and manage virtual resources of Solaris OS. Offering both sensors to observe the target as well as effectors to manage it, *Target Domain Manager* facilitates realization of virtual redeployment. *Adaptive Planner* is also supported by *Application Monitor* which delivers information about the state of application

¹ Currently, the application infrastructure realizes sensing only, however, the work is in progress on integrating the solution with the implemented migration mechanism [5].

components and level of traffic on their communication links. Currently, *Application Monitor* requires component modification – making the application adaptive-aware but work is in progress to use *Container Portable Interceptors* [24] to monitor the application state without any influence on the application business logic.

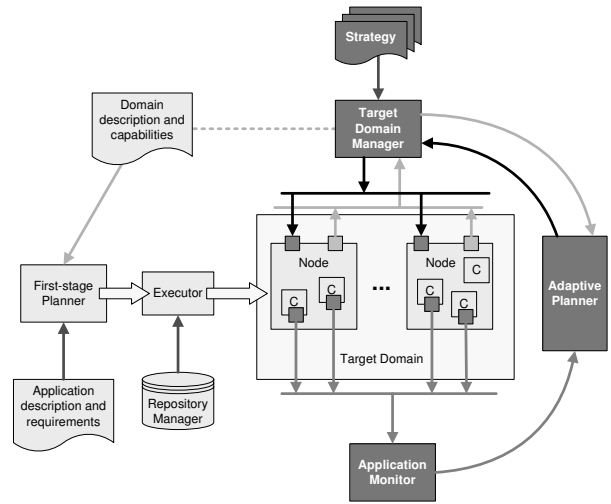


Figure 3. An overview of the architecture of the adaptive deployment tool

The most important and interesting part of the extended deployment infrastructure is the *Adaptive Planner* which circuits the possible adaptation loops. A key role in the planner operation is played by strategies delivering the expected adaptation logic. Strategies are procedures which process the domain structure and state, the current component-to-node assignment, and application monitoring data. Having this information, a strategy can make a decision whether any kind of rearrangement is required or not. New component deployment is then taken into account by *Adaptive Planner* which performs the required changes in the domain using *Target Domain Manager*.

To present the capabilities and verify the concept based on lightweight virtualization technology three testing strategies were implemented:

- ❑ a strategy maximizing application throughput,
- ❑ a strategy optimizing resource usage, and
- ❑ a strategy satisfying QoS requirements.

The following section presents an experimental study with details of the strategy for satisfying QoS requirements.

V. EXPERIMENTAL STUDY

The main goal of this study is to present that target domain adaptation by run-time control of Solaris projects is a valid concept which may be useful for example in tuning QoS-aware CCM applications.

The study was performed using the software configuration shown in Tab. 1. The system was installed on a single computer equipped with the Intel Pentium M745 1,8GHz processor and 768 MB of memory.

TABLE I. SOFTWARE CONFIGURATION

Software Type	Version
Java HotSpot	1.5.0_06 mixed mode sharing
JacORB	2.2
OpenCCM	0.9.0
Solaris	5.10

The architecture of the testing application is shown in Fig. 4. It includes a simple streaming service with three H.264 codec simulator components characterized by different computational complexity. Two of them implement the Extended Profile (XP) at level 1 and 2 whereas the third one implements the Baseline Profile (BP) at level 1. These codecs are used for raw video stream compression. The compressed video rate is measured and sent to the Application Monitor. The monitoring functionality is a part of the application not a middleware infrastructure, hence the investigated case has to be classified as an adaptation-aware application.

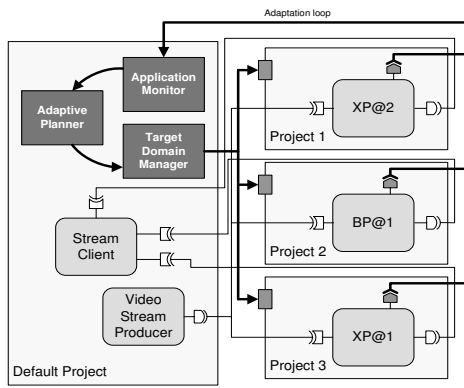


Figure 4. The architecture of the testing application with marked the adaptation loop.

Due to different complexity of the codecs the only way to control the rate of the generated data is the allocation of suitable amount of CPU shares for each of them. For this purpose each codec is started in a separate project and controlled via project effectors provided by the Target Domain Manager MBean. In this way the effectors are able to influence the performance of codec components separately, which is observed by the Application Monitor. The adaptation loop is closed by the Adaptive Planner which applies a given strategy using monitoring information and the effectors.

The goal of the strategy implemented by Adaptive Planner, in this example, is to meet user QoS requirements specified using a pairwise relative performance metrics. Since FSS offers relative guarantees too, it was easier for us to develop an algorithm satisfying relative than e.g. absolute guarantees.

In order to obtain a reference point for the further results, Fig. 5 presents a situation where all codecs are started in one project. This is, in fact, a common situation when an application is run without any QoS control. In this case codec components compete for CPU time resulting in transfer rates observed by the monitor as higher for components with low CPU requirements and lower for more demanding ones. The observation leads to the following inequality:

$$Tr1 < Tr3 < Tr2, \quad (1)$$

where $Tr1$, $Tr2$, $Tr3$ are the transfer rates obtained by $XP@2$, $BP@1$, and $XP@1$ respectively.

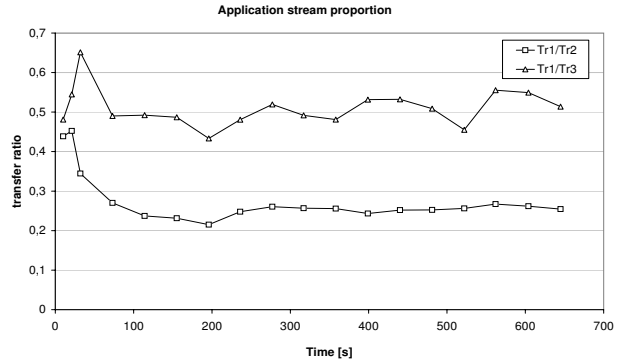


Figure 5. Transfer ratio of video data streams generated by codecs running in the same project.

Different situation is shown in Fig. 6. It illustrates that the ratio of the generated stream (output ratio) can be controlled according to client requirements (expected ratio). When codecs are started in different projects, Adaptive Planner is able to adapt the transfer rate of a codec to the requested level by changing the CPU shares allocated to the projects.

The adaptation strategy executed by the planner is as follows: during every time interval (arbitrarily set to 80 sec.) a linear extrapolation based on transfer rates is performed. The predicted value is decreased by a correction which is inversely proportional to the difference between the previously estimated result and the expected value.

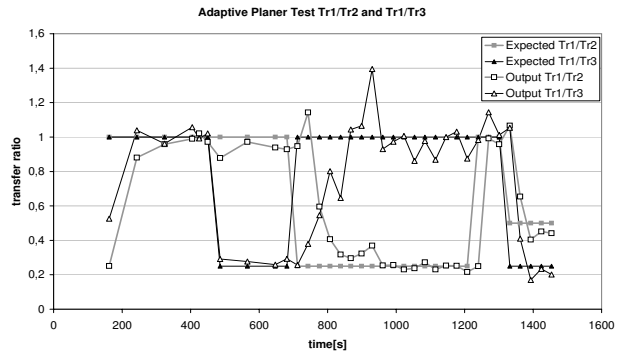


Figure 6. Adaptive control of transfer ratio of three streams.

The presented strategy can be easily replaced by a more sophisticated solution based on control theory [1][8]. Nevertheless, the obtained results are valuable enough to prove that this kind of adaptation is useful. In Fig. 7 the period of day when the relation between generated video streams should be sustained was selected. In this time the adaptation strategy is executed forcing $Tr1$, $Tr2$ and $Tr3$ streams to follow the requested proportion. During the rest of the time, $Tr1$ stream generation is dominated by $Tr2$ and $Tr3$ (ratio below 1). In this period of time the adaptation strategy tries to maximize transfer throughput.

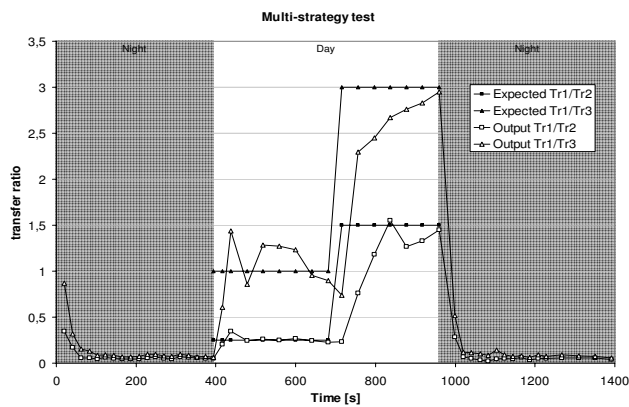


Figure 7. Periodical execution of adaptation strategy.

VI. CONCLUSIONS

The intention of this paper is to present that a feedback control system using the lightweight OS virtualization may greatly support deployment of component-based applications. Due to component-based architecture we can easily separate environment and application monitoring and management infrastructure from the application business logic. Moreover, support from feedback control and virtualization allows us to realize a virtual redeployment of application components instead of their physical movement or replication. This technique is complementary to other mechanisms used in dynamic component deployment.

Additionally, the presented study shows that virtualization of the target domain might be very useful for QoS-aware CCM applications not only in real-time scenarios [19] but also in common use. However, a full exploitation of the potential offered by the adaptive control via virtualized OS resources is a complex issue. Computer system models might be derived from control theory as P, PI and PID digital controllers but real component-based applications run over a distributed computer system are often far too complex to follow strict assumptions of the control theory. However, the proposed architecture creates an opportunity for further research in this area.

Obviously, the presented solution has its limitations. The main one is the requirement for retrofitting existing applications with monitoring sensors able to interact with the provided infrastructure. This will be solved by Container Portable Interceptors whose implementation is one of the aims of future work. An interesting research direction is also the integration of live migration mechanisms as application effectors in the architecture. This would open yet another possibility to control application performance.

REFERENCES

- [1] T.F. Abdelzaher, J.A. Stankovic, C. Lu, R. Zhang, Y. Lu, Feedback Performance Control in Software Services, *IEEE Cont. Systems*, 2003.
- [2] P. Antoniewski, L. Cygan, J. Cała, K. Zieliński, Extension of CCM Environment with an Adaptive Planning Mechanism, In *Proceedings of International Joint Conferences on Computer, Information, and Systems Sciences, and Engineering*, December 2006.

- [3] P. Backx, B. Van Den Bossche, B. Dhoedt, P. Demeester, AADD: Autonomic Adaptive Distributed Deployment of Component-Based Services, In *Proceeding of the EuroIMSMA 2005*.
- [4] K. Balos, and K. Zieliński, "JIMS - the Uniform Approach to Grid Infrastructure and Application Monitoring", *Cracow Grid Workshop 2004, Workshop Proceedings*, 2005, pp. 160–167.
- [5] J. Buisson, F. Andre, J.-L. Pizat, Performance and practicability of dynamic adaptation for parallel computing: an experience feedback from Dynaco, *IRISA research report*, 2006.
- [6] J. Cała, Migration of Components with OpenCCM, Technical report AGH-UST Distributed Systems Research Group, March 2006.
- [7] J. Dubus, P. Merle, Autonomous Deployment and Reconfiguration of Component-Based Applications in Open Distributed Environments. In *Proceedings of the 8th International OTM Symposium on Distributed Objects and Applications (DOA'06)*, November 2006, pp 26–27.
- [8] J.L. Hellerstein, Y. Diao, S. Parekh, D.M. Tilbury, *Feedback Control of Computing Systems*, A John Wiley & Sons, Inc. Publication, 2004.
- [9] A.A. Ivan, Partitionable Service Framework: Seamless Access to Distributed Applications, A Dissertation Presented to the Department of Computer Science of the New York University, September 2004.
- [10] J. O. Kephart, "Research challenges of autonomic computing", *ICSE '05: Proceedings of the 27th international conference on Software engineering*, ACM Press, New York, NY, USA, 2005, pp. 15–22.
- [11] T. Kichkaylo, Timeless Planning and the Component Placement Problem, *ICAPS 2004, Workshop on Planning and Scheduling for Web and Grid Services*, 2004.
- [12] B. Li, K. Nahrstedt, A Control-Based Middleware Framework for Quality of Service Adaptations, *IEEE Journal on Selected Areas of Communication*, September 1999.
- [13] X. Liu, X. Zhu, S. Singhal, M. Arlitt, Adaptive Entitlement Control of Resource Containers on Shared Servers, *Integrated Network Management*, 2005, pp. 163-176.
- [14] P.K. McKinley, S.M. Sadjadi, E.P. Kasten, B.H. Cheng, Composing Adaptive Software, *IEEE Computer*, July 2004, pp. 56–64.
- [15] M. Mikić-Rakić, Software Architectural Support for Disconnected Operation in Distributed Environments, A Dissertation Presented to the Faculty of the Grad. School Univ. of Southern California, 2004.
- [16] S.M. Sadjadi, A Survey of Adaptive Middleware, Technical Report MSU-CSE-03-35, *Comp. Sci. and Eng.*, Michigan State Univ., 2003.
- [17] J. Strassner "Policy-Based Network Management: Solutions for the Next Generation", *The Morgan Kaufmann Series in Networking*, San Francisco, CA, 2004.
- [18] S.J. Vaughan-Nichols, New Approach to Virtualization Is a Lightweight, *IEEE Computer*, November 2006, pp. 12-16.
- [19] N. Wang, D.C. Schmidt, M. Kircher, Towards an adaptive and reflective middleware framework for QoS-enabled CORBA component model applications, *Distributed System online special issue on Reflective Middleware*, 2003.
- [20] K. Zieliński, M. Jarzab, D. Wiczorek, and K. Balos, "JIMS Extensions for Resource Monitoring and Management of Solaris 10", *Advancing Science through Computation - ICCS 2006*, LNCS 3994, Springer-Verlag, Berlin/Heidelberg, 2006, pp. 1039–1046.
- [21] Autonomic Computing IBM Research. <http://www.research.ibm.com/autonomic>
- [22] Object Management Group, *Deployment and Configuration of Component-based Distributed Applications*, V4.0, April 2006.
- [23] Object Management Group, *CORBA Component Model*, V4.0, April 2006.
- [24] Object Management Group, *QoS for CCM final adopted specification*, April 2006.
- [25] Sun Microsystems, Inc., *Solaris 10 Resource Manager Developer's Guide*, 2005.