

DYNAMIC GRAPHICAL USER INTERFACE

AJMAL BEG

Author and/or publisher shall not be liable for any kind of direct and/or indirect loss as a result of using information in this book.

ISBN: 978-0-9805610-4-3

Dedicated to my family

Introduction

Many of the enterprise systems used in core business processes in large organizations consist of persistence layer, business logic layer and the presentation layer. Some of the enterprise systems completely separate the presentation layer from the business logic and persistence layer, thus allowing usage of multiple versions of the graphical user interface with the same business logic. These solutions further provide end-user means at the user client to manually switch the version of the graphical user interface at the presentation layer. One of the main merits of providing functionality of switching between different is that it allows the user to switch to low hardware resource consuming graphical user interface (GUI) version if there are not sufficient hardware resources available at the client machine. This book describes a method, which dynamically checks the availability of the hardware resources at the client and dynamically switches the graphical user interface without any manual intervention of the user. The merit of the proposed approach is that it help user run multiple resource consuming applications presentation layers concurrently at optimal performance while minimizing the risks of creating CPU or RAM bottleneck at the client machine.

Chapter 1

Need for dynamic graphical user interface

ERP systems are widely used in organizations of different sizes (Klaus et. al. 2000; Umar 2004). Many critical business applications such as ERP systems consist of following three computing layers (Frank 2004; Chan 1999; Schneider 2006):

- Persistence layer which resides on single or multiple database systems.
- Business logic layer which is stored and executed on the application servers.
- Presentation logic which is executed on web or windows clients of the users.

Each of the above computing layers consumes hardware resources such as CPU and memory.

Different approaches are used to optimize the utilization of the hardware resources on each of the above computing layers. For example, the hardware resources at the persistence layer are generally optimized by the database tuning experts using approaches such as:

- Optimizing of application server technical environment such as tuning of database data buffer and cursor caches (Schneider 2006).
- Using fast data storages (Seamons et. al. 1995).
- Optimal distribution of data files on disk storages (Pathak 1990).
- Creating of appropriate indexes on different database tables (Lifschitz 2005).
- Maintaining of database statistics to help database make right decision about access path selection (Ruberg 2003).
- Reorganization of the database indexes and files.
- Archiving of tables to reduce the size of very large database tables.

- Load balancing with multiple databases (Bouganim, 1996).
- Optimizing of batch jobs scheduling to flatten the peak load (Feitelson, 1997).
- Technical optimizing of operating system environment on which database runs (Musumeci et. al. 2002).

Typical approaches used by the technical and functional experts to optimize the hardware resource utilization at the application server are:

- Optimizing of database technical environment such as tuning of data buffers, memory paging areas and processes handling the data (Millsap, 2003).
- Technical optimizing of operating system on which application servers run (Musumeci et. al. 2002).
- Tuning of the expensive SQL statements which are issued by the business logic and puts extra load on the database resources (Burlison 2001; Henderson 2001; Gulutzan 2002).
- Program optimizing to reduce the volume of the data which is brought from database and saved in the internal tables at the application server (Crawford, 2000).
- Simplifying of business logic to reduce the resource requirements of the program.

User Interface is well explored area. Significant number of user interface tools, GUI design languages and methods has been investigated (Myers 1995; Bederson et. al. 2000; Myers et. al 2000; Abrams et. al 1999). However, optimization of presentation layer for large business applications such as ERP is not an extensive research focus area. The only approaches available to optimize the hardware utilization at the presentation layer in large scale business applications are:

- Switching to non-graphical version of the presentation layer. Such as accessing contents without rich graphical images.
- Switching to light version of the GUI controls where the presentation layer provides multiple GUI controls versions with capability to switch among them.

The first option is only available for those business applications which provide two versions of contents to be presented at the application layer. One version is less graphical than the other available version and consumes less hardware resources at the presentation layer. The demerits of adopting the first approach are:

- Extra workload and the development cost for the software manufacturer and the content developer as there is needs to prepare two versions of the contents.
- Non graphical version of the contents may be more difficult to comprehend compared to graphical version of the contents.
- Not all information can be presented in a non-graphical way. Such graphical contents are generally less informative compared to the graphical version of the contents

The second approach completely separates the presentation and the business logic layer and, provides at least one set of light GUI controls and another set of heavy GUI controls. The light GUI consumes less hardware resources and the end user can switch between heavy and the light GUI through easy to use interface to reduce the hardware resource consumption. ERP solutions from SAP provide the functionality at the presentation level to switch between two versions of GUI controls (Schneider 2006). Heavy version of GUI controls uses more resources compared to light version of GUI controls. The functionality allows switching among the two versions, thus allowing the client to be run on client machines which are not very powerful. The second approach is superior to the first approach as it does not have the above mentioned demerits of the first approach. However, the issues with adopting the second approaches from the point of non-technical end users are:

- The hardware of end user which runs the presentation layer has number of software

applications and other utilities installed such as AntiVirus Tool and Automatic Backup Tools. Different software applications run at the user PC without proper scheduling unlike background jobs which are scheduled after taking into account the available hardware resources. The hardware resource consumption at the client varies over time and is generally unpredictable. Permanently switching the GUI version to the light version may result in users using less graphical GUI interface even when the hardware resources are available to enjoy resource consuming but graphically rich user interface.

- The end user machine generally does not provide facility to exactly determine the cause of slow response of applications at the presentation layer. The slow performance of the program at the end user machine can be due to other potential problems at the business logic layer and the persistence layer. Just switching to a light version of GUI may not help improve the performance for the end user.
- Even when the merits of switching to light version of the GUI set are evident, it is difficult to determine how much performance can be gained by switching between light GUI set and heavy GUI set version.
- Manually changing between GUI sets can be frustrating experience sometimes, when resource consumption on the client machine is frequently changing.

To address the above issues, this publication describes a new method to allow automatic switching of the graphical user interface to the less graphical GUI control set when there is resource crunch on the client machine and switching GUI sets really helps improve performance of the presentation layer. Chapter 2 describes the architecture which allows the dynamic switching of the graphical user interface. Chapter 3 details the flow of enabling dynamic switching of the graphical user interface. Summary of the book is described in Chapter 4.

Chapter 2

Architecture

Figure 1 shows the overall architecture which allows the dynamic switching of the graphical user interface at the presentation layer while considering dynamic hardware resource consumptions at the client machine. The business application consists of three layers. The persistence layer exists at the database. The business logic layer exists at the middleware. The presentation layer is executed at the client machines of the business application server. The presentation layer can be a functionality installed on the client or it can be a program which runs in the browser at the client machine. At the end users machine a Test Program is scheduled, which runs periodically and determines following two aspects:

- Impact on performance when presentation layer switches from heavy GUI to a light GUI.
- Impact on performance when presentation layer switches from light GUI to heavy GUI.

GUI Performance Monitor Server plays a central role in the dynamic switching of the graphical user interface at the presentation layer. GUI Performance Monitor Server performs following functions:

- It receives request from client machines for generation of Test Programs.
- It installs the Test Programs on the client machines.
- It schedules the Test Programs.

The architecture shown in Figure 1 is a simplified version of corporate systems. The corporate system landscape may consist of multiple databases. Similarly, the clients may be accessing the business logic which is spread over multiple application servers. Multiple GUI Performance Monitor Servers using load balancing techniques may exist in the case of very large corporate network. In the case of a small system landscape, machine hosting application server or the database server may also provide the functionality of the GUI Performance Monitor.

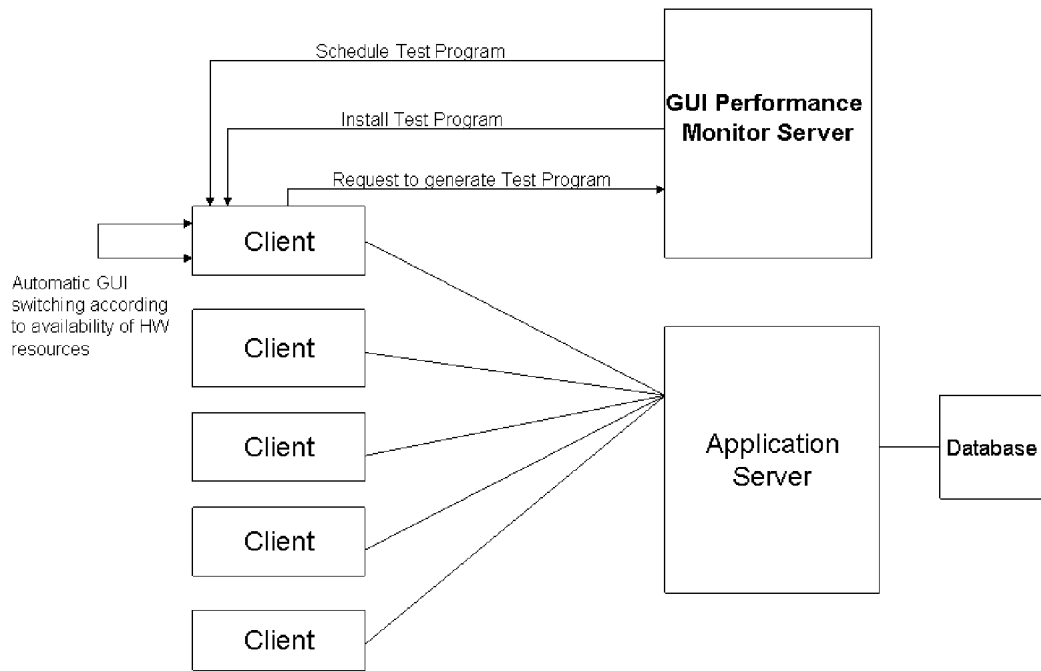


Fig 1: Architecture of dynamic graphical user interface approach

Chapter 3

Enabling dynamic graphical user interface

Figure 2 illustrates the steps involved in enabling dynamic graphical user switching at the client machine. The steps involved are:

- The program which requires dynamic graphical user interface is identified.
- The Description of the test is generated in XML to describe the test that is to be run on the client machine.
- The description of the test is transferred to the GUI Performance Monitor Server.
- If the a test already exist matching the Test Description then the Test Program is retrieved from the database and is installed on the client machine of the end user.
- If no Test Program with the received Test Description is available in database, a new Test Program is generated and installed on the client PC.
- The test is scheduled on the client machine, where it runs performance test periodically.

The remaining part of this chapter describes different steps which are generally expected to be involved in enabling dynamic graphical user interface at the client machine of the end user.

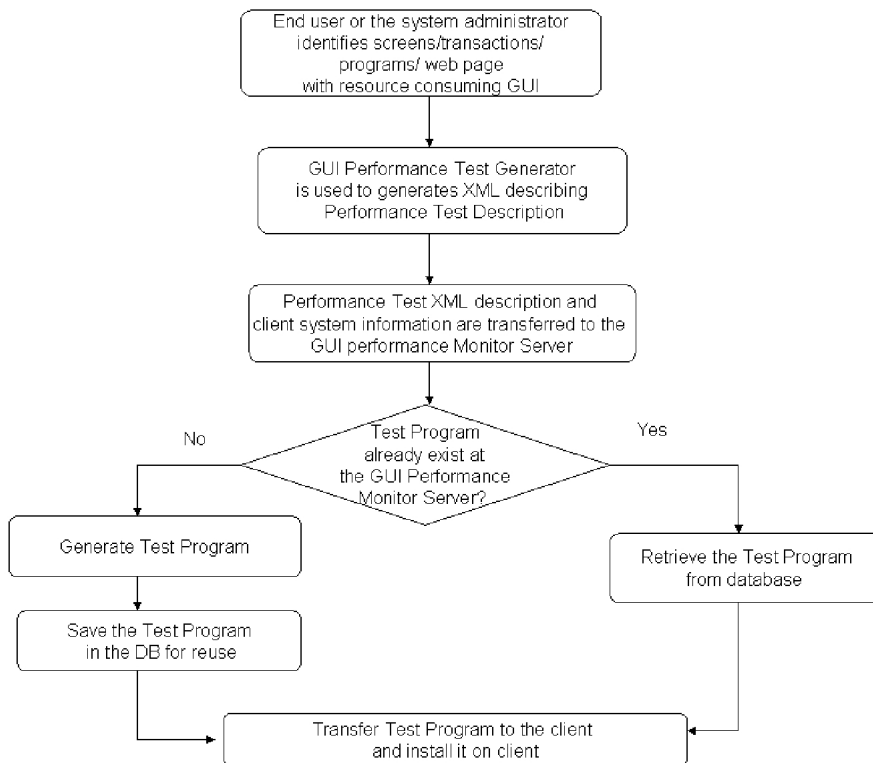


Fig 2: Generating and installing Test Program

3.1 Identifying programs with performance issues

There are two scenarios for using the dynamic graphical user interface:

- The dynamic graphical user interface can be enabled for all type of software or functionalities that run on the client machine.
- The dynamic graphical user interface can be enabled only for those programs or functionalities for which the end user complains about the performance

The step 1 in the flow shown in Figure 2 refers to the second scenario, in which the dynamic user interface is enabled only for the screens, programs or web pages which have the potential of consuming higher hardware resources. The end user or the administrator can determines such programs.

3.2 Generating performance test description

Figure 3 illustrates the user interface which can be used to generate XML describing the Performance Test Description at the second step in the above flow diagram. The end user may select all the elements in a screen to be evaluated for performance testing or just one of the elements contained in the program which consumes the most computing resources. The user interface allows selecting the controls within the program that can be tested for performance gains by switching to the GUI controls which consumes less computing resources. Let's assume a scenario, where a user of a business application generally displays large number of data in a table and experiences slower response. By switching from a table component which provides full spreadsheet capabilities to a simple table which does not provide calculation capability and only displays the table contents, may help the user meet the requirement of better performance for displaying data. The user wants to dynamically switch to a table component which consumes less computing resources at the time the client machine is experiences the resource crunch. The user selects the table as the component for which better performance is desired using the interface shown in Figure 3.

For an application, which provides capability to switch among two types of tables, one which is

with full spreadsheet capability and the one simple table which only display table, the test description is generated at the client in XML and sent to the GUI Performance Monitor Server. The task for generating the XML can be also done at the server. The XML describes the two table controls which may be changed dynamically.

```
<TEST NAME=TableWithLargeNumberOfRows>
  <IMPLEMENTATION ID=1111>
    <CLASSNAME>TableWithSpreadSheetCapability</CLASSNAME>
    <MANUFACTURER>SAP</ MANUFACTURER>
    <VERSION>3.0</VERSION>
    <PROPERTY NAME="NO_OF_COLUMNS" VALUE="10" />
    <PROPERTY NAME="NO_OF_ROWS" VALUE="1000" /> <PROPERTY
    NAME="EDITABLE_CELLS" VALUE="YES" /> <PROPERTY
    NAME="IMAGE_ALLOWED" VALUE="YES" />
    <PROPERTYNAME="CALCULATION_ALLOWED" VALUE="YES" />
    <PROPERTY NAME="MULTIPLE_FONTS_ALLOWED" VALUE="YES" />
  </ IMPLEMENTATION >
  <IMPLEMENTATION ID=2222>
    <CLASSNAME>SimpleTable</CLASSNAME>
    <MANUFACTURER>SAP</ MANUFACTURER>
    <VERSION>3.0</VERSION>
    <PROPERTY NAME="NO_OF_COLUMNS" VALUE="10" />
    <PROPERTY NAME="NO_OF_ROWS" VALUE="1000" /> <PROPERTY
    NAME="EDITABLE_CELLS" VALUE="NO" /> <PROPERTY
    NAME="IMAGE_ALLOWED" VALUE="NO" />
    <PROPERTYNAME="CALCULATION_ALLOWED" VALUE="NO" />
    <PROPERTY NAME="MULTIPLE_FONTS_ALLOWED" VALUE="NO" />
  </ IMPLEMENTATION >
</TEST>
```

The XML code below represents another test generation example, where a user faces performance problems while playing video player at the time of high HW resource consumption. In this case, only one video player is available and the only feasible way to improve performance at the client machine is to change the size of the video screen.

<TEST NAME=VideoImages>

<IMPLEMENTATION ID=aaa>

<CLASSNAME>LargeScreenVideoConfiguration</CLASSNAME>

<MANUFACTURER>SAP</ MANUFACTURER>

<VERSION>4.0</VERSION>

<PROPERTY NAME="FRAME_SIZE_UNIT" VALUE="PIXELS" />

<PROPERTY NAME="FRAME_WIDTH" VALUE="300"/>

<PROPERTY NAME="FRAME_HEIGHT" VALUE="200"/>

<PROPERTYNAME="IMAGE_QUALITY" VALUE="HIGH"/>

</ IMPLEMENTATION >

<IMPLEMENTATION ID=bbb>

<CLASSNAME>SmallScreenQualityVideoConfiguration</CLASSNAME>

<MANUFACTURER>SAP</ MANUFACTURER>

<VERSION>4.0</VERSION>

<PROPERTY NAME="FRAME_SIZE_UNIT" VALUE="PIXELS"/>

<PROPERTY NAME="FRAME_WIDTH" VALUE="200"/>

<PROPERTY NAME="FRAME_HEIGHT" VALUE="170"/>

<PROPERTYNAME="IMAGE_QUALITY" VALUE="MEDIUM" />

</IMPLEMENTATION>

</TEST>

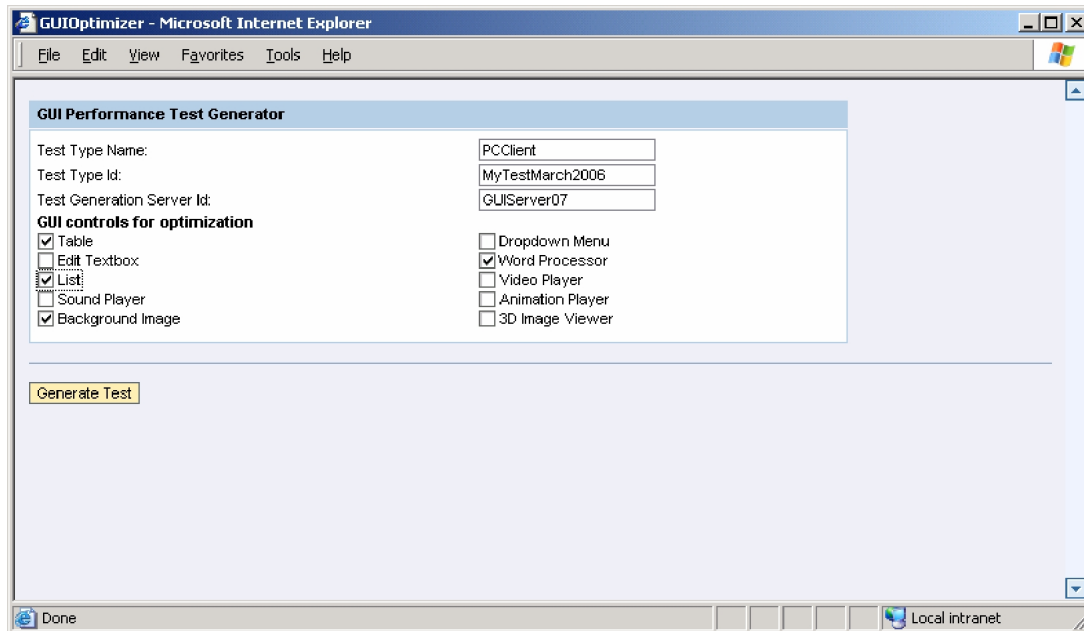


Fig 3: User interface for creating description of the performance test

3.3 Scheduling

Figure 4 shows a graphical user interface that can be used to schedule the Test Program on the end user machine. It can be used to select a test from number of different tests already created. It can be used to select the application and the server on which it is running. The presentation layer of the selected application will be monitored for dynamic switching of graphical user interface according to availability of the resources at the client machine. The graphical user interface can be used to describe the period of time during which the dynamic switching of the graphical user interface will be valid. The same interface can include the HW resource consumption thresholds at which the dynamic graphical user interface switching will be activated. The field with label “Monitoring triggering GUI Time(%)” in Figure 5 configures the Test Program to activate the dynamic switching of the graphical user interface, if time spent on generating the screens at the client is more than 50%

of the total time of processing at the presentation layer.

Figure 5 shows how the Test Programs run on the client machine. Test Programs generates dummy screens based on the Test Description and fills it with dummy data while measuring the time spent on generating the screen in background. In case, light GUI controls are in use, it looks for opportunity to switch back to heavy GUI controls as soon as the hardware resources are available on the client machine layer. At time when heavy GUI controls are in use, it runs tests to see if there is a hardware resource problem at the presentation layer and switching to light version of GUI control will help. If switching to alternative GUI control helps, it switches to alternative version of GUI controls.

In case, the presentation layer runs in the web client, where the html code is sent from a server, the switching between the light and heavy GUI at the presentation layer can be realized using different methods such as:

- Changing the contents of the JavaScript or html that is sent from the application server to the web client.
- Disabling script usage on the web client.
- Sending different graphical contents from the application server to the web client.
- Changing the plug-in being used

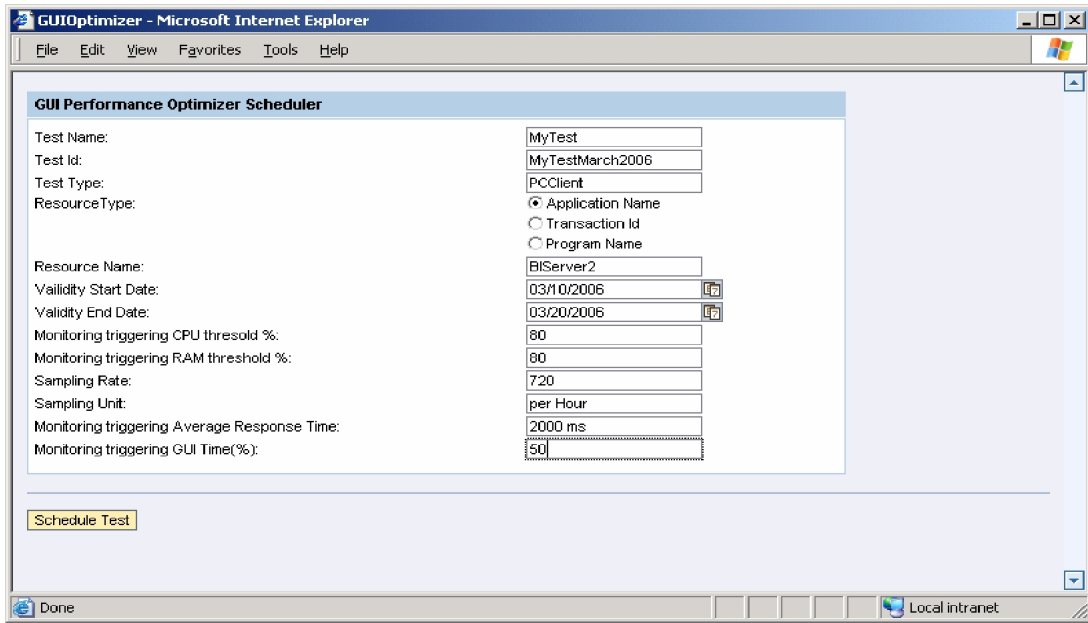


Figure 4: User interface for scheduling Test Program

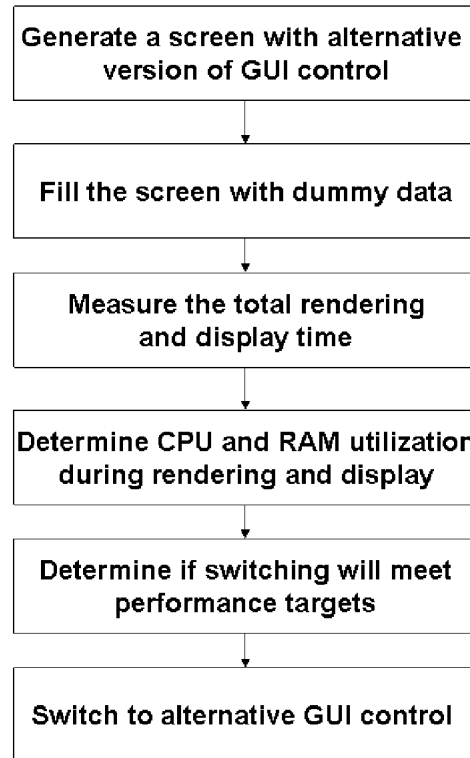


Figure 5: Functionality of Test Program running at the presentation layer

Chapter 4

Summary

Computing systems consisting of persistence layer, business logic layer and the presentation layer are extensively used in corporations. Persistence layer and the business logic layer are generally the focus for performance optimization. The presentation layer runs on the client machines and generally requires quite extensive computing resources for smooth running of presentation layer. For a company with large number of client machines, the cost of purchasing new client machines can be significant as graphical user interface at the presentation layer becomes more and more resource consuming due to arrival of more functional and graphical GUI controls. This book presents a method that can be used to dynamically change the user interface according to the availability of the hardware resources at the client machine without any manual intervention. It allows the user run concurrently presentation layer of multiple applications without creating resource crunch at the client PC. The described technique can be used also in case of web applications.

References

- [1] A. Umar, Third Generation Distributed Computing Environment. Nge Solution Inc, 2004.
- [2] B. A. Myers, User interface software tools, ACM Transactions on Computer-Human Interaction 2-1 (1995) 64-103.
- [3] B. B. Bederson, J. Meyer, L. Good, An Extensible Zoomable User Interface Graphics Toolkit in Java, Proceedings of ACM Symposium on User Interface and Software Technology (2000) 171-180.
- [4] B. Myers, S. E. Hudson, R. Pausch, Past, present and future of user interface software tools, ACM Transactions of Computer-Human Interaction 7-1 (2000) 3-28.
- [5] C. Millsap, Optimizing Oracle Performance, O'Reilly, 2003.
- [6] D.K. Burleson, Oracle High-Performance SQL Tuning, McGraw-Hill, 2001.
- [7] D. G. Feitelson , L. Rudolph , U. Schwiegelshohn , K. C. Sevcik , P. Wong, Proceedings of the Job Scheduling Strategies for Parallel Processing (1997) 1-34.
- [8] Gian-Paolo D. Musumeci, M. Loukides, System Performance Tuning, 2nd Edition, O'Reilly, 2002.
- [9] H. Klaus, M. Rosemann, G. G. Gable, What is ERP? Information Systems Frontiers, 2-2 (2000) 141-162.
- [10] I. Crawford, K. R. Wadleigh, Software Optimization for High-Performance Computers: Creating Faster Applications, Prentice Hall PTR, 2000.
- [11] K. E. Seamons, Y. Chen, M. Winslett, Y. Cho, S. Kuo, P. Jones, J. Jozwiak, and M. Subramanian, Fast and easy I/O for arrays in large-scale applications, Seventh IEEE Symposium on Parallel and Distributed Systems, Workshop on Modeling and Specification of I/O. (1995).
- [12] K. Henderson, The Guru's Guide to SQL Server Stored Procedures, Xml, and Html, Addison-Wesley Professional, 2001.
- [13] L. Bouganim, D. Florescu, P. Valduriez, Dynamic Load Balancing in Hierarchical Parallel Database Systems, VLDB (1996) 436-447.
- [14] L. Frank, Architecture for integration of distributed ERP systems, 104-5 (2004) 418-429.
- [15] M. Abrams, C. Phanourious, A. L. Batongbacal, S. M. Williams, J. E. Shuster, An Appliance-Independent XML User Interface Language, Computer Networks 11-16 1999 1695-1708.
- [16] N. Ruberg, G. Ruberg and M. Mattoso, Digging database statistics and costs parameters for distributed query processing, Lecture Notes in Computer Science, 2888 (2003) 301-318.
- [17] P. Gulutzan, T. Pelzer, SQL Performance Tuning, Addison-Wesley Professional, 2002.
- [18] R. M Pathak, A. Kumar, Y. P. Gupta, Reliability Oriented Allocation of files on Distributed

Systems, Proceedings of IEEE Symposium on Parallel and Distributed Systems. (1990) 886-893.

[19] S. Chan, Architecture Choices for ERP Systems, Proceedings 5th. Americas Conference on Information Systems. (1999) 210-212.

[20] S. Lifschitz, M. A. V. Salles, Autonomic Index Management, Second International Conference on Autonomic Computing. (2005) 304-305.

[21] T. Schneider, SAP Performance Optimization Guide, SAP Press, 2006.

[22] T. Schneider, Note 203924: Performance 4.6 – collective note, SAP Service Marketplace, 2006.