

POLITECNICO DI MILANO
Facoltà di Ingegneria dell'Informazione



POLO REGIONALE DI COMO

Master of Science in Computer Engineering

**Engineering of a set of Software Tools for
Server Farms Virtualization**

Supervisor: Prof. Luigi Casalegno

Tutor: Dr. Mauro Gatti, IBM Italy

Master Graduation Thesis by: Mostafa Ahmed Sharaf

Student Id. number: 721502

Academic year-2009/2010

POLITECNICO DI MILANO
Facoltà di Ingegneria dell'Informazione



POLO REGIONALE DI COMO

Corso di Laurea Specialistica in Informatica

**Ingegnerizzazione di un insieme di Strumenti
Software per la Virtualizzazione di Server Farm**

Relatore: Prof. Luigi Casalegno
Tutor: Dr. Mauro Gatti, IBM Italia

Tesi di laurea di: Mostafa Ahmed Sharaf
Matr. 721502

Anno Accademico-2009/2010

Acknowledgements

First, I would like to express my gratitude to Prof. Lugi Casalegno for his supervision on my master thesis. His suggestions and encouragement helped me very much in writing this thesis.

A special thank you goes to Dr. Mauro Gatti for making it possible to write my master thesis at IBM Italy. I enjoyed very much to work in IBM Italy, under his mentoring and supervision. Even when his time was very tight, he always had an ear for me. I am heartily grateful for the time he spent with me and for his great guidance and encouragement.

Furthermore, I want to thank the other members of WASFO team in IBM Italy: Salvatore Morsello, Giuliano Andrea Pagani, Maria Benedetta Riccelli, Elisa Tonello and Arsene Fansi Tchango, for the excellent help and support.

I owe my deepest gratitude to all the professors who taught me during my master courses in Politecnico di Milano. They always had time for my questions and whose help was a great support for achieving the master degree. I am also so grateful to Politecnico di Milano University, in particular to the head of Como Campus: Professor Roberto Negrini, and also Professor Piero Fraternali, Professor Pozzi Giuseppe, Professor Paolo Paolini.

As a friend, I would never forget my beloved friend Alina Pitu, who gave me a great support during the last year while I was in IBM Italy. I truly appreciate everything she made to me.

Finally yet importantly, I want to thank my family, particularly my mother for supporting me throughout all my years of study. In addition, I would like to thank Nevine Helmy for correcting the thesis and for her great encouragement. I finally thank God for giving me the opportunity to achieve a master degree from a great university as Politecnico di Milano University.

Table of Contents

Acknowledgements	3
Table of Contents	4
List of Figures	7
List of Tables	9
List of Graphs	10
Abstract	11
Sommario.....	12
1 Introduction	13
1.1 IT Optimization	14
1.1.1 System Virtualization.....	14
1.1.2 Server Consolidation	16
1.2 WASFO Goals	17
1.3 WASFO Users	18
2 State of Art	19
2.1 State of Art of WASFO	19
2.2 State of Art of IT Optimization tools	21
2.2.1 Data Collection Tools	21
2.2.2 Data Analysis and Business case spreadsheets.....	21
2.2.3 Data Analysis and Business case tools.....	22
3 IBM WASFO Software Development Methodology & Software Requirements.....	23
4 IBM WASFO Toolset.....	26
4.1 Overview.....	26
4.2 General Software Architecture	28
4.3 IBM WASFO Database	32
4.4 WASFO Shared Components	34
4.4.1 IBM.WASFO.Core	35
4.4.2 IBM.WASFO.ProjectDesign	37
4.4.3 IBM.WASFO.ProjectExplorer.....	41
4.4.4 IBM.WASFO.Authentication.....	45
4.4.5 IBM.WASFO.ReportGenerator	46

4.4.6	IBM.WASFO.Graphs.....	46
4.4.7	Infragistics .NET Controls.....	46
5	IBM WASFO Data Collector Tool	47
5.1	As-is WASFO Toolset for Data Collection	47
5.2	To-be WASFO Data Collector Tool	48
5.3	Software Architecture	50
5.3.1	Inventory Collection	50
5.3.2	Workload Collection	50
5.3.3	IBM.WASFO.DataCollector Component	51
5.3.4	IBM.WASFO.DCController Component.....	52
5.3.5	IBM.WASFO.DataCollectorGUI.exe Program	54
6	IBM WASFO Analysis and Optimization Tool	55
6.1	As-is WASFO Toolset for Analysis and Optimization.....	55
6.2	To-be WASFO Analysis and Optimization Tool	56
6.3	Software Architecture	58
6.3.1	IBM.WASFO.InventoryAnalysis Component	58
6.3.2	IBM.WASFO.WorkloadAnalysis Component.....	60
6.3.3	IBM.WASFO.Optimization Component	61
6.3.4	IBM.WASFO.AOController Component	61
6.3.5	IBM.WASFO.AnalysisOptimizationGUI.exe Program.....	63
6.4	Import Process of IDEAS International Spreadsheet.....	65
6.5	WASFO Matching Algorithm.....	67
6.6	Experimental Analysis for the Matching Algorithm	72
6.6.1	Customer1 project.....	72
6.6.2	Customer2 project.....	74
7	IBM WASFO Shipment Services.....	76
7.1	WASFO Shipment Web Services	76
7.2	Why WES 3.0?.....	76
7.3	IBM.WASFO.MTOM (Web Services Client).....	78
7.4	WASFO Shipment User Controls.....	80
7.4.1	IBM.WASFO.SendControl.....	80
7.4.2	IBM.WASFO.ReceiveControl	81

7.5	Web Service Configuration	83
8	IBM WASFO License Generator	84
9	IBM WASFO Deployment	86
10	Conclusion	91
11	Future Work	93
11.1	WASFO Toolset Architecture	93
11.2	WASFO Toolset Components.....	93
11.3	WASFO Matching Algorithm.....	93
11.4	WASFO Graphical user Interface (GUI).....	94
11.5	WASFO Database	94
11.6	WASFO Toolset Connectivity with WASFO Database	94
12	Appendix	95
A.	IBM WASFO Data Collector Tool.....	95
B.	IBM WASFO Analysis and Optimization Tool.....	95
C.	IBM WASFO Data Collector Setup	96
D.	Importing process of IDEAS International Excel Sheet (Reference servers)	96
E.	The process of Inventory Collection (IBM WASFO Data Collector Tool).....	97
F.	The process of Inventory Analysis (IBM WASFO Analysis and Optimization Tool).....	97
G.	Identifying the performance capacity of collected servers (Matching process)	98
H.	The InventoryAnalysis Table in WASFO Database, concerning the matching results between the servers in the tables "ServersInventory" & "ReferenceServers"	98
I.	WASFO ProjectDesign Custom Control (Project Wizard).....	99
J.	Exporting process of WASFO Projects (WASFO Data Collector Tool)	100
K.	Importing process of WASFO Projects (WASFO Analysis and Optimization Tool).....	101
L.	The design time of IBM.WASFO.ReceiveControl (WPF & XAML Coding).....	102
	Bibliography	103

List of Figures

Figure 1 Example of System Virtualization (VMware ESX version 2.0)	14
Figure 2 Example of Server Consolidation using VMware	16
Figure 3 Activity diagram of IBM WASFO Solution.....	27
Figure 4 WASFO Deployment Diagram.....	30
Figure 5 Component diagram of the libraries' dependencies in WASFO Toolset.....	31
Figure 6 Class Diagram of WASFO Core Library	36
Figure 7 Class diagram of ProjectDesign Component	38
Figure 8 Sequence diagram of ProjectDesign Component	39
Figure 9 State diagram of ProjectDesign Component	40
Figure 10 The ProjectExplorer Component (Custom control of type: Treeview)	42
Figure 11 Sequence diagram of ProjectExplorer Component.....	43
Figure 12 Class diagram of ProjectExplorer Component	44
Figure 13 The authentication Form of WASFO Toolset	45
Figure 14 Main form of WASFO Data Collector Tool.....	47
Figure 15 Component diagram of WASFO Data Collector Tool	49
Figure 16 Activity diagram of Inventory Collection Process (<i>Drawn by other WASFO Team members, IBM Italy</i>).....	50
Figure 17 Flow diagram of Workload Collection Process (<i>Drawn by Dr. Mauro Gatti, IBM Italy</i>).....	51
Figure 18 Class diagram of DataCollector Library	52
Figure 19 Class diagram of DCController Library	53
Figure 20 .NET Class diagram of the DataCollectorGUI.exe.....	54
Figure 21 Main form of WASFO Analysis and Optimization Tool.....	55
Figure 22 Component diagram of WASFO Analysis and Optimization Tool	57
Figure 23 UML Class diagram of InventoryAnalysis Library.....	59
Figure 24 .NET Class diagram of InventoryAnalysis Library.....	60
Figure 25 UML Class diagram of AOController library.....	62
Figure 26 .NET Class diagram of AOController library.....	63
Figure 27 .Net class diagram of AnalysisOptimizationGUI.exe	64
Figure 28 Diagram showing how to import IDEAS Sheet into IBM WASFO Database	65
Figure 29 Activity diagram of ETL Process for IDEAS International Sheet	66
Figure 30 Procedures of WASFO Matching Algorithm	69
Figure 31 Activity diagram of WASFO Matching Algorithm.....	70
Figure 32 Sequence diagram of WASFO Matching Algorithm	71
Figure 33 Component diagram of WASFO Shipment Web services.....	76
Figure 34 Class diagram of Shipment Web services.....	78
Figure 35 Class diagram of MTOM Component	79
Figure 36 Application Configuration file - WASFO Shipment Web Services (Client side).....	79
Figure 37 Class diagram of the component "IBM.WASFO.SendControl"	81
Figure 38 Class diagram of the component "IBM.WASFO.ReceiveControl"	82

Figure 39 Web Configuration file (XML file) of WASFO Shipment Web Services (Server side)	83
Figure 40 Authentication mechanism in WASFO Toolset (<i>Drawn by Dr. Mauro Gatti, IBM Italy</i>).....	84
Figure 41 Table LicenseKey.....	85
Figure 42 UML diagram of WASFO Solution	86
Figure 43 Welcome screen during the setup process of IBM WASFO Data Collector Tool	87
Figure 44 IBM WASFO Data Collector Tool - Application Folder.....	90
Figure 45 App.config XML file of IBM WASFO Analysis and Optimization Tool.....	90
Figure 46 Screenshot of WASFO Data Collector Tool	95
Figure 47 Screenshot of WASFO Analysis and Optimization Tool.....	95
Figure 48 Screenshot of the installing process of WASFO Data Collector Tool	96
Figure 49 Screenshot of the import process of IDEAS International Tables into WASFO Database	96
Figure 50 Screenshot of the inventory collection-user form.....	97
Figure 51 Screenshot of the inventory analysis- user form.....	97
Figure 52 Screenshot of the identified collected servers-user form.....	98
Figure 53 Relations between the Inventory Analysis Tables in WASFO Database.....	98
Figure 54 Screenshots of the ProjectDesign component	99
Figure 55 Screenshot of IBM.WASFO.SendControl component	100
Figure 56 Screenshot of the exporting process-step1 (compressing WASFO projects)	100
Figure 57 Screenshot of the exporting process-step2 (uploading WASFO projects)	100
Figure 58 Screenshot of the exporting process-step3 (Verifying files' integrity by hashing)	100
Figure 59 Screenshot of IBM.WASFO.ReceiveControl component	101
Figure 60 Automatic check in case of any found projects in the web server of WASFO.....	101
Figure 61 Manual check for any uploaded projects in the web server of WASFO.....	101
Figure 62 Screenshot of the importing process-step1 (decompressing WASFO projects).....	101
Figure 63 Screenshot of the importing process-step2 (downloading WASFO projects)	102
Figure 64 Screenshot of the importing process-step3 (Verifying files' integrity by hashing).....	102
Figure 65 Screenshot of the WPF & XAML Code of IBM.WASFO.ReceiveControl in the design-time	102

List of Tables

Table 1 Data Collection of Customer1 project	72
Table 2 Analysis results of both algorithms for Customer1 Project	73
Table 3 Data Collection of Customer2 Project	74
Table 4 Analysis results of both algorithms for Customer2 Project	75

List of Graphs

Graph 1 3D View of Customer1 project's data	72
Graph 2 Results of the old WASFO Matching Algorithm – Customer1 Project	73
Graph 3 Results of the new WASFO Matching Algorithm – Customer1 Project.....	73
Graph 4 3D View of Customer2 project's Data	74
Graph 5 Results of the old WASFO Matching Algorithm – Customer2 Project	75
Graph 6 Results of the new WASFO Matching Algorithm – Customer2 Project.....	75

Abstract

In the world of IT¹, enterprises face the challenge of how to spend a minimum cost on their IT infrastructure, while reaching the same business performances and targets. Hence, more conscious and reflective approaches are demanded.

The architectural design of the IT infrastructure requires a great effort to select the optimal design among multiple alternatives satisfying the given requirements. The problem of designing and sizing a virtualized server farm is one of the major problems in any IT infrastructures. Thus, WASFO2 has come to the reality to solve and overcome this problem.

WASFO is a joint project between IBM Italy, Politecnico di Milano and the Università degli Studi di Milano. Starting on October 2003, WASFO has been conceived to answer the need for virtualized server farm's sizing and design. My master dissertation is about the engineering of IBM WASFO Toolset. WASFO stands for *Workload Analysis for Server Farms Optimization*.

A software solution optimizes the design of server farms. It collects inventory and workload data from an existing server farm. Then based on the collected data, it finds the best virtualization design for the server farm after performing analysis and optimization through several operations and algorithms. Thus, WASFO allows the assessment, study and redesign of server farms from a single point. All these features make it unique in the arena of capacity planning tools for server virtualization.

Hence, the dissertation starts with a brief introduction to IT Optimization (Chapter.1) and after then, the state of art of WASFO and a concise overview about IT optimization tools (Chapter.2). Then detailed information is given about the software methodology used, the software requirements and specifications (Chapter.3), and after that a detailed description about the architecture of WASFO Toolset (Chapter.4). A chapter is then devoted to the description of WASFO Data Collector Tool (Chapter.5). It is followed by a description to WASFO Analysis and Optimization Tool (Chapter.6). Then, there is a chapter about the description of WASFO Shipment Services (Chapter.7), and then another chapter about a description to WASFO License Generator (Chapter.8). The last chapter is a detailed description of the deployment of WASFO Toolset (Chapter.9), and after that, a section concludes the work in the Project. The last section in the thesis is about the future work, which outlines the future possible evolutions of WASFO.

¹ IT; Information Technology

² WASFO; workload Analysis for Server Farms Optimization

Sommario

Nel mondo delle IT (Information Technology), le aziende affrontano la sfida di come limitare i costi per la loro infrastruttura IT, mantenendo inalterate le prestazioni e gli obiettivi della stessa attività. Di conseguenza, sono necessari approcci più consapevoli e ben ponderati.

La progettazione architettonica delle infrastrutture IT richiede un grande sforzo per individuare la progettazione ottimale tra le differenti alternative più corrispondenti agli obiettivi prefissati. Il problema della progettazione e del dimensionamento di una server farm virtualizzata è uno dei maggiori problemi in qualsiasi infrastruttura IT. Il progetto WASFO ha l'obiettivo di risolvere e superare questo problema.

WASFO è un progetto congiunto tra IBM Italia, Politecnico di Milano e l'Università degli Studi di Milano. L'acronimo WASFO indica *Workload Analysis for Server Farms Optimization*. A partire dall'ottobre 2003, WASFO è stato concepito per rispondere alle esigenze di dimensionamento e di progettazione di server farm virtualizzate. La presente tesi di laurea considera l'ingegnerizzazione dell'insieme di strumenti software per la virtualizzazione di server farms "WASFO Toolset".

WASFO è un soluzione software che consente di ottimizzare la progettazione e la virtualizzazione di server farm. WASFO raccoglie i dati sul carico di lavoro di una server farm esistente e, sulla base dei dati raccolti, trova il miglior progetto per la virtualizzazione della server farm esistente dopo averne eseguito una analisi ed una ottimizzazione attraverso diverse operazioni e algoritmi. In tal modo WASFO permette di valutare, studiare e riprogettare una server farm da un unico punto di vista: tali caratteristiche rendono WASFO uno strumento unico nel panorama degli strumenti di pianificazione e progettazione per la virtualizzazione di server.

1 Introduction

The WASFO toolset has been developed over the years under the leadership of Dr. Mauro Gatti (IBM Italy) and by many people. Today, many IBM partners and customers for IT Optimization are using WASFO in reality. The current version of WASFO is designed and developed using Microsoft Visual C++.Net 2008, Microsoft Visual C#.Net 2008, WPF³, XML⁴ Web Services, ADO.NET⁵, Microsoft Excel, Visio-UML⁶, IBM Rational Modeler, Microsoft Database Access.

The Toolset has currently three tools:

- I. **Data Collector Tool**; It is responsible for collecting inventory, workload, virtual machines, applications inventory and connections data from existing server farms.
- II. **Analysis and Optimization Tool**; It is responsible for analyzing, designing and optimizing the virtualization design of a server farm, based on the collected inventory data and collected workload data. Hence, it makes basic inventory and workload analysis in order to be used to solve the optimization problem by finding the minimum cost way to virtualize the current server farm.
- III. **License Generator Tool**; It is responsible for authenticating and authorizing users' access through authorized license keys.

During the master dissertation, I had a work plan that was composed of four stages:

- Create a new matching algorithm for identifying automatically the performance capacity of collected servers from an existing server farm. Herein, performance capacity means the estimate of the computational capability of the server.
- Design the architecture of the new tool; WASFO Analysis and Optimization Tool.
- Re-engineer the existing architecture of WASFO Data Collector Tool in order to modularize the code that is written previously in a non-modular way, modify the graphical user interface to look more professional and usable, and finally to add new functionalities to the current tool.
- Design and develop a shipment web service for importing and exporting WASFO Project files between the two WASFO tools; Data Collector Tool and Analysis and Optimization Tool, in order to facilitate the shipment process of those files between both tools, which was done manually in the past. The web service will use a new component for compressing /decompressing WASFO project files before/after the shipment in order to facilitate the transfer process of a large amount of files.

³ WPF; Windows Presentation Foundation

⁴ XML; Extensible Markup Language

⁵ ADO.NET; Active Data Object for Microsoft.NET

⁶ UML; Unified Modeling language

In this chapter, three main sections have been dedicated to IT and WASFO to give a broad view of the WASFO importance in IT World, whereby WASFO plays a vital role in IT Optimization;

- **IT Optimization** (Basic concepts and benefits of IT optimization, system virtualization and server consolidation)
- **WASFO Goals** (The main targets of WASFO Toolset)
- **WASFO Users** (Who are the users of WASFO Toolset?)

The following introduction about IT Optimization is mainly inspired by the website⁷ of Dr. Mauro Gatti, the project leader of WASFO.

1.1 IT Optimization

IT Optimization describes a set of techniques developed to optimize IT according to objectives established by the CIO⁸ or the IT Manager. Here are some of the IT optimization techniques:

- Techniques to improve the efficiency and effectiveness (business processes)
- Techniques to improve IT infrastructure

The optimization techniques for IT Infrastructure are as follows:

- System Virtualization
- Server Consolidation
- Operating System Consolidation
- Application Consolidation

1.1.1 System Virtualization

It allows multiple instances of the operating system to run concurrently on a host computer. The software/firmware, which provides such feature, is called Virtual Machine Monitor (Hypervisor);

- VMware ESX Server (SW⁹ for Intel servers)
- Microsoft Hyper-V (SW for Intel servers)
- PowerVM (firmware for IBM System p servers)

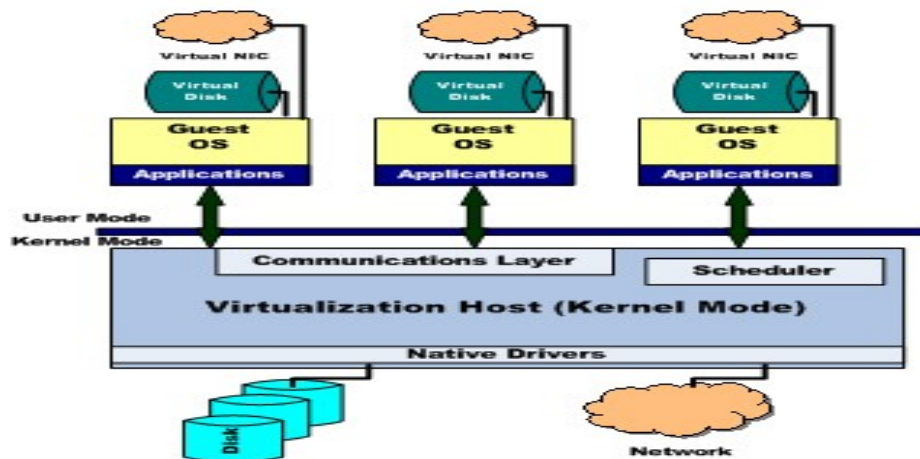


Figure 1 Example of System Virtualization (VMware ESX version 2.0)

⁷ <http://www.itdec.eu/>, an open web site dedicated to the application of formal methods to the IT *investment* and *design decision* processes. This web site is *open* in the sense that whoever can contribute by sending articles to us.

⁸ CIO; Chief Information Officer

⁹ SW; Software

A system virtualization project is always considered a server consolidation project, but it is not restricted to be an operating system or application consolidation project.

1.1.1.1 *Financial benefits*

1.1.1.1.1 Minimization of hardware maintenance costs

Costs of HW¹⁰ maintenance are significantly dependent on HW virtualization. It is possible to see 30-1 consolidation ratios in a system virtualization project, which results in about 30-1 maintenance savings.

“Detailed cost analysis in real projects using WASFO have shown that the savings of operating expenses concerning old servers maintenance may be enough to offset the initial capital expenditure of the overall project (servers, software and implementation) provided that tax shield is taken into account”, said by (Dr. Mauro 2010).

1.1.1.1.2 Minimization of LAN/SAN costs

A system virtualization project typically provides LAN and SAN savings. In addition, hypervisor-s can create virtual networks thereby further reducing the LAN costs. Thus, a system virtualization project can free many LAN/SAN resources (for example: network switch ports).

1.1.1.1.3 Minimization of floor spaces costs

System virtualization projects usually produce great space savings due to the reduced number of physical servers. “In a recent project we have estimated that 22 square meters could be freed with a system virtualization project with a 30-1 consolidation ratio”, said by (Dr. Mauro 2010)

1.1.1.1.4 Minimization of power consumption costs

The reduction of the servers’ number minimizes the cost of the power consumption:

- Power consumptions by servers
- Power consumption by datacenters refrigeration

1.1.1.2 *Intangible benefits*

1.1.1.2.1 Increased availability

Live migration of Virtual Machines makes it possible to increase the overall dependability of a server farm. Studies have shown that the overall availability of a server farm of Windows servers without any high availability protection is on average 99.9%.

“We can say that live migration can help to further reduce the planned and unplanned system downtime by dramatically reducing, let say for sake of simplicity by zeroing, the downtime due to HW failures. We know from statistical analysis that HW failures are 15-30% of the overall failures. So we should expect the overall availability to increase on average to 99.915-99.930%”, said by (Dr. Mauro 2010).

¹⁰ HW; Hardware

1.1.1.2.2 Green IT

New servers typically consume more than old servers do even though the technologies of modern power savings. However, the very high consolidation ratio is enough to replace such increases and deliver efficient power savings.

“An in-depth analysis on real data collected with WASFO tool has shown that a server farm with 200 servers can produce through system virtualization (30-1 consolidation ration) over 150,000 Euros power savings per year (with a cost of 0.18 Euros per KW¹¹/hour)”, said by (Dr. Mauro 2010).

1.1.2 Server Consolidation

It aims at reducing the number of servers. This can be achieved through the implementation of system virtualization, operating system consolidation or application consolidation.

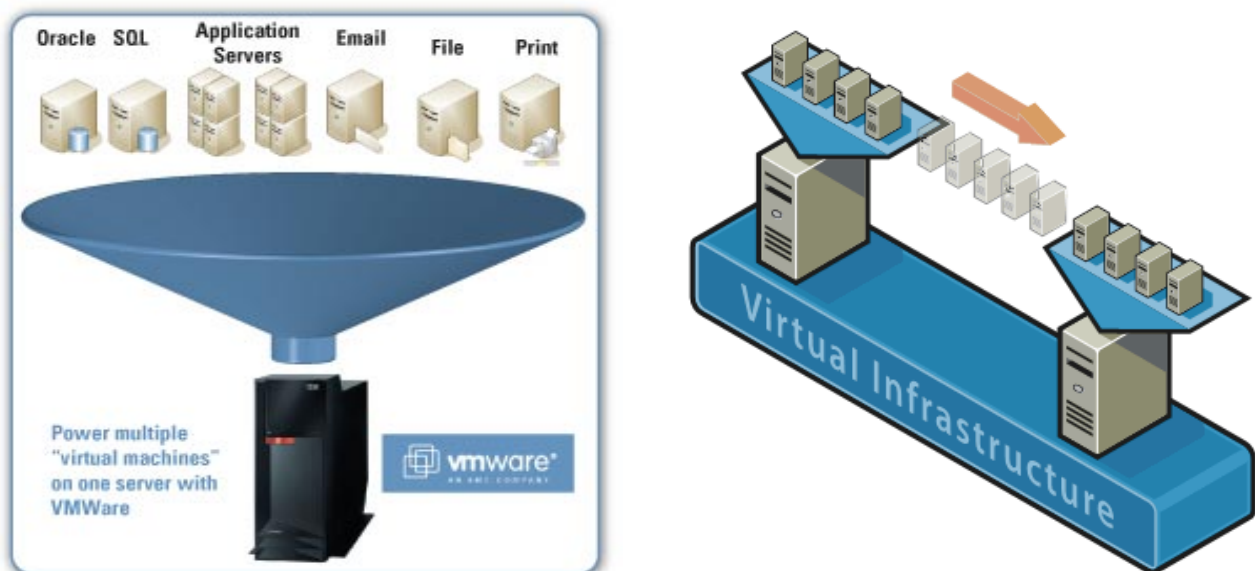


Figure 2 Example of Server Consolidation using VMware

1.1.2.1 Financial benefits

1.1.2.1.1 Minimization of hardware maintenance costs

The hardware maintenance costs are calculated per server and hence the fewer the servers, the less is the overall cost. These costs are not considered within the first 3 years after server purchase. However, they become explicit as soon as the warranties expire and the server is consumed. Maintenance costs may increase significantly, when the server exceed the *critical age*.

1.1.2.1.2 Minimization of network connectivity costs

The fewer the servers we use, the less number of ports and cables used for the LAN and SAN connections. In fact, the new servers typically use higher number of connections than those of

¹¹ KW; Kilo Watt

old servers. However, the consolidation ratio is usually so high enough to produce a significant reduction in network connectivity.

1.1.2.1.3 Minimization of physical spaces costs

When the number of physical servers decreases, the used datacenter space is reduced. This reflects a benefit on the utilization level of the datacenter; hence, in this case it can be fully utilized. However, this benefit depends primarily on the datacenter's utilization level.

1.1.2.1.4 Minimization of power consumption costs

As fewer servers we consume, as less power consumption we cost. Power consumptions are directly due to the servers, and due to the datacenter refrigeration.

1.1.2.2 *Intangible benefits*

1.1.2.2.1 Increased availability

Server consolidation usually reduces the overall number of failures in server farms, and henceforth minimizes the costs due to the management of these failures.

1.1.2.2.2 Green IT

Server consolidation clearly supports the principle of Green IT, because it reduces the number of used servers.

1.2 WASFO Goals

The main goal of WASFO is to minimize the total cost of acquisition, the management cost, and the energy consumption of server farms. Henceforth, WASFO is totally associated with an approach to solve IT optimization problems. The following are the sequence steps of how to use WASFO Toolset:

- Project's design and creation: the server farm to-be analyzed, consists of several systems which are divided into groups belong to:
 - Different areas
 - Different operating system technologies
 - Different organizations inside the company
- Inventory collection: collects inventory data from the existing server farm.
- Workload collection: collects workload data from the existing server farm.
- Inventory analysis: performs statistical analysis of the collected inventory data.
- Workload analysis: performs statistical analysis of the collected workload data.
- Optimal virtualization: finds the best virtualization design for the current server farm.
- Presentation: the best solution is created to be presented and discussed with the client.

Thus, with WASFO Toolset you avoid the typical "rules of thumb" that is frequently used in virtualization projects without a reasonable motivation. Indeed, WASFO Toolset answers typical questions from clients. These questions are usually investigated during a server consolidation project;

- Which servers should we virtualize? Moreover, which ones should not be virtualized?
- Which server models do we use?

- How many servers do we need?
- Which configurations should our virtualized servers have?
- How do we distribute the VMs¹² on the virtualized servers? Moreover, how should the servers be utilized after virtualization?

“WASFO software finds the best virtual machines allocation on both the new and legacy target servers; it also provides a projection, based on collected data, on how the servers will be utilized after the virtualization-consolidation project. In conclusion, the software tool can be an analytical guide to drive the customer to the best possible solution”, said by (Andrea Pagani 2009).

1.3 WASFO Users

WASFO can be used only by IBM employees. Typical users of WASFO Toolset are:

- IT Specialists
- IT Architects

IBM Business partners can use *WASFO for IBM BPs*¹³ (*the legal screening is being completed currently*). WASFO for IBM BPs only allows to collect data, hence IBM WASFO Data Collector Tool is used for this purpose. Collected data will have to be analyzed by an IBM employee. This restriction is due to the fact that licensing of IDEAS International Performance capacity tables is required.

WASFO is mainly to design an optimal server farm in virtualization projects. Thereby, WASFO guides sales, pre-sales, specialists and architects personnel from;

- IBM Systems Specialists/Architects
- IBM GTS¹⁴ Specialists/Architects (service delivery)
- IBM System Architects STG¹⁵
- IBM BPs (data collection)
- Techline (elaboration of data collected by IBM BPs)
- ITD¹⁶ Specialists/Architects

WASFO Toolset could also be used in strategic outsourcing projects in order to monitor physical server farms to-be-virtualized to IBM facility or to virtualize clients' servers that are already outsourced to IBM.

¹² VMs; Virtual Machines

¹³ IBM BPs; IBM Business Partners

¹⁴ GTS; IBM Global Technology Services

¹⁵ STG; IBM Systems and Technology Group

¹⁶ ITD; IBM Integrated Technology Delivery

2 State of Art

2.1 State of Art of WASFO

In this section, the previous WASFO Tool is investigated with comparison to the new WASFO Toolset. WASFO started in 2003, and its first release was issued in 2007. Since that time, WASFO has become a useful tool for the IBM x86 technical sales force and the IBM business partner network. In fact, WASFO has succeeded in 10s of virtualization projects in Italy and abroad as well. Although the old WASFO Tool was successful as an IT Optimization Tool, it does not have good software architecture, besides that it lacks many features. Consequently, it did face many technical problems. Some of these problems are as follows:

- Lack of Modularity
- Manual shipment of WASFO Projects
- Missing setup package for WASFO tool
- Missing reporting facility for WASFO projects
- The graphical user interface isn't professional and has many defects
- Generating simple graphs which sometimes does not show important details
- Using XML files to WASFO Data instead of having a database management system
- The optimization algorithms does not take into account important scenarios (e.g. currently virtualized server farms)
- Inefficient and non successful matching algorithm for the automatic identification process of the collected servers' performance capacity
- WASFO Tool is a single tool, since it performs data collection, analysis and optimization. This monolithic architecture was poor in terms of design and performance

After the second release in 2008, WASFO went through an intensive revision and improvement phase, both from the user and the technical perspectives. Currently the new WASFO is a toolset, which comprises three WASFO Tools:

1. **IBM WASFO Data Collector Tool**
2. **IBM WASFO Analysis and Optimization Tool**
3. **IBM WASFO License Generator Tool**

The improvements regarding the user's side are as follows:

- More user friendly interface
- New data collection from virtualized platforms and from different OS¹⁷ platforms
- Possibility of the choice between more IBM target platform families
- Addition capabilities to solve different decision issues
- Design of more detailed and comprehensive cost functions
- Financial analysis capabilities
- Generation of advanced graphs to help users in taking decisions

The improvements regarding the technical side are as follows:

- Entire code rewriting
- Software component's reusability
- Distributed architecture based on middleware paradigm using IIS¹⁸ Web Server

¹⁷ OS; Operating System

- Introducing SOA¹⁹ by building set of web services in the middleware layer
- Design and Implementation of WASFO Database
- User friendly and professional GUI²⁰
- Introducing professional GUI libraries to WASFO
- New matching algorithm for the automatic identification of the collected servers' performance capacity
- Import automatically the reference servers from IDEAS International sheet into WASFO Database
- Elaboration of the exact and heuristic solvers for the new mathematical models
- Design and development of advanced graphs
- Partial automation of data updates for server configurations
- Add more IBM systems as possible target platforms for the consolidations

In fact, my work in WASFO Project was part of this transformation process during the last year in 2009. I was responsible for designing and implementing the completely new architecture of WASFO Toolset. Hence, I created the new tool; WASFO Analysis and Optimization, and modified many parts in the old WASFO Data Collector Tool. In addition to contributing to the design and the implementation of the graphical user interface of WASFO Toolset, the development and the integration of the *Core*, *ProjectDesign*, *ProjectExplorer*, *DataCollector*, *DataCollectorGUI*, *InventoryAnalysis*, *AOController*, *AnalysisOptimizationGUI*, *SendControl*, and *ReceiveControl* components. Moreover, I developed the deployment packages for WASFO Tools.

Indeed the current WASFO is very different from the previous version. Currently, WASFO Toolset has two main tools, and middleware software:

- WASFO Data Collector Tool
- WASFO Analysis and Optimization Tool
- IBM WASFO Middleware Software

One of the most interesting development issues during my work was the shipment of WASFO projects' files. Now, WASFO has introduced a middleware layer to-be called IBM WASFO Middleware; IIS Web server that provides shipment web services to transfer remotely any project files between both tools. The biggest challenge was how to optimize the transmission of files of large size with zero error.

¹⁸ IIS; Internet Information Services, formerly called **Internet Information Server** that is created by Microsoft

¹⁹ SOA; Service Oriented Architecture

²⁰ GUI; Graphical User Interface

2.2 State of Art of IT Optimization tools

In this section, the state of art of IT Optimization tools is investigated. Currently, IT Optimization tools are classified into:

1. Data collectors
2. Data analysis and business case spreadsheets
3. Data analysis and business case tools

WASFO Team has performed a deep investigation for all these tools, and here below is a brief overview about each tool.

2.2.1 Data Collection Tools

2.2.1.1 *CDAT*

It stands for Consolidation Discovery and Analysis Tool. CDAT is a data collection tool that is used by IBM server consolidation specialists. It performs server discovery and data gathering of Windows, Linux, Solaris, HP-UX, AIX²¹ and Netware servers.

2.2.1.2 *IBM ATS SCOM Monitor*

The IBM ATS Server Consolidation Monitor (ASCM) is used to guide Server Consolidation studies. It is a Microsoft Windows browser application which runs on Windows .NET Framework. It has the ability to collect and report system architecture. Hence, it gathers system architecture and performance data from Windows computers and UNIX/Linux hosts.

2.2.2 Data Analysis and Business case spreadsheets

2.2.2.1 *Visian*

An IBM internal tool developed to help sizing Intel and UNIX target servers in consolidation study projects. VISIAN is an Excel-based IBM internal tool, which performs analysis based on the collected inventory and workload data. Nevertheless, it has no data collection capabilities.

2.2.2.2 *Zodiac*

It is an IBM Excel-based internal tool used to project the costs of ownership and growth over time, compare the base case, and alternate case in IT Optimization projects. It can be used in the following cases:

- Examine costs
- Investigate potential for automation
- Investigate savings in software costs
- Investigate potential reductions in carbon emissions

2.2.2.3 *Cobra*

It is an eventual derivative of Zodiac. Cobra is an IBM Excel-based internal tool. It is possible to switch to Cobra mode by accessing the Zodiac Spreadsheet. With Cobra, we can only gather summary data about the number of servers in each functional group.

²¹ AIX; Advanced Interactive eXecutive

2.2.2.4 *Race*

It is an IBM Pre-Sales technical support tool, which allows the development of IT Cost and value assessment for IBM Clients. Hence, it compares alternative infrastructure choices. Race stands for Right-Fitting Applications into Consolidated Environments. RACE provides re-hosting Applications into consolidated environments. It was designed to build business cases that supports clients' desire to move from a de-centralized computing model composed of discrete servers to a virtualized model running on fewer centralized servers. RACE was used mainly for sizing the mainframe as a replacement platform for Linux applications. Currently, it is being used to evaluate potential benefits of consolidating workloads from Competitive Intel/UNIX platforms into IBM Systems platforms.

2.2.3 Data Analysis and Business case tools

2.2.3.1 *Cirba*

Cirba has been selected as the standard software application for both for data collection and analysis;

- Global Technology Services-Server
- Service Product Line-Optimization and Integration Services

It is considered as the most complete and advanced tool among the previous ones. It is Java web-based software for datacenter intelligence.

3 IBM WASFO Software Development Methodology & Software Requirements

To design and develop the new WASFO Toolset, we pursued the iterative and incremental approach. This development methodology reduced project risk by breaking WASFO into smaller segments and providing easier change during the development process.

After we performed an intensive analysis to the previous design of WASFO, we decided to design the new WASFO Solution based on the principle of Component-based software engineering. This principle “*CBSE*²²” emphasizes the separation of complex functionalities available throughout a given software system into reusable components. This kind of architecture is similar to the SOA, whereby a system is composed of reusable services. In our case, WASFO is composed out of reusable components. Indeed, component-based software engineering is considered as a part of the starting platform for service orientation, where components can be converted into services. Thus, WASFO could support SOA in the future if needed.

In fact, WASFO Toolset has passed through several software requirements in order to optimize the architectural design, functionalities, graphical user interface and performance. These software requirements have been categorized into three types of requirements;

- **Design requirements:**

- *Optimization Design*

WASFO architecture design should be optimized to make best use of the available resources. Having a good choice of efficient algorithms and the implementation of these algorithms will benefit from writing good quality code. The architectural design of WASFO overwhelmingly affects the performance. The choice of WASFO optimization algorithms affects the efficiency more than any other item of the design.

- *New architecture (Distributed Architecture)*

Enable both WASFO Tools (Data Collector Tool & Analysis and Optimization Tool) to communicate remotely together in order to send/receive WASFO’s Project files “WASFO Data”. Hence, this will require middleware software based on a web server to handle this issue.

- *Modularity*

The software architecture is divided into components called modules. The related modules are grouped together in packages. In addition, appropriate design patterns and standardized code are applied.

²² CBSE; Component-based software engineering

- *Code optimization*

Avoid poor quality and redundant coding, modify WASFO to make it possible to work more efficiently and use fewer resources by means of high quality and well commented (.Net and XML comments) code.
- *Loosely coupling between presentation and business components*

Clear separation between presentation and business components; thereby avoid writing business code in the graphical components of WASFO.
- *Database design*

The old version of WASFO Tool did not have a database management system to store the projects' data of WASFO, and therefore the tool faced many technical problems in data storage, and performance as well. Thus, the new WASFO architecture requires the storage of projects' data in a relational database management system to satisfy the functional and non-functional requirements introduced by WASFO Toolset.
- *Software components' reuse*

Break down complexity and create components "reusable pieces" that can be shared and used by both tools: WASFO Data Collector and WASFO Analysis and Optimization Tool.
- *Extensibility*

New capabilities could be added to WASFO without major changes to the underlying architecture.
- **Functional requirements:**
 - *High quality user interface*

Provide flexible and professional interface in order to have a user friendly and easy to use environment.
 - *Import IDEAS Spreadsheet into WASFO Analysis and Optimization Tool*

Import automatically reference servers from the IDEAS spreadsheet into WASFO Database through WASFO Analysis and Optimization tool.
 - *New matching algorithm for performance capacity's identification*

Due to the fact that the old algorithm does not work properly and inefficient at all; we need to develop a new algorithm that work properly and guarantees better and efficient results.
 - *New methods for data collection*

Introduce VMware's inventory and workload data collection, and Linux I/O data collection as well.
 - *Improved and new optimization algorithms*

Support capacity planning of partially virtualized server farms, and provide stochastic noise modeling. Introduce new mathematical models.

- *Parallel execution of computations*
Use multi-threading techniques and concurrency control.
- **Non-Functional requirements:**
 - *High performance*
Reduce computation time of WASFO optimization and matching algorithms, plus a greater accuracy.
 - *Legal and licensing issues*
WASFO users must have a valid license and a registered account in order to use WASFO Toolset.
 - *Deployment*
Provide easy and flexible installation process to customers.
 - *Memory management*
Use the language-specific garbage collector, and efficient data types.
 - *Robustness*
Have the ability to cope with errors during execution or the ability of WASFO algorithms and data collection methods to continue to operate despite any abnormalities in the input, calculations, servers...etc.
 - *Efficiency*
Balance the resources' consumption corresponding to WASFO Algorithms. WASFO should executes more rapidly, and operate with less memory storage or other resources.
 - *Technical Documentation & User Documentation*
Produce technical documentation that is necessary to facilitate future development and maintenance by using .NET and XML comments inside the code. In addition, produce user documentation to guide users with a professional description about all WASFO steps to optimize the design of a virtualized server farm.

4 IBM WASFO Toolset

4.1 Overview

The WASFO toolset is a set of tools designed and developed to optimize the virtualization design of server farms through data collection, analysis and optimization. WASFO Solution currently consists of IBM WASFO Middleware-software (based on IIS Web Server for hosting IBM WASFO Shipment Web Services), and two main tools:

- IBM WASFO Data Collector Tool
- IBM WASFO Analysis and Optimization Tool

The scenario begins when the customer installs WASFO Data Collector Tool, and then starts to create a new project, which usually contains group(s) of servers, and subgroup(s) of servers according to the design of the existing server farm in the customer site. These groups and subgroups represent the different departments, organizations, policies or operating systems...etc within the customer's company.

After then, WASFO Data Collector Tool collects firstly the inventory data, and secondly the workload data of the existing server farm. When the tool completes the collection process of inventory and workload data, the customer will be able to upload the entire project's data (collected data of the existing server farm) into IBM WASFO Middleware by a specific web service dedicated for exporting WASFO Projects into the server.

After then, specialists or architects in IBM or IBM Partner use WASFO Analysis and Optimization Tool to download the customer's data from IBM WASFO Middleware through a web service dedicated for importing WASFO Projects into the tool. Once the project's data is loaded, the tool will be ready to navigate through the project's structure and performs statistical inventory analysis on the collected servers, and then statistical workload analysis on the collected servers.

After the analysis phase is completed, specialists or architects will be able to find the optimal virtualization design of the customer's server farm. At the end, a presentation is prepared by these specialists or architects in order to discuss the proposed solution with the customer. Indeed, WASFO finds the best allocation of virtual machines on the new servers and the legacy servers as well. It provides also a forecast on how the servers will be utilized after the virtualization-consolidation solution, based on collected data of the sever farm.

As we can see, the main target of WASFO Toolset is to find the best optimization design of how to virtualize a server farm in order to minimize the total cost of acquisition, the management cost, and the energy consumption of the targeted server farm. Using such tool can help significantly in optimizing the design of the IT Infrastructures in companies.

Here below is the activity diagram that shows the flow of the main activities inside WASFO Tools and the shipment's activities between WASFO Toolset and IBM WASFO Middleware:

Server Farms Optimization with WASFO

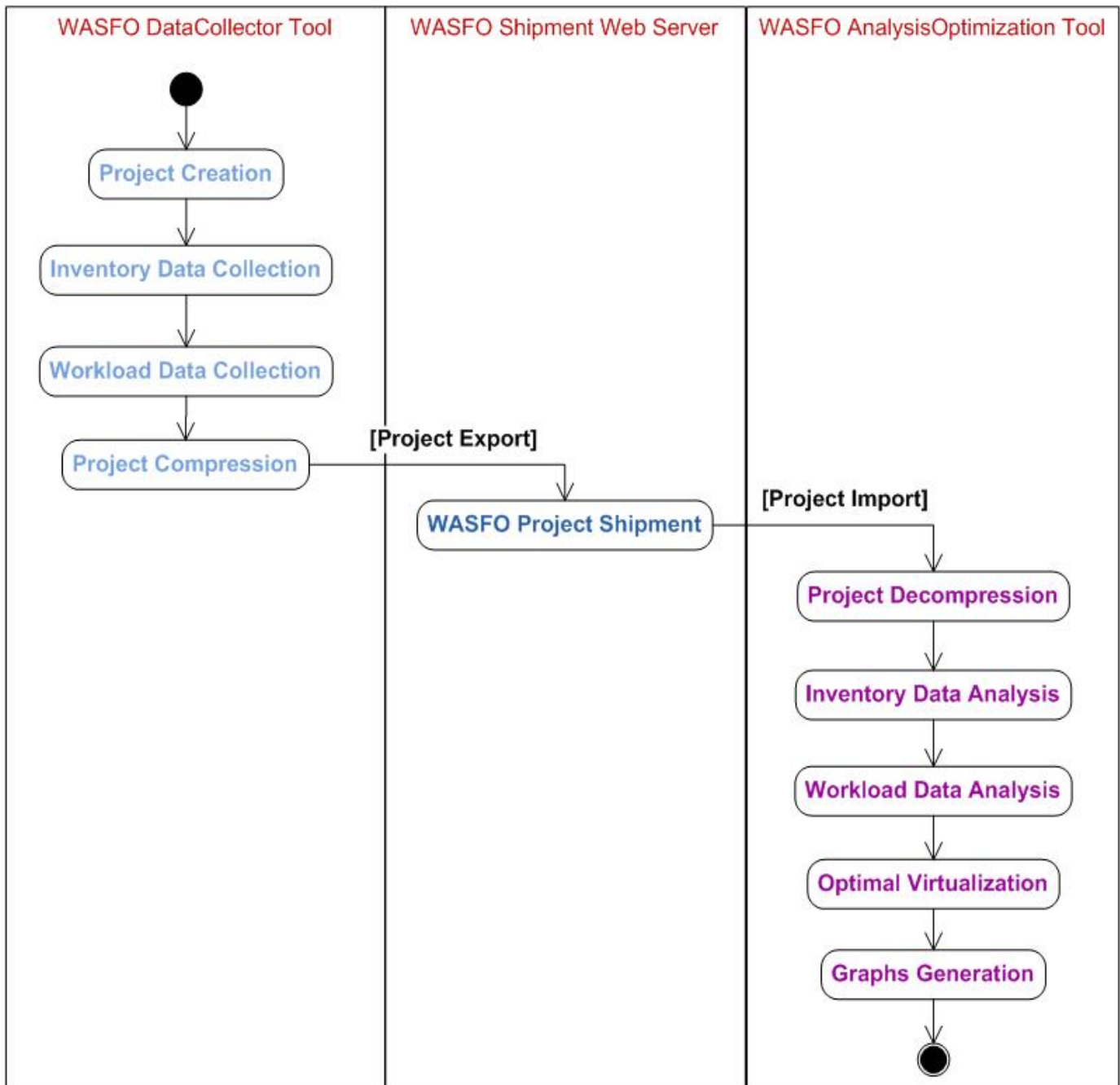


Figure 3 Activity diagram of IBM WASFO Solution

4.2 General Software Architecture

At the beginning WASFO was a single tool, and hence It was designed as a single tier architecture. This design suffered from many technical problems in terms of inefficiency, poor performance, non-modularity and redundancy of business functionalities. Moreover, having the presentation, the application processing, and the data management altogether in one code without a clear logical separation was very inappropriate and an obstacle in modifying the design of WASFO or even in updating it.

Hence, we had to analyze the whole situation and think carefully whether we should re-engineer the current architecture or engineer a new one in order to find the optimal design for WASFO, with respect to the new software requirements and specifications, plus the user requirements. Moreover, we had to put in our account that the old version of WASFO carried on the data collection, data analysis and data optimization altogether in one tool. However, now we need to split the data collection functionalities from the data analysis and optimization ones in order to optimize WASFO's Design.

As a result, WASFO currently is a toolset that has two main tools; WASFO Data Collector Tool, and WASFO Analysis and Optimization Tool. In fact, one of the most interesting issues during the design phase of the new architecture was that both tools are standalone applications which share set of business functionalities particularly in the core level.

Hence, I had to develop software components to build those common functionalities in the form of class libraries and usercontrol libraries. Each component packages a group of classes and functionalities which have a logical relation based on OOP Paradigm and also based on our structure's design. Indeed, we classified these shared components as the following:

1. Class libraries contain the common business functionalities;
2. Usercontrol libraries contain the common presentation functionalities.

Thus, I made advantage of following the principle of components' reuse by building these common libraries to be shared by both tools in order to optimize the architecture design of WASFO Toolset. Besides that, I followed this principle in designing all the functionalities of WASFO Toolset in the form of libraries in order to break the whole complexity of WASFO functionalities into small functions, and so I could manage to group all the related functions in thier corresponding library. Thus, the concept of building all the functionalities in libraries has introduced intuitively a logical separation of the 3 software layers in WASFO's design;

1. Presentation layer: represented by the usercontrol libraries and all the graphical components
2. Business layer: represented by the class libraries and the business components
3. Data layer: represented by WASFO Database, and the data interface classes inside the IBM.WASFO.Core Library

As we see, this logical separation could be very helpful in updating or modifying the architecture design of WASFO Toolset in the future when needed. For example, it is currently possible to build WASFO in 2 or 3 tier architecture model in case if WASFO will be used remotely by clients in the future or something like that. So we could separate physically the three software layers of WASFO to achieve a three tier model; presentation layer (client side), logical layer (server side), and data layer (database server).

After then, WASFO Toolset has evolved in terms of functional requirements, since we needed to build a communication link between both tools for the sake of importing and exporting the projects' files which are created by WASFO Tools. Hence, it came to my mind the development of a Middleware Software which will be responsible for communicating WASFO Tools together. Indeed, middleware is especially integral to modern information technology based on XML, SOAP²³, Web services, and SOA. Thus, WASFO Middleware consists of two loosely coupled web services, which connect both tools together to be able to send and receive projects' files to each other in the form of SOAP messages. Currently, the middleware acts as a messaging software, however the idea of building web services has introduced the concept of Service-Oriented Architecture.

Indeed, SOA has been used in WASFO in the form of exposing web service for files' export and web service for files' import, accessible over the network in order to allow WASFO Tools to combine and reuse them in their applications. These web services and their corresponding consumers communicate with each other by passing data in a well-defined, shared format.

Here below is the deployment diagram of WASFO Toolset, which shows the different physical nodes of WASFO Solution:

1. IBM WASFO Data Collector Tool
2. IBM WASFO Analysis and Optimization Tool
3. IBM WASFO Middleware – Software for Shipment Web services

In addition, the diagram shows the most important shared libraries between both WASFO tools (it will be discussed later in the section of WASFO Shared Libraries). You can see also the communication protocol between the three physical entities; the request-response protocol (HTTP²⁴), the exchanging protocol (SOAP), and the format of the exchanged messages (XML).

From figures 4 and 5 you can observe that there is a common rule for naming our WASFO components. This rule is a standard for naming any component in WASFO Toolset; **“CompanyName.ProgramName.ComponentName”**. Thus, in our case the component will be named in this way **“IBM.WASFO.ComponentName”**. In this way, we guarantee to overcome any internal or external naming conflict that might happen with the components. In figure 5,

²³ SOAP; Simple Object Access Protocol

²⁴ HTTP; HyperText Transfer Protocol

a component diagram shows in details the relations between all WASFO components (shared and unshared) for both WASFO Tools and their dependencies upon each other.

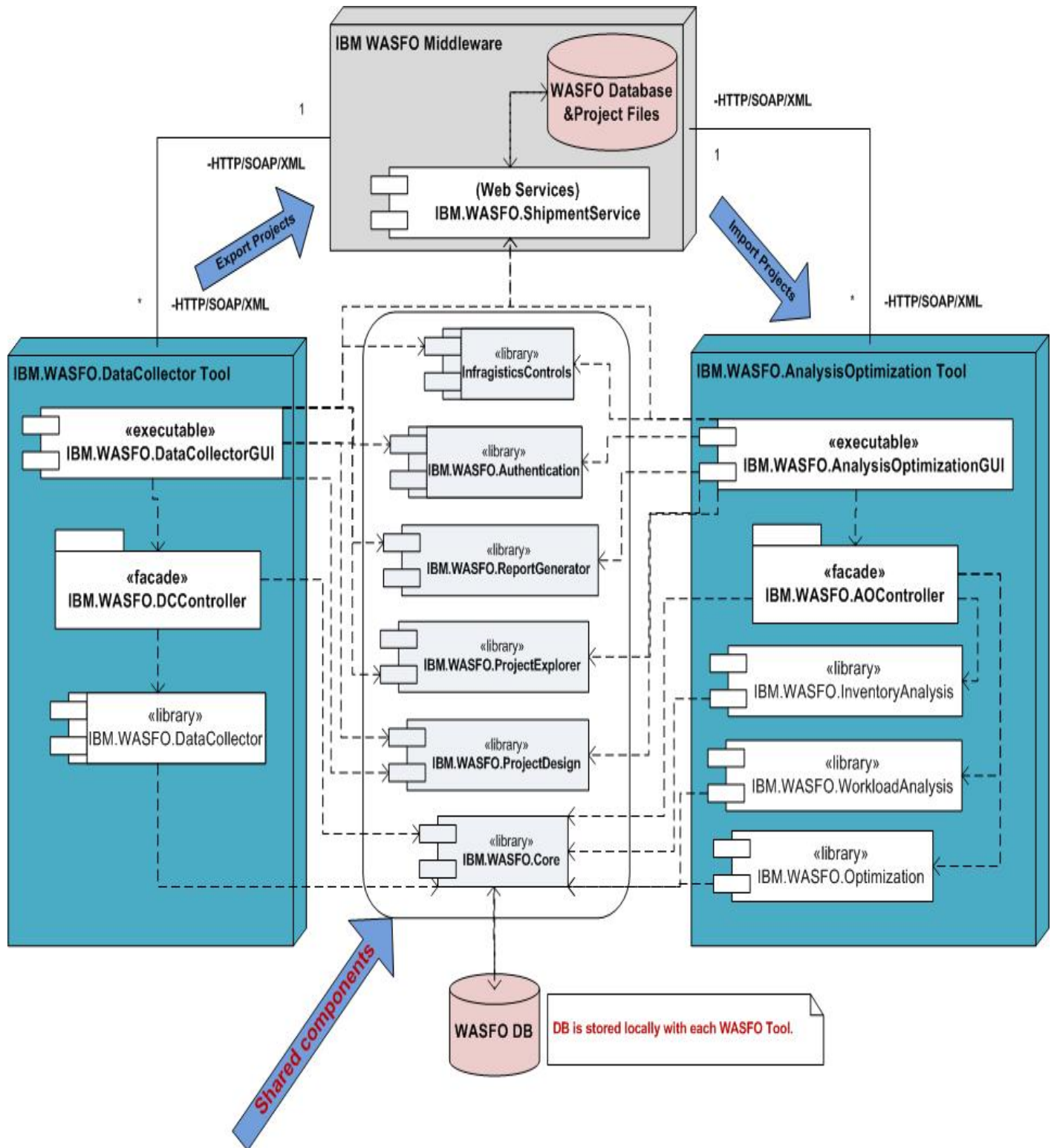


Figure 4 WASFO Deployment Diagram

4.3 IBM WASFO Database

In the past, the old version of WASFO did not use a database management system to store its data, while XML files were chosen to store WASFO Projects' data, thinking this could be enough. However, after involving WASFO in real projects, it was obvious that the choice was a short-term decision and as a consequence, many technical problems have arisen due to the over simplicity in the data storage's technique. Hence it became very clear that using simple files to store WASFO's data was not wise and inappropriate design in our case.

Although I was not involved in the design and the development of WASFO Database, I took part in taking critical decisions about the quality of the database design due to my main role as an architect of the new architecture of WASFO Toolset. Thus, we arrived to a decision of using a database engine to overcome the previous problems of storing WASFO's data. The design and the implementation of WASFO Database started last year, and now the current database system is implemented in Microsoft Access. It is being used by both WASFO tools; WASFO Data Collector Tool and WASFO Analysis and Optimization Tool.

The choice of Microsoft Access for storing WASFO's data is due to the following reasons:

1. The simple design of the database at the beginning of the development of WASFO;
2. Microsoft Access is considered as a light-weight relational database management system, hence it is a good portable database;
3. Microsoft Access is a Microsoft product and it is usually installed within the package of Microsoft Office, hence it is supported and installed by all windows platforms;
4. Because of the 2nd and 3rd reasons, each WASFO Tool could have a local copy of WASFO Database. Thus the connection with the database is done locally, which should be fast.

In fact, one of the major issues in the design of WASFO Database is that WASFO Data Collector shares all its data tables in the database with WASFO Analysis and Optimization Tool, not vice versa. Hence, WASFO Analysis and Optimization Tool has its own data (tables) for the analysis and optimization processes. Based on this fact, we had to think about the best way to design the database, and whether we should keep the current database to be shared by both tools, or build a separate database for each tool.

Indeed, having a single database shared between both tools is an inefficient design, since the data tables of WASFO Analysis and Optimization Tool is exposed to WASFO Data Collector Tool, and consequently we are sharing undesired data with WASFO Data Collector Tool. However, this is the current solution of WASFO Database due to its simplicity.

On the other hand, if we design two separate databases (database for WASFO Data Collector Tool and another one for WASFO Optimization and Analysis Tool), the database of the Analysis and Optimization Tool will need to import the database of the Data Collector Tool in order to

make the analysis and optimization based on the collected data. Indeed, this could be an optimal solution to design the databases, since we will keep both databases separate avoiding the problem of having undesired data tables between WASFO Tools, and hence we will need only to import WASFO Data Collector's tables into the database of the other Tool. The import process can be done through the new IBM WASFO Middleware, which could merge the two databases together to be ready for usage by WASFO Analysis and Optimization Tool. Thus, each database will work locally with its tool, and this would be efficient in terms of database connection.

Currently, the installation process of WASFO Data Collector, accompanies the installation of WASFO Database to be used for storing the collected data from server farms. However, this does not happen with WASFO Analysis and Optimization Tool, because it depends on the collected data from WASFO Data Collector Tool in order to be able to proceed with the analysis and optimization processes. Thus, WASFO Data Collector Tool calls a specific web service to upload (export) all the projects' data of the tool including the database into IBM WASFO Middleware (installed in IBM Site). After that, WASFO Analysis and Optimization Tool will be able to download (import) all the available projects' data including the recent database from IBM WASFO Middleware through a specific web service.

In fact, the choice of Microsoft Access is inefficient due to the following points:

1. The need for more efficient and advanced database, since Microsoft Access does not have some advanced features such as advanced design tools, stored procedures, triggers and advanced constraints...etc;
2. We must buy the private license of Microsoft Access (even though it should not be a problem since we already have a Microsoft Office License);
3. Why don't we use IBM Database-DB2 "free of charge"? Hence, we can benefit from having a database server that centralizes the data layer of WASFO Toolset.

Indeed, DB2 would be a good choice to build WASFO Database. since it is an IBM relational model database server and so it is reasonable to use it for IBM WASFO Toolset without the need to buy a license for the database. This would certainly help in minimizing the development cost of WASFO. In addition, using DB2 for WASFO will introduce the idea of having a database server in WASFO Toolset Architecture, and this could bring us to have a single database server (for example: IBM WASFO Database Server) to-be managed and used by IBM WASFO Middleware. In this way, we could avoid having a local copy of the database with WASFO Data Collector or WASFO Analysis and Optimization tools, relying on the advantage of the remote connection with the database server.

Despite the fact that the database performance will be much better using DB2 than with Microsoft Access, however a question mark remains; WILL this new design affect the performance of the database connection with WASFO Toolset? This question arises from the fact that the central database will be located in the server side, not in the client side and hence all the connections will be done remotely. One solution is that each WASFO tool can work with a

local copy of the database in the machine where it's running, then when the transactions are completed, the tool connects to the central server and updates the database server with the new updated data.

As we can see, the whole database design is going to be modified in the near future. Thus, we need to think carefully about the optimal design of WASFO Database, and foresee how it is going to be used by WASFO Toolset in order to guarantee better and more efficient performance in the long term.

4.4 WASFO Shared Components

They are the common components between the two main tools of WASFO Toolset (WASFO Data Collector Tool, WASFO Optimization and Analysis Tool). The concept of having shared components between both tools was one of the most interesting ideas in my architecture design. During the design phase of the new architecture of WASFO Toolset, I observed that there are many classes and functionalities which are shared and used extensively by the data collection part, the analysis and the optimization part as well. Thus, I found out that applying the software principle of components' reuse would be the best solution to solve the major problem of the code redundancy and non-modularity. In this way, we could provide an optimized architecture, efficient object oriented design, non-redundant and modular code.

In fact, these components have been designed and implemented before starting the development phase of WASFO Data Collector Tool, and WASFO Analysis and Optimization Tool. This was due to the fact that there is an extensive usage and complex dependencies on the functionalities of these components by both WASFO tools.

The previous release of WASFO was totally developed by using managed C++/CLI²⁵. However, this was a non-professional choice since C++/CLI is a new programming language introduced in .Net platform, and unfortunately it was not fully supported by all the .Net facilities like for example C#.Net programming language (the .Net primary language). Thus, we decided to use C#.Net as possible to build these shared components.

The choice of the programming language was strongly depending on whether we are going to build new functionalities from scratch in the new library or we are going to move many existing functionalities from the old version of WASFO into the new library. Actually the oldest components are developed in C++/CLI and the newest ones in C#.NET.

In fact, some of the functionalities have been taken from the old version of WASFO Tool. However, these existing functionalities have been divided and moved into the new libraries after being modified and optimized according to WASFO requirements and specifications.

²⁵ C++/CLI; C++/Common Language Infrastructure, is a Microsoft's language specification intended to replace Managed Extensions for C++.

The shared components which, have been implemented for WASFO Toolset are as follows:

1. IBM.WASFO.Core (C++/CLI)
2. IBM.WASFO.ProjectExplorer (C++/CLI)
3. IBM.WASFO.ProjectDesign (C++/CLI)
4. IBM.WASFO.Graphs (C#.NET)
5. IBM.WASFO.MTOM (C#.NET)
6. IBM.WASFO.ShipmentService (C#.NET)

The components which, will be implemented for WASFO Toolset in the future are as follows:

7. IBM.WASFO.Authentication (C#.NET)
8. IBM.WASFO.ReportGenerator (C#.NET)
9. IBM.WASFO.DCCController (already designed using C#.NET)

We bought also a license of .NET Controls Library from Infragistics²⁶ for the sake of WASFO GUI in order to improve and enhance the graphical user interface of both WASFO Tools. In the following sections, detailed descriptions are given about each WASFO's shared library.

4.4.1 IBM.WASFO.Core

It is the core library of WASFO Toolset, since it is used mostly by all WASFO components, the dependent libraries including the unimplemented libraries yet, are the following:

1. IBM.WASFO.ProjectDesign
2. IBM.WASFO.ProjectExplorer
3. IBM.WASFO.DataCollector
4. IBM.WASFO.InventoryAnalysis
5. IBM.WASFO.WorkloadAnalysis
6. IBM.WASFO.Optimization
7. IBM.WASFO.AOController
8. IBM.WASFO.Authentication (unimplemented yet)
9. IBM.WASFO.ReportGenerator (unimplemented yet)
10. IBM.WASFO.DCCController (unimplemented yet, however has been designed)

As mentioned before, the old version of WASFO did not use libraries so the code related to this library was previously implemented in the application of WASFO. The code was non-modular and complex. Thus, I had to break down the complexity of the code by separating all the classes that are frequently used by both WASFO Tools and put them in the core library. As a consequence, the core library has currently the basic classes and functionalities which are shared between all WASFO components. The library has also the data interface layer with WASFO database which is called OleDbInterface. This interface is responsible for data manipulation and queries with the database.

²⁶ Infragistics; A company which is considered as one of the world leaders in user interface development tools

4.4.2 IBM.WASFO.ProjectDesign

The ProjectDesign component is a user control library which is composed of a professional main menu, customized forms and some other classes. This control is used in the GUI of both WASFO tools as a menu strip at the top of the main form. The menu has the basic items needed for designing WASFO projects. One of the important menu items is the “*New Project*” item; It is mainly designed for the creation process of new WASFO projects through a wizard that consists of a sequence of 5 customized forms. The wizard starts when users selects the “*New Project*”.

Figure 7 shows the class diagram of the ProjectDesign component. There is the class “ProjectDesignController” that hosts the menu strip. It acts as the controller of the menu; adding menu items, removing menu items and adding handlers for menu items. In addition, it receives requests/events from users in the form of actions, and then redirects these requests to the proper handlers in order to handle the fired events.

In the diagram you find also the customized forms of the wizard dedicated for creating new WASFO projects. The forms of the wizard are as follows:

1. AddProject Form
2. AddDatacenters Form
3. AddGroupsOfServers Form
4. AddSubgroupsOfServers Form
5. AddServers Form

In figure 8, you can see the sequence diagram that describes the creation process of a new project through WASFO Project Wizard. During the creation process, users are able to navigate forward and backward between the wizard forms. Normally, the wizard starts with the “AddProjectForm” to add the project’s details, then users select “Next” button to go to the “AddDatacentersForm” to add datacenters to the created project, then users select “Next” button to go to the “AddGroupsOfServersForm” to add groups of servers that should group some existing servers in the server farm of the company, after then users select “Next” button to go to the “AddSubgroupsOfServersForm” to add subgroups of servers inside the created groups in order to better separate logically the servers, and finally users select “Next” button to go to the last form “AddServersForm” to add the existing servers of their server farms inside the previously created groups and subgroups. The wizard completes when users select “Finish” button in order to add the new project’s structure into the database with the help of the class “ProjectWizard”. Thus, the ProjectDesign component depends on the IBM.WASFO.Core library for any database handling issues. Indeed, the class “ProjectWizard” is responsible for initializing forms, navigating between forms, handling projects’ structure and adding new projects into the database. Moreover, WASFO’s wizard is so efficient since it can be used also for modifying the structure of existing projects at any level, hence users can start the wizard by selecting the desired project/group/subgroup/server node in the ProjectExplorer control without the need to start the wizard from the beginning.

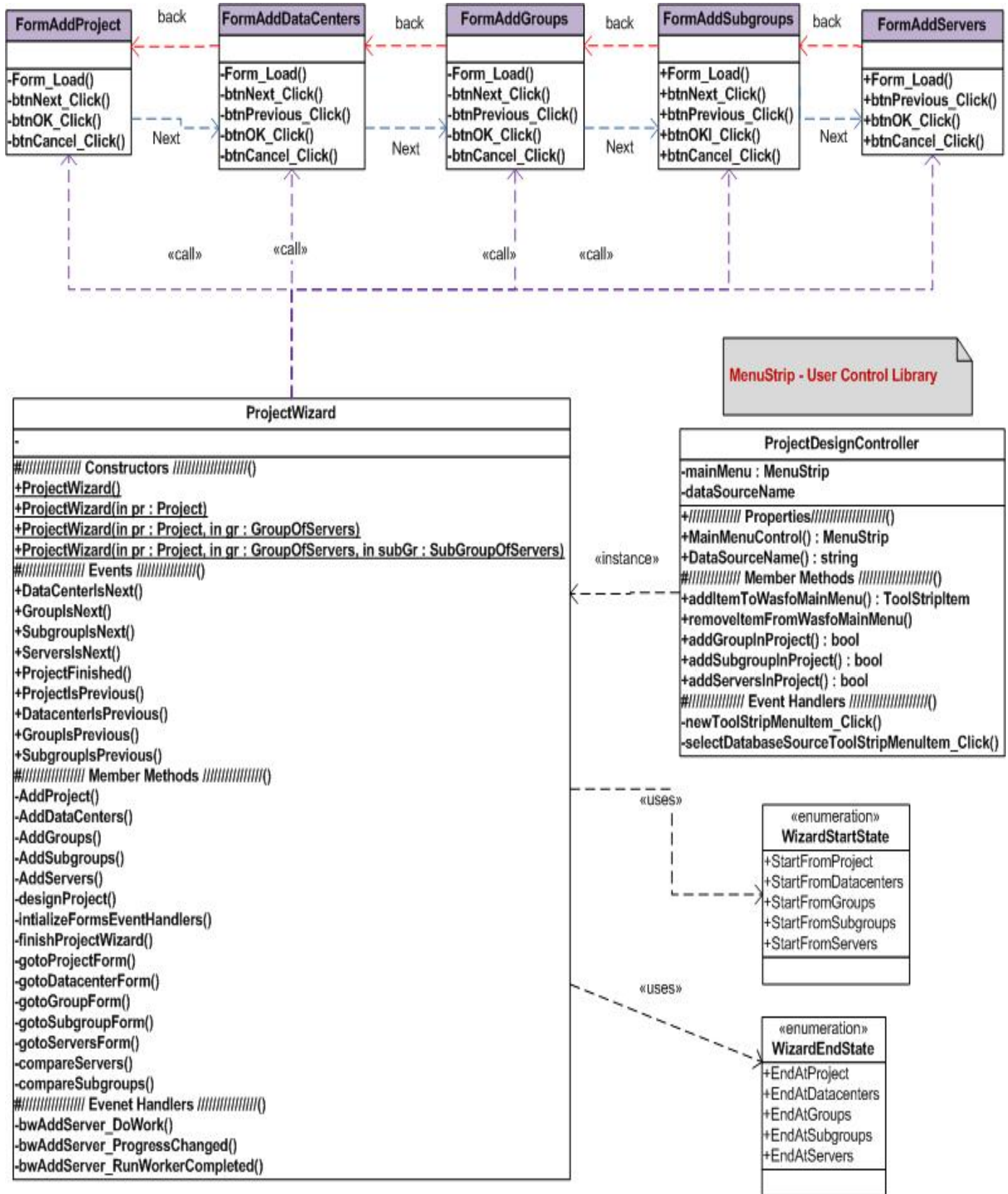


Figure 7 Class diagram of ProjectDesign Component

Sequence Diagram of WASFO Project Wizard

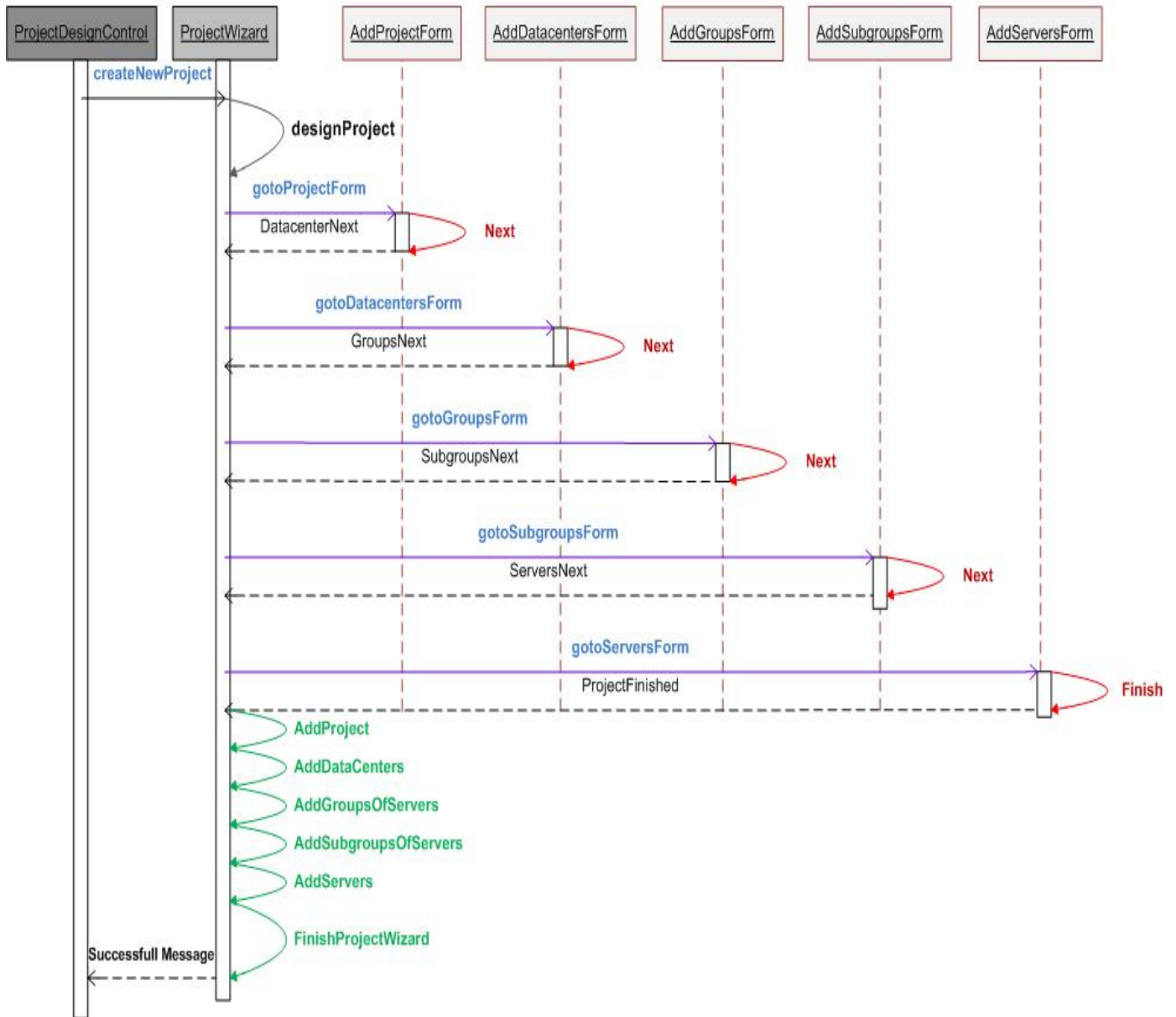


Figure 8 Sequence diagram of ProjectDesign Component

In the figure below, a state diagram describes the different states of WASFO Project Wizard during the creation of a new project. The states are as follows:

1. NewProject
2. NewDatacenters
3. NewGroupsOfServers
4. NewSubgroupsOfServers
5. NewServers

As mentioned before, the class “ProjectWizard” controls the procedures of the wizard, and hence it controls the state transitions based on users’ actions. In each visual form, users have the ability to select between several buttons to navigate and proceed between the forms; the button “Next” is to proceed, the button “Previous” is to move backward, the button “Cancel” is to cancel the wizard, and finally the button ‘Finish’ is to add the new project. The interesting fact in WASFO wizard that users could finish the wizard at any level, without being restricted to continue till the end, being sure the wizard will save the project’s data into the database until the form in which users has reached. Thus, the wizard is designed to be fast, user-friendly and flexible with users to allow the efficient design of WASFO projects. In figure (60) at section (I) in Appendix, you can find a screenshot of the Wizard, which shows the creation process of a new WASFO project.

State Diagram of WASFO Project Wizard

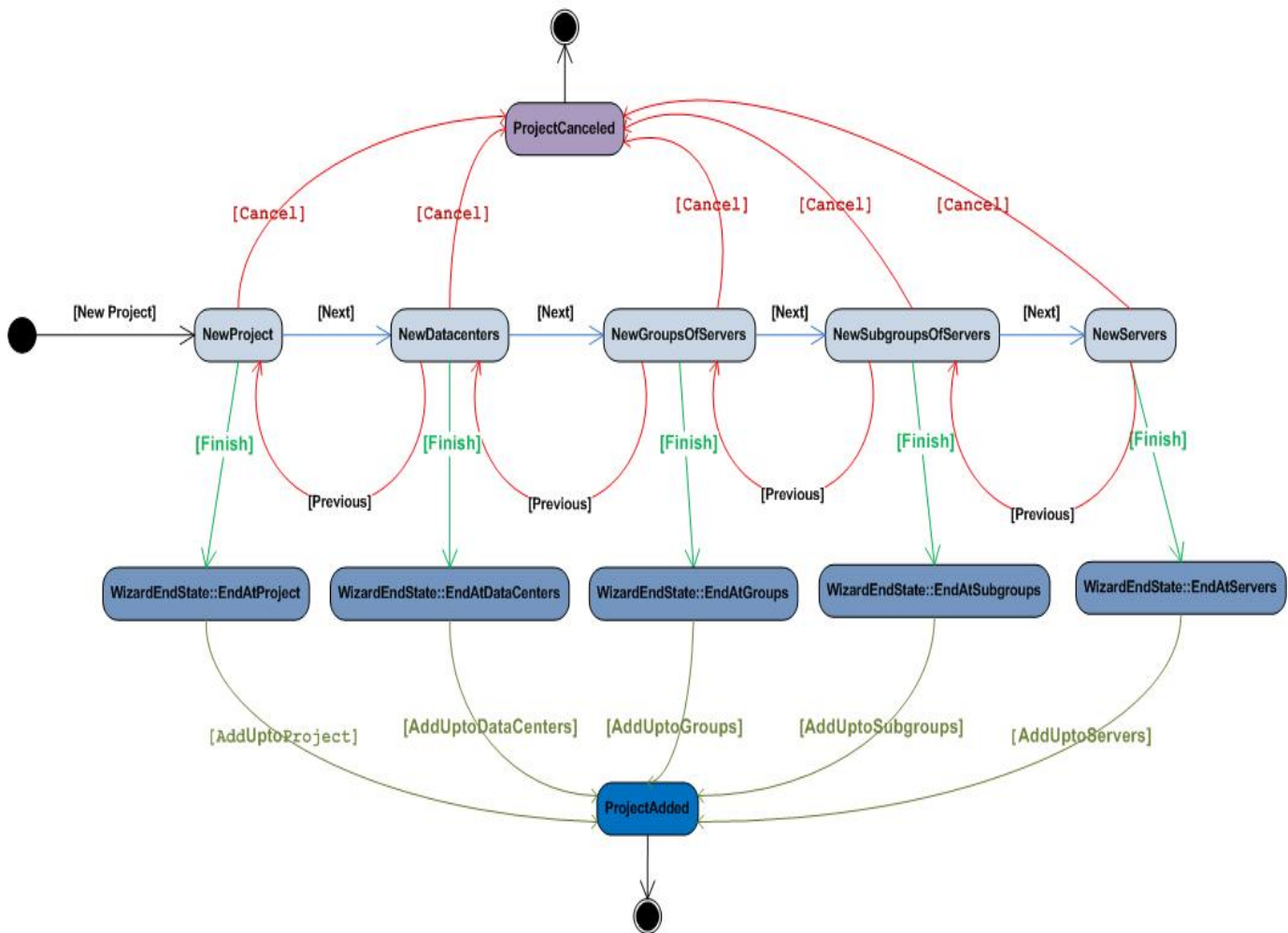


Figure 9 State diagram of ProjectDesign Component

4.4.3 IBM.WASFO.ProjectExplorer

The ProjectExplorer component is a user control library which is composed of a treeview, context menus and customized forms to be used in the GUI of each WASFO Tool. The treeview enables users to explore the structure of WASFO projects' through the tree nodes. The WASFO Project has the following nodes' structure:

- Project Node (**Root**)
 - Servers Node (*All servers inside the project*)
 - Server Node
 - Groups of Servers Node (*All groups of servers inside the project*)
 - Group of Servers Node
 - Servers Node (*All servers inside a specific group of servers*)
 - ❖ Server Node
 - Optimizations Node
 - ❖ Optimization Node
 - Constraints Node
 - ❖ Constraint Node
 - Subgroup of Servers Node
 - ❖ Server Node

Each node in the treeview is represented by a treenode object. This object carries information about the node such as nodeId, tag, name, icon, projectId...etc. Actually, the ProjectExplorer connects with the database of WASFO in order to get all the needed information for representing a project's structure and for making it accessible and navigable to users. Thus, it depends on the *IBM.WASFO.Core* library for any database handling issue. Figure 10 shows a diagram which explains the hierarchical structure of projects' nodes and their different levels.

In figure 11, a sequence diagram shows two different processes of loading information for project's nodes in ProjectExplorer. The diagram shows in details the sequence of operations for each case between 3 main participants (each participant resides in a different library/component):

1. IBM .WASFO.DataCollectorGUI or WASFO.AnalysisOptimizationGUI Tool
2. IBM.WASFO.ProjectExplorer Component
3. IBM.WASFO.Core Component

Case 1, describes how to load detailed information (project, datacenters, groups, subgroups and servers) of a project's nodes using the **Eager-Loading** technique. This case happens when WASFO tool starts up or after the user creates a new WASFO project. Hence, the treeview controller starts to get all information needed from the database to create a project node in the treeview of WASFO. Due to the long duration of this process, it runs in a background thread to avoid hanging the user interface of WASFO tool, and so users could interact with the program efficiently without the need to wait until the ProjectExplorer completes loading all projects' nodes (For example users could access the loaded nodes while other projects' nodes being loaded).

Description of ProjectExplorer Treeview of WASFO Project

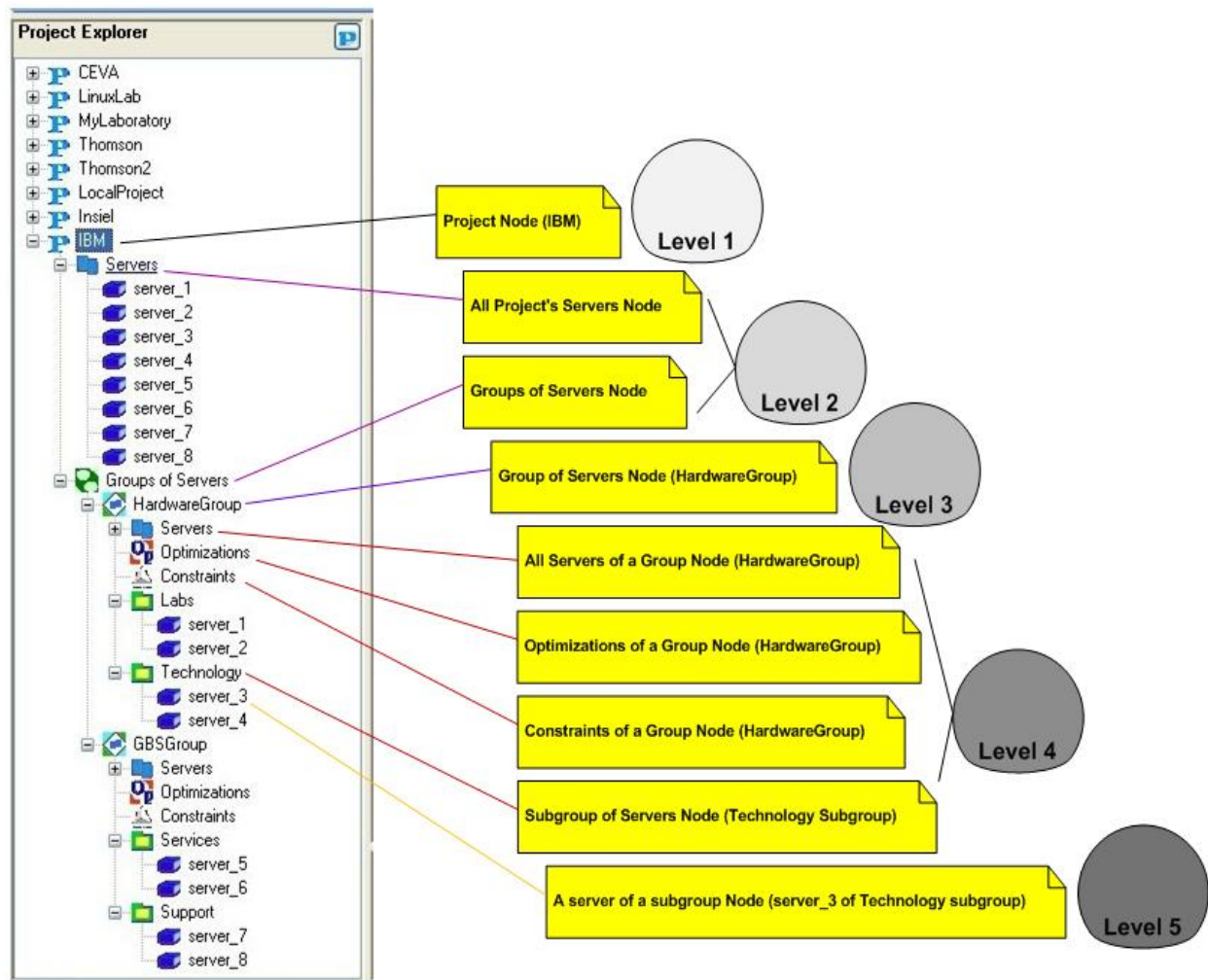


Figure 10 The ProjectExplorer Component (Custom control of type: Treeview)

Case 2, shows how to load limited information (project, datacenters) of a selected project's node using the *Lazy-Loading* technique. This happens when the users select a project's node in the treeview. Hence, the selected project becomes the current selected project in ProjectExplorer component. In addition, to add more efficiency to the component, I implemented an algorithm for storing the selected projects in a dynamic list (resizable array) in the cache memory, in order to avoid re-loading these nodes again, alternatively ProjectExplorer will get the cached node and its related information directly from cache memory. However, I had to optimize using the cache memory by restricting the size of the list to avoid causing a heavy load on the cache, which as a consequence might cause inefficiency.

The scenario starts when a project node (or any sub-node in a project node) is selected, the algorithm checks automatically if the project node was selected before. If it was previously selected, so the algorithm will increase its rank by 1 (herein rank means number of selections); every element in the list has a corresponding rank. If the selected node is new, the algorithm will check if the list has reached its maximum size, then based on this, it behaves as the following:

1. If not; the project node will be added automatically to the list.
2. If yes; the array will remove from the list the node which has the lowest ranking (minimum number of selections, which means that the node is not frequently used)

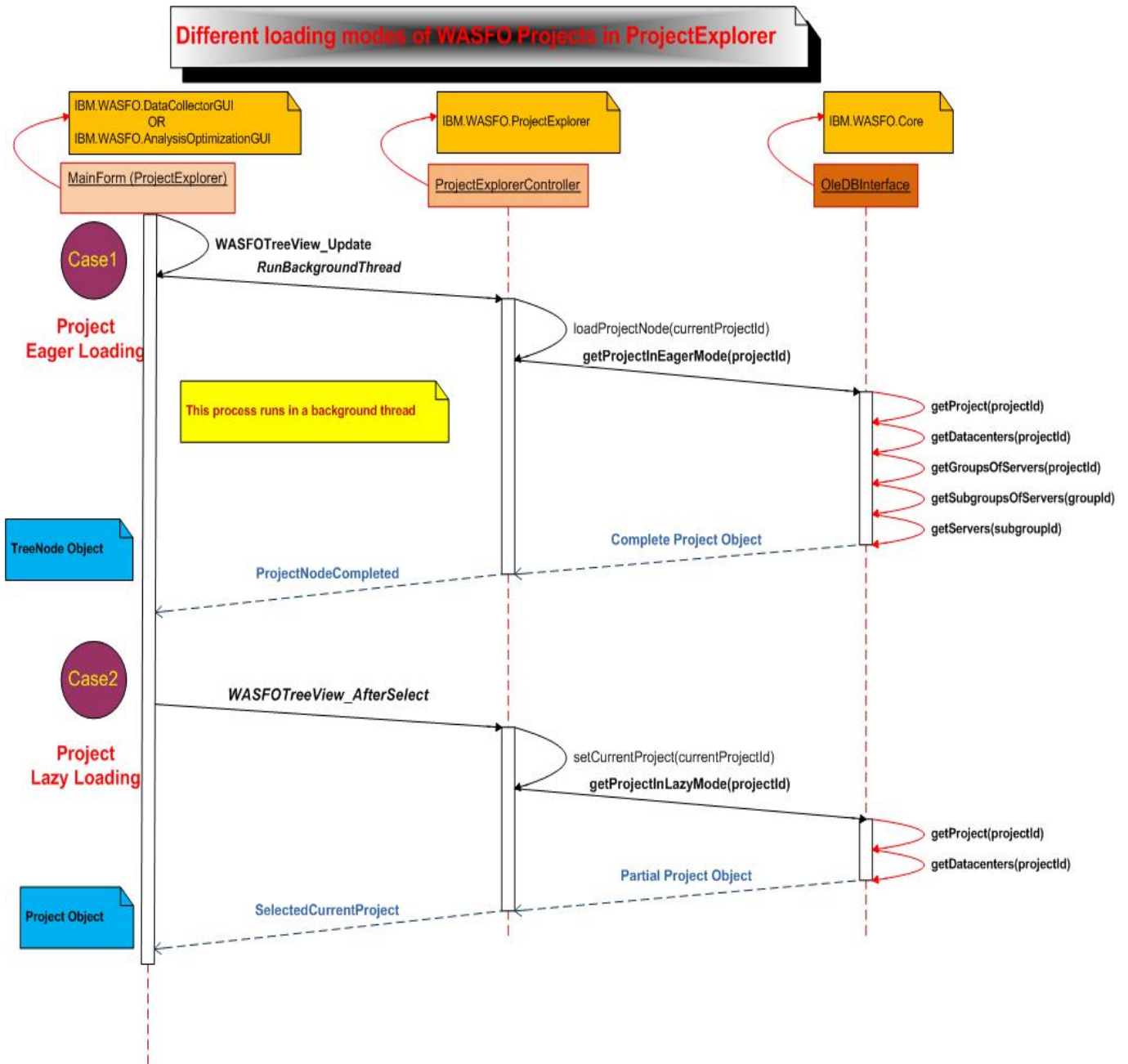


Figure 11 Sequence diagram of ProjectExplorer Component

In figure 12, a class diagram of the ProjectExplorer component shows the class *ProjectExplorerController* which hosts WASFO’s Treeview and its context menu. The controller controls and performs all the operations regarding initializing, loading and manipulating WASFO project’s node. In addition, you can see all the customized forms which represent the different operations that can be done by the ProjectExplorer component. These operations are initialized by accessing the context menu. They allow users to add, modify, search and delete any node in the project, and the corresponding object in the database.

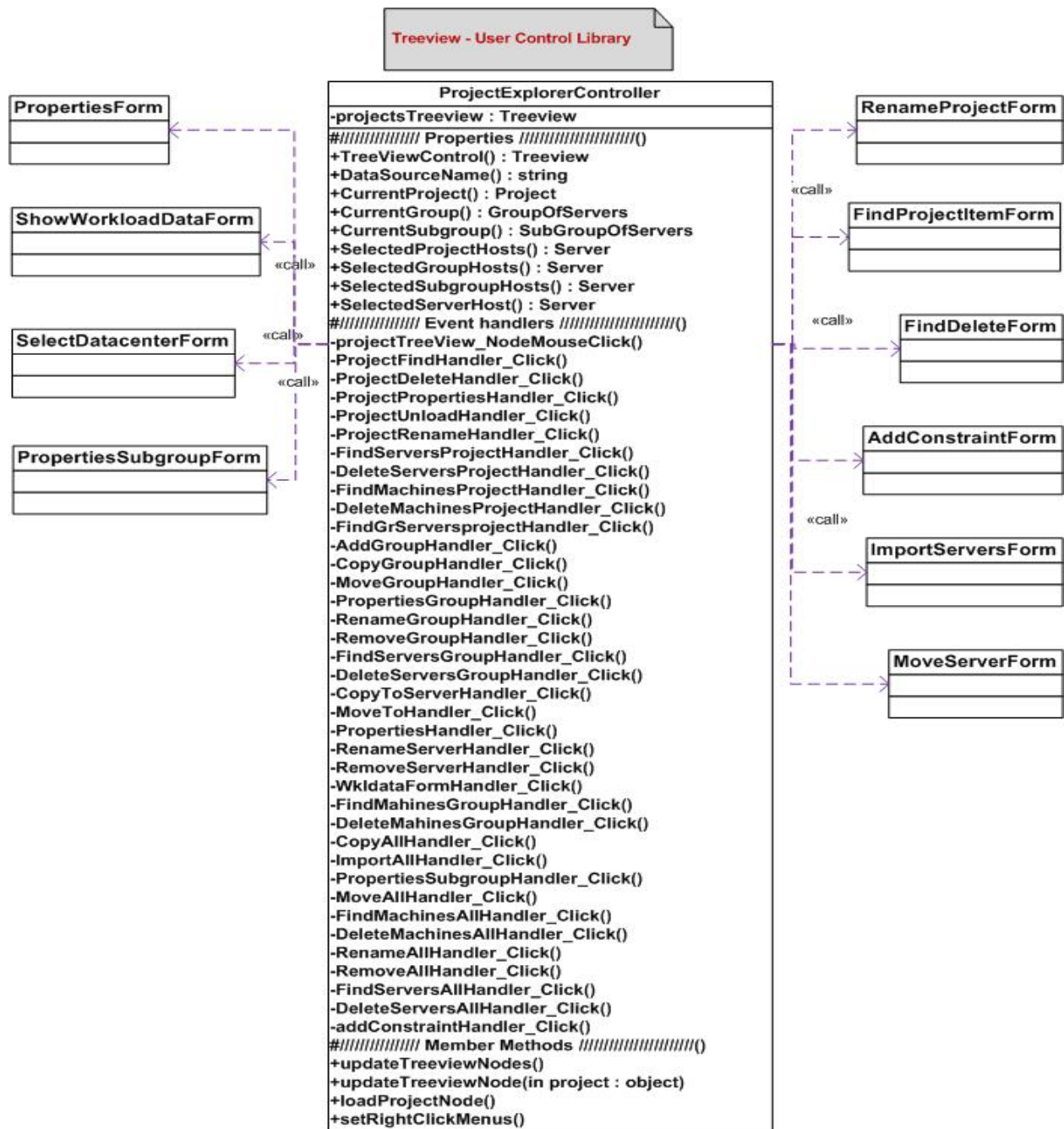


Figure 12 Class diagram of ProjectExplorer Component

4.4.4 IBM.WASFO.Authentication

A visual component will be responsible for authenticating WASFO users. Currently, the logic of the authentication process is implemented separately inside both WASFO Tools; consequently, it is duplicated in each tool. Thus to overcome this problem, we should build this component to-be shared between WASFO Tools. The component will include the current login form of WASFO. Hereunder, you can find a figure about the *WASFO Authentication Form* that is currently used by WASFO Tools. Hence, the component will depend on IBM.WASFO.Core Library to authenticate users based on the users' license keys and the users' data in the database. Indeed, this component is not implemented yet because it is still under design.

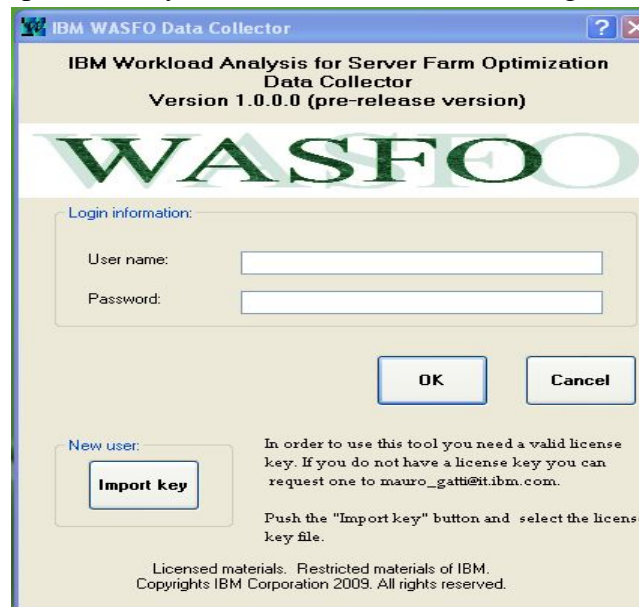


Figure 13 The authentication Form of WASFO Toolset

The authentication process is quite complicated in WASFO, since it comprises three different inputs to verify and validate the users successfully. The three inputs are the following:

1. Username
2. Password
3. Import the user's license key

The authentication process begins once the user inserts the required inputs and clicks on the button OK. After that, the authentication component calls a sequence of specific functions to check and verify the user identity. The authentication's procedures are as follows:

1. Check if the *UserName* and the *Password* exist and are correct in WASFO database in the table "LicenseKeys";
2. Get the *SecretKey* of the user from the table, and then the *LicenseKeyType*;
3. Get the *ComputerName* of the user from the table;
4. Compute the Salt; the salt's output of the user's password;
5. Perform SHA-1 Hashing for a combination of the *UserName*, *Password*, *LicenseKeyType* and the *SecretKey*. The output represents the computed LicenseKey of the user;
6. Compare the output of the hashing process with the imported license key of the user;

7. If equal, then get the user security credentials by performing SHA-1 hashing to the following combination;
 - a. The user's entries in the table "LicenseKeys": *UserName*, *Password*, *SecretKey*, *ComputerName*, *Salt*.
 - b. Plus, the *LicenseKey* that has been computed previously.
8. Then, compute the SHA-1 hashing of the following combination;
 - a. Username (inserted by the user)
 - b. Password (inserted by the user)
 - c. Secret Key (fetched from the database based on the *UserName* and the *Password*)
 - d. License key (imported by the user)
 - e. Computer Name (obtained programmatically from the user's running machine)
 - f. Salt of the user's password (computed based on the inserted password by the user)
9. Compare this hashing output with the user's security credentials (above calculated);
10. If equal, then the user is identified by WASFO, and hence logs in successfully.

4.4.5 IBM.WASFO.ReportGenerator

A component will be responsible for generating reports in excel and PDF format concerning the results of WASFO processes. It will take the results of the processes as inputs and produces reports as outputs. The processes could be the data collection, inventory analysis, workload analysis or optimization design for a virtualized server farm. The generated reports are very useful in understanding the results, and discussing them with the customers. The library will be shared by both tools of WASFO; WASFO Data Collector Tool (for data collection's processes) and WASFO Analysis and Optimization Tool (for analysis's processes and optimization's processes). This library is not designed yet.

4.4.6 IBM.WASFO.Graphs

A component responsible for generating advanced 2D-graphs concerning the results of WASFO processes. It depends on NetAdvantage-Infragistics .NET Graph control. It takes the results of WASFO processes as inputs and produces graphs as outputs. The processes of WASFO could be the data collection, inventory analysis, workload analysis or optimization design for a virtualized server farm. Indeed, using graphs would be of great benefit in understanding and better visualizing WASFO results. The library is shared by both tools of WASFO; WASFO Data Collector Tool and WASFO Analysis and Optimization Tool.

4.4.7 Infragistics .NET Controls

NetAdvantage for .NET is a comprehensive suite of Windows Forms, Windows Presentation Foundation-WPF, Silverlight controls, components, and tools for the .NET platform. The user controls are included in a private library called InfragisticsControls. We used these controls in the GUI of WASFO Data Collector Tool and in the GUI of WASFO Analysis and Optimization Tool in order to add awesome power and performance to the graphical user interface of the toolset. In fact, the library has been used particularly for generating good look and feel graphs. Hence, it will be used more in the future like for example generating professional reports...etc.

5 IBM WASFO Data Collector Tool

5.1 As-is WASFO Toolset for Data Collection

The old version of WASFO is a single tool which included the functionalities of data collection from existing server farms. The main functionalities are the following:

1. Inventory Collection
2. Workload Collection

These functionalities are used by IBM business partners, however WASFO tool included also the functionalities of data analysis and optimization (inventory analysis, workload analysis and data optimization), hence the user could have also the ability to switch from the *Data Collection-Mode* to the *Analysis and Optimization-Mode* through the main menu of WASFO Tool, and vice versa.

Obviously, this was an inefficient and non-professional way to use the analysis and optimization mode of WASFO since its use is only allowed to IBM employees. However all users could use this feature once they have WASFO Tool installed on their PC. Thus, we had to separate the data collection functionalities of WASFO in a new tool to-be called “IBM WASFO Data Collector Tool”. Below you can see a screenshot of the old WASFO version for data collection mode. As you might observe the limited and inefficient GUI of the data collection mode.

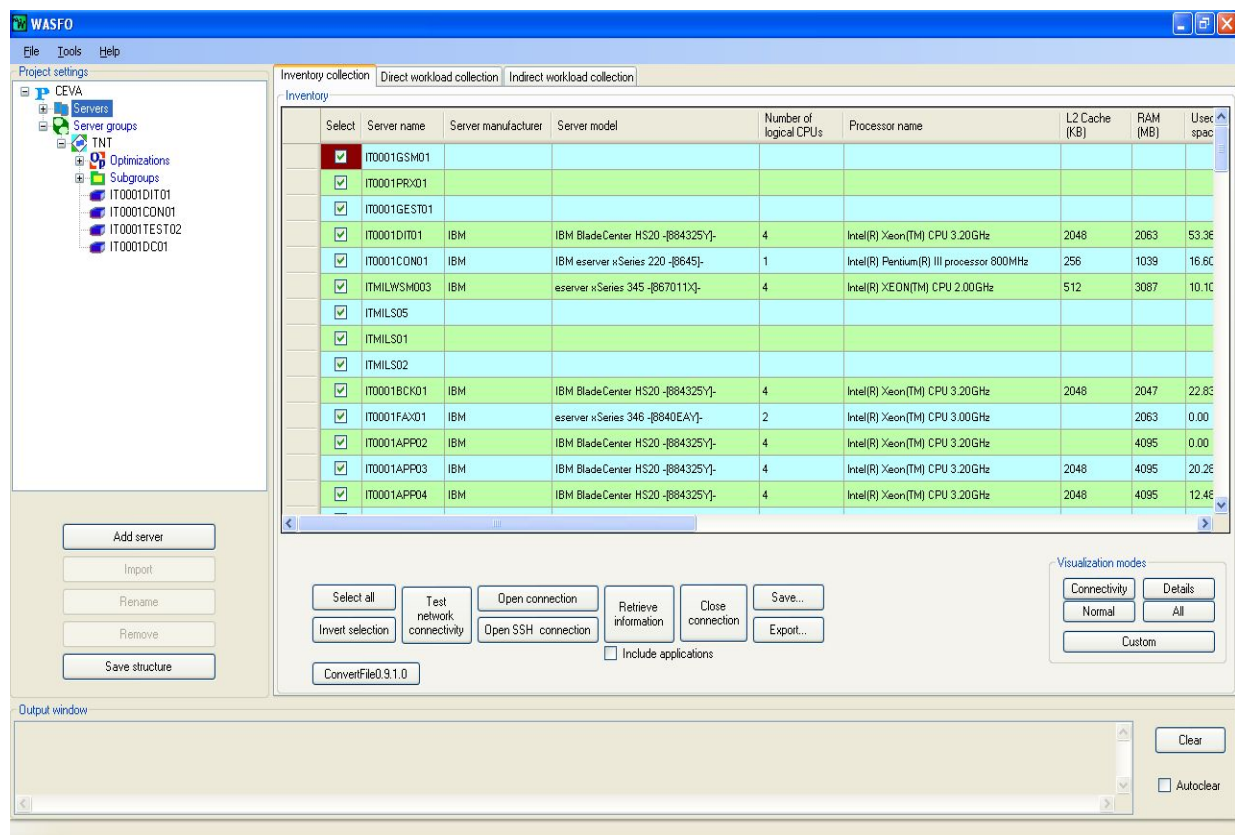


Figure 14 Main form of WASFO Data Collector Tool

5.2 To-be WASFO Data Collector Tool

After a deep analysis that was performed on the release of the old WASFO, we came up with the new tool “IBM WASFO Data Collector Tool”, that is responsible for collecting data from existing server farms for the sake of designing an optimal virtualized server farm. The tool currently collects:

- Inventory data (server model, CPU²⁷ model, number of CPUs...etc)
- Workload data (processor utilization, memory utilization...etc)
- Virtual Machines data (name of virtual machines, processor shares, memory share...etc)
- Applications inventory (name and version of installed applications)
- Connections data (connected server’s IP address, used network protocol...etc)

The servers’ data can be collected from several operating systems such as:

- Windows
 - By using Windows Management Instrumentation-*WMI*.
- Linux
 - By executing standard Linux command through a SSH²⁸ connection. The connection is created by using *PUTTY.exe* tool.
- AIX
 - By using a Telnet connection.
- VMware ESX and vSphere
 - By using web services. VMware API provides a mechanism that makes it possible to collect inventory and workload data.

Based on WASFO requirements and specifications, the new tool has been designed and developed in a modular and optimized way. Thus, we guarantee that the process of data collection is quick, easy, and efficient. Indeed, the main features of the current WASFO Data Collector are as follows:

- Data collection does not have to be installed on monitored systems (*agent-less*)
- Data collection exploits multi-threading techniques to provide fast and efficient data collection methods

The data collection process should work efficiently with well-managed server farms. However, it could be a tough process if some conditions have not been met such as the following:

- Understanding the LAN structure.
- Awareness of where firewalls are and what they do.
- Right security credentials.
- Inexistent servers that exhibit various forms of corruption.

Figure 15 demonstrates the UML component diagram that shows the dependencies between WASFO Data Collector Tool and its related libraries. Some of these libraries are shared libraries; hence, WASFO Analysis and Optimization Tool are using these libraries as well (as previously discussed). As mentioned before, all the shared libraries are implemented except for *IBM.WASFO.ReportGenerator* and *IBM.WASFO.Authentication*.

²⁷ CPU; Central Processing Unit

²⁸ SSH; Secure Shell, **a network protocol** allows data to be exchanged between two networked devices. It is used mainly on Linux and UNIX systems.

Regarding the dedicated (unshared) libraries for the Data Collector tool, as you can see below in figure 15 they are as follows:

- IBM.WASFO.DataCollector
- IBM.WASFO.DCController

WASFO DataCollector library was implemented using the old class structure for data collection. Hence, a new version of the library has been implemented to overcome the design problems of the old version but still not yet integrated (this new version will be discussed later). Regarding WASFO DCController library, it has been designed, and hence it should be implemented in the near future along with future work for WASFO.

In addition, there is the *EXE* program of the tool; “*IBM.WASFO.DataCollectorGUI.exe*”. This program will use the controller “*IBM.WASFO.DCController*” to call any data collection method. The controller acts as a facade, which receives requests from the user interface of the tool and then sends them to the proper handlers in the DataCollector library, after that the controller responds to the user with the results. *IBM.WASFO.DataCollector* library depends on *IBM.WASFO.Core* library for performing database operations or any needed basic functionalities. Indeed, *DataCollectorGUI* represents the presentation layer of the tool, while *DCController*, *DataCollector*, and some other libraries will represent the business layer of WASFO Data Collector Tool. From the diagram below, we can observe that most of the components are dependent on the library of *IBM.WASFO.Core*. Thus, the tool is a collection of combined components that form transparently an integrated tool. This is definitely so efficient and powerful since these components could be reused in other tools. Moreover, they can be updated and modified easily because of their high modularity. Detailed descriptions about these components will be given later in this chapter.

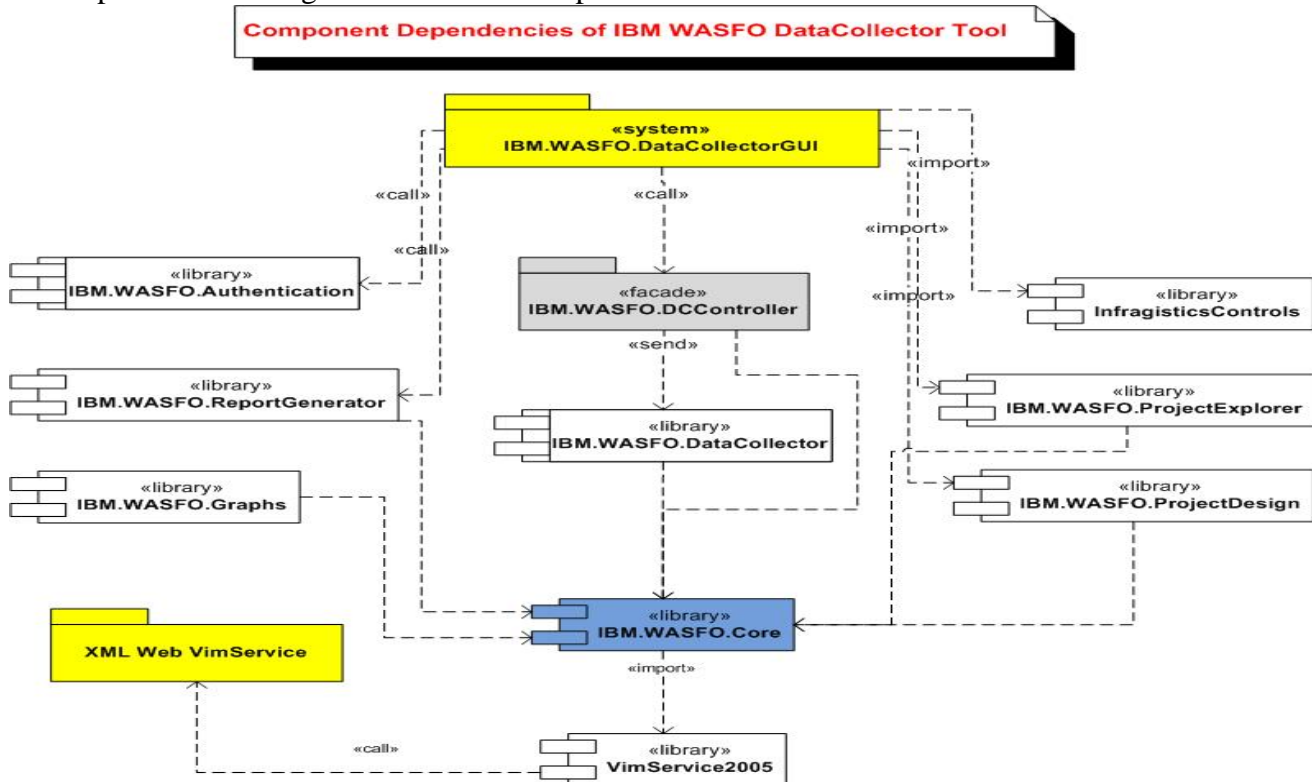


Figure 15 Component diagram of WASFO Data Collector Tool

5.3 Software Architecture

5.3.1 Inventory Collection

The first process to be performed by WASFO Data Collector Tool is to collect the inventory data of an existing server farm. Inventory data is needed information to further design and optimize the existing server farm. It corresponds to the running servers in a customer's company such as server model, server manufacturer, processor model, number of processor cores, amount of installed memory and used storage space...etc. The tool is flexible and customizable since it permits users to choose the inventory items they require for collection without being forced to collect all inventory items, which are offered by the tool. Figure 16 demonstrates the activity diagram about the process of the inventory collection. It shows the sequence of activities that the process follows when it collects the inventory data from a server farm. As you can see, the process ignores any unready servers in order to avoid any obstacles during the process. When the collection is completed, the data will be saved into WASFO database by the help of *IBM.WASFO.Core* library for further analysis.

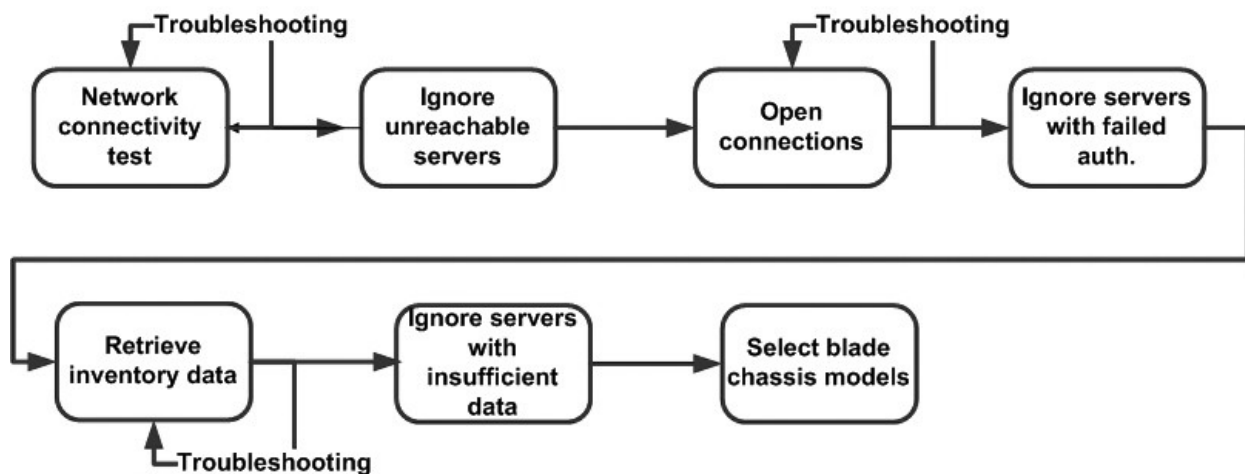


Figure 16 Activity diagram of Inventory Collection Process (Drawn by other WASFO Team members, IBM Italy)

5.3.2 Workload Collection

The second process to be performed by WASFO Data Collector Tool is the workload collection. In this process, the tool collects the workload data of an existing server farm. This data is so important for the design and the optimization of the server farm, since it gives us a real time indication about the servers' utilization in a running server farm. Workload data could be the processor utilization, memory utilization, disk I/O utilization or network I/O utilization. Indeed, users have the facility to select the workload items they need for the collection process. Hence, this gives them more flexibility to optimize the design of their server farms. Figure 17 shows the flow diagram of the workload data collection from a server. It shows the activities of data collection from a server that has one or more virtual machines. At the end of the process, a workload file is created for each machine in the server (Machine means a running virtual machine on a physical server). All the workload files are saved into a folder called "Workload Files" in the local desk, where WASFO tool is installed. When the collection is completed, the machines' data will be saved into WASFO database by the core library for further analysis.

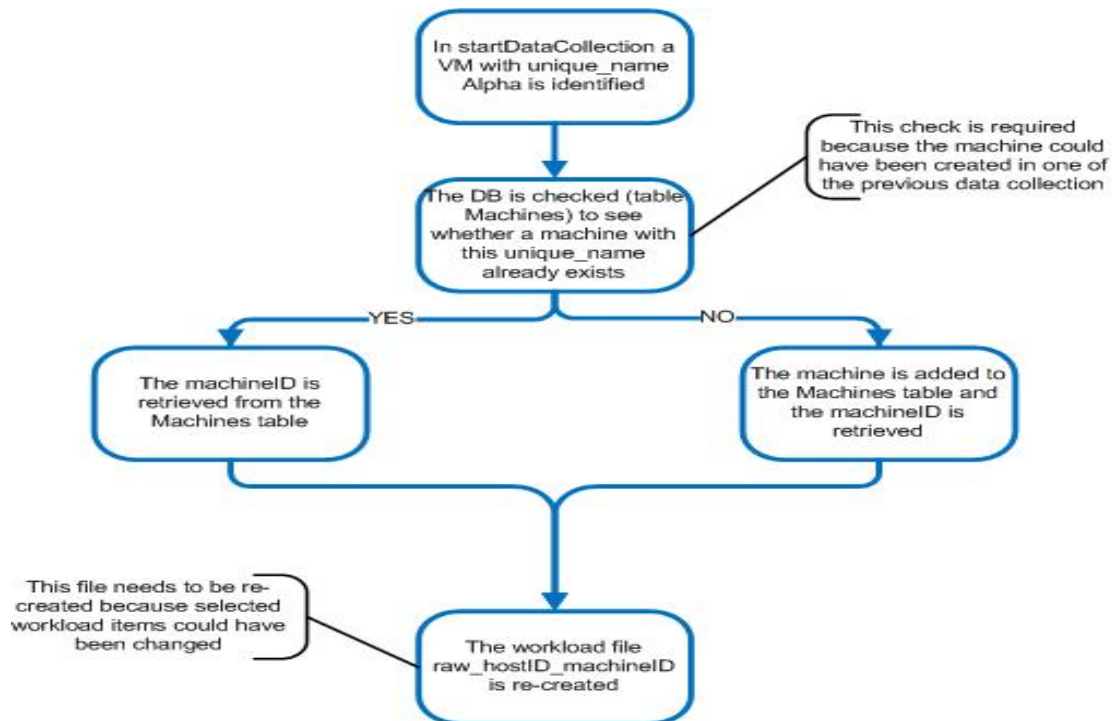


Figure 17 Flow diagram of Workload Collection Process (Drawn by Dr. Mauro Gatti, IBM Italy)

5.3.3 IBM.WASFO.DataCollector Component

This library is the main library of the Data Collector tool, since it provides the data collection's functionalities. The code in this library was implemented before inside the tool directly without being modular. Moreover, it was not well designed and hence the performance was inefficient. Thus, I separated all the classes related to the data collection from WASFO Tool into the DataCollector library, in order to modularize the code and also re-design it better. It is now being used successfully by WASFO Data Collector Tool.

However, due to the inefficient performance of the library, we had to re-design the class structure of the old code in order to make the library more efficient and modular. Hence, the new version of the library is ready but not yet integrated with WASFO Data Collector tool. Currently the new version of the DataCollector library is designed based on Go4 design patterns using .NET language. The design patterns that have been used are as follows:

1. Factory Method pattern
2. Facade pattern
3. Product pattern

In figure 18 below, you can see the class diagram of the component and the hierarchical structure of the classes with respect to the types of the data collectors. As mentioned before, WASFO Data Collector collects data from several operating systems such as Windows, Linux, VMware or AIX. In fact, for each operating system, two classes of the data collection are implemented; One class for collecting inventory data, and the other class for collecting workload data. These classes are designed as several products-designing pattern. The factory method-design pattern

“*MonoDataCollectorFactory*” is used for creating instances of the required data collectors for data collection purpose. The facade-design pattern “*PoliDataCollectorHandler*” is designed to be the interface of the Data Collector library. In other words, WASFO Data Collector Tool performs inventory or workload data collection on server farms through this facade by calling the appropriate method in the facade, providing it the desired servers and their security credentials. Based on the type of the server’s OS, the related data collection’s class is called.

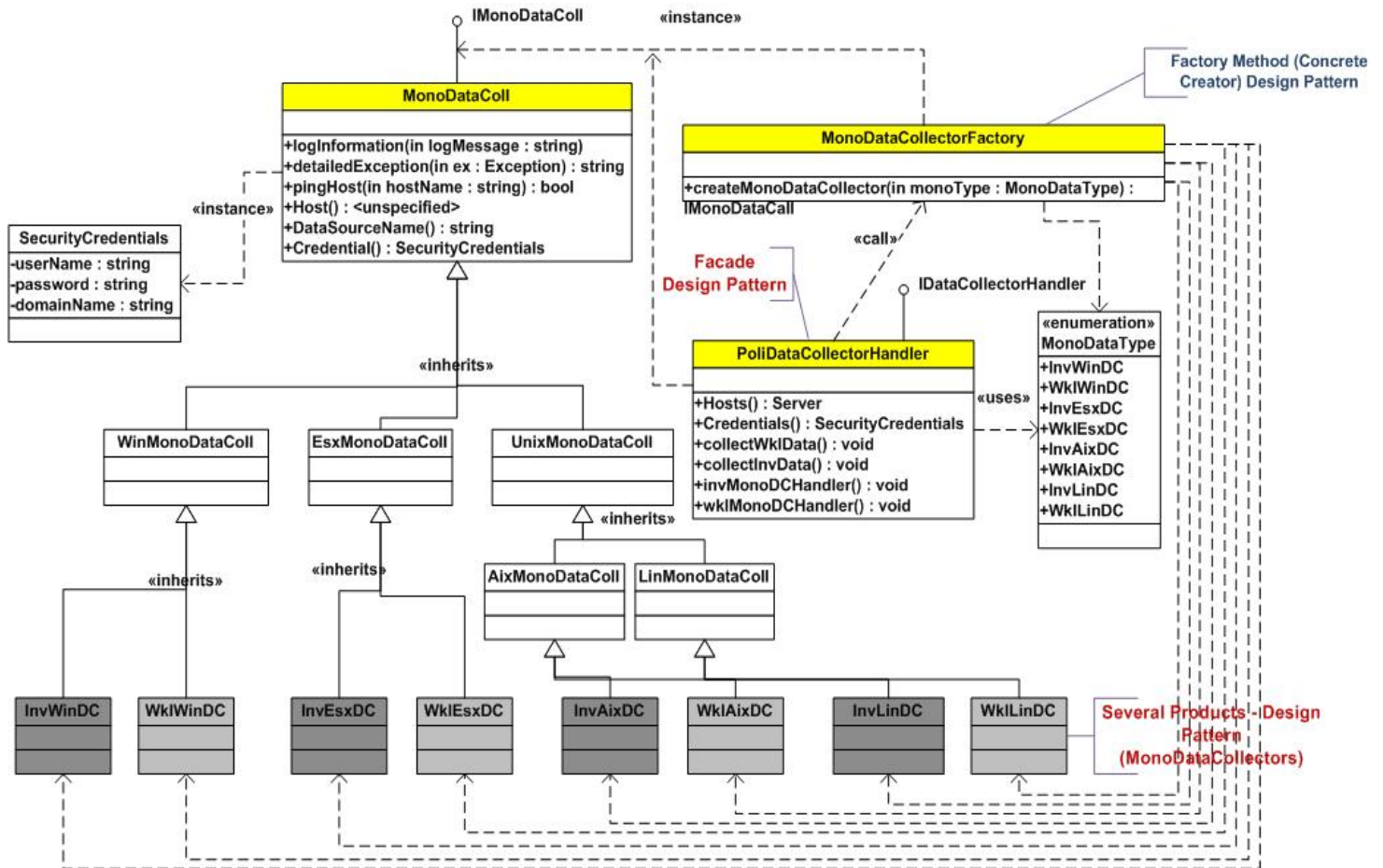


Figure 18 Class diagram of DataCollector Library

5.3.4 IBM.WASFO.DCController Component

This library will be mainly a gateway between the DataCollector library and the Data Collector Tool (user interface). This component has been designed, however it is not implemented yet. Thus, it should be implemented and integrated with the tool in order to complete the current architecture of WASFO Data Collector. As mentioned before, it acts as facade for the business functionalities concerning the data collection. Besides this, it carries all the requests/responses from/to any business component used by the tool, and so it hides the complexity from the presentation layer. We can imagine it as an interface layer between the presentation and the business layers. I have designed the library using the Go4 design patterns in .NET language:

1. Facade pattern
2. Singleton pattern

3. Proxy pattern

The facade pattern-classes (*InventoryCollection*, *WorkloadCollection* and *MachineCollection*) act as interfaces between the DCController component and the referenced library “DataCollector”. Hence, all the methods of the referenced library are provided and called in their corresponding facade classes. For example, all the methods in the DataCollector library concerning the inventory collection’s functionalities are called in the InventoryCollection-facade class.

The proxy pattern-classes (*InvCollController*, *WklCollController*, *MachineCollController*) act as proxies for the facade classes, in front of WASFO Data Collector Tool. In other words, they represent the interface controllers of the DCController component, whereby each controller (proxy) is responsible for its corresponding facade class. Indeed, each proxy class implements a corresponding Interface (*IInventoryCollection*, *IWorkloadCollection* and *IMachineCollection*) to its facade class. This interface represents a contract between the facade class and its proxy class, and so we guarantee the data integrity between both classes.

The singleton pattern is built inside each controller to guarantee that only one instance will be created for every controller in order to provide more efficiency in the performance. Thus, as you can see the DCController component has a double-sided interface; interface between the “DCController component” and the “DataCollector library”, and another interface between the “DCController component” and “WASFO Data Collector Tool”.

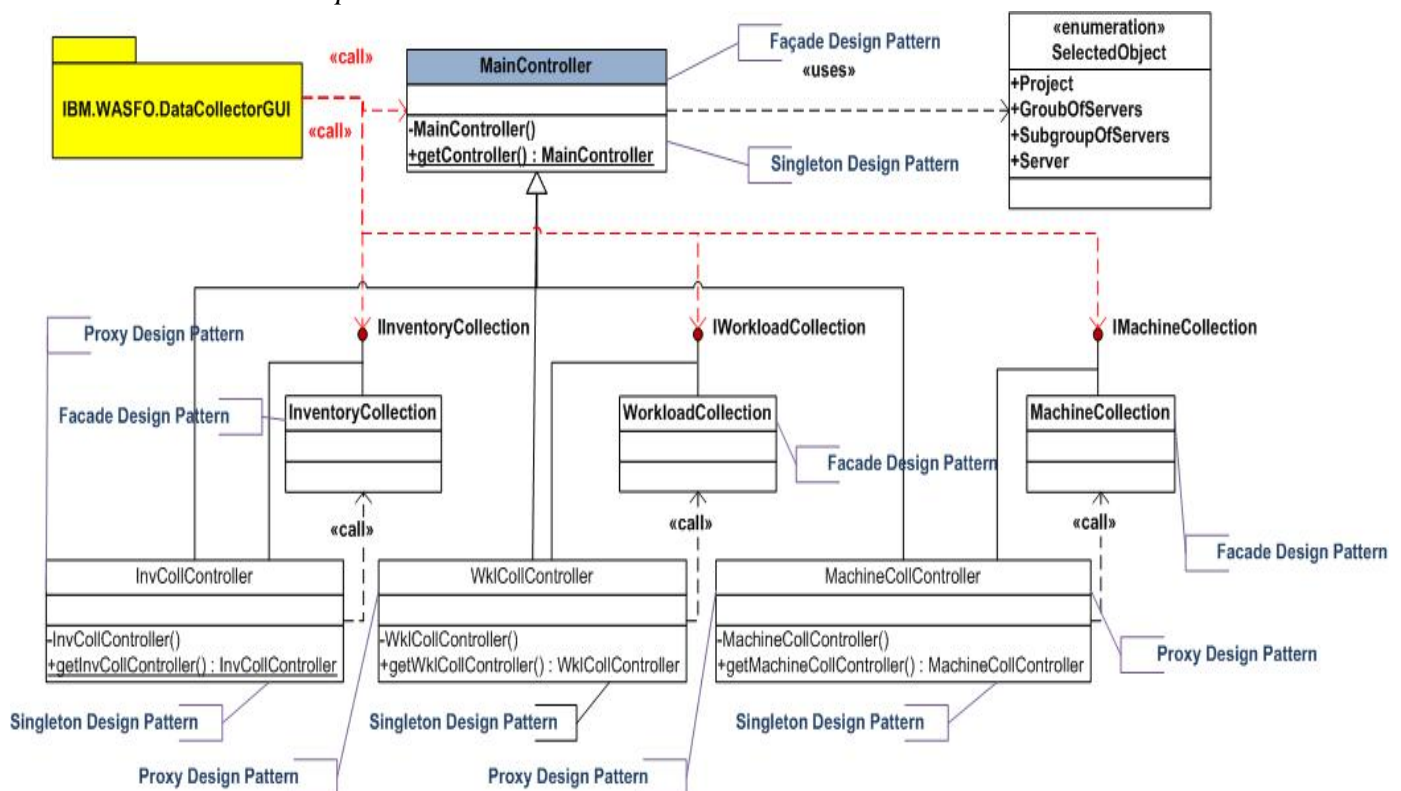


Figure 19 Class diagram of DCController Library

In the above figure (Fig. 19), a class diagram of the component shows the classes and their corresponding design patterns. It shows also how the “*IBM.WASFO.DataCollectorGUI.exe*” calls the controllers to request business functionalities. As you can see, there is also a main controller which have the common functions needed for the sub-controllers. Hence, all the sub-controllers inherit from this main controller.

5.3.5 IBM.WASFO.DataCollectorGUI.exe Program

The “*IBM.WASFO.DataCollectorGUI.exe*” is the executable program of WASFO Data Collector Tool. It consists of the main class (the entry point of the program), GUI classes, utility classes, visual forms and user controls (*ProjectDesign*, *ProjectExplorer* and *SendControl* components) which all, represent the graphical user interface of the tool. In addition, the tool is composed of a collection of libraries that represent the business layer (data collection). The data layer of the tool is represented by WASFO database. Indeed, the database and the *EXE* program are both installed using the package setup that is called “*IBM.WASFO.DataCollectorSetup.exe*”.

In order to make the tool portable and easy to use, we created an application configuration file called “*App.config*” in XML format to be installed with the tool. This file carries all the configurations needed for the tool to run successfully such as the database’s path, workload files’ path, shipment web services’ reference...etc. Indeed, this file can be modified in the run-time, in case if the user needs to change some configurations. Hereunder, is the class diagram of WASFO Data Collector Tool. As we can see, the class structure is clear, whereby it contains only visual forms, their related classes and some utility classes. This is due to the fact that the business layer is built in the form of business components, which are used by the tool as reference libraries .The main form of the tool is the “*FormMain*”, which starts after the appearance of the welcome screen. The caller to the main form is the main entry of the tool; the class “*Program*” that contains the “*Main*” method of the program. There is also the class “*Resources*” which manages all the resources needed for the tool. In section A at Appendix, you will find a screenshot of the graphical user interface of WASFO Data Collector Tool.

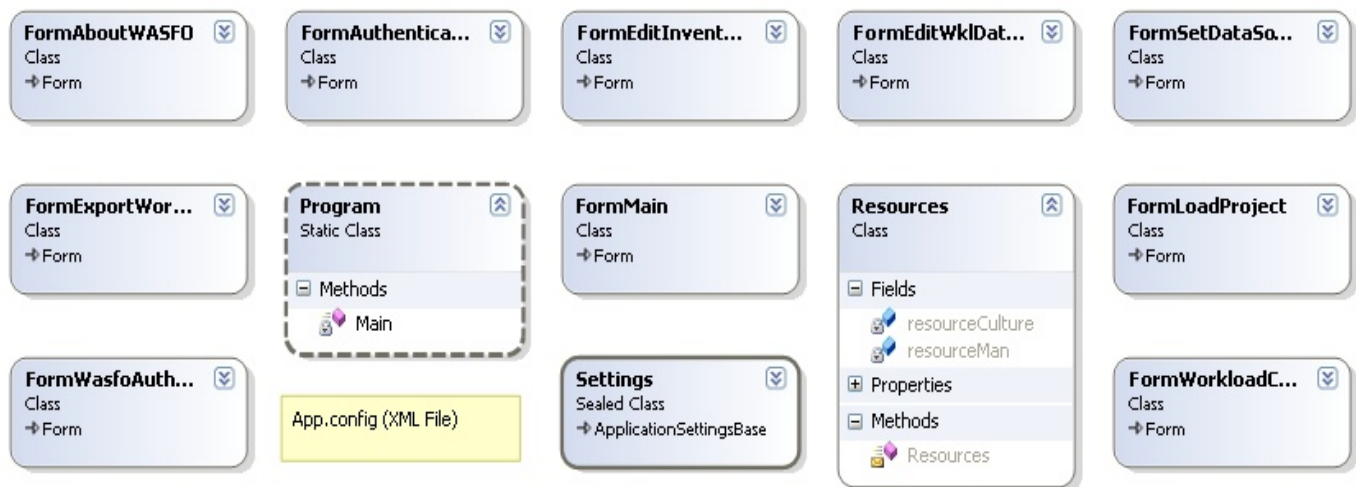


Figure 20 .NET Class diagram of the DataCollectorGUI.exe

6 IBM WASFO Analysis and Optimization Tool

6.1 As-is WASFO Toolset for Analysis and Optimization

Previously WASFO was a single tool which included the functionalities of data analysis and optimization for server farms virtualization. The main functionalities are the following:

- Inventory analysis
- Workload analysis
- Optimal virtualization design
- Graphs generation

Although these functionalities are supposed to be used by IBM employees, however WASFO Tool included also the functionalities of data collection (inventory and workload data), hence the user has the ability to switch from the *Analysis and Optimization-Mode* to the *Data Collection-Mode* through the main menu of WASFO Tool, and vice versa. This is useless since the data collection functionalities are only used in customers' sites not in IBM. Moreover, all users could use the analysis and optimization functionalities, once they have WASFO Tool installed on their PC. Obviously, this was an inefficient and non-professional way to use WASFO for server farm's analysis and optimization. Thus, we had to separate the analysis and optimization functionalities in a new tool to-be called "IBM WASFO Analysis and Optimization Tool". Here below, you can see a screenshot of the old WASFO version for analysis and optimization mode. As you might observe the simple and limited GUI of the analysis and optimization mode.

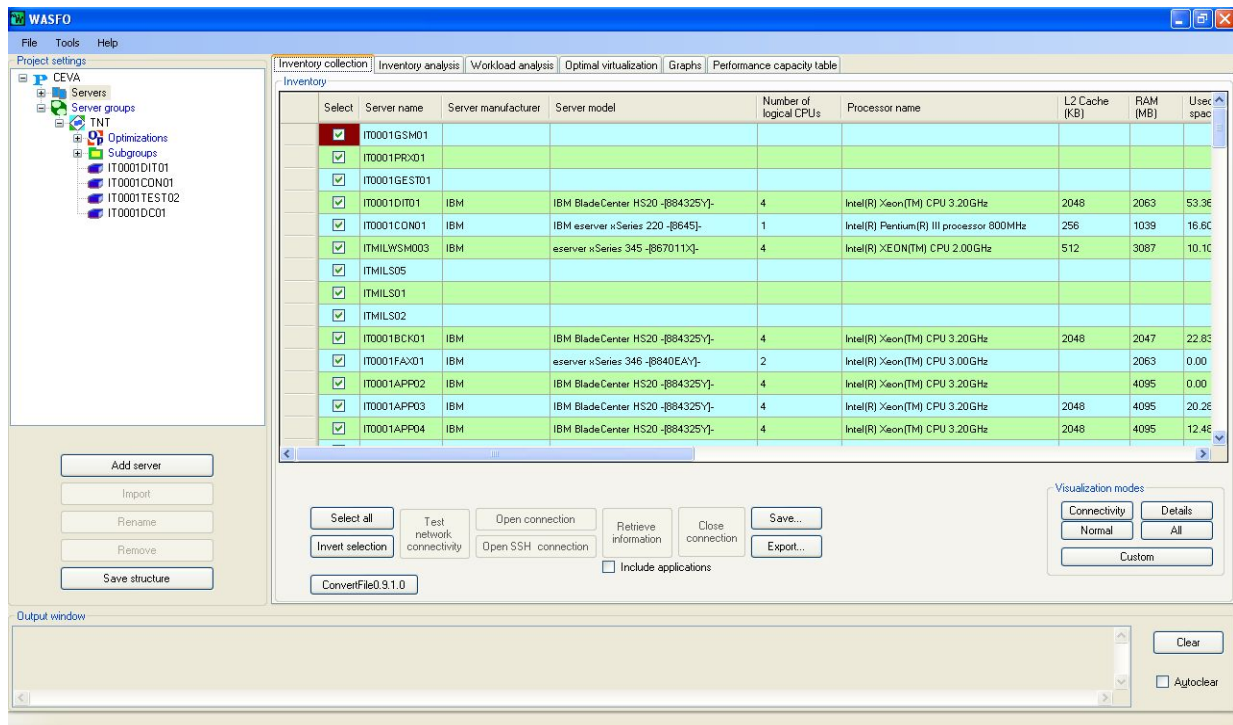


Figure 21 Main form of WASFO Analysis and Optimization Tool

6.2 To-be WASFO Analysis and Optimization Tool

After the analysis phase regarding the old version of WASFO, we decided to build a new tool called “IBM WASFO Analysis and Optimization Tool”, that will be responsible for analyzing inventory and workload data to solve the optimization problem by finding the minimum cost way of a virtualized server farm. WASFO Analysis and Optimization Tool currently provides the following functions:

- Inventory analysis
- Import of IDEAS International tables
- Workload analysis
- Optimal virtualization design
- Graphs and reports generation

The inventory analysis process is about finding automatically the performance capacity of the collected servers based on their data. The performance capacity means an estimate of the computational capability of a server. The analysis process depends on identifying the matched server in IDEAS International tables. WASFO Analysis and Optimization makes such a process much faster by means of algorithms that support the identification process.

The workload analysis process is concerned with the analysis of the workload data of collected servers. This process is complex because it passes through several sub-processes; Firstly the handling process of data holes (missing data), and secondly the normalizing process, and finally performing statistical analysis so that we consider that the collected data is just one instance of a stochastic process.

For the virtualization process, WASFO Team has implemented a mathematical model that is using Binary Programming. With such model, the tool using an Optimization Engine could compute the optimal way to virtualize a server farm. Herein optimal design means finding the minimum cost including costs related to servers’ purchase cost, power consumption, floor space utilization...etc.

According to WASFO requirements and specifications, the new tool has been designed and developed in a modular and optimized way, in order to guarantee that the analysis process and the optimization process are quick and efficient. Indeed, the main features of the current tool are as follows:

- Efficient matching algorithm; identifies the performance capacity of the collected servers
- Data analysis exploits the multi-threading technique to provide efficient analysis methods
- Powerful optimization; Identification of the minimum cost solution to the virtualization problem

Here below is the UML components diagram that shows the dependencies between the components and WASFO Analysis and Optimization Tool. Some of these components are shared libraries; hence they are being used by the other tool; WASFO Data Collector Tool as well. As you see, the dedicated components for the Analysis and Optimization Tool are as follows:

- IBM.WASFO.InventoryAnalysis
- IBM.WASFO.WorkloadAnalysis
- IBM.WASFO.Optimization

- IBM.WASFO.AOController

In addition, there is the *EXE* program of the tool; “*IBM.WASFO.AnalysisOptimizationGUI.exe*”. This program uses the controller “*IBM.WASFO.AOController*” to call any analysis and optimization methods. The controller acts as a facade, which receives requests from the GUI of the tool and then sends them to the proper handlers in the required libraries, after then, the controller responds to the user interface with the results. The specific components (mentioned above) of the tool depend on *IBM.WASFO.Core* library for any database operations or any needed basic functionalities. Indeed, the *AnalysisOptimizationGUI* represents the presentation layer of the tool, while *AOController*, *InventoryAnalysis*, *WorkloadAnalysis*, *Optimization* libraries and some other libraries represent the business layer of WASFO Analysis and Optimization Tool. From can figure 22, we can observe that most of the components are dependent on the library of *IBM.WASFO.Core*. Thus, the tool is a collection of combined components that form transparently an integrated tool. This is definitely so efficient and powerful since these components could be reused in other tools. Moreover, they can be updated and modified easily due to their high modularity. Detailed descriptions about these components will be given later in this chapter.

Component Dependencies of IBM WASFO OptimizationAnalysis Tool

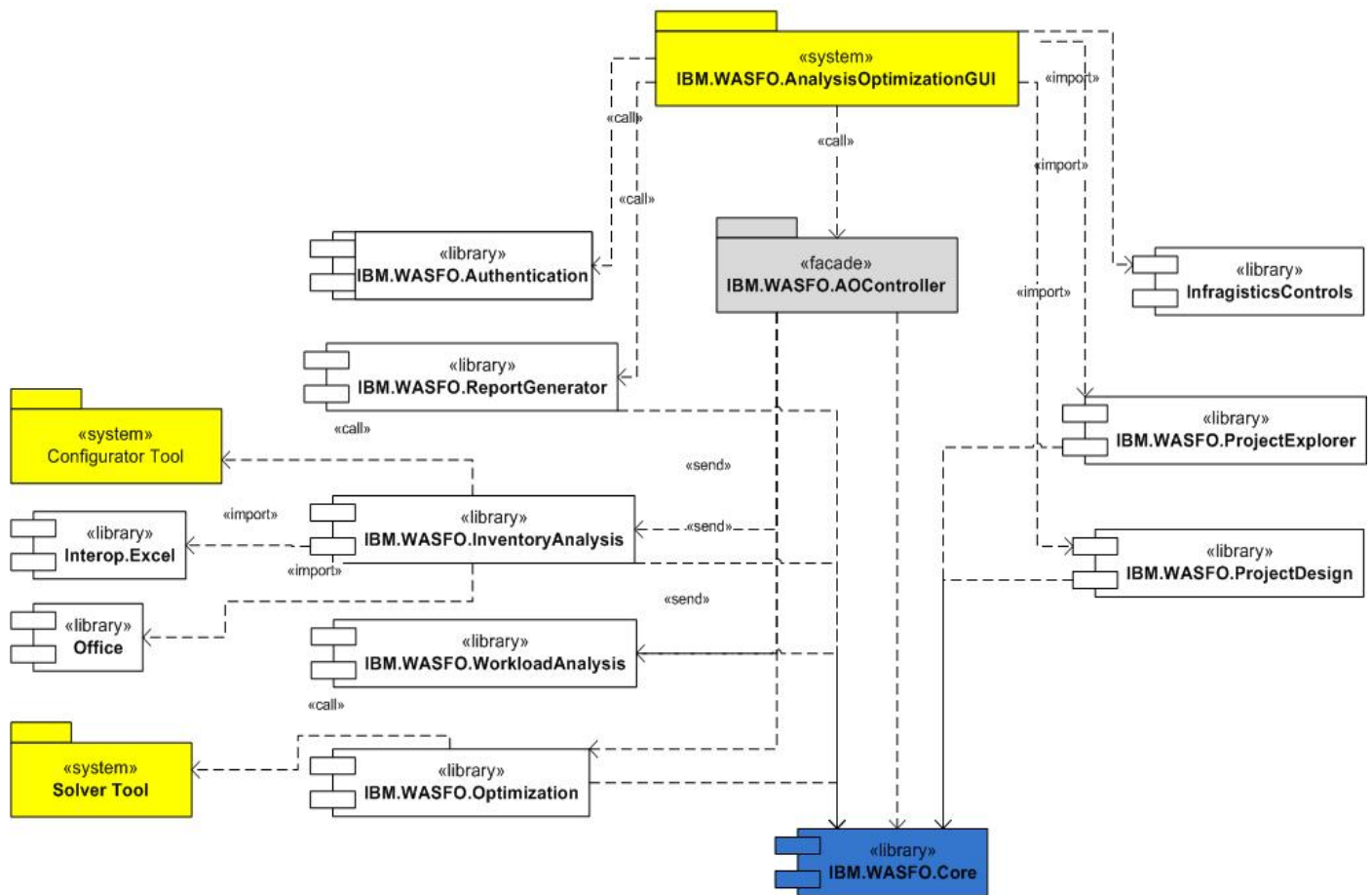


Figure 22 Component diagram of WASFO Analysis and Optimization Tool

6.3 Software Architecture

6.3.1 IBM.WASFO.InventoryAnalysis Component

This library is designed to perform all the functionalities related to the inventory analysis based on the collected inventory data from a server farm. I was involved in the design and the implementation of this library with another team member (Andrea Pagani).

WASFO Data Collector Tool collects data from hundreds of different servers, and hence we need to provide a mechanism to identify automatically the performance capacity of the collected servers. This is actually done through the *InventoryAnalysis* library by the help of the IDEAS International Sheet's data, which provides a spreadsheet that contains the performance capacity's estimate of thousands of server models. RPE and OLTP RPE are the used metrics for the computation of the performance capacity, and it is obtained by calculating the average results of five different benchmarks. In case of the unavailability of any of the required data used for the computations, extrapolations are used.

Hence, I designed and implemented a matching algorithm in the library to identify automatically the performance capacity. Starting from the collected server's data, the algorithm is able to identify in the IDEAS International spreadsheet the correct server model and configuration. Indeed, the matching algorithm exploits the IDEAS data and therefore, WASFO Analysis and Optimization Tool requires a license to use this data. This is one of the main reasons that only IBM employees use the tool. The matching algorithm will be discussed in details later in this chapter.

In figure 23, you will find the class diagram of the *InventoryAnalysis* library. As you can see in the diagram, there are several design patterns, which have been used to design the classes' structure. The used patterns are as follows:

1. Strategy pattern
2. Context pattern
3. State pattern

These design patterns are integrated altogether to build WASFO Matching algorithm. Hence, the strategy pattern represents the different strategies by means of algorithms (*MatchByMemory*, *MatchByFrequency* and *MatchByFrequencyMemory*), which are used to find the best match to the collected server out of the reference servers in the IDEAS spreadsheet. The context pattern represents the context of the matching process, where the matching algorithms are applied to find the ideal match by mixing and filtering the results of these algorithms. The state pattern represents the different cases (*SingleAlgorithmState*, *DoubleAlgorithmState* and *TripleAlgorithmState*) which, occur in the matching context during the matching process. Whenever the matching context switches the running algorithm to an alternative one, the matching state will change based on the number of the applied algorithms. This is very useful in identifying which results will be taken in order to be mixed, filtered and finally prioritized to find the best match. There is also the class "*MatchingHelper*", which acts as utility class for the "*MatchingContext*" and the "*MatchingAlgorithm*" classes. Beside the matching functionality, IBM.WASFO.InventoryAnalysis library has other important functionalities such as the following:

- Import of the IDEAS International spreadsheet automatically to the tool, and then stores the extracted data and information from the spreadsheet into WASFO database.
- Manages the Blade Center Chassis related to a collected server.
- Parses and extracts all the needed information from the configuration file used for configuring the collected servers inside a server farm.

In figure 24, you can find the .NET class diagram concerning the *InventoryAnalysis* library. It shows more details about the methods, properties, delegates and events of the classes. However, it does not show the design patterns that were used for designing the class diagram of the library.

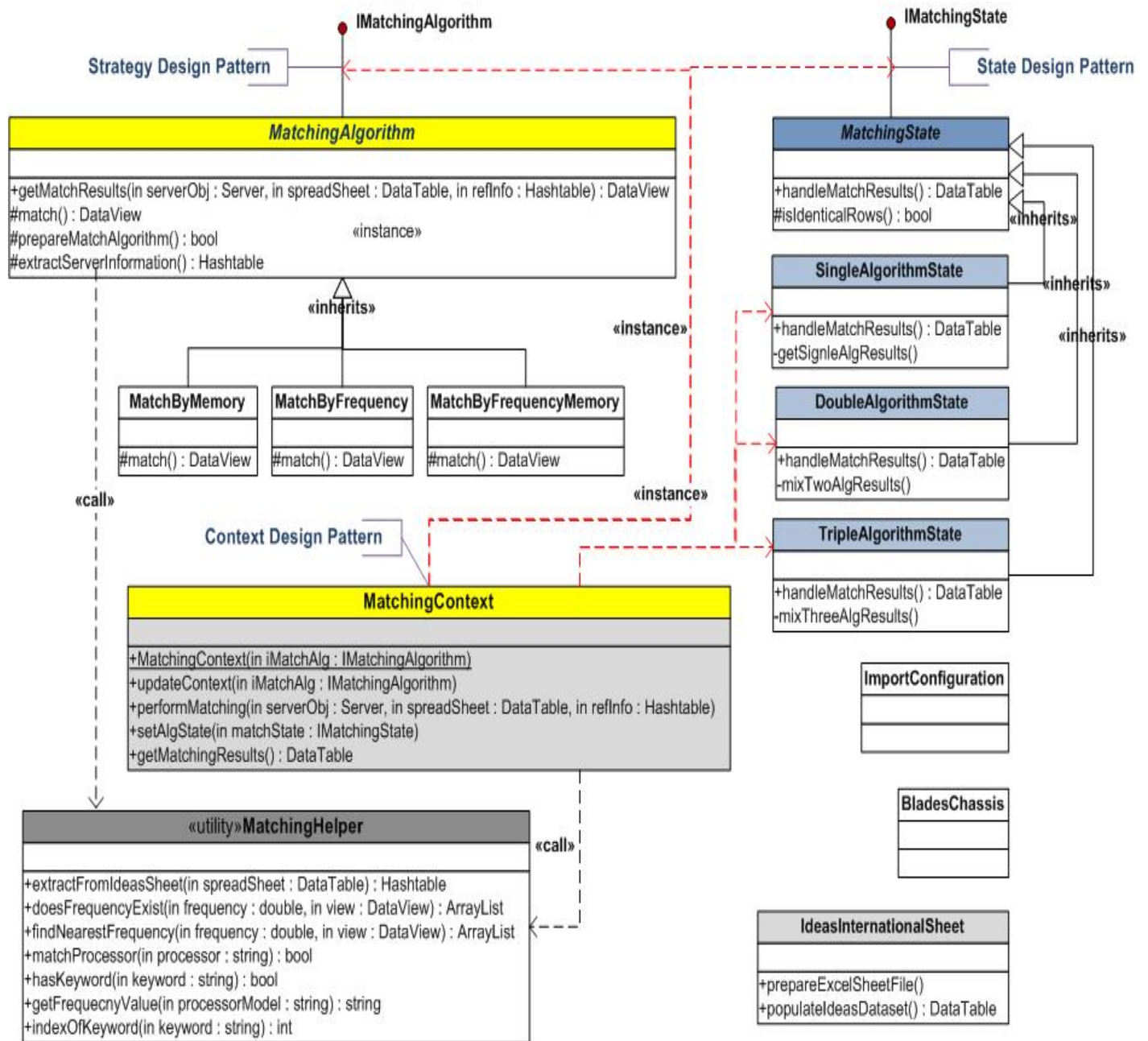


Figure 23 UML Class diagram of InventoryAnalysis Library

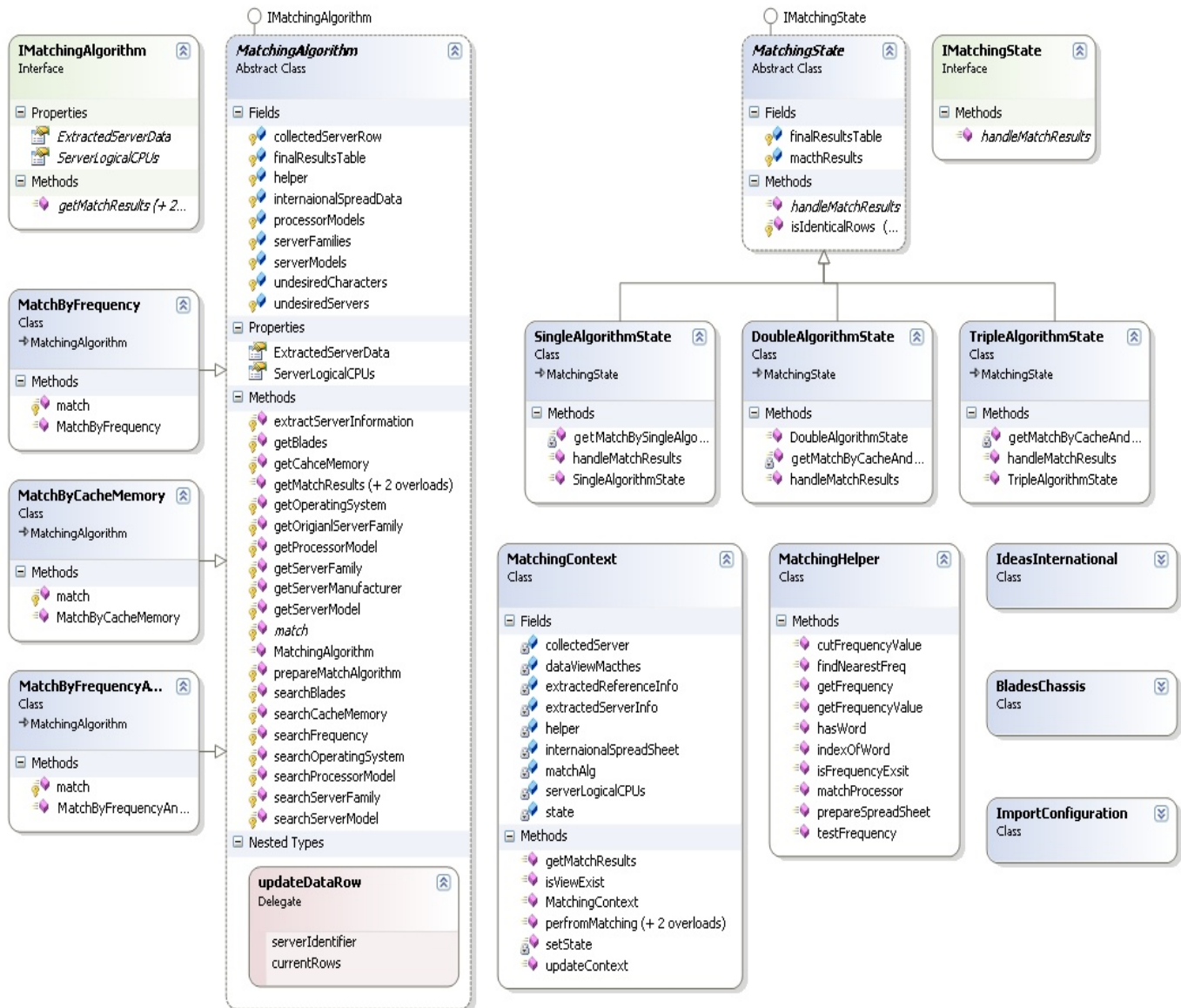


Figure 24 .NET Class diagram of InventoryAnalysis Library

6.3.2 IBM.WASFO.WorkloadAnalysis Component

This library is designed to perform all the functionalities related to the workload analysis based on the collected workload data from a server farm. It has been implemented by the project leader Dr. Mauro Gatti, and (Andrea Pagani 2009). Collected workload data from an existing server farm, usually contains some data holes “some missing data in the workload time series”. This could happen due to several reasons such as the monitored server has been turned off or a network disconnection has occurred to the server.

The optimization algorithms currently cannot support handling such holes; therefore we need to fill in these gaps in the workload data. For this problem, WASFO uses some interpolation algorithms to fill in these holes when possible. After filling these holes, the data are normalized.

After the normalization, we will be able to compare the collected workload data from one server with those collected data from another server. Indeed, the collected workload data can be absolute data or complex data. The absolute data (e.g., free memory) can be compared easily even without normalization. However, the complex data (e.g., processor utilization) cannot be compared the same as the absolute data, particularly if the the servers are different. For this reason we need the normalization process for the workload data. Hence, for the normalization, we take into our account that the collected servers have different performance capacity. Herein performance capacity means any metric that is used for measuring a server's capability according to some criterion.

As we can see, the normalization process is very important in solving the optimization problem. The last step is to pass the normalized data through a basic statistical analysis by computing the moving average and moving variance, thus the optimization does not depend directly on the raw data but on the statistics of this data.

6.3.3 IBM.WASFO.Optimization Component

This library is designed to perform all the functionalities and processes related to optimize the design of a virtualized server farm based on the analysis done for the collected data. It has been implemented by other members in WASFO team; (Tchango 2010) and (Andrea Pagani 2009). In fact, many mathematical models regarding the server farm's virtualization have been designed and implemented, since the birth of WASFO. These models are varying in their complexity, hence the more complex they are the longer computation times they take. Every model of these models depends on a specific target function. The target function describes the overall cost of a virtualized server farm, in order to better minimize this cost. Thus, the model will be able to identify a server farm that meets the required operational requirements at a minimum cost.

6.3.4 IBM.WASFO.AOController Component

This library is mainly a gateway between the business components (*InventoryAnalysis*, *WorkloadAnalysis* and *Optimization* libraries) and the Analysis and Optimization Tool (graphical user interface). As mentioned before, it acts as facade for the business functionalities regarding the data analysis and optimization. Besides that, it carries all the requests/responses from/to any business component used by the tool, and so it hides the complexity from the presentation layer. It is basically an interface layer between the presentation and business layers. I have designed the library using the Go4 design patterns in .NET language:

1. Facade pattern
2. Singleton pattern
3. Proxy pattern

The facade pattern-classes (*InventoryAnalysis*, *WorkloadAnalysis*, *Optimization* and *OptimalVirtualization*) act as interfaces between the AOController component and the referenced libraries "*InventoryAnalysis*, *WorkloadAnalysis*, *Optimization* and *OptimalVirtualization*". Hence, all the methods of the referenced libraries are provided and called in their corresponding

facade classes. For example, all methods in the InventoryAnalysis library concerning the inventory analysis’s functionalities are called in the InventoryAnalysis facade-class.

The proxy pattern-classes (*InvCollController*, *InvAnlController*, *WklAnlController* and *OptimizationController*) act as proxies for the facade classes in front of WASFO Analysis and Optimization Tool. In other words, they represent the interface controllers of the AOController component, whereby each controller is responsible for its corresponding facade-class. Indeed, each proxy class implements a corresponding Interface (*IInventoryAnalysis*, *IWorkloadAnalysis*, *IOptimization* and *IOptimalVirtualization*) to its facade class. This interface represents a contract between the facade and its proxy class, hence we guarantee the integrity between the classes.

The singleton pattern is built inside each controller to guarantee that only one instance will be created for every controller to provide more efficiency in the performance. Thus, as you can see the AOController component has a double-sided interface; interface between the "AOController" and the "Referenced libraries", and another interface between the "AOController" and "WASFO Analysis and Optimization Tool". In figure 25, the class diagram of the library shows the classes and their corresponding design patterns. It shows also how the *AnalysisOptimizationGUI* calls the controllers to request business functionalities. As you can see, there is also a main controller which have the common functions needed for any sub-controller. In figure 26, the .NET Class diagram shows in details the methods and the properties of the classes.

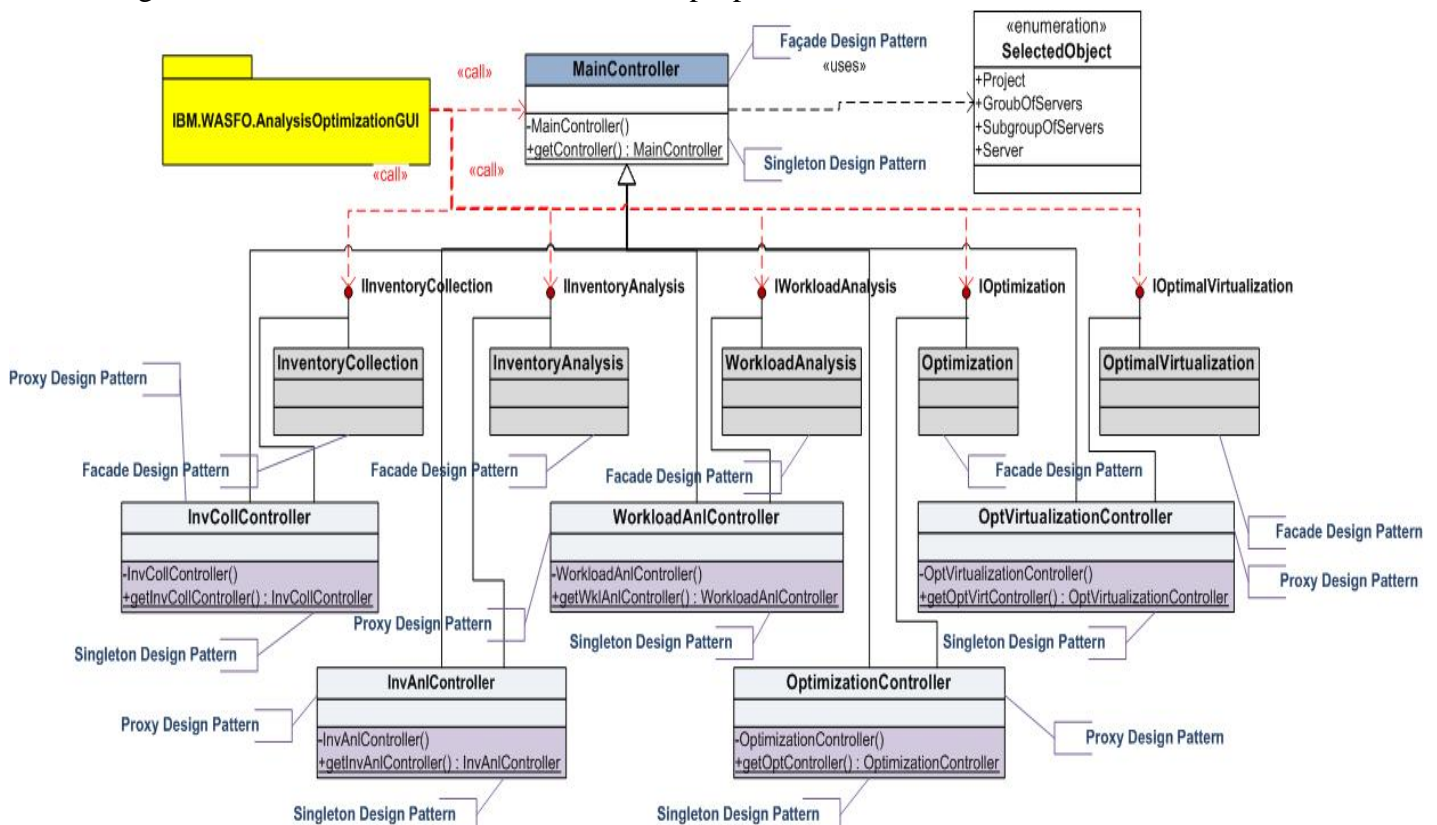


Figure 25 UML Class diagram of AOController library

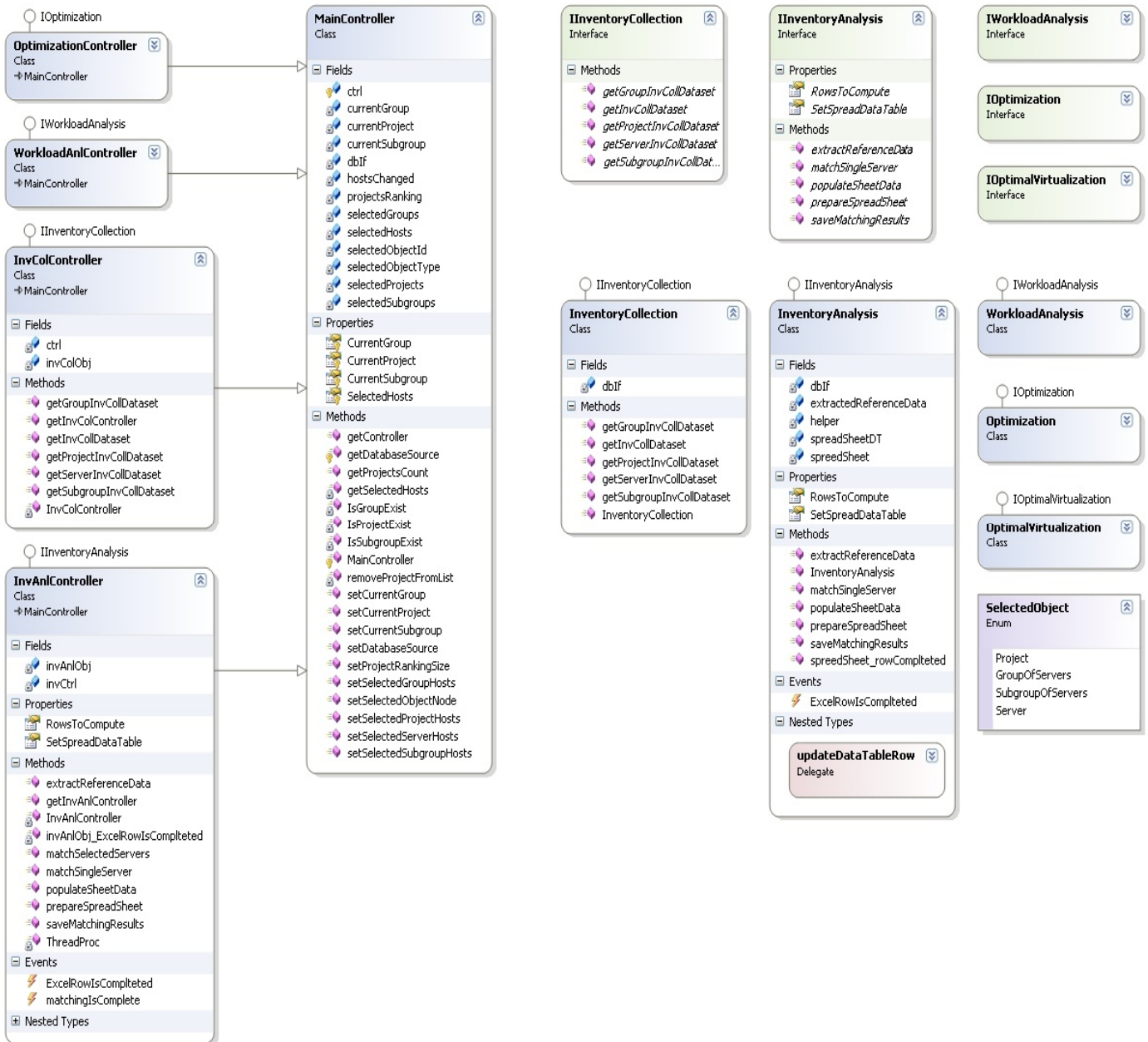


Figure 26 .NET Class diagram of AOController library

6.3.5 IBM.WASFO.AnalysisOptimizationGUI.exe Program

The “*IBM.WASFO.AnalysisOptimizationGUI.exe*” is the executable program of WASFO Analysis and Optimization Tool. It consists of the main class (the entry point of the tool), GUI classes, visual forms and user controls (*ProjectDesign*, *ProjectExplorer* and *ReceiveControl* components), which represent the graphical user interface of the tool. In addition, the tool is composed of a collection of libraries which represent the business layer (data analysis and data optimization). The data layer of the tool is represented by WASFO database. The database and the *EXE* program “*IBM.WASFO.AnalysisOptimizationGUI.exe*” are both installed using the package setup that is called “*IBM.WASFO.AnalysisOptimizationSetup*”. In order to make the

tool portable and easy to use, I created an application configuration file called “*App.config*” in XML format to be installed with the tool. This file carries all the configurations needed for the tool to run successfully such as the database’s path, *IDEAS* spreadsheet’s path, workload files’ path, shipment web services’ reference...etc. Indeed, this file can be modified in the run-time, in case if the user needs to change some configurations.

Here below, is the class diagram of the Analysis and Optimization Tool. As you see, the class structure is clear, whereby it contains only visual forms, their related classes and some utility classes. This is due to the fact that the business layer is built as business components, which are used by the tool as reference libraries. The main form of the tool is the “MainForm” class, which starts after the appearance of the welcome screen. The caller to the main form is the main entry of the tool; the class “*Program*” that contains the “*Main*” method of the program. There is also the class “*Resources*” which manages all the resources needed for the tool. In section **B** at Appendix, you will find a screenshot of the graphical user interface of WASFO Analysis and Optimization Tool.



Figure 27 .Net class diagram of AnalysisOptimizationGUI.exe

6.4 Import Process of IDEAS International Spreadsheet

As mentioned before, the *InventoryAnalysis* component provides the possibility to import the IDEAS excel sheet's data into WASFO Analysis and Optimization Tool. In the figure below, you can see an overview about the importing process through the tool. The process begins when the user selects to import a spreadsheet; a file dialog prompts the user to choose the desired IDEAS excel file. Then, the program verifies that the file exists and if so, it stores the file's path in the *App.config* XML file of the tool. Then, the tool calls a background thread for the importing process, which consequently uses the *InvAnlController* instance to call the import method of the class *IdeasInternationalSheet* through the *InventoryAnalysis* component.

In figure 29, you can see the activity diagram of the importing process for the IDEAS spreadsheet. The diagram shows the performed activities and the interactions between the *InventoryAnalysis* component and the tool. Indeed, the import process is considered as an ETL (Extraction, Transformation and Loading) process, where the desired data is extracted from the IDEAS spreadsheet for further processing. Then the data is mapped from the spreadsheet format (Excel sheet) into the Data Table format (database table) after performing data transformation by means of removing unnecessary data and adding extracted information out of the given data (e.g. extracting the frequency value from a *ProcessorModel* or the server family from a *ServerModel*). After that, the data is finally loaded into the database in the table "*ReferenceServer*".

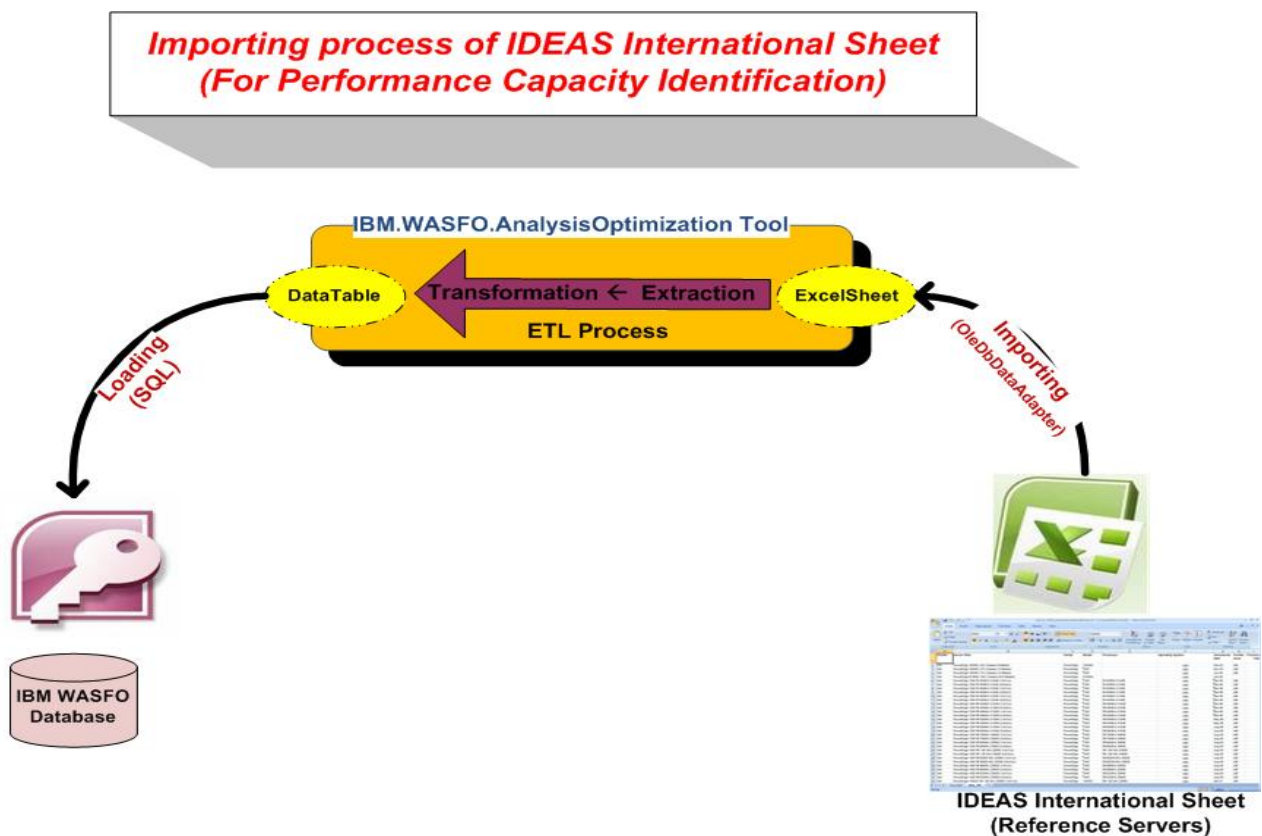


Figure 28 Diagram showing how to import IDEAS Sheet into IBM WASFO Database

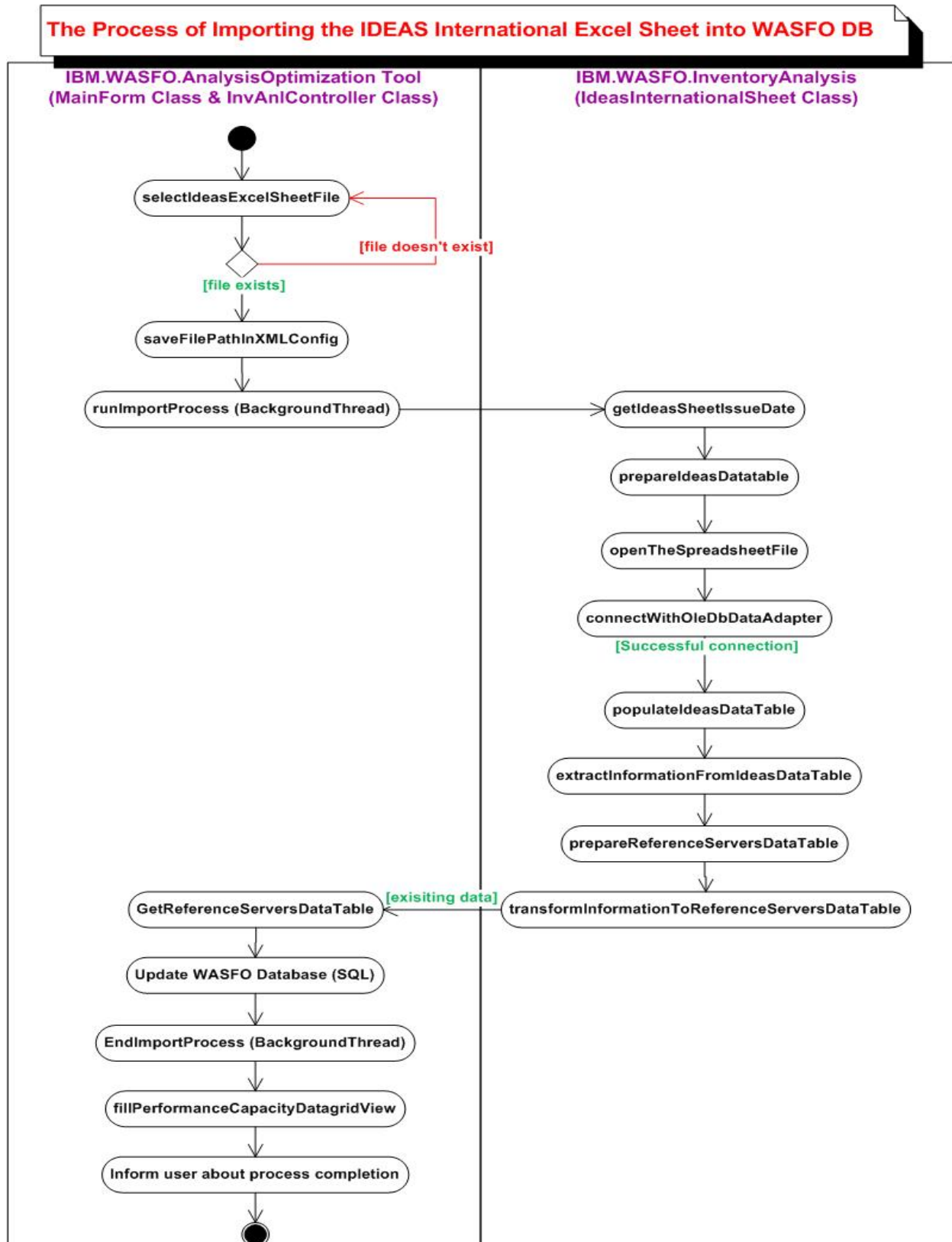


Figure 29 Activity diagram of ETL Process for IDEAS International Sheet

6.5 WASFO Matching Algorithm

After we collect the inventory data from a server farm. The performance capacity of the collected servers should be identified, and hence we need to match these servers with the corresponding ones out of the reference servers in the IDEAS data table. In order to provide an automatic mechanism for such identification process, we need a matching algorithm that is based on the collected data of a server, it finds the best match with those of the reference servers. Thus, we can get better results while optimizing the virtualization design of the server farm. The server's collected data is as follows:

- Server manufacturer
- Server model
- Processor model

This data is necessary for identifying the collected server, otherwise the server will be considered unknown, and so the matching algorithm will ignore it from the collected servers' list. This is due to the fact that if there is no information about the server (the manufacturer or the model), it is impossible to match unknownserver with one of the reference servers. However, some times the collected data misses the server manufacturer or the server model so in this case the algorithm tries to find the best match based on the processor model and the available data (server manufacturer or server model), but if both are missing it is totally impossible.

On the other hand, the information we get regarding the reference servers after importing the IDEAS international spreadsheet into the database in the table "*ReferenceServer*" is more broad:

- Server manufacturer
- Server family
- Server model
- Processor model
- Processor speed (Frequency in HZ)
- Cache memory (KB)

The reason for this is that the spreadsheet stores the servers' data in columns format, so it was easy for the *InventoryAnalysis* component to either import this data as it is (server manufacturer, server family, server model and processor model), or extract and transform the required information out of it (processor frequency and cache memory). Thus, this gives us more flexibility in identifying correctly the best match to a collected server.

Until now the matching problem looks simple, however we faced a major problem in matching the collected data with the reference servers; The problem is that the IDEAS data table has a fixed format and order of the data. But on the other hand, the format of the collected data is totally random and unknown. In other words, there is no specific syntax or order of the data we collect from the servers due to the fact that the servers are produced by different vendors, and

moreover they run different operating systems. In addition, WASFO Data Collector Tool uses different data collection methods to collect data from these servers based on the server's type and its running operating system. Thus, all these factors result in having different data formats of the collected data. For example, you might find that the collected data from a known server is as follows;

- Server manufacturer → “hewlett-packard”.
- Server model → “proLiant- ML350 G pentium3 11300 mhz 512 kb”.

Now, if we match this server with one of those corresponding servers in the IDEAS data table (*ReferenceServer*), we would find that the matched server will have this fixed format

“HP ProLiant ML350 G2 PIII 1.13GHz 512KB (1ch/1co)”

As you can see, in order to match both servers, we need a specific algorithm for such purpose. Using only a string matching algorithm is insufficient. We still need more advanced criterion to be able to map the collected data to the reference data successfully.

In figure 30, you can see a visual view about the taken steps of WASFO Matching algorithm. The procedures that the algorithm follows are as following:

1. Extract information from the “ReferenceServer” data table;
 - List of server families, list of server models and list of processor models.
2. Extract keywords from the collected server, using the previously extracted lists;
 - Server's (vendor, family, model) and processor's (model, speed, cache memory)
3. Load the data table of all the reference servers in order for the matching algorithms to filter it with the help of the found keywords;
4. Call the first algorithm “MatchByFrequency” to filter the reference servers using the extracted keywords in a certain sequence (the cache memory keyword is excluded);
5. Call the second algorithm “MatchByMemory” to filter the reference servers using the extracted keywords in a certain sequence (the processor frequency keyword is excluded);
6. Call the third algorithm “MatchByMemoryAndFrequency” to filter the reference servers using the extracted keywords in a certain sequence (all the keywords are used);
7. Then, the three dataviews (above results) are mixed in one datatable, and then the results are ordered based on the number of similarities between the 3 dataviews. Hence the best matches, are the ones that got high rank. Herein, matching rank means the number of similarities between the 3 dataviews. For example, rank 3 means that a reference server is ideally found in the 3 dataviews. Thus, the servers with the lowest ranks will be eliminated from the results, and the servers with the highest ranks will be shown to the user. In case that there are no matches between the 3 dataviews, all the datatable's results will be returned to the user to have the possibility to select the best match manually;
8. Then, the final results will be published into the inventory analysis's gridview through the user interface of WASFO Analysis and Optimization Tool. Finally, the save option will be activated to allow the user to store the data results in WASFO database (Table InventoryAnalysis).

In figure 31, you can see the activity diagram of the matching algorithm. It shows the performed activities and the interactions between the *InventoryAnalysis* library and the matching classes (*MatchingContext*, *MatchingAlgorithm*, *MatchingHelper* and *MatchingState*) in order to match a collected server with its corresponding reference server. The diagram describes in details all the matching activities starting from the user’s request to match a single server or a list of servers.

In figure 32, you can see the sequence diagram regarding the steps taken by the different matching algorithms for identifying a collected server during the inventory analysis process. The diagram shows the procedures of each matching algorithm. In fact, the three matching algorithms inherit the common matching functionalities from the parent algorithm, which mainly extracts information from the IDEAS data table “*ReferenceServer*”, and also extracts the keywords from the collected data server. Hence, the extraction process is performed once by the parent matching algorithm, before the sub-matching algorithms start to run. Then, each algorithm performs filtration steps based on its strategy, and at the end returns a dataview which represents the matching results.

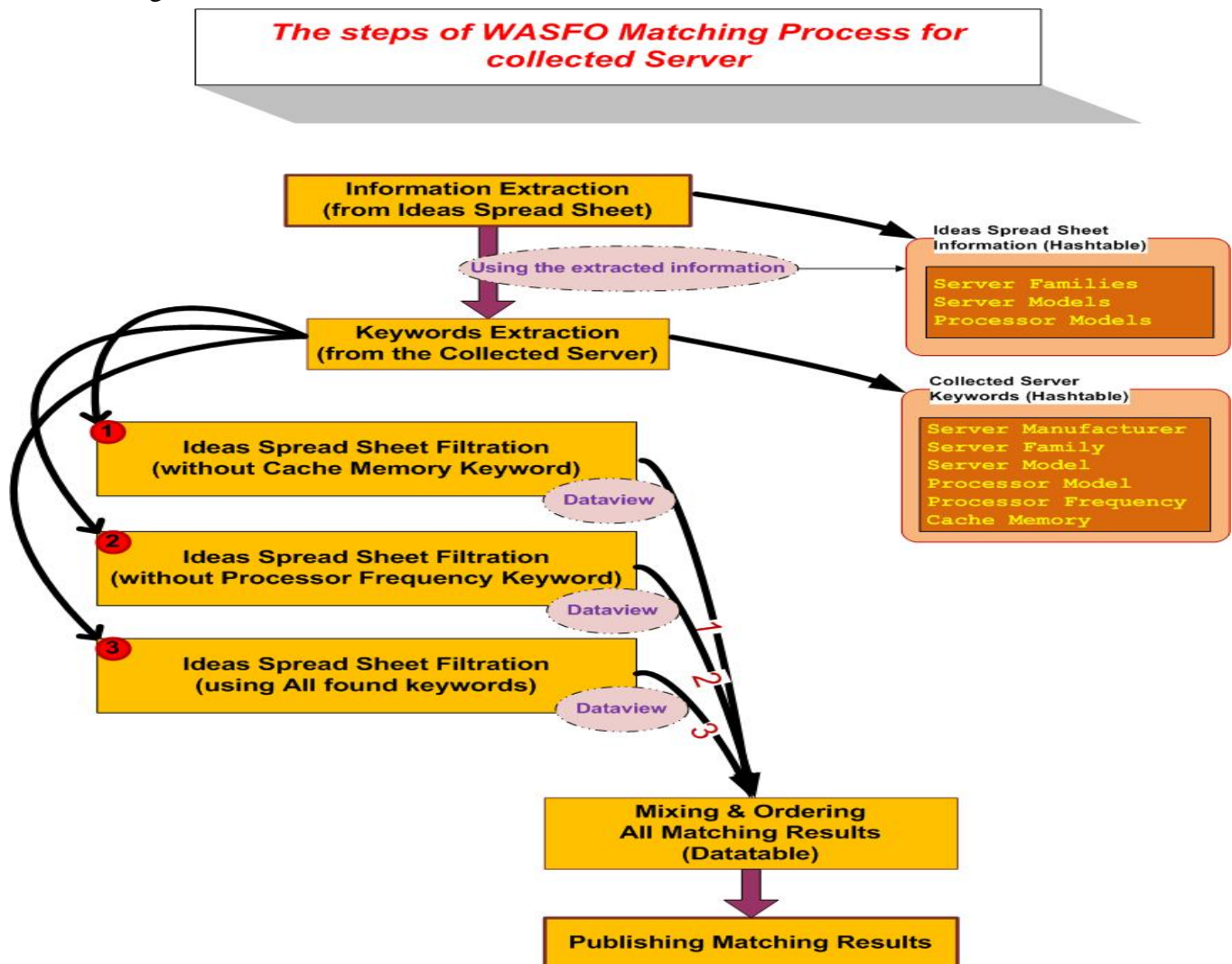


Figure 30 Procedures of WASFO Matching Algorithm

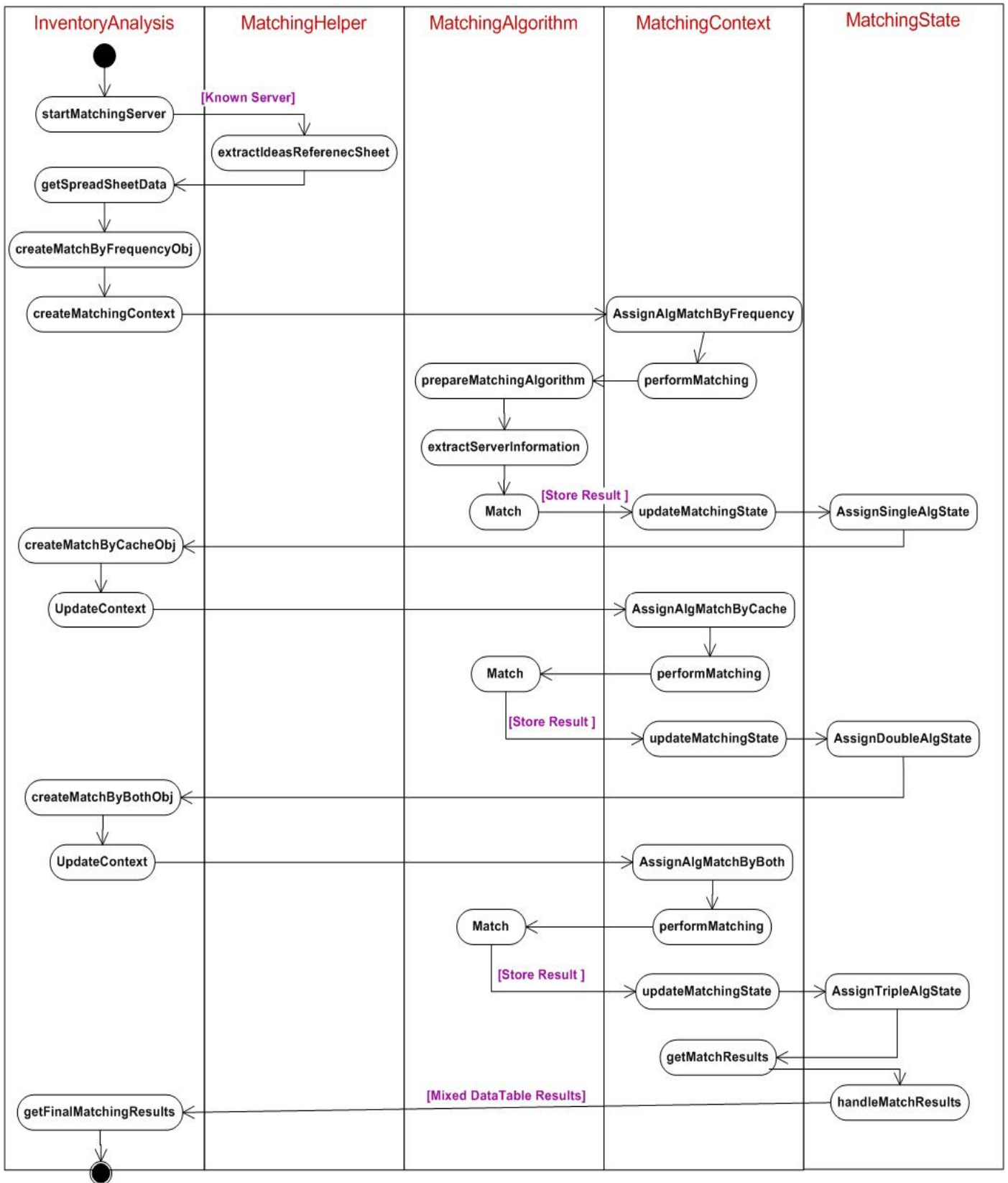


Figure 31 Activity diagram of WASFO Matching Algorithm

Sequence Diagram of Matching Algorithm

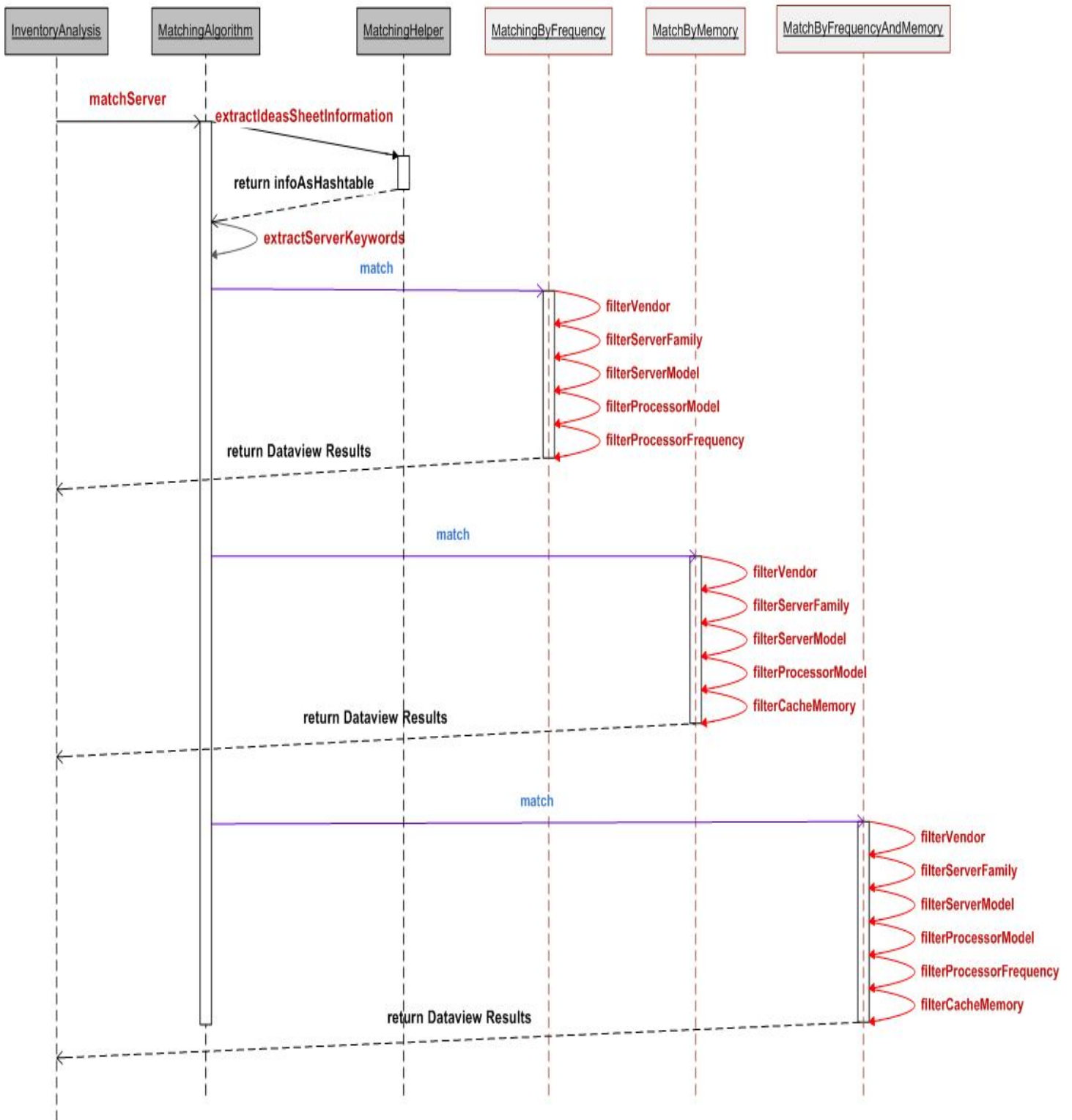


Figure 32 Sequence diagram of WASFO Matching Algorithm

6.6 Experimental Analysis for the Matching Algorithm

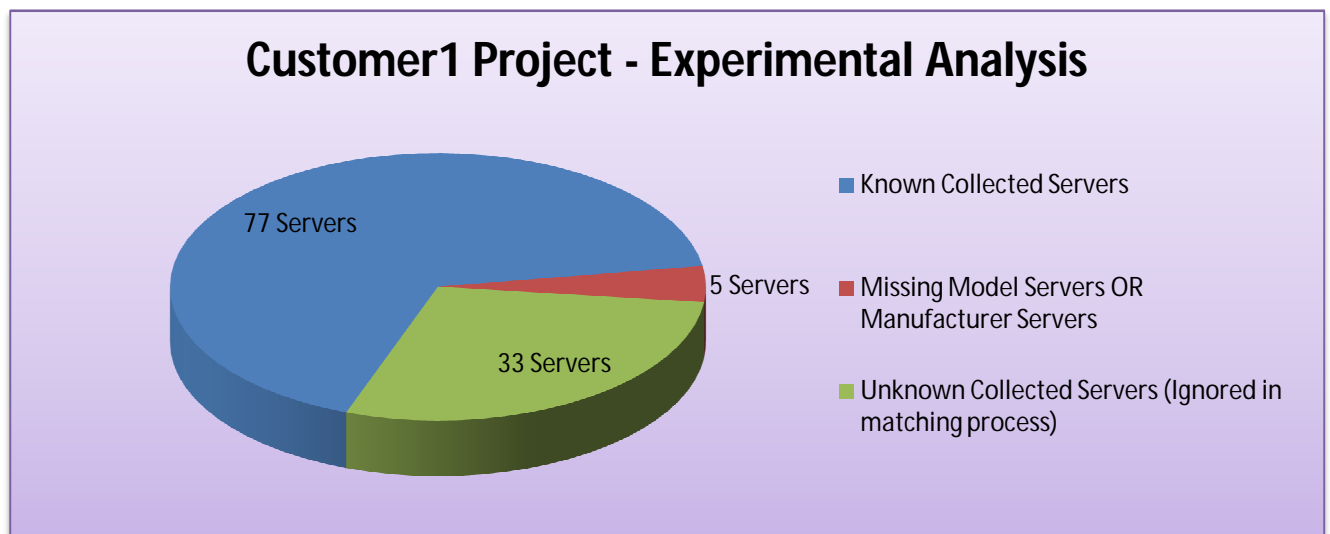
In order to verify the execution performance, correctness and efficiency of the new matching algorithm in WASFO Analysis and Optimization Tool, I had to do a comparison between the old and the new matching algorithms by means of a deep experimental analysis based on real data. The experimental analysis has been performed on previously collected data from real projects of two IBM customers. Due to the confidentiality of these projects, only the summery data of the performed analysis will be mentioned. Hence, the first customer will be called “Customer1” and the second one will be called “Customer2”.

6.6.1 Customer1 project

This project is of a small scale, where the data was collected from a server farm that is composed of 115 servers. The data was collected by the old version of WASFO Tool. Table 1 shows, that the collected servers are classified between known and unknown servers’ data. The known servers can also be classified into 100% known servers’ data or known servers with some missing data (missing server’s model or missing server’s manufacturer). In graph 1, you can see a 3D graph that shows a visual statistics regarding the collected servers’ data.

Data Collection of Customer1 Project	
<i>Total Servers</i>	<i>115 Servers</i>
Known Servers	77 Servers
Unknown Servers	33 Servers
Known Servers (missing server model/manufacturer)	5 Servers

Table 1 Data Collection of Customer1 project

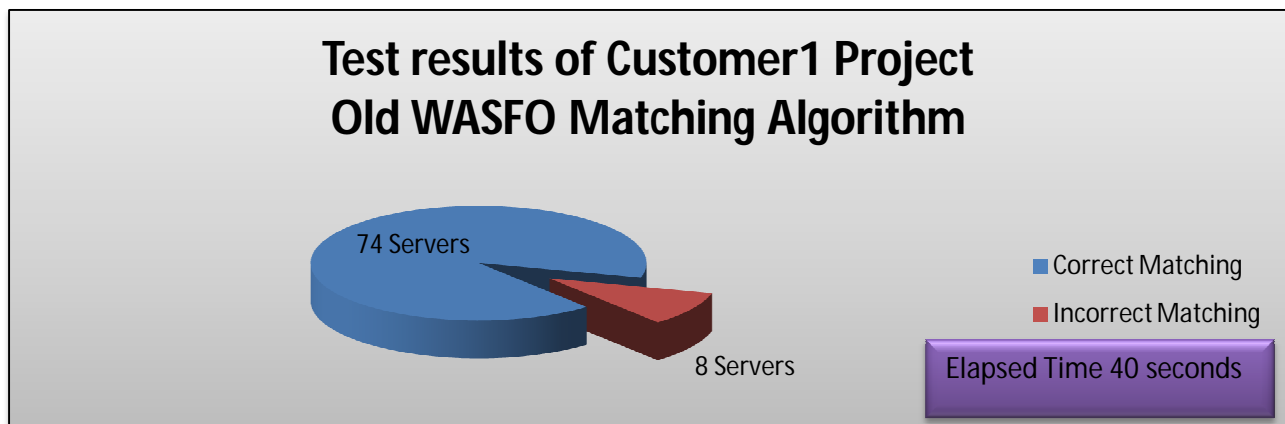


Graph 1 3D View of Customer1 project’s data

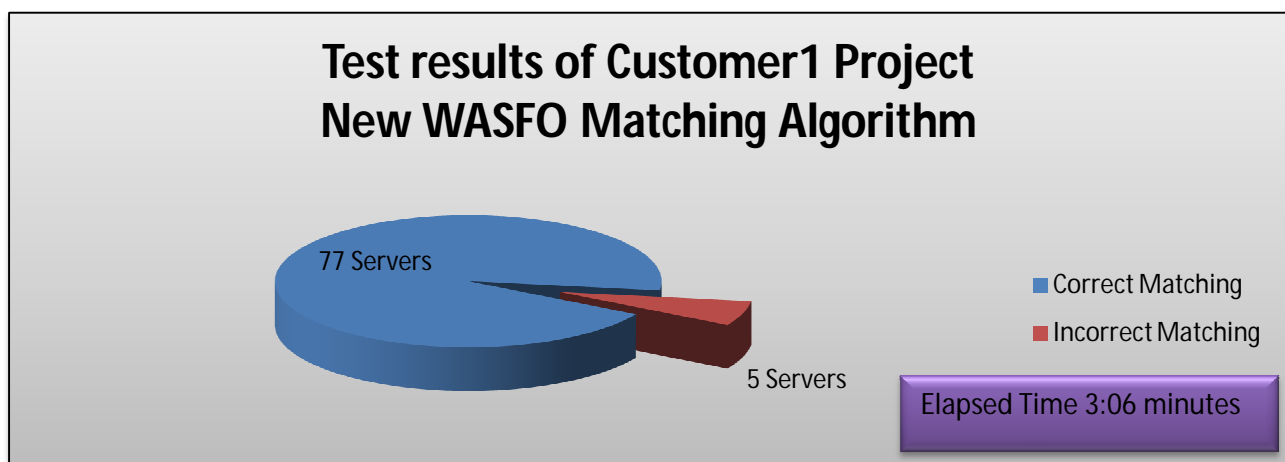
In table 2, you can find the comparison of the analysis results between the old and the new algorithms. As you can see, the new algorithm achieved better results than the old one by having a higher number of correctly matched servers, and by having less number of incorrect matched servers. In addition, if we look carefully at the data, we find that incorrect matched servers regarding the new matching algorithm are the servers whose server model or/and server manufacturer is/are missing! This indicates that the new algorithm almost achieved 100% success. Indeed, the results of both algorithms are generally excellent because the collected servers' data is of a good quality, which consequently helped in better matching the servers. In graphs 2 and 3, you can see better view of the analysis's results of both algorithms.

Analysis Results of the algorithms for Customer1				
	Ignored Servers	Correct Matching	Incorrect Matching	Elapsed Time
<u>Old Matching Algorithm</u>	33	74	8	40 Seconds
<u>New Matching Algorithm</u>	33	77	5	03.06 Minutes

Table 2 Analysis results of both algorithms for Customer1 Project



Graph 2 Results of the old WASFO Matching Algorithm – Customer1 Project



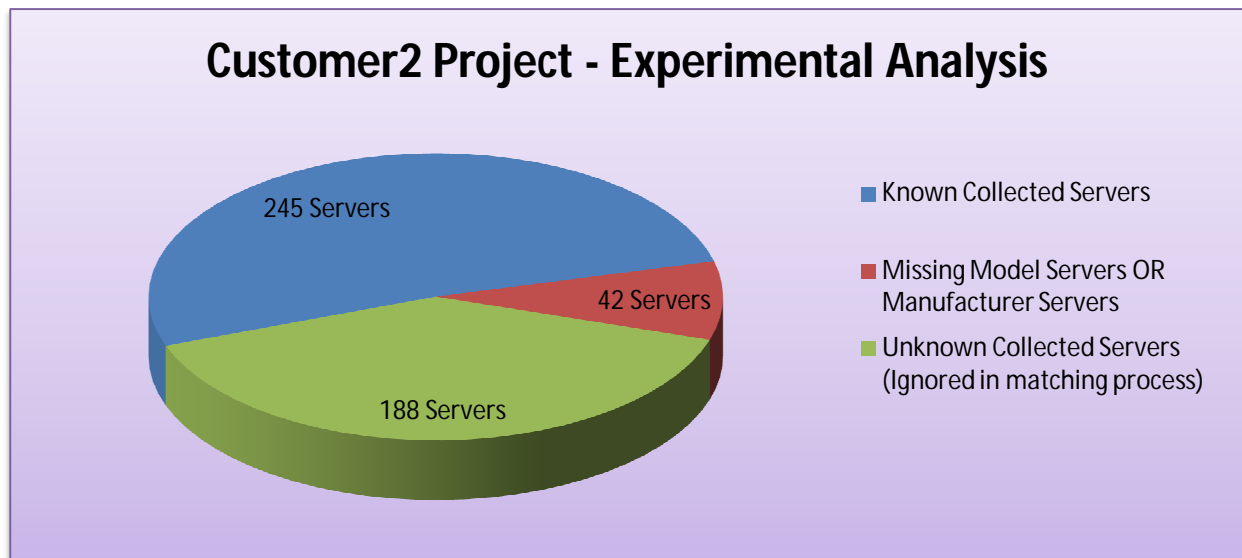
Graph 3 Results of the new WASFO Matching Algorithm – Customer1 Project

6.6.2 Customer2 project

This project is of a medium scale, where the data was collected from a server farm that is composed of 475 servers. The data was collected by the old version of WASFO Tool. As you can see in table 3, the collected servers are classified between known and unknown servers' data. The known servers can also be classified into 100% known servers' data or known servers with some missing data (missing server's model or missing server's manufacturer). In graph 4, a 3D graph shows a visual statistics regarding the collected servers' data.

Data Collection of Customer2 Project	
<u>Total Servers</u>	<u>475 Servers</u>
Known Servers	245 Servers
Unknown Servers	188 Servers
Known Servers (missing server model/manufacturer)	42 Servers

Table 3 Data Collection of Customer2 Project



Graph 4 3D View of Customer2 project's Data

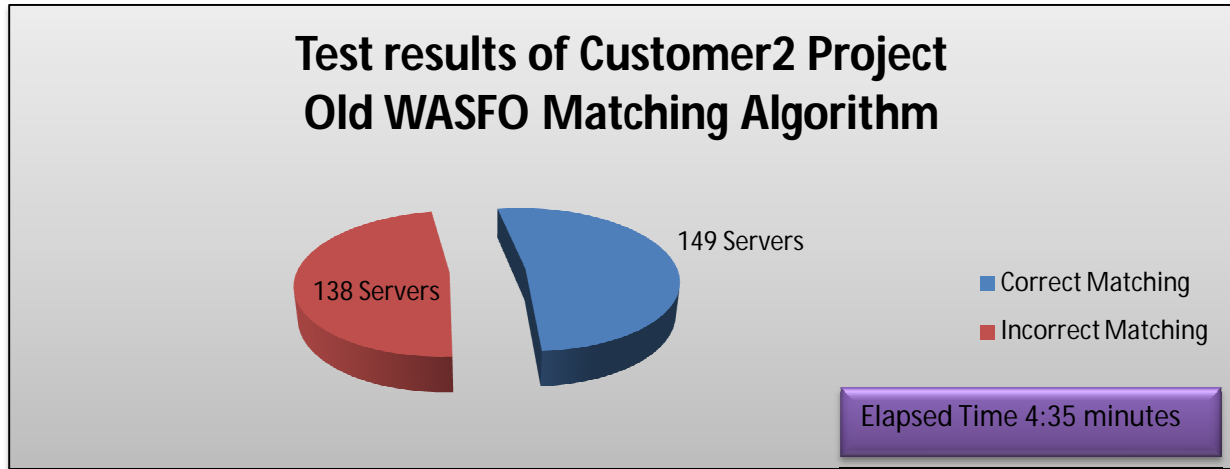
In the table below, you can find the comparison of the analysis results between the old and the new algorithms. As you can see, the new algorithm achieved significantly better results than the old one by having a higher number of correctly matched servers, and by having less number of incorrect matched servers. Moreover, by comparing the data results between both algorithms, we find that in case of the old matching algorithm, the percentage of incorrect matched servers (138) to the correct matched servers (149) is nearly equal to 50 %. This means that almost half of the well-known collected servers are not identified correctly. Hence, this is certainly inefficient and unacceptable.

On the other hand, if we look at the results of the new algorithm, we find that incorrect matched servers are the servers whose server model or/and server manufacturer is/are missing! This

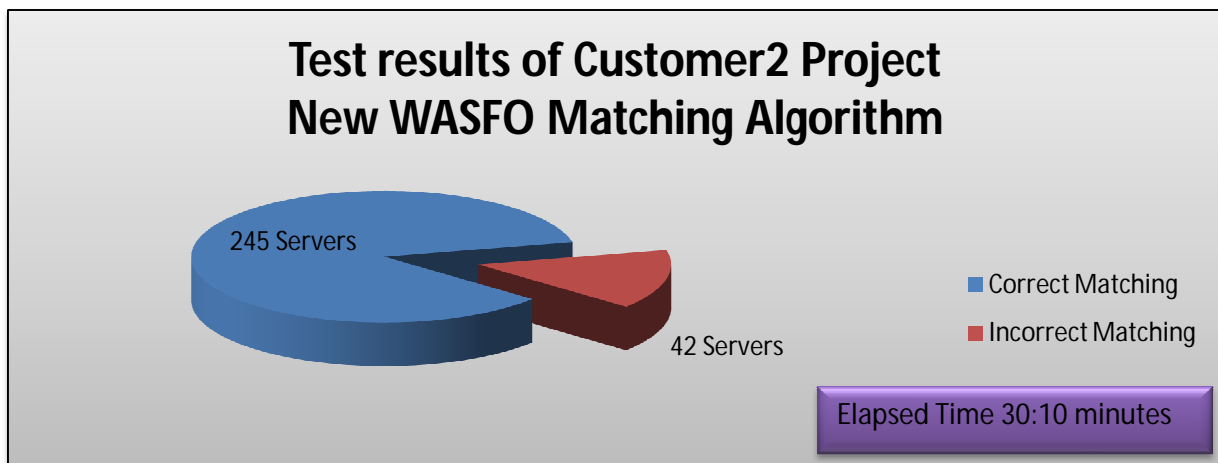
indicates that the new algorithm achieved 100% success in matching the known servers. In graphs 5 and 6, you can see better view of the analysis’s results of both algorithms.

Analysis Results of the algorithms for Customer2				
	Ignored Servers	Correct Matching	Incorrect Matching	Elapsed Time
<u>Old Matching Algorithm</u>	188	149	138	04.35 Minutes
<u>New Matching Algorithm</u>	188	245	42	30.10 Minutes

Table 4 Analysis results of both algorithms for Customer2 Project



Graph 5 Results of the old WASFO Matching Algorithm – Customer2 Project



Graph 6 Results of the new WASFO Matching Algorithm – Customer2 Project

As a conclusion, we realize that the new algorithm achieved better results than the old one. However, if we look at the elapsed time used for each algorithm, we notice that the old algorithm is quicker in computations than the new one. Nevertheless, this is due to the fact that, it has very simple string algorithm, which is not sufficient in matching the collected servers.

7 IBM WASFO Shipment Services

7.1 WASFO Shipment Web Services

It represents the middleware layer of WASFO Solution. Indeed, it acts as middleware software between WASFO Tools, in order to ship WASFO projects (database and project files) from WASFO Data Collector Tool to WASFO Analysis and Optimization Tool. The shipment process is performed through specific web services for importing and exporting files of large size in an optimized way. Hereunder, is the component diagram of WASFO middleware and the corresponding WASFO Shipment Web service's client of each WASFO Tool. The web services have been implemented using the WSE 3.0²⁹ for Microsoft® .NET. WSE 3.0 enables developers to build secure Web services based on the latest Web services protocol specifications. Moreover, it provides a simpler method for sending binary data in small chunks over HTTP web services.

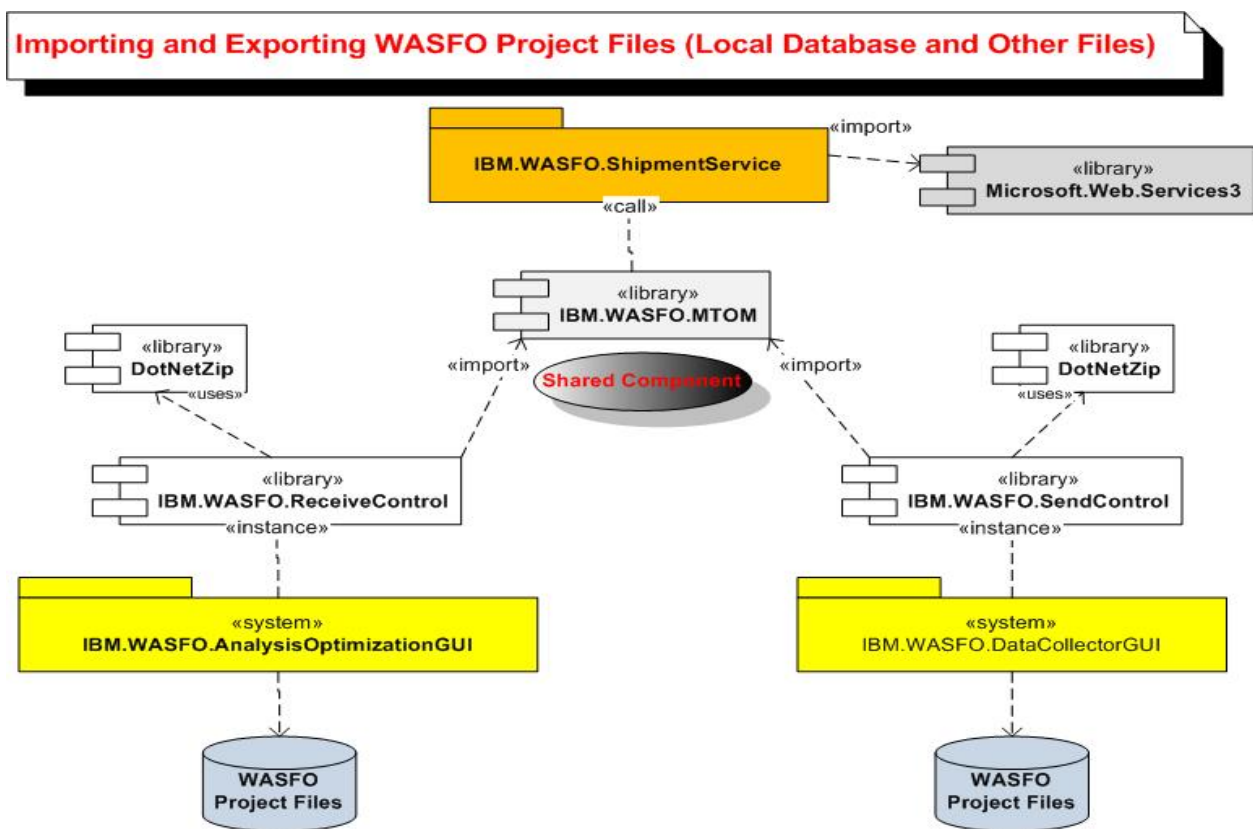


Figure 33 Component diagram of WASFO Shipment Web services

7.2 Why WES 3.0?

In order to send a large file across a web service under .NET platform, I had to understand how to do that, particularly how the web service call, Internet Information Service and .NET can all

²⁹ WSE 3.0; Web Services Enhancements 3.0

fit together. We might think to send a file in one single array of bytes as an input parameter to a web service call. Then this call will be sent to the web server as a single request. This looks simple; however, it is totally inefficient in case if the file's size is beyond the configured *MaxRequestLength* of the application or if the request results caused an IIS timeout. Moreover, there is no way to provide a file's transfer feedback to the graphical user interface of the application, because there is no indication about the progress of the transfer until it is either completed or failed. We also could think about transmitting files via web services using Direct Internet Message Encapsulation-*DIME*. It is a good approach however it has a significant problem that the binary contents of the messages are sent outside the SOAP-Envelope of the XML messages. This means although the messages are secured, their DIME attachments might not be secured.

An optimal solution for transmitting a file, notably one of large size, via web services is using WSE 3.0, which provides the ability to send large amounts of binary data efficiently and securely via the MTOM³⁰ specification (W3C³¹ SOAP). MTOM reduces the size of messages on the wire helping in scenarios of low bandwidth. Moreover, WSE 3.0 automatically handles the binary encoding of the transmitted data inside the web service's message when MTOM is turned-on in the client and the server applications. The most interesting fact of MTOM is the way in which it encodes the binary data; it sends the binary data in its original binary form, without any increase in the size of the data because of its text's encoding. Normally any binary data in the SOAP message is encoded as text using the Base64³² encoding, which results in increasing the size of the binary data by 33%. However, with MTOM this never happen, and so this reflects how powerful and efficient WSE is.

Indeed, MTOM sends the file as chunks one-by-one and appends them to the file on the server. MTOM fully complies with the other Web services specifications like WS-Security, so the entire message is secured. In figure 34 below, you can see the class diagram of WASFO Shipment Web Services. IBM.WASFO.ShipmentService is called in our context Shipment Web services. These web services are built in the class *Shipment*, and they are as the following:

- **SendChunk**: uploads a file into the server in the form of chunks.
- **ReceiveChunk**: downloads a file from the server in the form of chunks.
- **GetFileSize**: calculates the size of a file in bytes.
- **GetListFiles**: retrieves a list of files inside a folder.
- **GetMaxRequestLength**: one useful and important feature is that MTOM provides the *MaxRequestLength* setting on the server. Thus, this web service enables the client to stay within acceptable request sizes on the server during the transfer process. Thus, the overall result is a self-controlled file transfer that will adapt to changing network conditions during the transfer.

³⁰ MTOM; Message Transmission Optimization Mechanism

³¹ W3C; The World Wide Web Consortium

³² Base64; It is a generic term for any number of similar encoding schemes that encode binary data by treating it numerically and translating it into a base 64 representation.

- **VerifyFileHash:** verifies the integrity of a file by computing the MD5³³ hash value of the file's stream before and after the transmission in order to check that the file received is identical to the file sent. Thus, the web service determines whether any changes have been made to the file during the transfer.

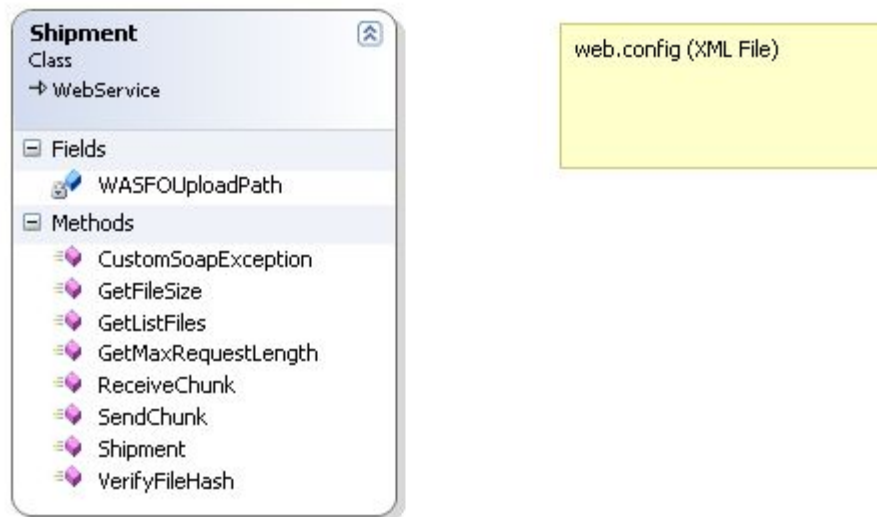


Figure 34 Class diagram of Shipment Web services

After the shipment web service completes uploading the WASFO project into the web server of WASFO, the project is then saved in a folder named by the user id in the local desk of the server. So in this way, the web server could manage the stored projects belong to different users.

7.3 IBM.WASFO.MTOM (Web Services Client)

This component represents the Web service client of WASFO Web services. Hence, it has a client reference to the Shipment Web services. This client contains a custom control of type .NET Background Worker, and so it acts as a background process in any .NET Application. Indeed, IBM.WASFO.MTOM carries the shipment process of WASFO projects through this custom control. Hence, both WASFO tools can send/receive files into/from the web server of WASFO Middleware with the help of this component. In the figure below, is the class diagram of the component. As you can see, the two main client classes are as follows:

- **DownloadBackgroundWorker:** calls the *ReceiveChunk* service for downloading files.
- **UploadBackgroundWorker:** calls the *SendChunk* service for uploading files.

Both of them inherit from the class *MTOMBackgroundWorker*. This parent class is also a client class to some Shipment web services (*GetFileSize*, *GetListFiles*, *GetMaxRequestLength...etc*), and has some common functionalities needed for the transmission process (*VerifyLocalFileHash*). For example, it calls the web service “*VerifyFileHash*” to perform MD5

³³ MD5; Message Digest algorithm 5

file hash before or after the transmission on the server application to verify that the received file is identical to the sent file.

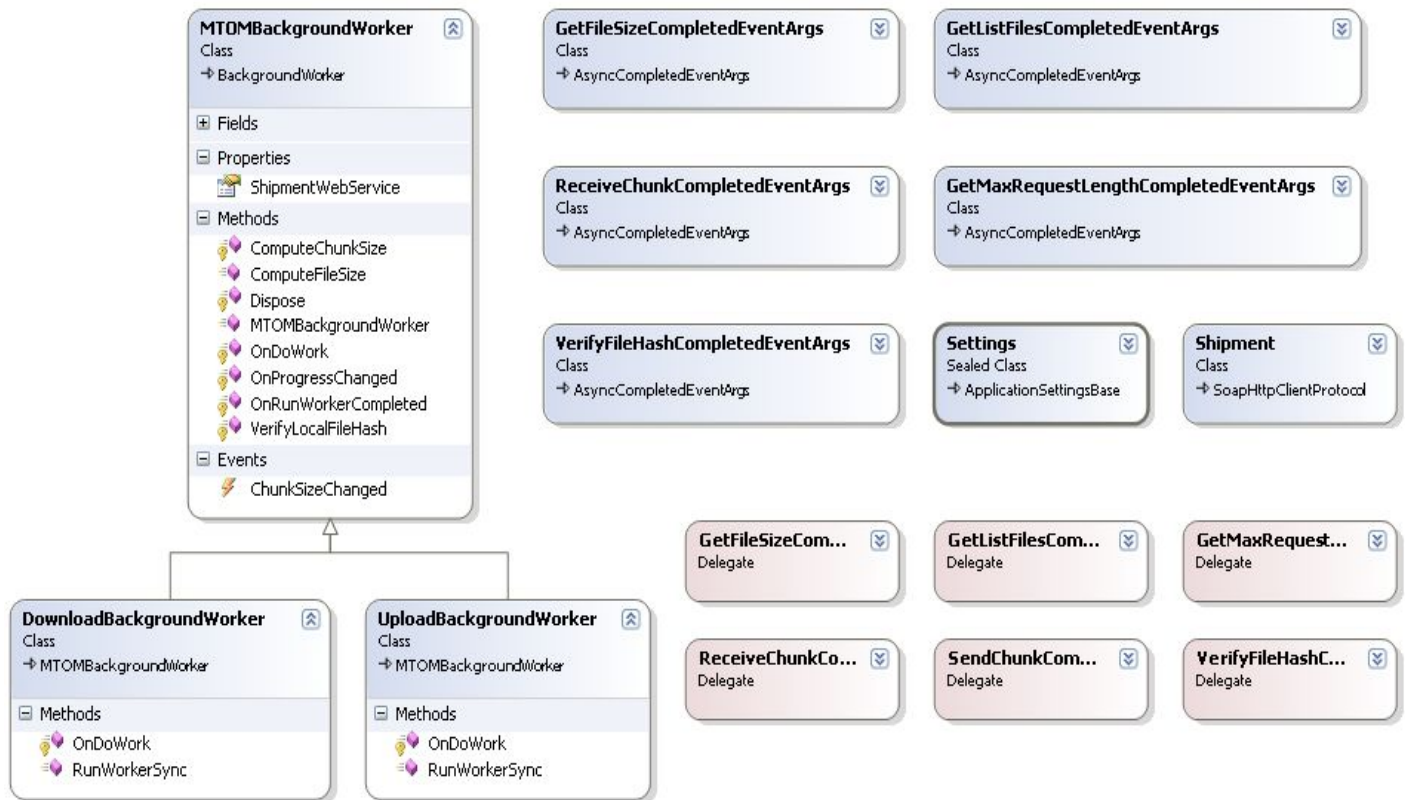


Figure 35 Class diagram of MTOM Component

Here below, you find a screenshot of the *app.config* file shown in figure 36; this file is necessary to run *MTOM* in the client side of the application. It contains sections referring to the WSE 3.0 assembly and a messaging *clinetMode* setting for the client.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <section name="microsoft.web.services3" type="Microsoft.Web.Services3.Configuration.WebServicesConfiguration, Mi
  </section>
    <sectionGroup name="applicationSettings" type="System.Configuration.ApplicationSettingsGroup, System, Version=2.0.
  </sectionGroup>
    <section name="IBM.WASFO.Shipment.Properties.Settings" type="System.Configuration.ClientSettingsSection, System,
  </section>
  </configSections>
  <microsoft.web.services3>
    <messaging>
      <mtom clientMode="On" />
      <maxMessageLength value="-1" />
    </messaging>
  </microsoft.web.services3>
  <applicationSettings>
    <IBM.WASFO.Shipment.Properties.Settings>
      <setting name="ShipmentWebService" serializeAs="String">
        <value>http://localhost:3543/WebService/Shipment.asmx</value>
      </setting>
    </IBM.WASFO.Shipment.Properties.Settings>
  </applicationSettings>
</configuration>
```

Figure 36 Application Configuration file - WASFO Shipment Web Services (Client side)

7.4 WASFO Shipment User Controls

They are mainly designed to be used as custom user controls in the GUI of WASFO tools for the sake of exporting or importing projects' files between WASFO tools. Both controls import the Shipment Web Services' client (IBM.WASFO.MTOM), in order to be able to call WASFO Shipment Web Services to export or import WASFO projects through WASFO Middleware Web Server. The two custom controls are as the following:

- IBM.WASFO.SendControl
- IBM.WASFO.ReceiveControl

In order to make the shipment process through WASFO Middleware more quick and efficient; I developed functionalities for compressing WASFO project's files (database, workload files...etc) before the export process of Shipment Web services, and also other functionalities for decompressing these projects' files after the import process. Thus, I could minimize the transfer load on the shipment web services by reducing the size of the transferred files, notably of large size before sending them. The received projects are always kept compressed in the server side to be ready for sent.

For such purpose, I used the DotNetZip Library. The library is licensed by Microsoft Public License (Ms-PL). It is a free, small, fast, and easy library for manipulating zip files. The CodePlex³⁴ powers this library, which is used mainly by .NET Applications installed on Windows PCs with the full .NET Framework to easily create, read, and update zip files. Indeed, the zip compression and decompression is so efficient with this library. Hence, the DotNetZip Library is used by both WASFO Shipment user controls in order to provide the compression or decompression functionalities for any files.

These custom controls are built by the latest .NET GUI Framework technology; Windows Presentation Foundation. WPF is an advanced graphical system for rendering user interfaces in Windows applications. It is based on DirectX that provides hardware acceleration and enables modern user interface features such as transparency, gradients, and transforms...etc. WPF uses the new user interface markup language of Microsoft .NET; XAML³⁵. This language is used to define UI³⁶ elements, data binding and events...etc.

7.4.1 IBM.WASFO.SendControl

A visual component that is imported by WASFO Data Collector Tool to export WASFO projects into the Middleware web server of WASFO. In figure 37, you can see the class diagram of the component. The component contains a visual form that allows users to export their projects' files. The export process runs always in a background thread to avoid interrupting the main thread (GUI) of WASFO Data Collector Tool. During the uploading process, the user interface provides the user with the files' transfer feedback, through a progress bar, which shows the progress of the transfer until the transmission completes. We should remember that the component uses WSE 3.0 in order to use MTOM feature while exporting WASFO projects.

³⁴ The CodePlex; Open Source Project Community

³⁵ XAML; Extensible Application markup Language

³⁶ UI; User Interface

7.4.1.1 Compression Process

When the user requests to export WASFO projects, the IBM.WASFO.SendControl calls the DotNetZip library to perform a compression for the required files, then the component calls the MTOM library to upload the zipped projects into the web server of WASFO (Middleware layer). In fact, the compression process uses the following options to compress WASFO projects:

- Format: ZIP64.
- Encryption: Advanced Encryption Standard 56 (AES56).
- Encoding: IBM Code Page 437 (IBM437).
- Password: The zipped project always has a protected password for security reasons.
- Compression Level: High Speed.

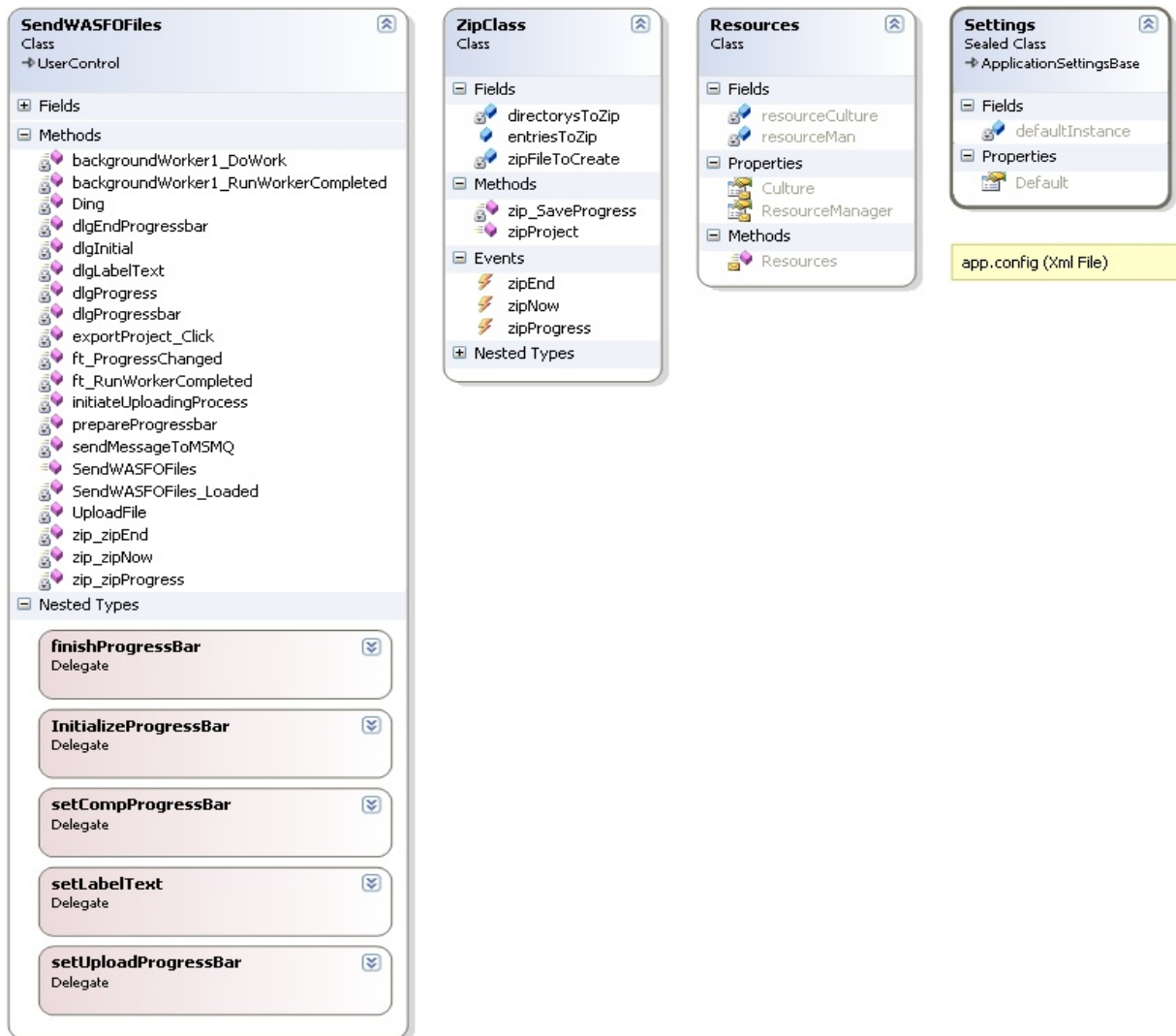


Figure 37 Class diagram of the component “IBM.WASFO.SendControl”

7.4.2 IBM.WASFO.ReceiveControl

A visual component that is imported by WASFO Analysis and Optimization Tool to import WASFO projects from the Middleware web server of WASFO. Below you can see the class

diagram of the component. It contains a visual form that allows users to import their projects' files. The import process runs always in a background thread to avoid interrupting the main thread (GUI) of WASFO Analysis and Optimization Tool. During the downloading process, the user interface provides the user with the files' transfer feedback, through a progress bar, which shows the progress of the transfer until the transmission completes. In addition, the component provides a function, which allows the user to check manually if there are any related projects in the web server of WASFO. If it finds any projects, it prompts the user if s/he wants to import the found projects. This function runs also automatically in a background thread every certain interval to check if there are any available projects in the web server. If so, it informs automatically the user whether to download those available projects immediately. Indeed, the component uses WSE 3.0 in order to use MTOM feature while exporting WASFO projects.

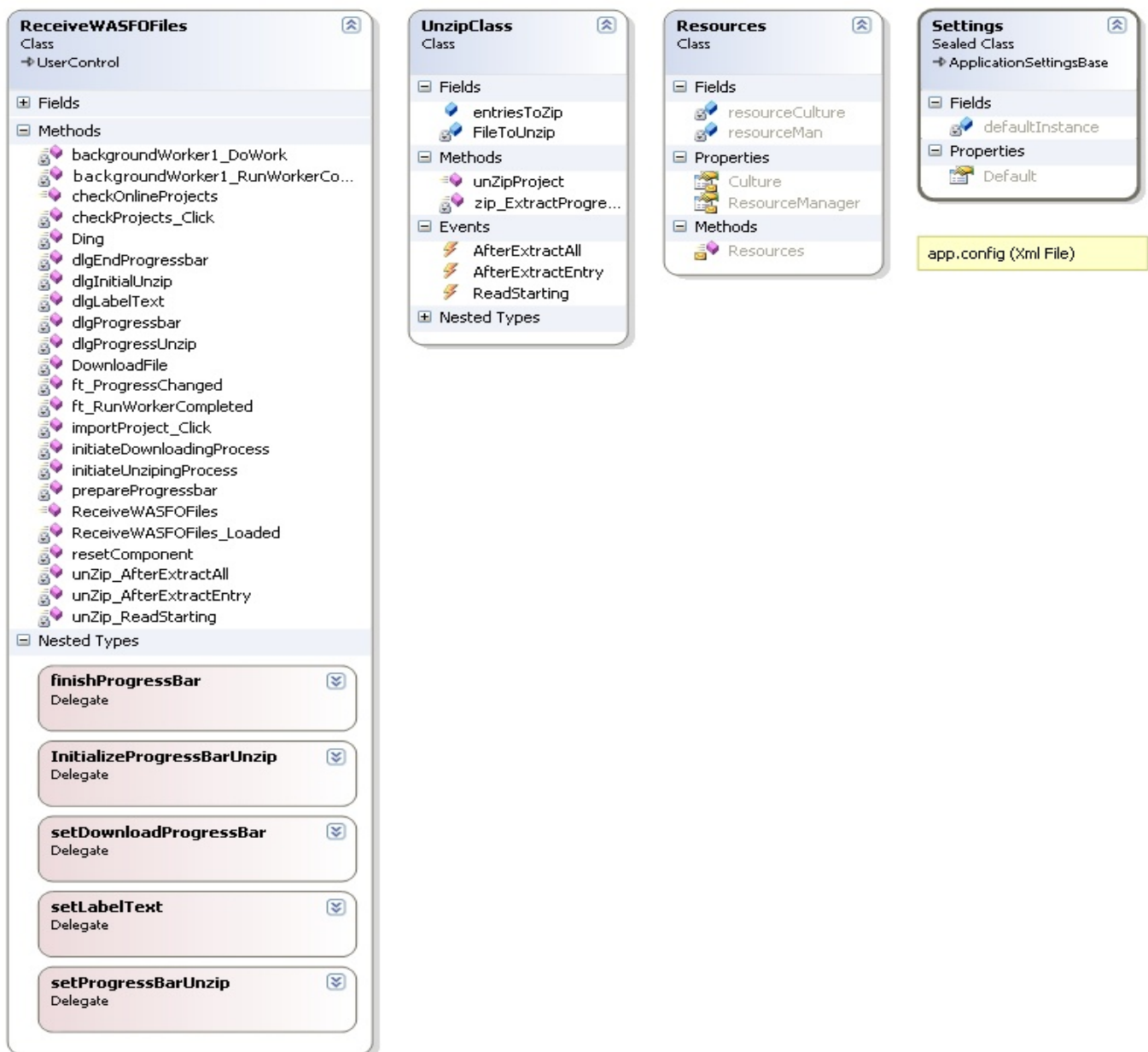


Figure 38 Class diagram of the component "IBM.WASFO.ReceiveControl"

7.4.2.1 Decompression Process

When the user requests to import WASFO projects, the IBM.WASFO.ReceiveControl calls the MTOM library to download the available projects on the web server of WASFO (Middleware layer), and after that the component calls the DotNetZip library to perform a decompression for the received projects (zipped projects). In fact, the decompression process follows the same options that were used for compressing WASFO projects:

- Format: ZIP64.
- Encryption: Advanced Encryption Standard 56 (AES56).
- Encoding: IBM Code Page 437 (IBM437).
- Password: The same password that was used while compressing the file.
- Compression Level: High Speed.

7.5 Web Service Configuration

Any Web service that uses WSE 3.0 must refer to Microsoft.web.services3 in its web configuration file. This file is needed to configure all the settings related to a web service in order for the web service to run successfully. As you can see below the screenshot of the *web.config* file is shown in figure 39; this file is necessary to run MTOM in the server side of the application. It contains sections referring to the WSE 3.0 assembly and the messaging *serverMode* setting for the web server.

```

<?xml version="1.0"?>
<configuration>
  <configSections>
    <section name="microsoft.web.services3" type="Microsoft.Web.Services3.Configuration.WebServicesConfiguration, Microsoft.Web.Services3, Version=3.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" />
  </configSections>
  <sectionGroup .../>
  <appSettings>
    <add key="UploadPath" value="UsersUpload"/>
    <!-- relative or absolute path -->
  </appSettings>
  <system.web>
    <webServices>
      <soapServerProtocolFactory type="Microsoft.Web.Services3.WseProtocolFactory, Microsoft.Web.Services3, Version=3.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" />
      <soapExtensionImporterTypes>...</soapExtensionImporterTypes>
    </webServices>
    <compilation .../>
    <httpRuntime maxRequestLength="409600" executionTimeout="300"></httpRuntime>
    <pages>...</pages>
    <httpHandlers>...</httpHandlers>
    <httpModules>
      <add name="ScriptModule" type="System.Web.Handlers.ScriptModule, System.Web.Extensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" />
    </httpModules>
    <microsoft.web.services3>
      <messaging>
        <mtom serverMode="Always"/>
        <maxMessageLength value="-1" />
      </messaging>
    </microsoft.web.services3>
  </system.web>
  <system.codedom>
    <compilers>...</system.codedom>
  </system.codedom>
  <system.webServer>...</system.webServer>
  <runtime>
    <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
      <dependentAssembly>...</dependentAssembly>
      <dependentAssembly>...</dependentAssembly>
    </assemblyBinding>
  </runtime>
</configuration>

```

Figure 39 Web Configuration file (XML file) of WASFO Shipment Web Services (Server side)

8 IBM WASFO License Generator

This tool is designed to create a valid license for WASFO user to be authorized to access one of WASFO Tools based on the user's type. The tool is an internal tool in IBM, which is managed by Dr. Mauro Gatti, the project leader of WASFO. He has designed and developed this tool to secure the access to WASFO Tools against any misuses or fraud actions. As you can see below, this is a general picture of how user's license is generated, and how the user can be authenticated through the authentication Form (WASFO Login Form) of WASFO Toolset.

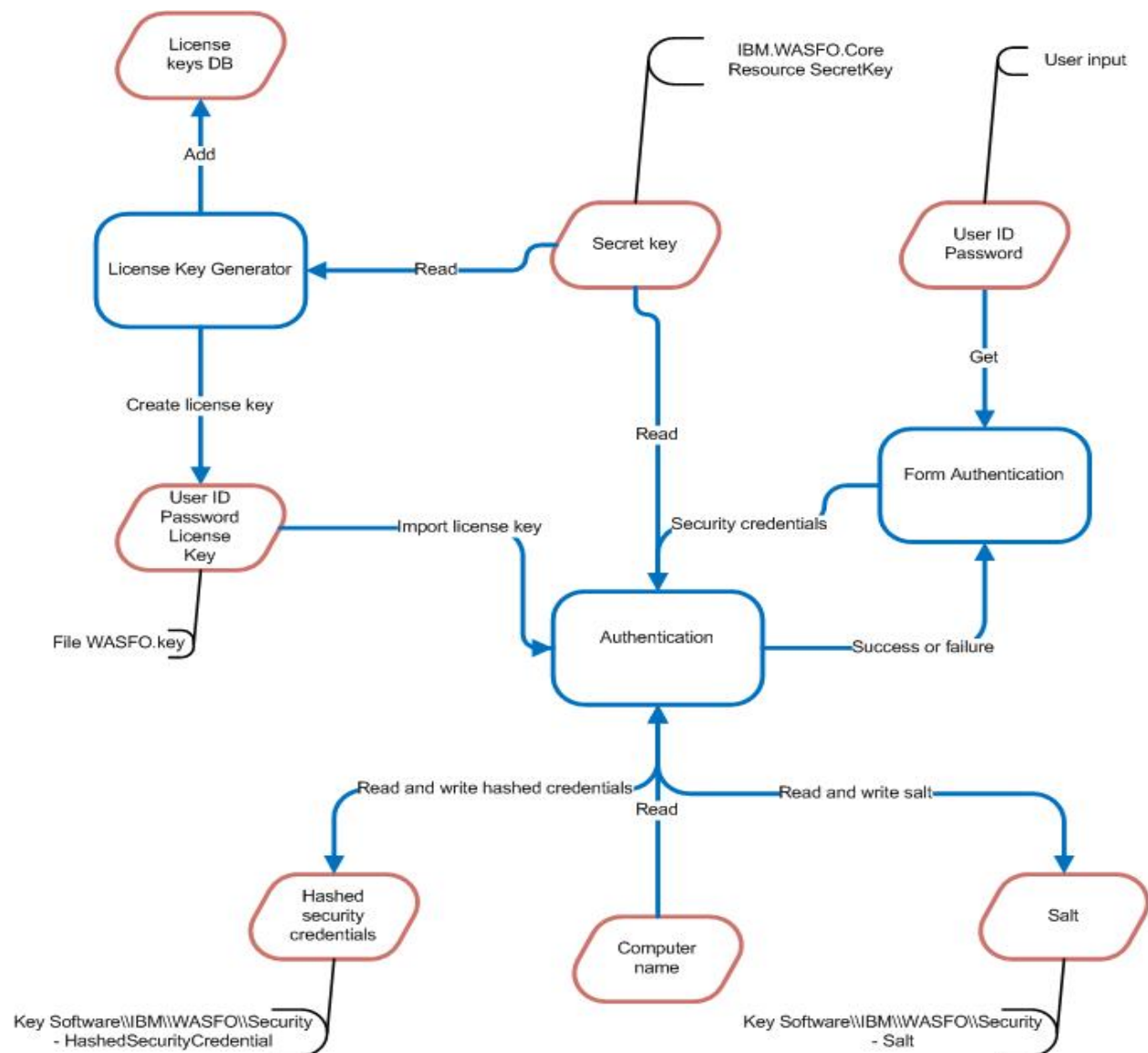
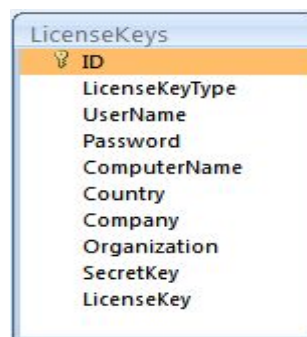


Figure 40 Authentication mechanism in WASFO Toolset (Drawn by Dr. Mauro Gatti, IBM Italy)

WASFO License Generator generates a license file in XML format, which represents a license key to be imported afterward during the user's login. The license contains the hashing result of;

1. User credentials
 - a. UserID → the user’s email
 - b. Password → the user’s password
2. Type of the license key
 - a. License for WASFO Data Collector Tool
 - b. License for WASFO Analysis and Optimization Tool);
3. Secret key

The “*SecretKey*” Object represents the secret key (random value) that a specific algorithm inside the *IBM.WASFO.Core* library creates it automatically. After that, the license tool creates the “*LicenseKey*” Object using the *IBM.WASFO.Core* Library, then it stores this generated object in the table “*LicenseKeys*” for further authentication (as shown below in figure 41).



ID
LicenseKeyType
UserName
Password
ComputerName
Country
Company
Organization
SecretKey
LicenseKey

Figure 41 Table LicenseKey

In the cryptography world, our license key is called a message digest, which is obtained by computing the SHA-1³⁷ hashing of a combination of the id, password, license type and secret key of the user. The length of the license key is 160 bits. The cryptographic hash function has four main properties:

- Easy to compute the hash value for any given license;
- Infeasible to find a license that has a given hash;
- Infeasible to modify a license without changing its hash;
- Infeasible to find two different licenses with the same hash.

For the authentication purpose (using *IBM.WASFO.Authentication*), a hashing process of the user’s security credentials is computed in order to identify the corresponding user and his/her machine that is running an instance of WASFO tool. The user’s security credentials represents the output of the SHA-1 hashing process for the combination of the user’s id, user’s password, secret key, imported license key, user’s computer name and salt. The salt is an output, which results from a key derivation function that takes two inputs (random bits and user’s password). The output of this key derivation function is stored as the encrypted version of the password. This output is then used as a part of the security credentials of the user.

³⁷ SHA-1; Secure Hash Algorithm-1

9 IBM WASFO Deployment

As we already know, WASFO Toolset has four main entities:

1. IBM WASFO Data Collector Tool (data collection)
2. IBM WASFO Analysis and Optimization Tool (data analysis and optimization)
3. IBM WASFO Middleware Software (IIS, hosts WASFO Shipment Web Services)
4. IBM WASFO Database (data storage)

Hereunder, you can see a real picture of WASFO Solution after the deployment:

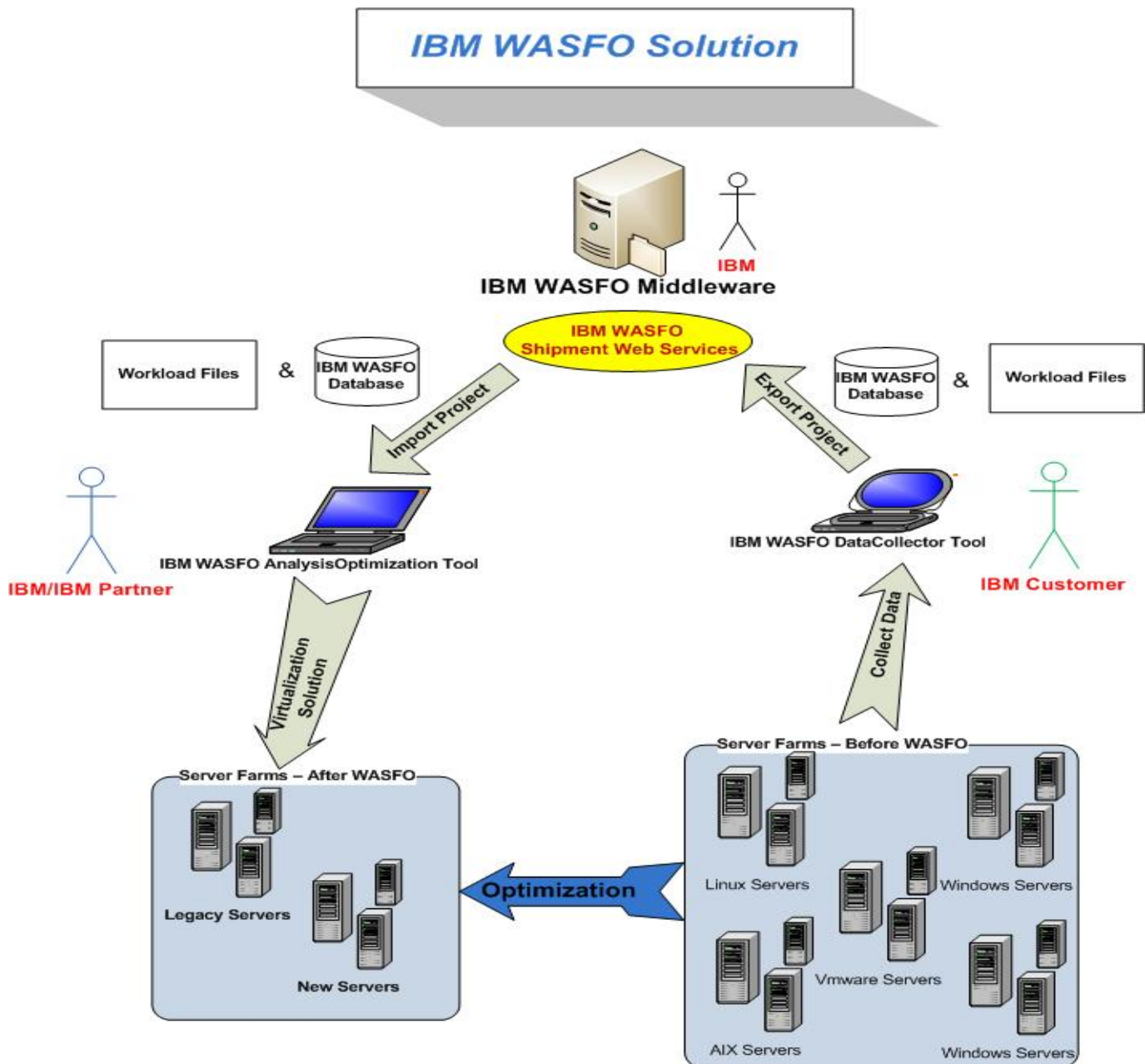


Figure 42 UML diagram of WASFO Solution

Hence, two deployment packages have been developed for WASFO Toolset in order to enable users to install the desired WASFO Tool in their PCs. Both tools are installed in a release mode because they are stable, and so there is no need to install them in a debug mode in users' machines. The two-setup packages for WASFO are:

1. IBM.WASFO.DataCollector.Setup
2. IBM.WASFO.AnalysisOptimization.Setup

During the setup process of WASFO, users must insert a setup key to verify their validity to use WASFO Tool. Hereunder, is an example of one of the screenshots that appears during the installing process of IBM.WASFO.DataCollector Tool:

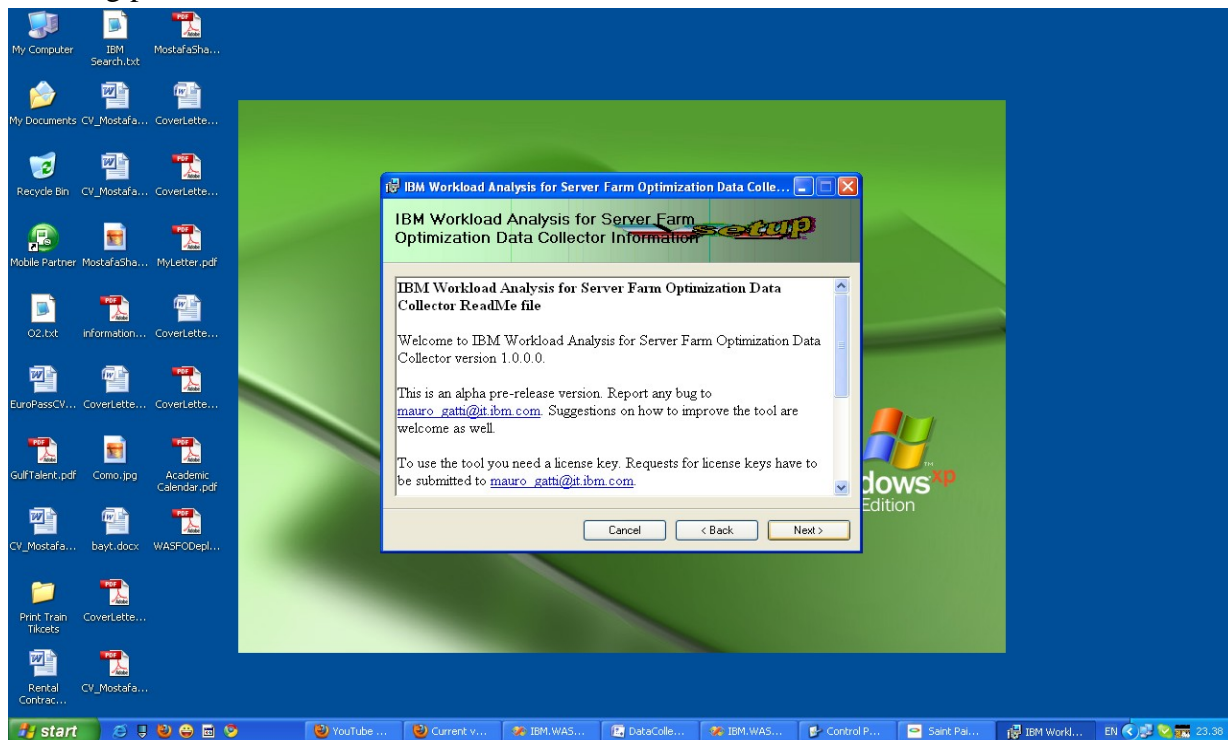


Figure 43 Welcome screen during the setup process of IBM WASFO Data Collector Tool

In order to install WASFO Tools successfully, there are three mandatory pre-requisites that must be found in the running machine prior to the setup process;

- Microsoft.NET Framework 3.5 or later.
- Windows Installer 3.1 or later.
- Microsoft Visual C++ 2005 SP1³⁸ Redistributable Package (x86). This package installs the runtime components of Visual C++ Libraries required to run applications developed with Visual C++ on a computer that does not have Visual C++ 2005 installed.

Without installing these pre-requisites, the installation process of the desired WASFO Tool will definitely fail, warning the user about the reason of the failure. The interesting thing is that the

³⁸ SP1; Service Pack1, is a collection of updates, fixes and enhancements of Microsoft Visual C++ 2005

setup process detects automatically any missing pre-requisite(s) in the user's machine, and if so it prompts the user to install the missing pre-requisite(s) automatically online from the provider's website (Microsoft Website) or either manually from the local PC or the local network.

Indeed, building the setup packages of WASFO Tools was a quite complex process, due to the high dependencies between the libraries of WASFO both for the shared and unshared libraries. Thus, I had to take care of the ordering to build (compile) these libraries, based on their dependencies upon each other. Hence, I started by the root library "*IBM.WASFO.Core.dll*", which has no dependencies upon any other libraries except for "*Vimservice2005.dll*" library, and then I continued building the subsequent libraries starting from the less dependent libraries until the high dependent libraries. You can see in figure 5, the hierarchy of the libraries' dependencies above in page (30) at section (5.2) in Chapter.5 "*IBM.WASFO.Toolset*".

Hence, after installing WASFO DataCollector Tool into the user's PC, the application folder will contain the following structure (*after implementing and integrating the new components*):

- EXE Program file; IBM.WASFO.DataCollectorGUI.exe
- App.config file
- Libraries
 1. Vimservice2005.dll
 2. Vimservice2005.XMLserializers.dll
 3. Ionic.Zip.dll (the DotNetZip library)
 4. Infragistics2.Win.v8.3.dll
 5. Infragistics2.Shared.v8.3.dll
 6. Infragistics2.Win.UltraWinChart.v8.3.dll
 7. Microsoft.ReportViewer.Common.dll
 8. Microsoft.ReportViewer.ProcessingObjectModel.dll
 9. Microsoft.ReportViewer.WinForms.dll
 10. Microsoft.Web.Services3.dll
 11. IBM.WASFO.Core
 12. IBM.WASFO.Authentication (new component to-be introduced)
 13. IBM.WASFO.ReportGenerator.dll (new component to-be introduced)
 14. IBM.WASFO.Graphs.dll
 15. IBM.WASFO.MTOM.dll
 16. IBM.WASFO.SendControl.dll
 17. IBM.WASFO.ExcelInterface.dll
 18. IBM.WASFO. ProjectExplorerer.dll
 19. IBM.WASFO. ProjectDesign.dll
 20. IBM.WASFO.DCCController.dll (new component to-be introduced)
 21. IBM.WASFO.DataCollector.dll (new version to-be integrated)
- Folders
 1. Database Folder (WASFODB)

2. Icons Folder (Icon files)
3. Tools Folder (putty.exe, plink.exe)
4. Profiles Folder (log for servers' data collection)
5. Workloads Folder (Workload files)

On the other hand, after installing WASFO Analysis and Optimization Tool into the user's PC, the application folder will contain the following structure (*after implementing and integrating the new components*):

- EXE Program file; IBM.WASFO.AnalysisOptimizationGUI.exe
- App.config file
- Libraries
 1. Vimservice2005.dll
 2. Vimservice2005.XMLserializers.dll
 3. Ionic.Zip.dll (the DotNetZip library)
 4. Infragistics2.Win.v8.3.dll
 5. Infragistics2.Shared.v8.3.dll
 6. Infragistics2.Win.UltraWinChart.v8.3.dll
 7. Microsoft.ReportViewer.Common.dll
 8. Microsoft.ReportViewer.ProcessingObjectModel.dll
 9. Microsoft.ReportViewer.WinForms.dll
 10. Microsoft.Web.Services3.dll
 11. IBM.WASFO.Core
 12. IBM.WASFO.Authentication (new component to-be introduced)
 13. IBM.WASFO.ReportGenerator.dll (new component to-be introduced)
 14. IBM.WASFO.Graphs.dll
 15. IBM.WASFO.MTOM.dll
 16. IBM.WASFO.ReceiveControl.dll
 17. IBM.WASFO.ExcelInterface.dll
 18. IBM.WASFO. ProjectExplorer.dll
 19. IBM.WASFO. ProjectDesign.dll
 20. IBM.WASFO.InventoryAnalysis.dll
 21. IBM.WASFO.WorkloadAnalysis.dll
 22. IBM.WASFO.Optimization.dll
 23. IBM.WASFO.AOController.dll
- Folders
 1. Database Folder (WASFODB)
 2. Icons Folder (Icon files)
 3. Solvers Folder (solveModel.exe)
 4. Workloads Folder (Workload files)

Here below you can find a screenshot regarding the contents of the application folder of IBM WASFO Data Collector Tool after the setup process. You will find also in figure 45, another

screenshot regarding the application configuration file of IBM WASFO Analysis and Optimization Tool. As you can see, the application configuration file is in XML format.

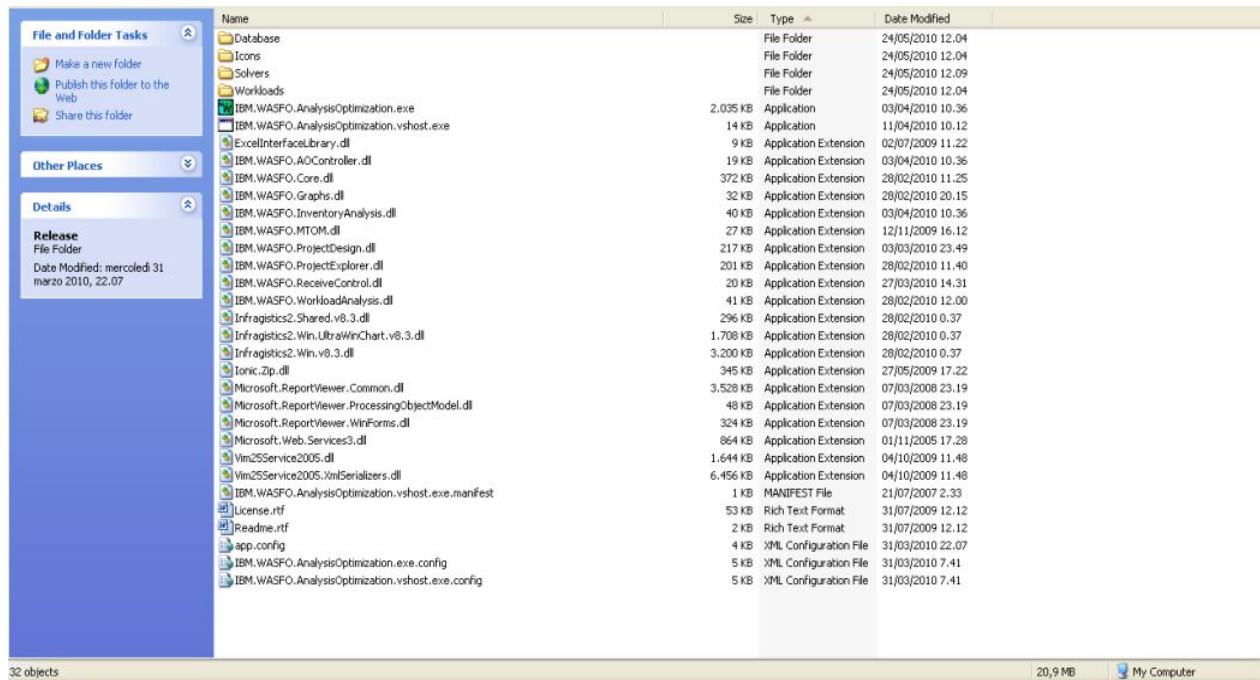


Figure 44 IBM WASFO Data Collector Tool - Application Folder



Figure 45 App.config XML file of IBM WASFO Analysis and Optimization Tool

10 Conclusion

WASFO Toolset has now a completely new architecture. In addition, most of the algorithms, functionalities, policies have been refined and improved in order to increase the efficiency and the execution performance of WASFO Toolset.

Currently WASFO Toolset is composed of five physical entities:

- **IBM WASFO Data Collector Tool**
(for data collection)
- **IBM WASFO Analysis and Optimization Tool**
(for data analysis & optimization)
- **IBM WASFO License Generator Tool**
(for creating users' licenses)
- **IBM WASFO Database**
(for data storage)
- **IBM WASFO Middleware Software**
(for the shipment purpose of WASFO projects between the two main tools)

In fact, using web services for the shipment of WASFO Project files and WASFO Database has evolved the single-tier architecture (monolithic application) of WASFO to a distributed architecture. The distributed architecture is based on IBM WASFO Middleware that acts as a central Web Server (represented by Internet Information Server-IIS) for any requests coming from WASFO Data Collector Tool or WASFO Analysis and Optimization Tool. Currently, the middleware software is being used for exporting projects' data from the Data Collector Tool, and for importing them to the Analysis and Optimization Tool.

Hence, I think there is a lot to be introduced for IBM WASFO Middleware in the near future, which will certainly help WASFO to cope with the current trends and software technologies, particularly regarding the web services technologies and SOA.

Indeed, WASFO has made significant steps, changes and updates during the latest year, which improved it to be more robust than before. Moreover, these changes will make the future development and modification of WASFO Toolset is easier and flexible. Here are the main changes and updates, which have been made for WASFO during the latest year:

- A. New architecture
- B. Building a middleware in the form of exposed web services
- C. New database design
- D. User friendly and flexible graphical user interface
- E. Using Web services;
 - VMware Web service
 - WASFO Shipment Web services
- F. Building shared libraries and components
- G. Parallel execution of computations using threading techniques
- H. Automatic import of reference servers from IDEAs Spreadsheet into WASFO DB
- I. New and efficient matching algorithm for finding automatically the performance capacity of collected servers in existing server farm

- J. New optimization algorithms (Advanced mathematical and analysis models)
- K. New Data Collection methods;
 - VMware inventory and workload data collection
 - Linux I/O data collection

Actually, the interesting part of the architecture design is that the two main WASFO Tools, share set of business functionalities and graphical user controls. Therefore, I had to apply the principle of software components' re-use by designing and building set of shared components in the form of class libraries and custom control libraries during the engineering of WASFO Optimization and Analysis Tool, and during the re-engineering of WASFO Data Collector Tool. This helped significantly in improving the architecture design of WASFO Toolset. The following are the components in which I contributed to its development:

- *IBM.WASFO.Core*
- *IBM.WASFO.ProjectDesign*
- *IBM.WASFO.ProjectExplorer*
- *IBM.WASFO.DataCollector*
- *IBM.WASFO.DataCollectorGUI*
- *IBM.WASFO.InventoryAnalysis*
- *IBM.WASFO.AOController*
- *IBM.WASFO.AnalysisOptimizationGUI*
- *IBM.WASFO.ShipmentService*
- *IBM.WASFO.MTOM*
- *IBM.WASFO.SendControl*
- *IBM.WASFO.ReceiveControl*

In addition, building a database management system to store WASFO data was an appropriate decision, because it has resolved many technical problems that were arisen because of using XML files for the data storage. Although, the current design of the database system is not the optimal one, however, it is considered a significant improvement for representing the data layer of WASFO. Thus, the current database system is implemented by using Microsoft Access, and hence it is being used by both WASFO tools; WASFO Data Collector Tool and WASFO Analysis and Optimization Tool.

The development of WASFO solution included taking care of the performance engineering, software efficiency, memory management, scalability (by means of distributed architecture), multithreading techniques, software components' re-use, and code modularity. For the performance of the workload analysis and the optimization, it was crucial to make it sure that computations did not last forever.

To conclude, this current architecture has increased significantly the efficiency of the performance, and added a greater modularity and robustness in the whole software architecture. Moreover, the graphical user interface has been improved significantly than the old GUI, in order to add more flexibility and provide high quality and more user-friendly interface to the user.

11 Future Work

11.1 WASFO Toolset Architecture

The current architecture of WASFO Toolset has introduced the layer of WASFO Middleware through SOA; hence continuing the design and development under the trend of SOA for WASFO, would be of a great benefit to the current distributed architecture. In addition, WASFO should support parallel computation for performing algorithms and data collection to increase the performance and the efficiency of the toolset. For example, cluster computing and grid computing would be very useful and beneficial in our case.

11.2 WASFO Toolset Components

Indeed, new libraries still need to be introduced for WASFO Toolset such as the follows:

1. IBM.WASFO.Authentication (shared library, to-be developed using C#.NET)
2. IBM.WASFO.ReportGenerator (shared library, to-be developed using C#.NET)
3. IBM.WASFO.DCController (non-shared library, to-be developed using C#.NET)
4. IBM.WASFO.DataCollector (non-shared library, the new version is implemented by using C++.NET and will be integrated with the current WASFO Data Collector tool)

These new components will definitely improve the performance and efficiency concerning the business layer of WASFO Tools, notably WASFO Data Collector Tool. In addition, some existing libraries will need to be modified particularly the core library which in a deep need for great improvement. Although the class design of the core library has been improved significantly, the code inside the classes still needs to be modified and optimized due to its complexity and lack of modularity. Indeed, using design patterns and software best practices will certainly help in having reliable software architecture and modular code.

11.3 WASFO Matching Algorithm

The new matching algorithm is in a stable status now, however we still need to test it in many real projects in order to verify its correctness, speed and efficiency. In the future, it is important that the algorithm will have the self-learning ability, in order to update it easily and adapt it to any new cases. Thus, introducing a kind of artificial intelligence-AI technique can help in creating a smart matching algorithm that is capable of recognizing unknown collected servers (servers with missing information) from previous experiences and learning. The current algorithm uses some strategies and criteria for filtering and matching collected servers such as; Server Vendor, server model, processor model, processor frequency and cache memory. I expect that more strategies and criteria will be introduced in the future to maximize the efficiency of the algorithm and to increase the matching probability for the collected servers.

11.4 WASFO Graphical user Interface (GUI)

The current graphical user interface is based on .NET built-in controls and custom controls. Although it looks good, we still need to provide more professional and flexible user interface. This can be achieved by customizing the user controls for our specific purposes. In addition, we have bought the license of the NetAdvantage-Infragistics.Net library last year. It is a private library, which has many professional and usable .NET controls. Hence, it is a good idea to make advantage of this professional library to improve WASFO GUI. Another important thing is the new application development framework; WPF “Windows Presentation Foundation” that should be used as a new technology for the GUIs in Microsoft.NET. WPF provides a professional look and feel user interface of .NET Applications, which will increase the usability and flexibility of WASFO GUI.

11.5 WASFO Database

The current database design has significant problems that should be modified in order to become more efficient and less complex. WASFO database has been created by using Microsoft Access, which is not suitable any more to handle the complexity of the database design and the volume of stored data by WASFO Toolset. For these reasons, I recommend to re-design the database using DB2, which is very suitable since it is an IBM Database Engine. In this way, we will be able to use stored procedures, triggers, constraints and rules, which in consequence will improve the performance of WASFO Database. Moreover, DB2 is a database server, which is able to centralize the data layer of WASFO Toolset, henceforth both WASFO tools could communicate remotely to each other through the database server.

11.6 WASFO Toolset Connectivity with WASFO Database

The code regarding the database operations is written in one big class “*OleDbInterface*”. This design is not the most suitable and efficient way to handle the database. Hence, it should be re-modified in the near future to overcome the increasing complexity of the class, and to improve the efficiency of WASFO’s data connectivity with the database. This is can be done by splitting this DB³⁹ class into several classes each one acts as database adapter that is responsible for specific tables and tasks in the database. Currently, we are using the database-connected mode of ADO.NET in order to connect to WASFO Database for any queries. Hence, this is not always efficient in every case; henceforth, using the latest .NET specific-database language “LINQ⁴⁰” would facilitate the connectivity with our WASFO DB, and make it more efficient. In addition, having the database in DB2 will enable us to use advanced features such as stored procedures, triggers, and constraints...etc.

³⁹ DB: Database

⁴⁰ LINQ; Language INtegrated Query

12 Appendix

A. IBM WASFO Data Collector Tool

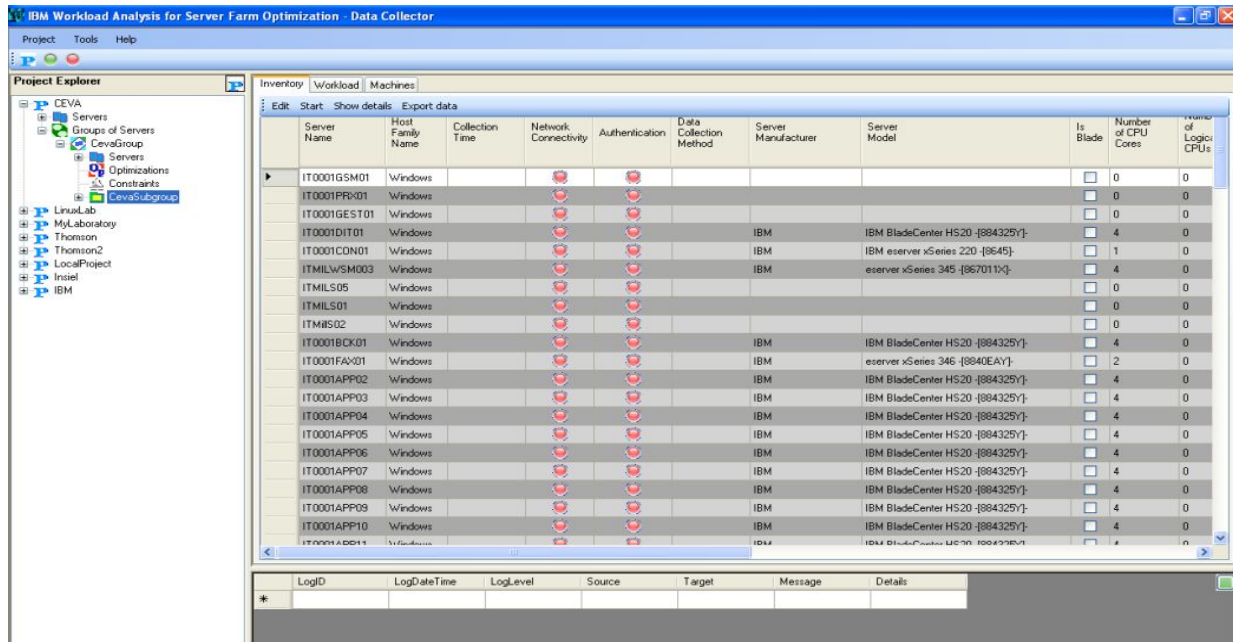


Figure 46 Screenshot of WASFO Data Collector Tool

B. IBM WASFO Analysis and Optimization Tool

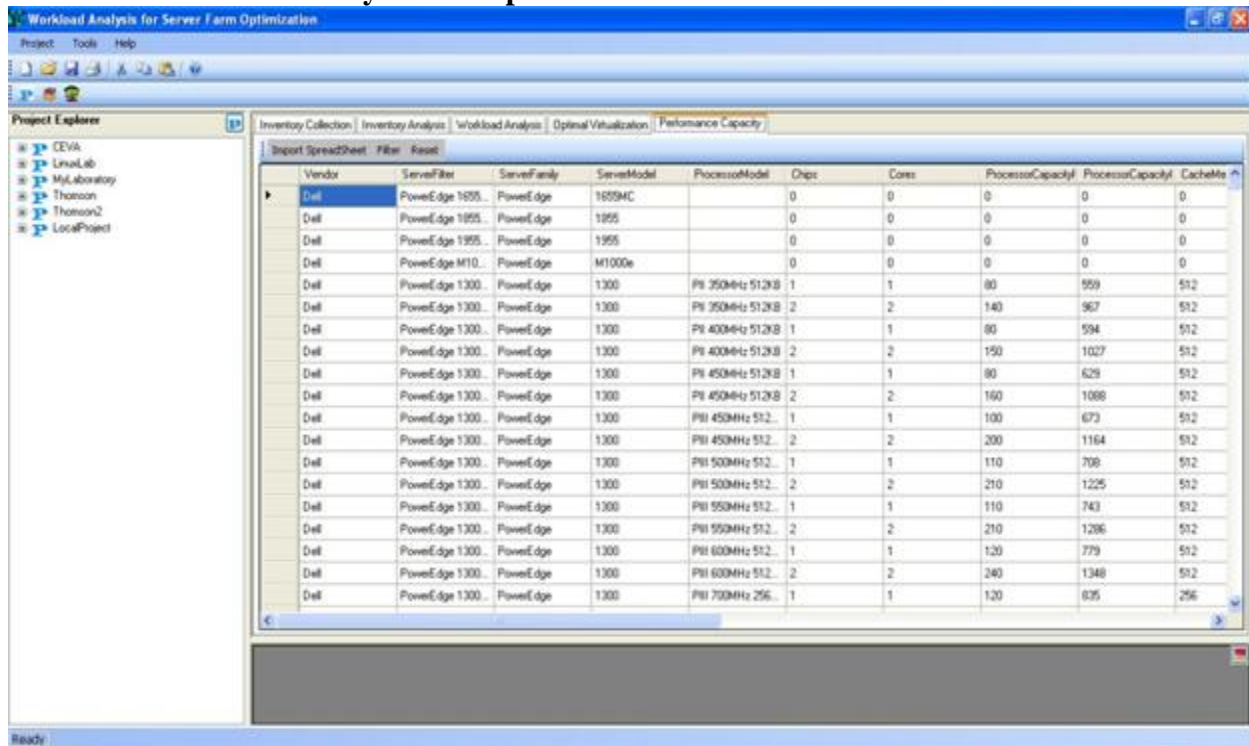


Figure 47 Screenshot of WASFO Analysis and Optimization Tool

C. IBM WASFO Data Collector Setup

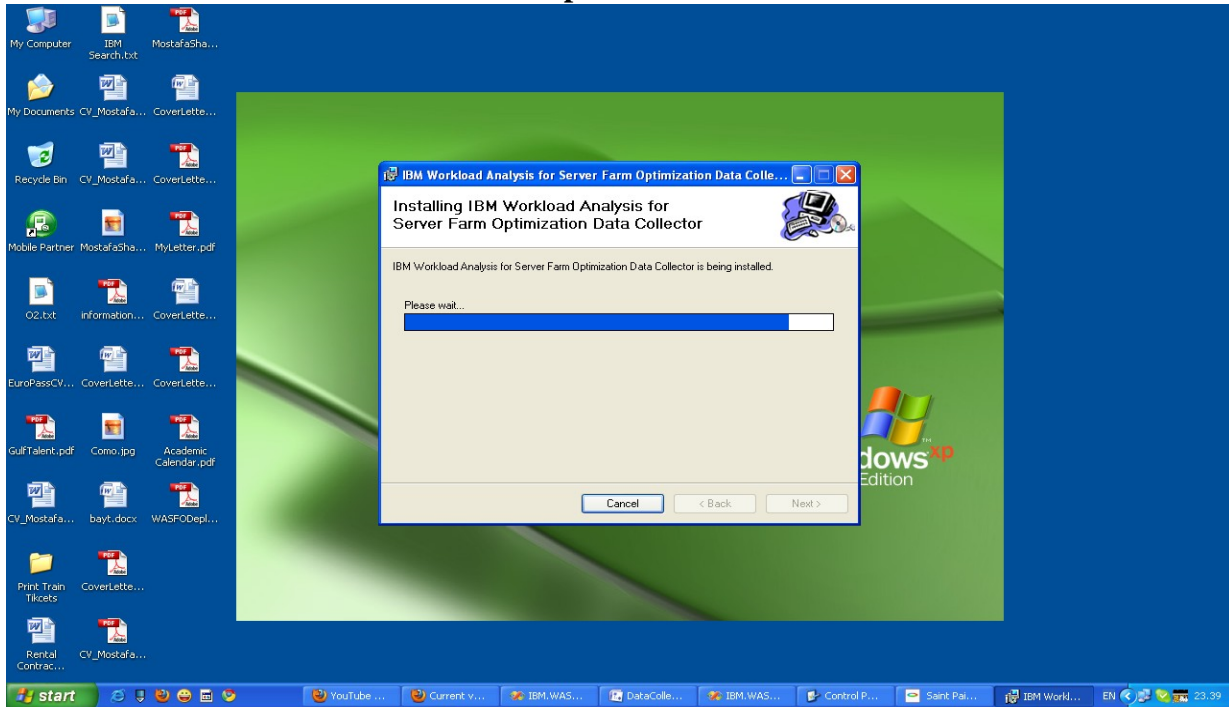


Figure 48 Screenshot of the installing process of WASFO Data Collector Tool

D. Importing process of IDEAS International Excel Sheet (Reference servers)

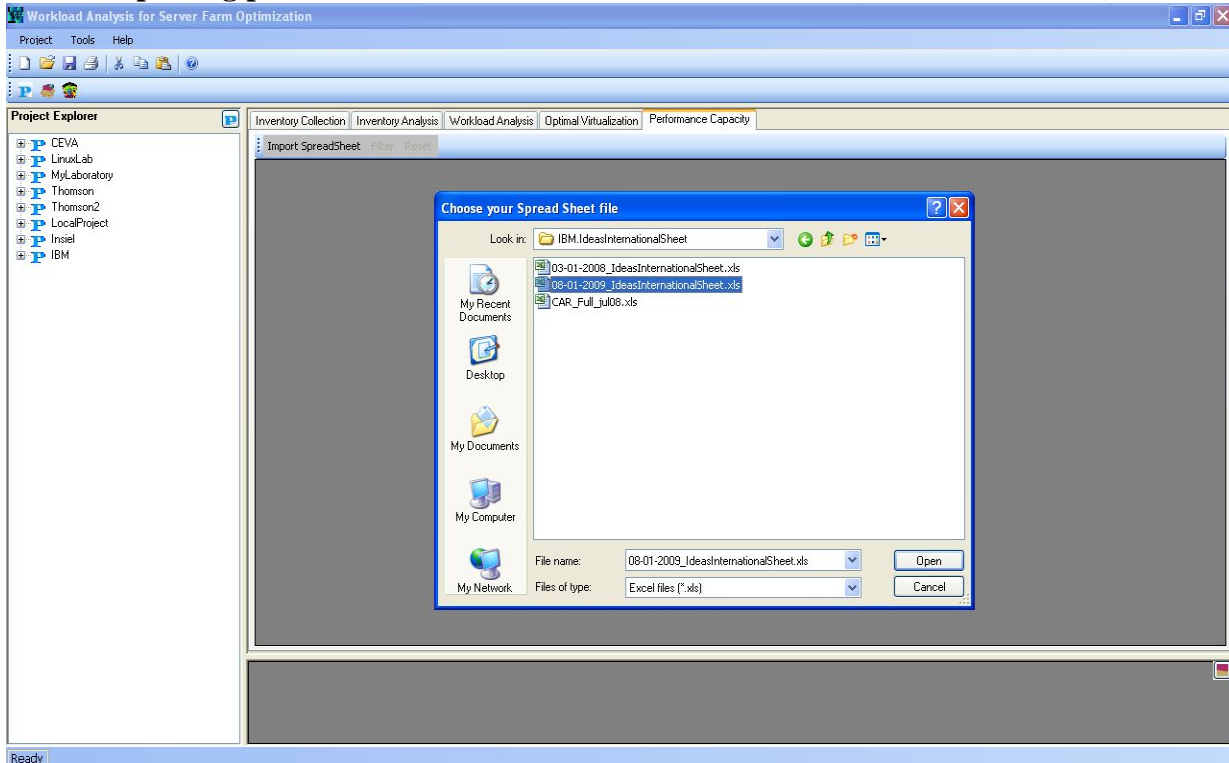


Figure 49 Screenshot of the import process of IDEAS International Tables into WASFO Database

E. The process of Inventory Collection (IBM WASFO Data Collector Tool)

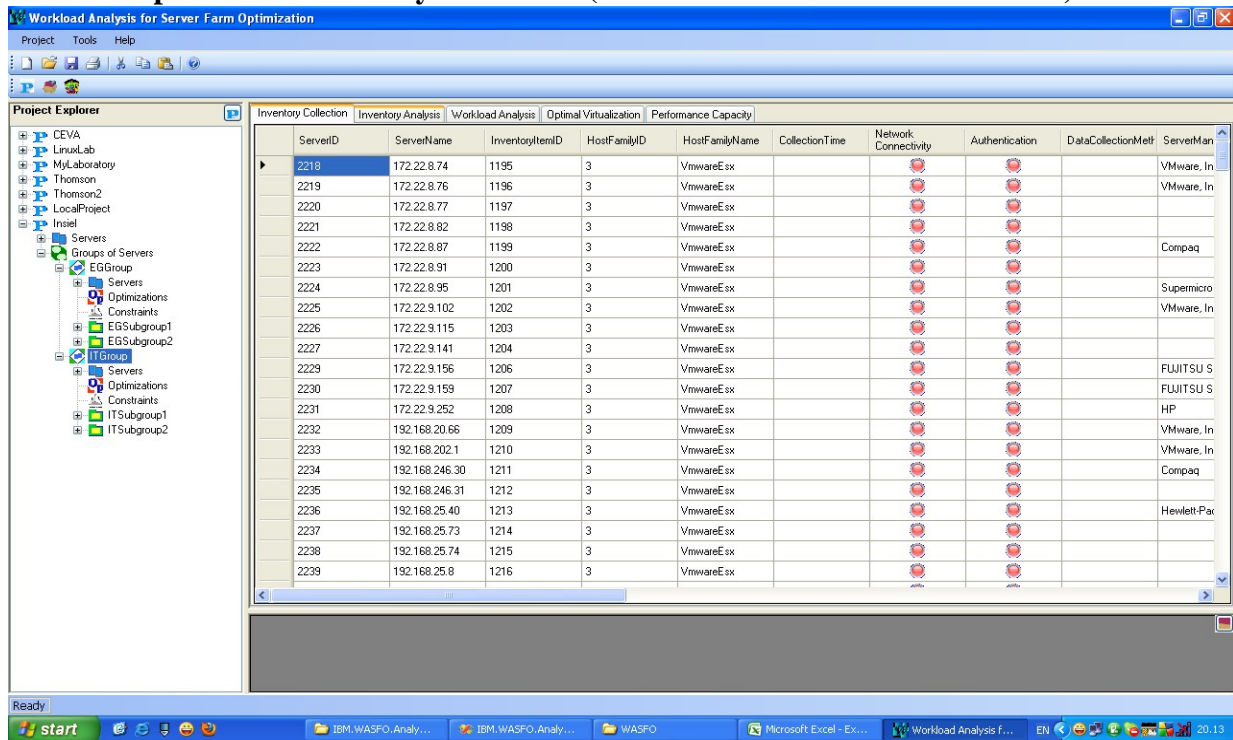


Figure 50 Screenshot of the inventory collection-user form

F. The process of Inventory Analysis (IBM WASFO Analysis and Optimization Tool)

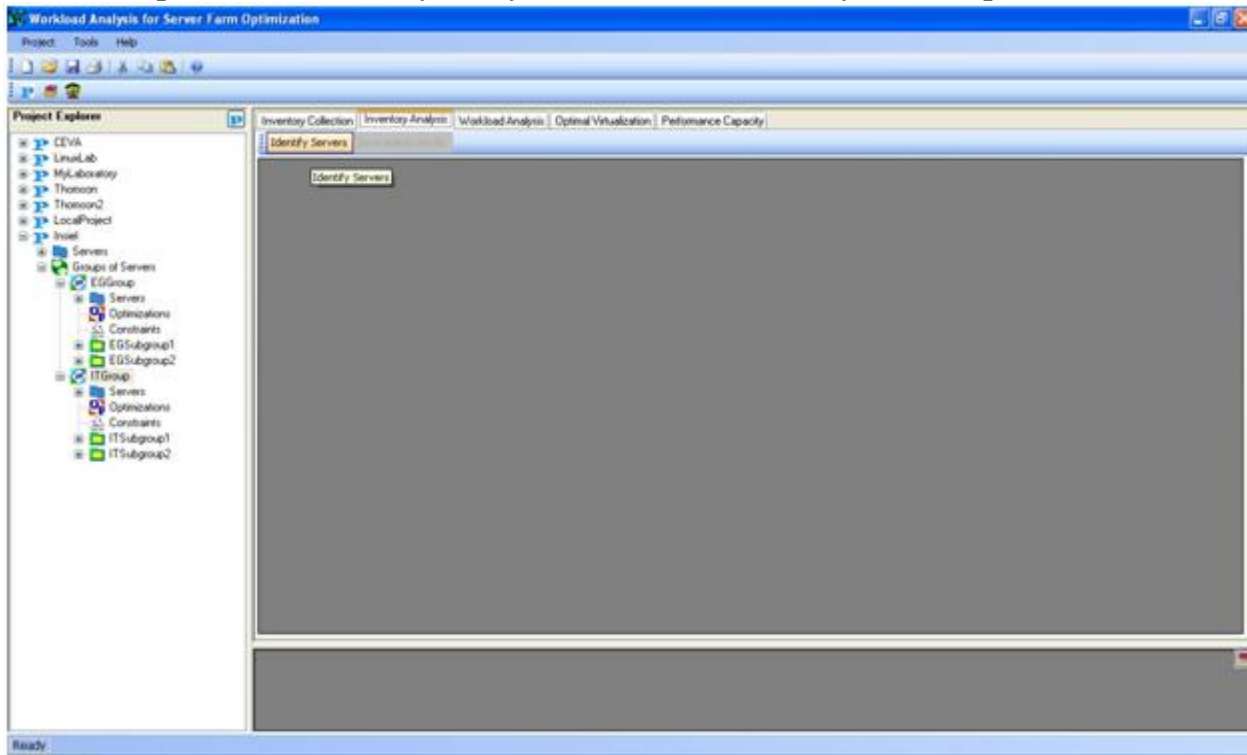


Figure 51 Screenshot of the inventory analysis- user form

G. Identifying the performance capacity of collected servers (Matching process)

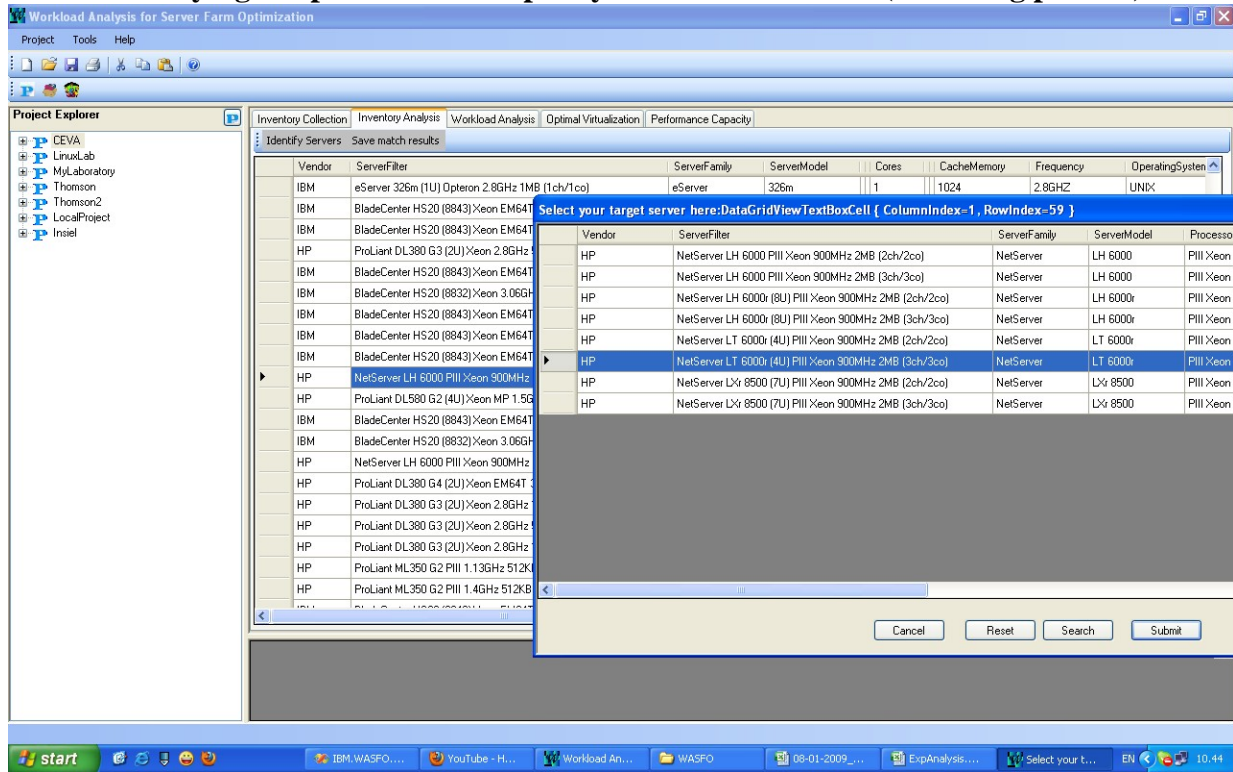


Figure 52 Screenshot of the identified collected servers-user form

H. The InventoryAnalysis Table in WASFO Database, concerning the matching results between the servers in the tables “ServersInventory” & “ReferenceServers”

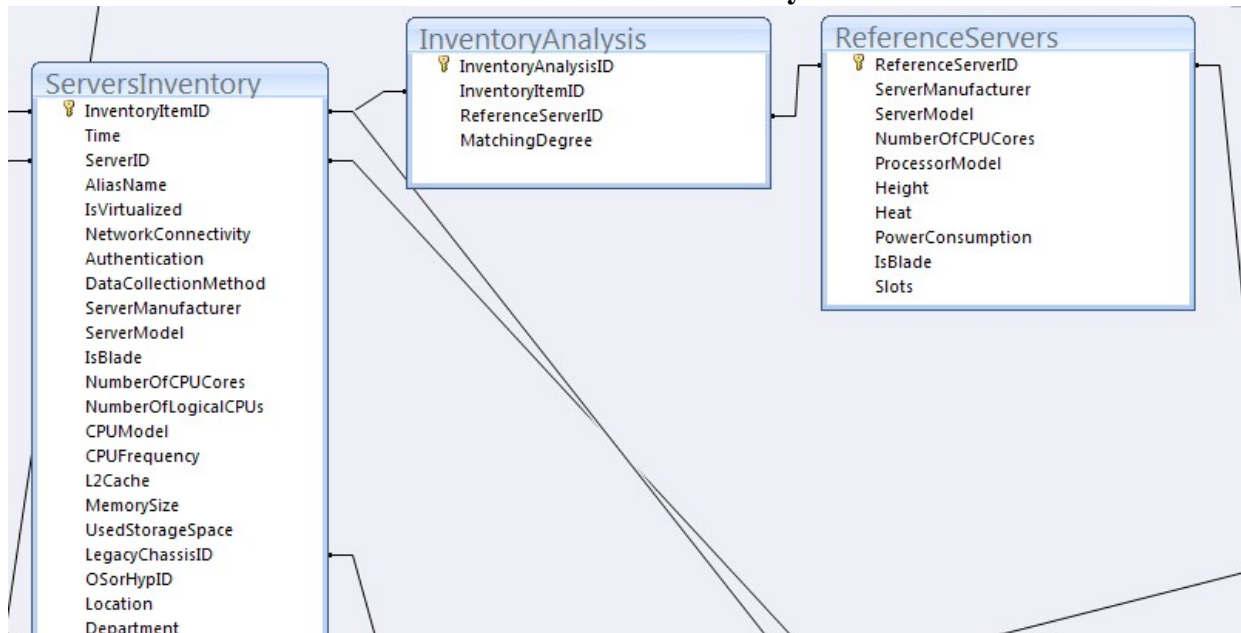


Figure 53 Relations between the Inventory Analysis Tables in WASFO Database

I. WASFO ProjectDesign Custom Control (Project Wizard)

Description of ProjectDesign Wizard of WASFO Project

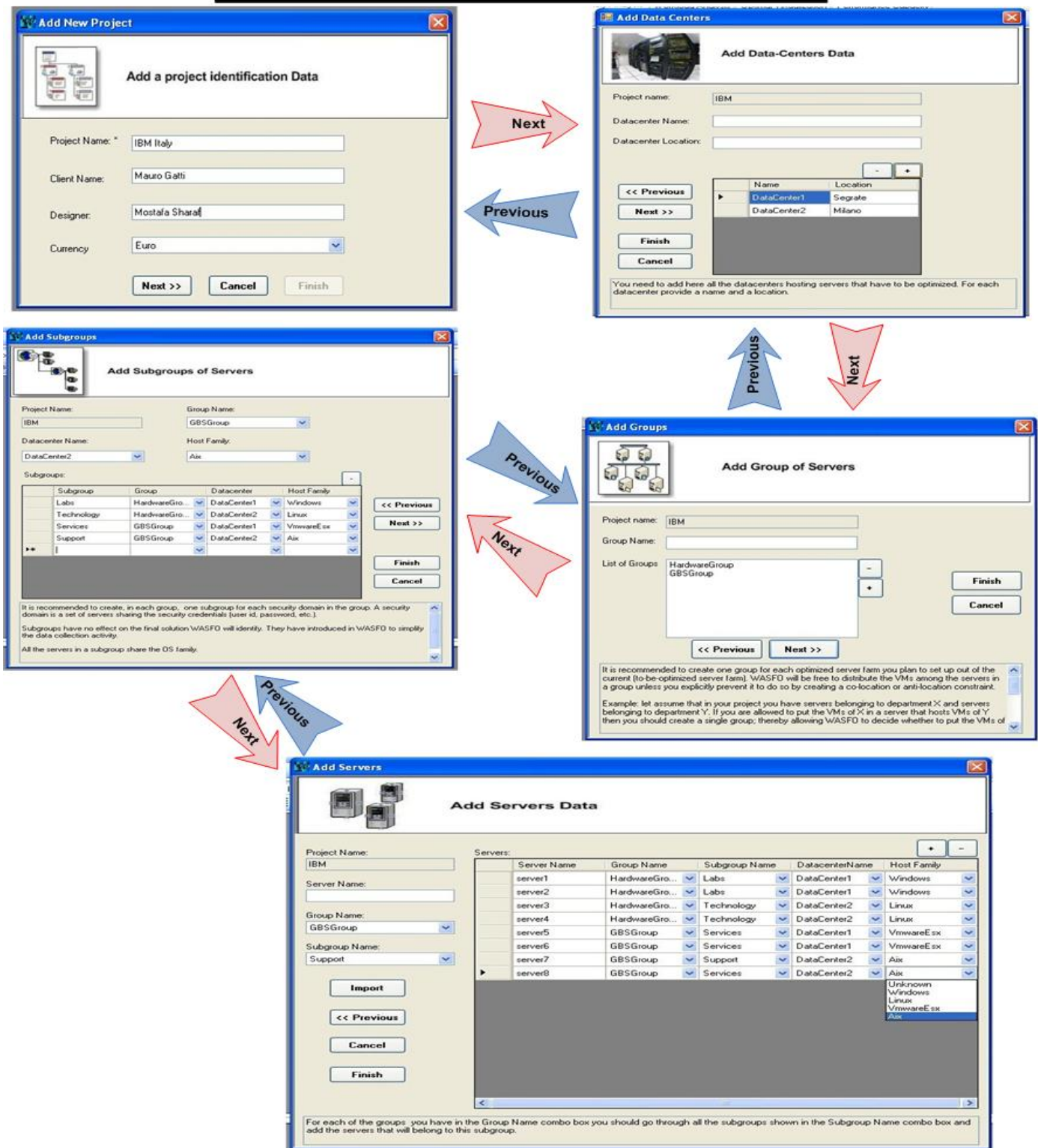


Figure 54 Screenshots of the ProjectDesign component

J. Exporting process of WASFO Projects (WASFO Data Collector Tool)

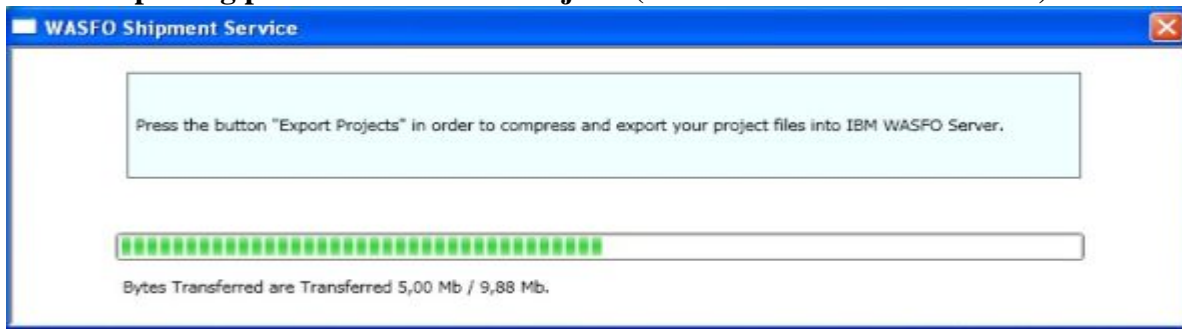


Figure 55 Screenshot of IBM.WASFO.SendControl component

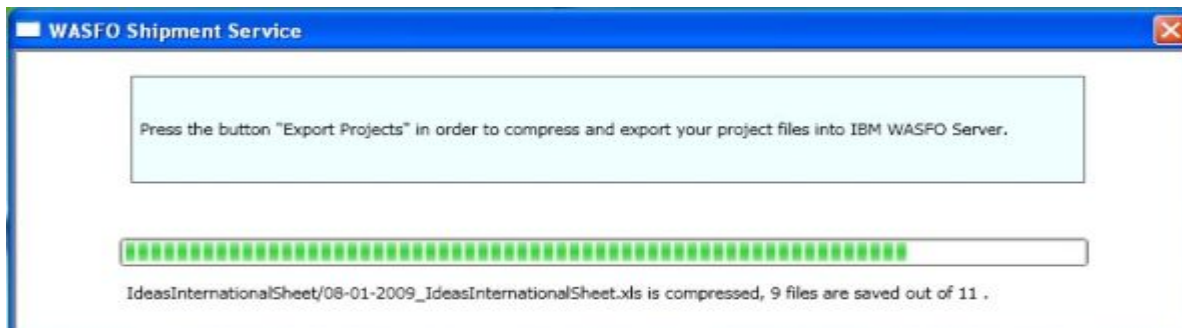


Figure 56 Screenshot of the exporting process-step1 (compressing WASFO projects)

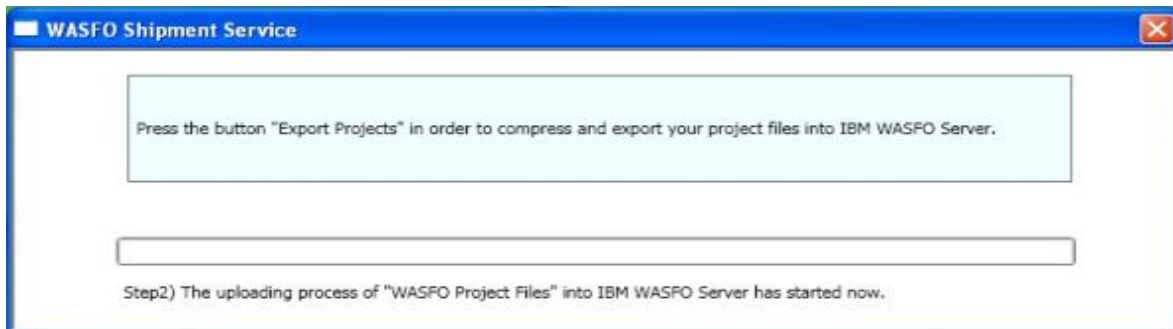


Figure 57 Screenshot of the exporting process-step2 (uploading WASFO projects)

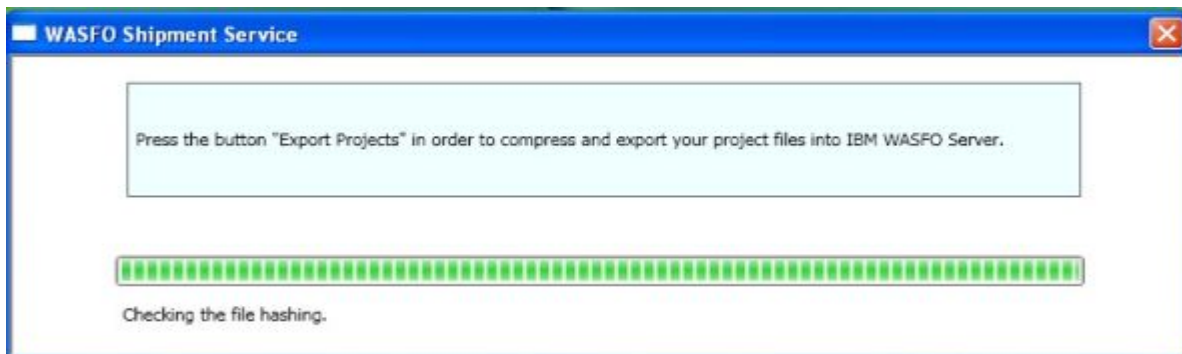


Figure 58 Screenshot of the exporting process-step3 (Verifying files' integrity by hashing)

K. Importing process of WASFO Projects (WASFO Analysis and Optimization Tool)

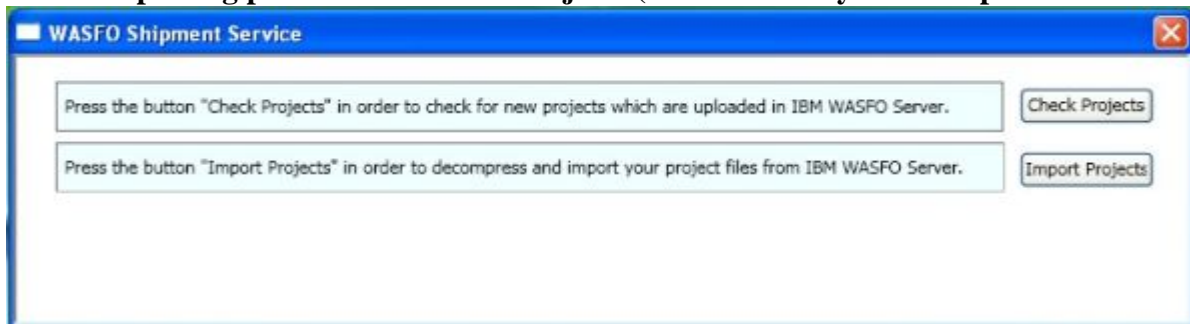


Figure 59 Screenshot of IBM.WASFO.ReceiveControl component

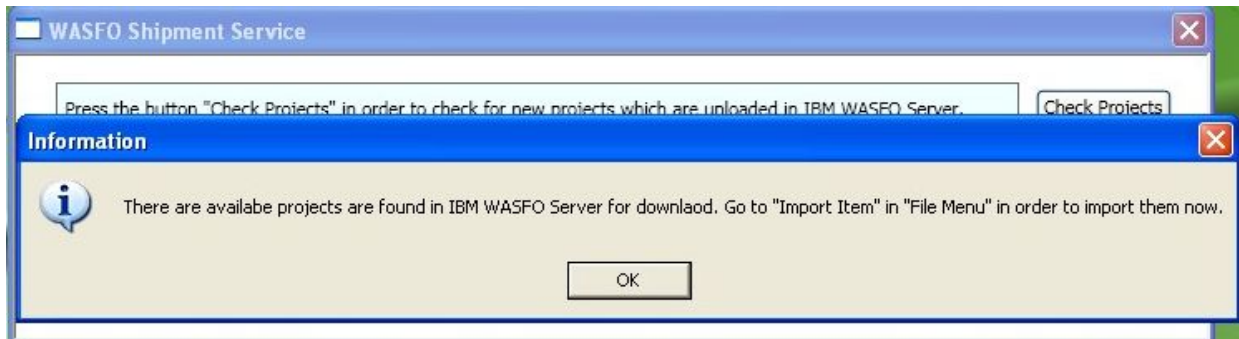


Figure 60 Automatic check in case of any found projects in the web server of WASFO

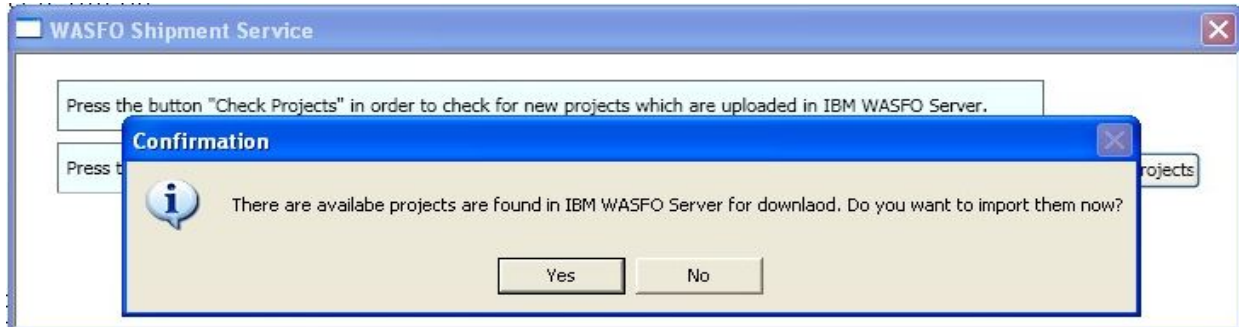


Figure 61 Manual check for any uploaded projects in the web server of WASFO

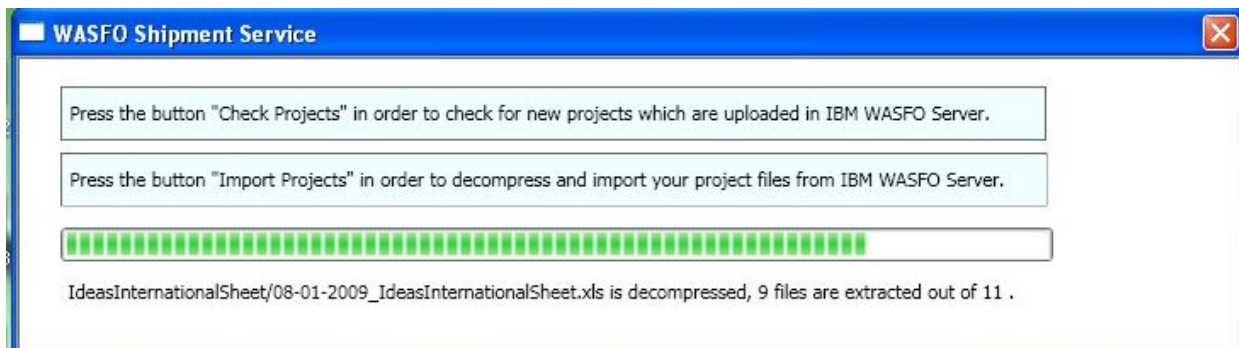


Figure 62 Screenshot of the importing process-step1 (decompressing WASFO projects)

Bibliography

(Europe), Mauro Gatti-IBM System x™ Architect. *IBM Italy, Executive presentation, "Workload Analysis for Server Farm Optimization (WASFO) Version 0.9.1.0"*. Milan.

(Italy), Mauro Gatti-IBM System x™ Architect (Europe) and Salvatore Morsello-IBM System x™ Specialist. *IBM Italy, Education Presentation, "Workload Analysis for Server Farm Optimization (WASFO) Version 0.9.1.0"*. Milan.

AMD. "AMD64 Virtualization Codename "Pacifica" Technology." *Secure Virtual Machine Architecture Reference Manual*. May 2005. <http://www.mimuw.edu.pl/~vincent/lecture6/sources/amd-pacifica-specification.pdf> (accessed April 2009).

Andrea Pagani, Maria Benedetta, Elisa Tonello. *Master Thesis, "An analytical approach to decision problem in IT optimization projects"*. Milan: IBM Italy, 2009.

Bittman, P. Dawson and T. J. *Virtualization changes virtually everything*. Gartner Research, 2008.

Carr, N. G. *IT doesn't matter*. Review, Harvard Business, 2003.

Community of Software development and Design developer, The Code Project. <http://www.codeproject.com/KB/cpp/DimeBufferedUpload.aspx>. <http://www.codeproject.com> (accessed November 2009).

—. <http://www.codeproject.com/KB/cpp/wsaltroute.aspx>. <http://www.codeproject.com> (accessed November 2010).

—. <http://www.codeproject.com/KB/webservices/SoapMSMQ.aspx>. <http://www.codeproject.com/> (accessed November 2010).

—. <http://www.codeproject.com/KB/XML/MTOMWebServices.aspx>. <http://www.codeproject.com/> (accessed November 2009).

Company, Infragistics. <http://www.infragistics.com/dotnet/netadvantage.aspx#Overview>. <http://www.infragistics.com/> (accessed June 2010).

D. Ardagna, C. Francalanci, G. Bazzigaluppi, M. Gatti, F. Silveri, M. Trubian. "A Cost-oriented tool to support server consolidation." *ICEIS Proceedings*. Miami, Florida, USA: ICEIS , 2005.

D. Ardagna, C. Francalanci, M. Trubian. *A Cost-oriented Approach for Infrastructural Design*. ACM Symposium on Applied Computing, Milan: Politecnico di Milano & Università degli Studi di Milano, 2004.

D. Ardagna, C. Francalanci, M. Trubian. *A Multi-Model Algorithm for the Cost-Oriented Design of the Information Technology Infrastructure*. Paper, Milan: Politecnico di Milano & Università degli studi di Milano.

D. Ardagna, C. Francalanci. *A cost-oriented methodology for the design of web based IT architectures*. Paper, Milan: Politecnico di Milano, permitted by SAC Madrid, Spain, 2002.

D. Ardagna, E. Conforti, C. Francalanci, M. Gatti, S. Lucchini, S. Morsello, M. Trubian. Method, system and computer program for configuring server farms at minimum cost. Politecnico di Milano - IBM (European Patent) Patent Patent pending EP06123004.

Dr. Mauro, Gatti. *IT Optimization*. 2010. <http://www.itdec.eu/> (accessed June 2010).

Erich Gamma, Richard Helm, Ralph Johnson, John M. Vlissides. *Design Patterns, Elements of Reusable Object-Oriented Software*.

Friedman, Mark. "The reality of Virtualization for Windows Server." paper.

Gatti, Dr. Mauro. "Planning IT Infrastructure Change in Server Virtualization Projects with the WASFO Tool." *Conference Estimating Exchange*. IBM Academy, 2008.

Gil Neigher, Amy Santoni, Felix Leung, Dion Rodgers, Rich Uhlig. "Intel Virtualization Technology: Hardware Support for Efficient Virtualization." *Intel Technology Journal*. August 10, 2006. <http://download.intel.com/technology/itj/2006/v10i3/v10-i3-art01.pdf>.

Horton's, Ivor. *Beginning Visual C++ 2008*. Wiley Publishing, Inc., 2008.

IBM Italy, Mauro Gatti, IBM System x™ Architect (Europe) and Salvatore Morsello, IBM System x™ Specialist. *Presentation, Education, "Workload Analysis for Server Farm Optimization (WASFO) Version 0.9.1.0"*. Milan.

J. S. Robin, C. E. Irvine. *Analysis of the Intel Pentium's Ability to Support a Secure Virtual Machine Monitor*. Paper, U.S. Air Force & Naval Postgraduate School.

M. Salsburg, P. Karnazes, B. Maimone. *It May Be Virtual - But the Overhead is Not*. March 2008. http://www.cmg.org/measureit/issues/mit39/m_39_1.html (accessed March 2009).

Marco Antonello Tromboni, Luca Terribile. *Master Thesis, "Una metodologia e uno strumento software per la progettazione ottimale di una server farm virtualizzata"*. Milan: Politecnico di Milano & IBM Italy, 2008.

Mauro Gatti-IBM System x™ Architect (Europe), Salvatore Morsello-IBM System x™ Specialist (Italy). *IBM Italy, Exercises presentation, "Workload Analysis for Server Farm Optimization (WASFO) Version 0.9.1.0"*. Milan.

MSDN, Microsoft. *Microsoft.NET Resources*. <http://msdn.microsoft.com/en-us/default> (accessed May 2010).

Open Source Project Community, CodePlex. <http://dotnetzip.codeplex.com/>. <http://www.codeplex.com/> (accessed June 2010).

Tchango, Arsene FANSI. *Master Thesis, Servers Consolidation Problem Models and Algorithms Application to the IBM WASFO Tool*. Milan: IBM & Politecnico di Milano, 2010.

Wikipedia. *Design Patterns*. http://en.wikipedia.org/wiki/Design_Patterns/ (accessed April 2010).

—. *MTOM*. http://en.wikipedia.org/wiki/Message-oriented_middleware (accessed May 2010).

—. *Software Deployment*. http://en.wikipedia.org/wiki/Software_deployment (accessed April 2010).

—. *Software Design*. http://en.wikipedia.org/wiki/Software_design/ (accessed May 2010).

—. *Software Development*. http://en.wikipedia.org/wiki/Software_development_process (accessed April 2010).

—. *Software Requirements*. http://en.wikipedia.org/wiki/Software_requirements_specification (accessed April 2010).

—. *Software Testing*. http://en.wikipedia.org/wiki/Software_testing (accessed April 2010).

—. *Systems Architecture*. http://en.wikipedia.org/wiki/Systems_architecture (accessed April 2010).

—. *Visual Studio*. http://en.wikipedia.org/wiki/Visual_Studio (accessed May 2010).

—. *Web Services*. <http://en.wikipedia.org/wiki/Webservice> (accessed April 2010).

VMWARE. *VMware Resources*. <http://www.vmware.com/resources/techresources/> (accessed April 2010).