OBJECT ORIENTED DESIGN AND IMPLEMENTATION OF A WEB BASED DISTRIBUTED SIMULATION AND CONTROL SYSTEM

by

Mahmut Kurşun

BS. in M.E., Boğaziçi University, 1997

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
The requirements for the degree of
Master of Science
in
Industrial Engineering

Boğaziçi University

2001

ACKNOWLEDGMENTS

I would like to express my gratitude to Ali Tamer Ünal, for his continuous support and his indispensable advice throughout my study.

I would also like to give special thanks to Orhan Kayalar and Park Holding for their support.

I am thankful to my family, and especially to my mother, for their continuous support throughout my entire study life.

My last but not the least thanks are to Yasemin Savcı, who was always with me for all those years.

ABSTRACT

OBJECT ORIENTED DESIGN AND IMPLEMENTATION OF A WEB-BASED DISTRIBUTED SIMULATION AND CONTROL SYSTEM

In this study we have presented an object oriented design and implementation of a web based distributed simulation and control for real time systems. We have developed a system, which enables both controlling and simulating of any given manufacturing environment at any scale, from anywhere in the world via internet in an asynchronous manner. All the manufacturing components in a factory, or the manufacturing environment as a whole, can be represented with the help of the system's virtual manufacturing component objects in a highly customizable form. The proposed architecture has the ability to minimize the execution load of a complex simulation by distributing the simulation to many physical computers, and can simulate third-party simulator objects designed according to the system standard when looked at the system simulation perspective. And when looked from the control perspective, one will see that the proposed system enables the control of any manufacturing machine of a factory from any source, which can either be the web based clients, or a third-party enterprise-wide software running anywhere.

In the proposed system, the developed Real Life Machine Server objects represent the counterparts of real manufacturing components. With the help of the developed DCOM/Proxy Application Server software, a seamless communication between the web based clients or any other third party software, and the virtual manufacturing components is established. Finally the WebFrontEnd application software represents the proof of concept for the integration of web based clients to the system.

ÖZET

NESNE YÖNELİMLİ WEB TABANLI DAĞITIK BENZETİM VE KONTROL SİSTEMİNİN DİZAYN EDİLMESİ VE GERÇEKLEŞTİRİLMESİ

Bu çalışmada gerçek zamanlı sistemlerin benzetimi ve kontrolünü sağlayabilecek nesne yönelimli web tabanlı dağıtık bir altyapının dizaynını ve gerçekleştirilmesini sunduk. Geliştirdiğimiz sistem sayesinde herhangi bir büyüklükteki herhangi bir üretim ortamının dünyanın herhangi bir yerinden internet vasıtasıyla ve asenkron biçimde benzetimi veya kontrolü mümkün hale gelmiştir. Bir fabrikanın tüm üretim komponentleri, veya üretim ortamının kendisi, sistemin yüksek derecede özelleştirilmeye müsait sanal üretim komponenti objeleri ile temsil edilebilir. Benzetim perspektifinden bakıldığında, ortaya konan mimari kompleks bir benzetimin çalıştırılma yükünü, o benzetimi bir çok fiziksel bilgisayara dağıtarak hafifletebilme yeteneğindedir, ve üçüncü kişilerin sistemin standartlarına uygun yarattığı objeleri de benzetimde kullanabilir. Ve kontrol perspektifinden bakıldığında ise, ortaya konan sistem bir fabrikanın herhangi bir üretim makinasının her hangi bir kaynak, ki bu web tabanlı kontrolörler veya üçüncü kişiler tarafından işletme-geneli için geliştirilmiş herhangibir yerde çalışan bir yazılım olabilir, tarafından kontrol edilebilmesini sağlamaktadır.

Ortaya konulan sistem dahilinde geliştirilen Gerçek Hayat Makine Sunucu objeleri gerçek üretim bileşenlerini temsil etmektedir. DCOM/Proxy Uygulama Sunucu yazılımı sayesinde web tabanlı kontrolörleri veya üçüncü kişi uygulamaları ile, sanal üretim bileşeni objelerinin kesintisiz iletişimi sağlanmaktadır. Son olarak da WebÖnYüzü yazılımı web tabanlı kontrolörlerin sistem ile entegrasyonu fikrinin kanıtsal uygulamasını oluşturmaktadır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

ASP	Asynchronous Simulation Protocol or Active Server Pages
CMB	Chandy, Misra, and Bryant a parallel simulation protocol
CORBA	Common Object Request Broker Architecture
DCOM	Distributed Component Object Model
ERP	Enterprise Resource Planning
GUI	Graphical User Interface
HLA	High Level Architecture
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IIOP	Internet Inter-ORB Protocol
MAP	Manufacturing Automation Protocol
MMS	Manufacturing Message Specification
MPI	Message Passing Interface
MSMQ	Microsoft Message Queue Server
OMG	Object Management Group
ORB	Object Request Broker
OSI	Open System Interconnection
RAD	Rapid Application Development
RPC	Remote Procedure Calls
ROT	Running Object Table
TCP/IP	Transmission Control Protocol/Internet Protocol
VMD	Virtual Manufacturing Device
XML	Extensible Markup Language

1. INTRODUCTION

There is a growing recognition that current manufacturing enterprises must be agile; that is, capable of operating profitably in a competitive environment of continuously changing customer demands.

The advances of information and factory automation technology in manufacturing systems have been remarkable in recent years. Manufacturing systems are being created on larger and more complicated scales than ever before. Therefore, it has become necessary to accurately predict and control the flow of material and information during the planning and design phase of such a manufacturing system. It is also important to evaluate many alternative plans in a short time due to rapid changes in manufacturing cycles and to keep costs at a minimum. For these reasons, manufacturing system simulators are often used as a support tool to design an actual manufacturing system [1].

Such simulation systems are often specialized stand-alone and/or single purpose systems [2]. To avoid numerous systems, each specialized for a specific task, that have to be maintained and for which the operators have to be trained, a common, unifying approach of interoperability, re-usability is needed. And, as simulators depend on software with particular uses, the software often limits designers to changes determined by the software rather than their purposes or needs. [1]. Also for a large scale manufacturing system whose subsystems are often found at different locations, it will be necessary to facilitate a new technique in developing a simulation system capable of representing such a large system [3].

More or less similar concerns are in the control area. Many researchers [4,5,6,7,8] have proposed various formal models to facilitate the development of control software [5]. While the approaches of these models are usually generic enough to be applied to any kind of flexible manufacturing system configurations, the complexity in modeling and computation and interoperability problems with existing enterprise-wide software has been recognized as the main barrier for their successful implementations in the large-scale, real world flexible manufacturing systems.

Motivated by the need of a highly customizable, reusable, distributed and readily implementable simulation and control systems for industrial, small to large scale manufacturing systems, we choose a different approach in this study and combine simulation and control systems in a message driven web based system. We use tiny virtual manufacturing component objects, which have the primitive methods, those can be found in every type of machine. These objects may act as counterpart of a real-life manufacturing components, or may act as a real-time execution interface of the real-life machines. On top of these, we add a message driven asynchronous communication and management layer between these objects and any other third party enterprise-wide software or our web based control front end for remote location independent clients.

The layout of the study is as follows: In chapter two, the literature is reviewed. The problems are defined in chapter three, and the system is proposed in chapter four. In chapter five the software is documented. And conclusions and discussions are stated in chapter six.

2. LITERATURE REVIEW

In reviewing the literature we focused generally on the simulation systems and control systems and intensely tried to identify some keywords like distributed, reusable, interoperable, asynchronous, web-based, and component-based in them. Our first starting point for this study was the simulation need for the performance measurement of an object oriented distributed scheduling system from a previous M.S. thesis by Oğuz [9]. By reviewing the simulation area especially first we confronted with many different simulation systems, which included both monolithic and distributed systems. The monolithic systems tried to enhance their execution times by parallel simulation technology and by executing them on shared memory multiprocessor systems (CMB-SMP) [10]. Most of the simulation systems also heavily invested on the modeling capability of the systems by some nice graphical user interfaces (GUI). When looked to distributed simulation systems we see several systems, which include RTI (Run-time Infrastructure) of the HLA (High Level Architecture)[2,11], MPI-ASP (Message Passing Interface – Asynchronous Simulation Protocol) [12], DVF (Distributed Virtual Factory) concept that consists of distributed precise simulation models connected by several Time Bucket algorithms [3], and others. After analyzing the object oriented distributed simulation systems, we concentrated on the control systems, which founded on set of distributed resource controllers, and a central system supervisor controlling them. And tried to find and analyze systems that can handle both simulation and control. In the mean time, we heavily invested in the software technology, which can be used to implement these concepts.

2.1. Distributed Simulation Architectures

As manufacturing systems become automated, so-called automation islands have appeared in factories. Nowadays, most manufacturing systems consist of several automation islands, such as direct numerical control systems (DNC), flexible manufacturing systems (FMS), automated cell systems, automated guided vehicle systems (AGV) and so on. In the course of development, many simulation studies have been performed to obtain an effective design of the automated systems. Some simulation

systems developed for those studies were extended for use in operational decision making at the shop floor. This resulted in the advent of simulation islands, each corresponding to the real automation island in the factory.

Besides dealing with material and information flow, the Distributed Virtual Factory (DVF) concept that consists of distributed precise simulation models connected by several synchronization mechanisms named Time Bucket algorithms [3] mainly focuses on the cost analysis of products by implementing ABC (Activity Based Costing). This system is a distributed simulation system, mostly satisfies the structural features of a factory and the requirements for a factory-wide simulation system when it is developed on a distributed computer system installed as the infrastructure of DVF. Each subsystem in the DVF structure at the area level can be modelled on one processor and a transportation system, T-Process, which connects areas, is modelled on one processor transferring works. The functions for information exchange amongst areas are also necessary to model the collection and the dispatch of information occurring time to time. One processor can be allocated to model a global decision making system collecting the ordinary status reports of areas for decision making.

In MPI-ASP [12] the parallel simulation engine CMB-SMP [10], which we handled as monolithic simulation system, is extended as for distributed simulation. It is based on an extended asynchronous simulation protocol [10]. The algorithm was modified by incorporating the MPI library for message passing. MPI-ASP lacks several important features like interoperability and reusability. In the comparison between HLA [12], its speed of execution beats HLA, but as said before HLA beats MPI-ASP in terms of reusability and interoperability.

Currently the only software architecture for heterogeneous simulation-based distributed systems with interoperable and reusable components is the High Level Architecture for Modeling and Simulation (HLA) [2].

The High Level Architecture (HLA) is a general purpose architecture for simulation reuse and interoperability. The HLA was developed under the leadership of the Defense Modeling and Simulation Office (DMSO) to support reuse and interoperability across the

large numbers of different types of simulations developed and maintained by the DoD. The HLA Baseline Definition was completed on August 21, 1996. It was approved by the Under Secretary of Defense for Acquisition and Technology (USD(A&T)) as the standard technical architecture for all DoD simulations on September 10, 1996. The HLA was adopted as the Facility for Distributed Simulation Systems 1.0 by the Object Management Group (OMG) in November 1998. The HLA was approved as an open standard through the Institute of Electrical and Electronic Engineers (IEEE) - IEEE Standard 1516 - in September 2000. The HLA MOA was signed and approved in Nov. 2000 [11].

The HLA is defined by [2]:

i. Rules which govern the behavior of the overall distributed simulation (Federation) and their members (Federates).
ii. An interface specification, which prescribes the interface between each federate and the runtime infrastructure (RTI), which provides communication and coordination services to the federates.
iii. An object model template (OMT) which defines the way in which federations and federates have to be documented using the Federation Object Model (FOM) and the Simulation Object Model (SOM), respectively, federations can be viewed as a contract between federates about how a common federation execution is intended to be run.

The time management services (one of six service groups provided by the HLA interface) allows the transparent execution of federates under different time regimes (e.g. real time, time stepped, event driven, continuous). Even a mixed combination of conservative and optimistic synchronization policies is supported which allows the exploitation of the optimistic approach among `optimistic' federates [2].

Federates have to comply with the HLA specifications: as far as the RTI is concerned, they are treated as black boxes, hiding their true functionality: simulations, live players (man-in-the-loop and hardware-in-the-loop) as well as management tools, passive viewers, sensor platforms and information systems can participate [2].

The RTI software package consists of libraries linked to the federates, a central RTIExecutive process (RTIExec) which provides a naming service and acts as a first point of contact for federates as a well-known service, and an FederationExecution process (FedEx) which is created for each federation execution [2].

Although HLA is now a stable standard and the number of military applications based on HLA is growing rapidly, civil applications remain the exception.

2.2. Object Oriented Control of Manufacturing Systems

The possible operation model of a manufacturing system is designed during the simulation phase by means of evaluating the simulation / scheduling results. These are the results that assist in getting the 'best' manufacturing system model, too. If the 'best' or an appropriate model was chosen for implementation and the implementation is done the next task is the operation (control) of the system [13].

Recent discrete manufacturing and / or assembly systems (FMS / FMA) are more and more often using MAP / MMS (Manufacturing Automation Protocol / Manufacturing Message Specification) [14], because this technology is widely available from many vendors and really gives a safe and open solution according to the demands of OSI (Open System Interconnection). Many users do not exactly know that they have such interconnections, they just enjoy the useful features of MAP [13].

Our proposed systems functionality is very similar to the SSS (Simulation and Scheduling System) [13] In this study SSS were designed and developed in a way that later the simulation might be changed to a real FMS environment, and the expert system (ES) would be the real-time controller. The prototype applications of SSS were developed with the real data layout, capacities, process plans, machine parameters, etc. of some academic and industrial FMS. The application specific and independent parts were separated in the expert systems [13].

The practical problems of the communication of expert systems in CIM applications can be divided into two parts. One is the hardware–software connection (physical) and the

other is the logical one between the controller(s) and controlled devices [13]. If this decomposition is not so sharp many problems may occur during the development and specially in maintenance of the software later on [13].

Most controller and controlled device vendors offer good (proprietary) solutions to communicate and also vendor independent standards are available. In the CIM area there are more accepted models or modeling tools to describe the objects of an FMS. In the communication point of view the most promising one is the object oriented view of the so-called MMS (Manufacturing Message Specification), which is originally an application layer protocol in the MAP OSI networks. MMS gives a so-called VMD (Virtual Manufacturing Device) view about each resource of the FMS. It was realized that this specification is good on the higher level of the FMS to give a communication oriented view about the network elements and their resources [13].

The object-oriented view of MMS allows to design the VMD model of a certain device, and it is immediately possible to use this model as a specification of the communication where the services (what one can do with a given object of the VMD) are defined and they are working in the MAP networks [13].

A generic event control framework for a class of modular flexible manufacturing systems is analyzed in [5]. In this study the reconfigurability of control system for various FMS implementations and control policies is achieved, through a control framework, which is defined as a set of distributed resource controllers and a central system supervisor coordinating them. The resource controllers are further classified into workstation, transporter, and stocker controllers. As the controllers exchange a series of events according to pre-defined protocols, they are modeled as event handlers in which control actions are made based on the event occurrences. Specifically, for each controller, an event-based control structure specified in terms of generic logical and performance control functions, is presented.

2.3. Composition Approach and Reusability

The composition approach tries to identify functional, geographical or otherwise specified components which could be implemented as separate software (and hardware, if necessary). Often components of one system will be similar to those of existing systems, so that reusability of components will be beneficial. If the functionality of an overall system is split into components, then the communication and coordination between components must be addressed. In order to minimize bandwidth and latency requirements, the separation into components requires careful consideration of this issue [2].

Furthermore, the composition approach exploits the black-box character of components by using sets of components based on different implementations where appropriate. This results in one function (e.g. real-time environmental monitoring) to be performed by a monitoring component which is available as a `real' component (connected to the real-world monitoring system), as a simulated component, and as a play-back component (both `off-line' components based on different sources) [2].

While the real component is restricted to real-time environments, the other implementations can also be used in an `as-fast-as-possible' environment (e.g. for forecasting and analysis). This implies that the time management of the component (for the monitoring component: real time) often has to be extended to other time management schemes (such as discrete event, time stepped, or hybrid methods) [2].

2.3.1. Components

Components are software modules which are self-contained, are usable by themselves (stand alone), and are completely defined by their interface to the outside world [2]. The interface describes the information needed to operate the component as well as the information the component is able to provide; at the same time, the name space (what names are used for objects, properties, etc.) is defined (and should be compatible with the rest of the system) [2].

This 'black-box' property creates the opportunity to define a set of components which are identical in interface but different in 'nature', such as one component being implemented as a connection to a real-world information-generating process, another one being based on simulation, etc.

2.3.2. Reuse

Software developers often meet the problem of creating new components of an application that someone probably previously has already produced. Without having effective reuse tools, usually it is more natural to create new components from scratch than to seek for useful elements in other programs and / or systems [13].

In the field of simulation and control of flexible manufacturing systems this issue often occurs, when different systems with some similar features have to be managed. The basic components of different FMS and FMC (Flexible Manufacturing Cell) are the same type of machine tools, robots, transfer equipment, etc. In the relevant aspects they usually differ from each other only in their quantity and working parameters. This fact itself breeds the idea of reuse of FMS and FMC elements [13].

There are several elements defined during the analysis, design and implementation of a simulation model, as ideas, concepts, object classes, and lines of source code created, etc., that should be reused in new applications. Application of reuse methodology and practice will reduce the effort in developing new simulation models to assist the design of new systems. According to the European Software Institute the additional cost of producing reusable software is about 20 per cent, while the cost reduction by reuse is about 40 per cent in average. The same study states that 2–5 uses reuses result in a payback of the investment [13].

Our virtual manufacturing component objects represent the previously mentioned FMS' basic components, and thanks to the DCOM architecture our model can be reused even in the binary level, besides source code level.

2.4. Web-based Simulation

Web-based simulation is an emerging theme in simulation research and practice. Driven largely by the phenomenal growth in the world wide web (WWW) and its attendant technologies, it is tempting to view web-based simulation as nothing more than a technology push. To a certain extent, it is just that. A significant portion of the literature surrounding web-based simulation involves a re-dressing of the same old emperor in new technological garb. However, a few researchers in the field have described the possibility for the web to fundamentally alter the practice of simulation modeling [6].

The term web-based simulation emerged in the mid-1990s, although the exact pedigree of its coinage is unknown. Within the modeling methodology track of the 1996 Winter Simulation Conference, the topic of web-based simulation is formally introduced to the simulation community at large. Several potential impacts of web technologies on simulation, giving particular attention to education and training, publications, and simulation programs areas are described in this conference. The proliferation of web content is described as a "kind of twenty-first century gold rush" and simulation researchers and practitioners admonished to be proactive in defining the relationship of the web and simulation [6].

3. PROBLEM DEFINITION

For large scale manufacturing systems as multinational giants whose plants are often geographically distributed worldwide, it will be necessary to facilitate a new technique in developing a simulation and control system capable of representing such a large system.

In designing complex and large-scale systems, cost and time usually prohibit physical prototyping and experimentation with such systems. Accordingly, it is essential that simulation techniques be developed, which allows people to study the behavior of these systems before they are actually built and to evaluate design alternatives.

Existing software development environments for discrete-event simulation have adopted either a language-based approach or a library-based approach. In either case, programmers are provided with a set of model definition primitives together with a set of distributed programming primitives for object definition and inter-object communication and synchronization. These primitives are provided either as language extensions or as functions implemented as library routines. These approaches have advantages such as type checking and optimized code generation provided by the former and familiar programming environments facilitated by the latter.

However they suffer from the following three major limitations:

- **Lack of portability**. Simulation models developed using one simulation language or library might not be easily ported to another environment. The programmers will be required to learn new language constructs and perhaps an entirely new set of program development tools.
- **Lack of interoperability**. Components of a simulation model are required to be programmed in the same host language dictated by the simulation language or library routines used.
- **Lack of capability to execute over the Internet and the Web**. Simulation models either perform sequential execution on a single workstation or run on parallel

computers or LAN-connected workstations, which cannot scale over the Internet and the Web infrastructures.

The former two limitations may lock simulation application development into a particular environment and result in inflexible models and higher model development cost, while the latter may jeopardize the capability of deploying large-scale simulation applications over the Internet and the Web infrastructures.

Another important issue is the management of such a large system. It has to be also possible to control every single manufacturing machine regardless of where it operates. The system has to be able to identify this manufacturing machines for controlling purposes also as it has to identify it for simulation purposes.

Let's identify this problem in an example, and say we are a multinational company called MultiNat Inc. MultiNat Inc. has various manufacturing plants worldwide. Let's say it has its head quarter in Atlanta, U.S., and plants in Rome, Italy and Istanbul, Turkey. This system has to be able to manage and control such a distributed environment. Let's have an exaggerated example, and say that the manager in the Headquarter in Atlanta want to control and monitor a specific CNC machine located in Istanbul, Turkey. In the meantime the member of the board of MultiNat is on vocation in France, but reported that there is a problem in the Rome plant in Italy. But unfortunately he cannot find a computer even a laptop to access internet, but thanks God, he has his WAP enabled phone. He quickly login to the system via the WAP interface, and monitor the current situation in the Rome plant. Or take a case, where our multinational company MultiNat Inc. want to replace its current scheduling software in its headquarter, but before the implementation the managers want to be sure that the new system will fit their system perfectly. So what to do now, they have to simulate the current infrastructure to see whether the newly proposed system will fit well in to the system, so they run the software on the virtually designed environment. And to be able to colloborate any software system the current infrastructure has to use well-known standards.

Now let's define the problems stated in the previous example:

- The system has to be able to work securely and reliably over the world wide web.
- The system has to be composed of components each acting as real life components,
- as one may also define even a small manufacturing machine.
- The system has to communicate securely and reliably with each of its components, also with each of the subsystems.
- The system has to use a technology that is built upon the set of common standards.
- The system has to provide interfaces to all possible devices, as any kind of computer, Palm or WAP enabled cellular phones.
- The system has to provide interfaces to all possible type of computers with all possible operating systems. Sun Sparc with FreeBSD, Unix as the operating system, Macintoshes with MacOS as the operating system or a Desktop PC or Laptop with Microsoft or Unix as operating system has to be the possible candidates who can use the system.
- The system has to be robust to every possible downtime arising from connection problems, or other internal problems.

4. PROPOSED SYSTEM

4.1. Technology Overview

With these problems in our mind, we propose a state of the art system that has an object oriented and distributed infrastructure, and uses set of standards that are widely accepted as common standards.

The system is composed of three layers, client's web browser or the GUI layer, DCOM-Proxy/Application Server layer, and DCOM objects layer.

We propose a system composed of objects, which will represent their real life manufacturing component counterparts. With real life manufacturing components we mean machines, jobs, whatever related with manufacturing, and give these objects 'black-box' property [11], which creates the opportunity to define a set of components which are identical in interface but different in 'nature', such as one component being implemented as a connection to a real-world information generating process, another one being based on simulation, etc.

These objects have to be distributed over a TCP/IP network, and have to talk with the controlling application on a standard language. There are two competing distributed object technologies named CORBA (Common Object Request Broker Architecture) and DCOM (Distributed Component Object Model). A brief explanation of these technologies are presented in Appendix E. We choose DCOM as the object communication technology in our system.

A user will able to interact with the system by using Internet. To provide Graphical User Interface (GUI) the web browser is used, regardless of operating system, platform or browser vendor. The only constraints of the web browser are that it has to be able to interpret Javascripts and be able to understand frame structure. This leads us to use browsers IE4.0+ and Netscape 3.0+ for Microsoft Windows Platform, and Netscape 3.0+ for other platforms.

The interaction of web browsers with the system's internal DCOM objects is provided with a middle layer called DCOM-Proxy/Application Server. In order to ensure robustness to communication faults, an asynchronous messaging system is implemented as the messaging infrastructure between the web browser clients and the main system. To not re-invent the wheel, a custom made messaging system is not implemented, on the contrary a well-known messaging infrastructure is used. We choose Microsoft Message Queuing (MSMQ) mechanism as the messaging infrastructure. With its transactional architecture MSMQ will provide queue mechanism for both clients, and the server application.

The objects and users data will be stored in a database. Again to force the usage of common standards, we select the ODBC (Open Database Connectivity) structure as the database communication protocol. With the help of ODBC standard we will use any RDBMS (Relational Database Management System) compliant database. Some of possible choices include Oracle Server, Sybase Adaptive Server, Microsoft SQL Server,... etc., and thanks to the ODBC standard replacing a RDBMS will not require any code change in the developed system programs. Any other software will communicate with our storage system in case of necessity with an ODBC connection.

ASP (Active Server Pages) scripting language is used in the Web-Front-End Software, and IIS (Internet Information Server) is used as the web server. The web based application called "WebFrontEnd" is developed with Microsoft Visual Interdev. The DCOM-Proxy/Application server is developed with Microsoft Visual Basic 6.0, a rapid application development (RAD) tool.

The security issues are solved by placing a secure web server on the DMZ (De-militarized Zone), behind the firewall.

To satisfy programming language and system independence for the sake of completeness some remarks have to be made. There is no constraint for the further development of this system with any other language since the DCOM, and MSMQ architectures provide COM (Component Object Model) interfaces, which could be called through any Win32 application. Also, the ASP scripting language may be replaced with an open source alternative PHP (Hypertext Preprocessor) scripting language easily, since the

new PHP4 version for Win32 platforms has a built-in COM API. The web server may be replaced with an open source alternative Apache Web Server for Win32 platforms, but at the time of writing this document the Apache software was at the beta version for Win32 platforms. The MSMQ mechanism is not dependent just for the Microsoft platform, it can be used also on the IBM AS 400 platforms, and support for Unix based systems are currently in the development. And for the last word, the DCOM objects are not dependent on the Microsoft platform, since current developments to port DCOM objects to Unix and other operating systems succeeded.

As it can be seen the proposed systems technological view represents a platform independent, object oriented and distributed, reliable, scalable and robust system.

4.2. User Based Overview of the System

In this section the architectural overview of the system will be presented. The graphics mostly obeys to standard UML (Unified Modeling Language) Use Case Diagram rules. The system is presented in Figure 4.1. The system can be composed of four main functional use cases, which are Access Control System, Send Control Command, Get Status, and Select Accessible object. The actors are Manager/Controller, Objects DB, System Administrator and Messaging System. The manager/controller points his/her browser to the internet address of the system. By loading the web page, the system identifies that the current user is not authenticated, and redirects him/her to the login page. The Access Control System use case is presented in Figure 4.2. The manager/controller's web browser is displaying the login page as shown in Figure 4.3.

System asks for the user name and password. And Manager/Controller inputs his/her user name and password on to the system. The system checks the username and password that is delivered to the system via the user database. If the system authorizes current user, the system redirects him/her to the main control page, otherwise the system redirects him/her to the same login page.

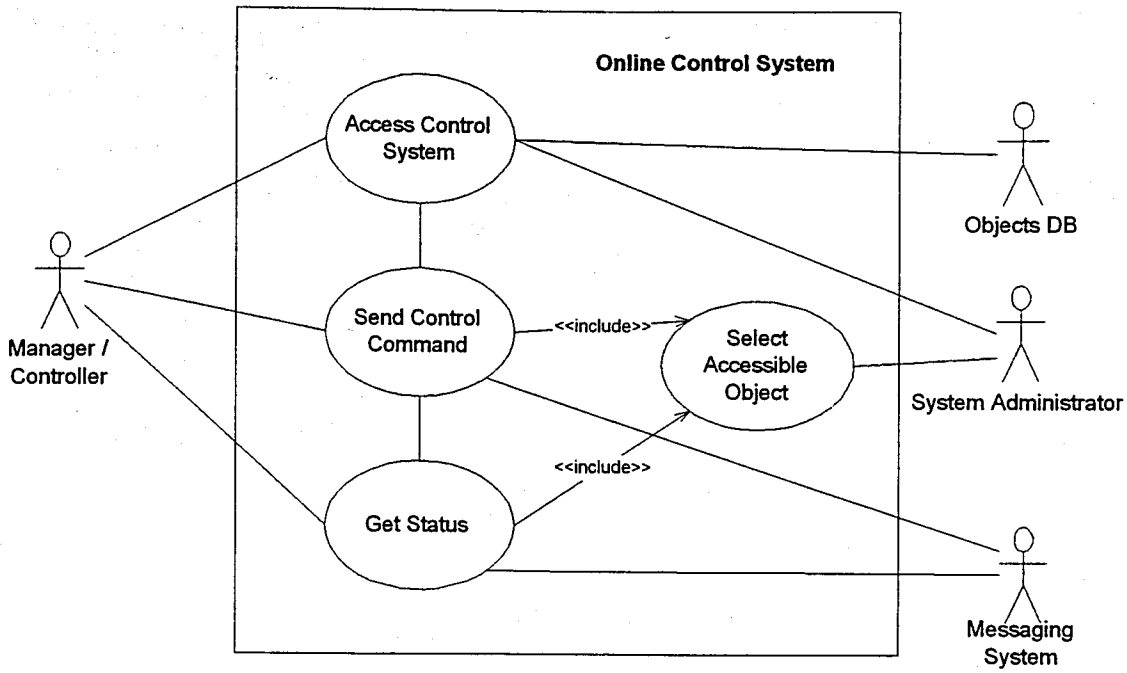

Figure 4.1. Online control and simulation system use case

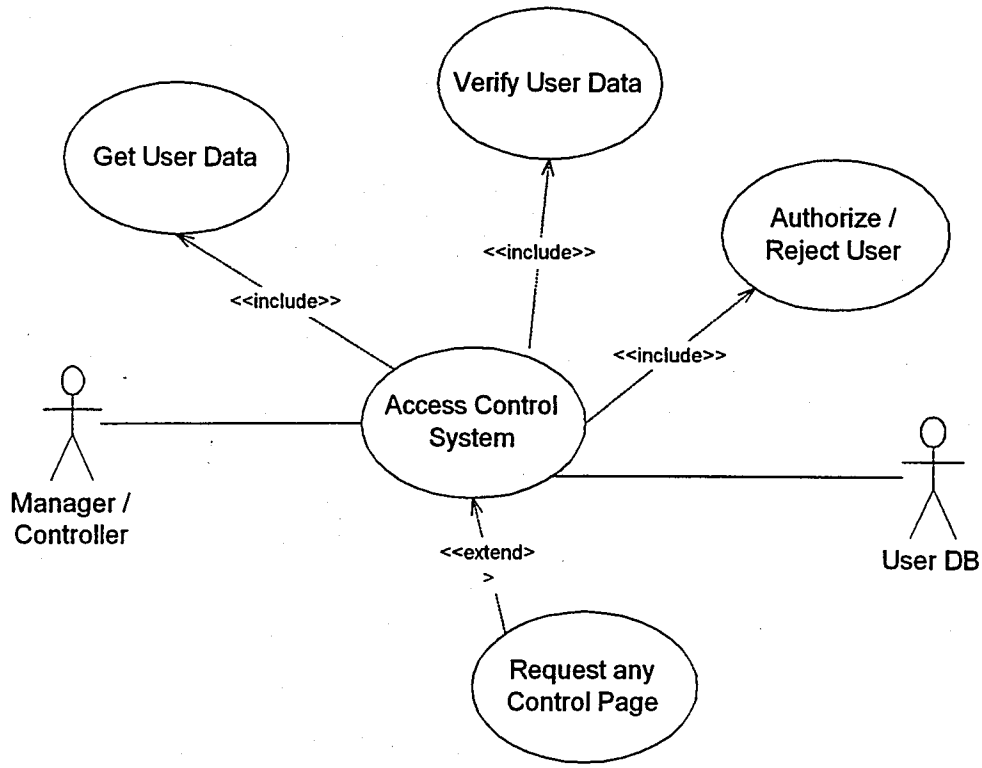

Figure 4.2. Access control system use case

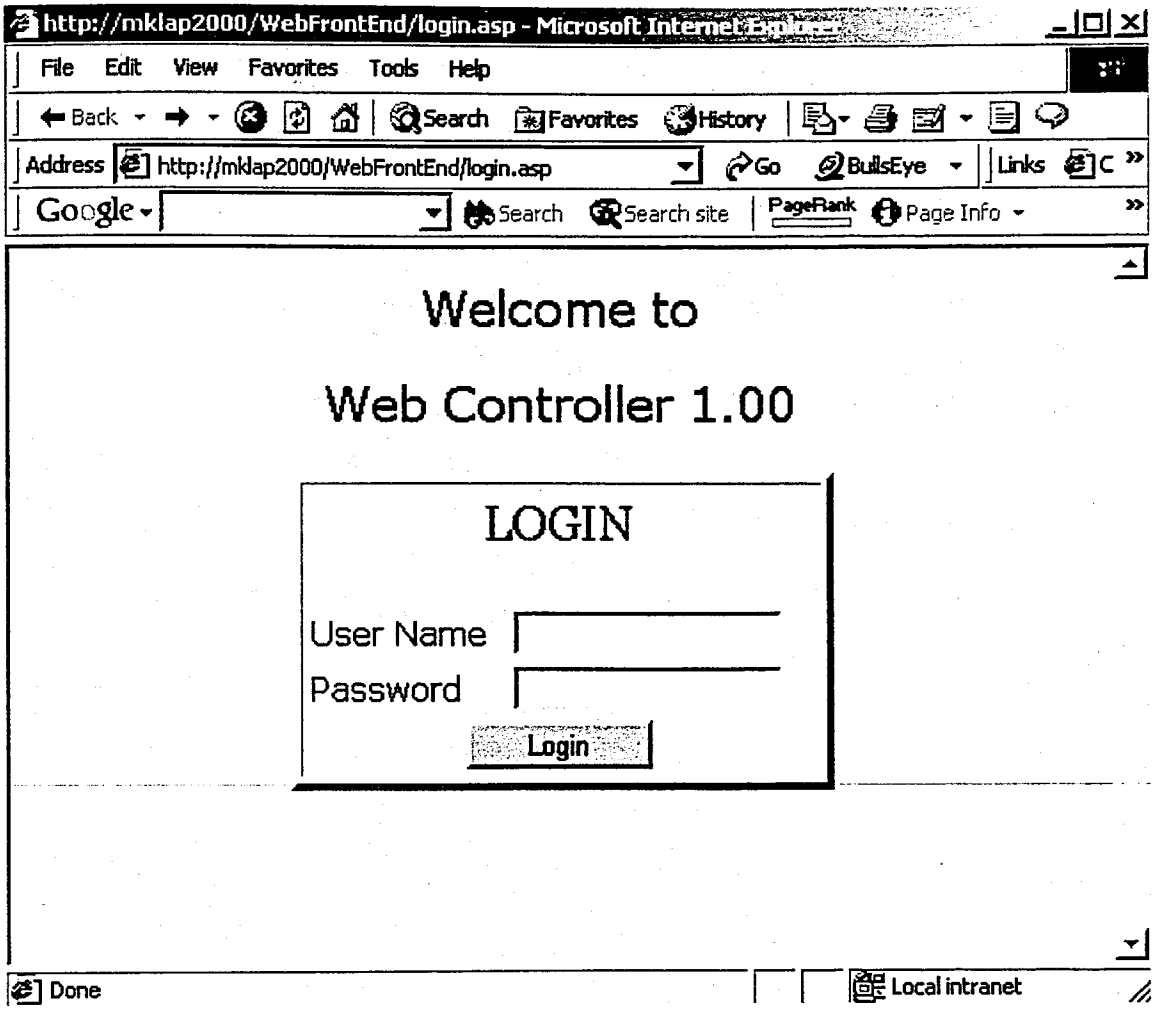

Figure 4.3. Login GUI of the system

Manager/Controller is authenticated through the login page, and redirected to the main control screen. All of the frames in the current screen are on their initial states; i.e. showing only welcome information. The main menu of accessible objects list is in its initial state; i.e. showing only types of available objects (machines, jobs,...) as shown in Figure 4.4.

The label 1 on Figure 4.4 shows the title of the web browser, which is the GUID[1] number. All clients requesting the control page are assigned a GUID number to be

[1] GUID stands for Globally Unique Identifier, a 128-bit (16-byte) number generated by an algorithm designed to ensure its uniqueness. This algorithm is part of the Open Software Foundation (OSF) Distributed Computing Environment (DCE). A GUID is a 128-bit value consisting of one group of eight hexadecimal digits, followed by three groups of four hexadecimal digits each, followed by one group of twelve hexadecimal digits. [17]

uniquely identified. By requesting this control page, a communication channel in the messaging system will be created, and allocated for the incoming client. The details of this mechanism will be explained in the messaging infrastructure section. The frame with the menu of objects is labeled with A. The virtual manufacturing components are displayed according to their types in a tree-view in this menu. For our system we used two types of objects named Jobs, and Machines. But for the ease of understanding we will mainly use the machines type in this study. The virtual machines are listed under the type of machines with their physical name followed by their physical location on the current network. As an example the first machines physical name is "Torna" and its physical location is on the physical computer called "mklap2000", and therefore it is listed as "Torna-mklap2000".

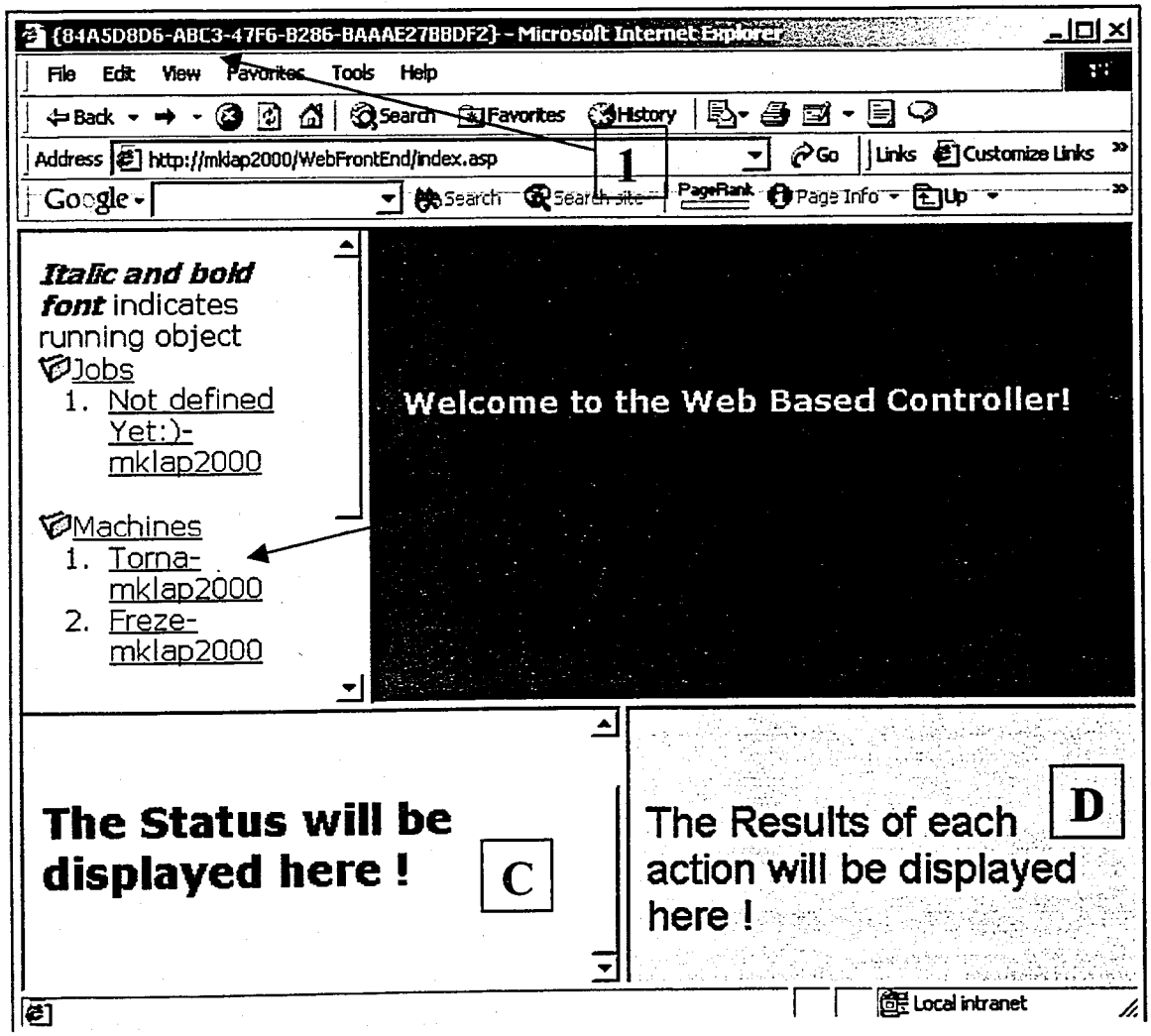

Figure 4.4. Initial state of the main controller screen

The frame B will represent the whole control related functions. The status info will be displayed on frame C, and the results of each function call will be displayed on frame D. If the function calls succeed, a log of the last action will be displayed on frame D, otherwise it will present the detailed error code.

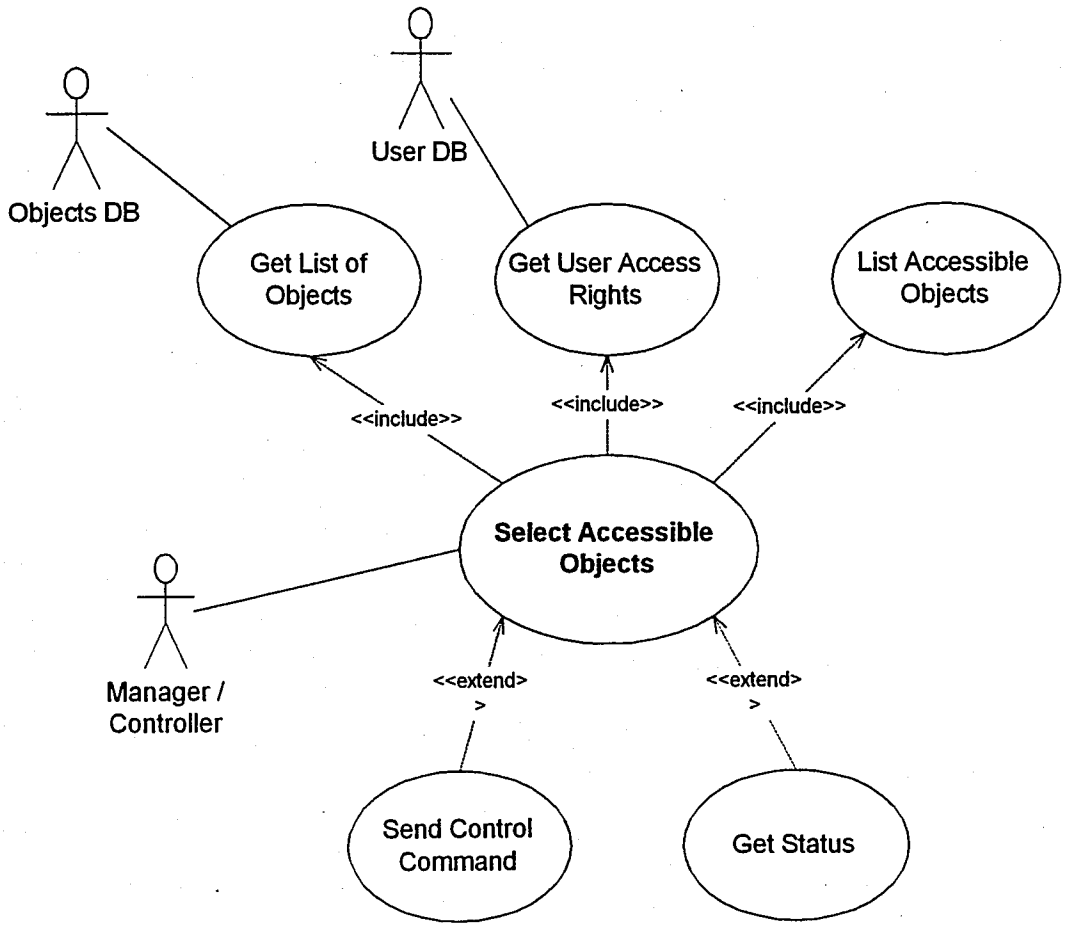

Figure 4.5. Select accessible objects use case

The "Select Accessible Objects" use case is presented on Figure 4.5. It is both used on the "Send Control Command" and "Get Status" transactions. In the proposed system an access right value is assigned to every object. The users of the system also have their access right values. In this transaction the system retrieves the list of objects from the objects database. It also retrieves the object access rights of the current user from user database. The system will list only the objects with access right value less than or equal to the current users access rights value. If the current users access right is not enough to list any object, than the current Manager/Controller will not be able to control any of the system objects.

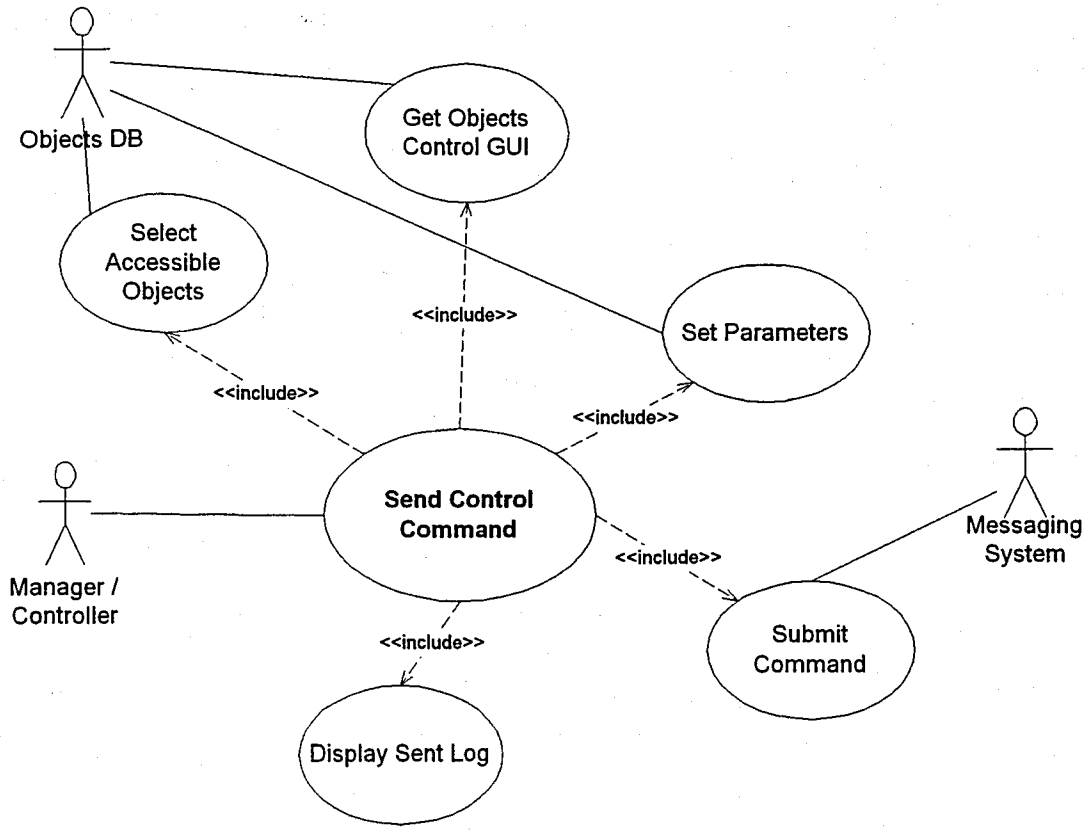

Figure 4.6. Send control command use case

The "Send Control Command" use case is presented on Figure 4.6. Manager/ Controller selects from list of accessible objects an object. The selected objects control table is displayed by selecting an accessible object with the current state information retrieved through the objects database. The current controllable state is displayed on the Objects Control GUI as shown in Figure 4.7. If the object has been loaded / initialized before, the Manager/Controller may not set any new parameters. If the selected object has not been loaded / initialized before, Manager/Controller sets appropriate parameters of the selected object through objects control GUI. Manager/Controller pushes the submit button, and submits his/her command through the Objects Control GUI's submit buttons.(load, start, pause, stop). By pushing the appropriate button, the relevant information is sent to the messaging system. After sending the command, the last operations log is displayed on the log window as shown in frame D of Figure 4.4.

In the following case represented on Figure 4.7, manager/controller selects "Torna on mklap2000" for controlling. The system identifies that no other manager/controller controls this object currently, and therefore not initialized before. So the system presents the available set of parameters for the initialization of this object on the Objects Control GUI, represented on frame B in Figure 4.7. There are default values for these parameters, which are stored on the objects database, and these values will be displayed to the manager/controller for the sake of simplicity. If the manager/controller now pushes any of the four buttons, namely Load, Start, Pause, Stop button, the Send Control Command transaction will begin. There is one thing to note here. Although in the whole of the system even the states of the buttons are ordered according to objects real state, this is not performed for the first initial state.

If the Manager/Controller do not pushes any of the Objects Control GUI's submit buttons, the current page will be displayed indefinitely, but after 15 minutes the session timeout will occur, and the connection of the current browser will die.

The "Get Status" use case is represented in Figure 4.8. The Manager/Controller selects again from list of accessible objects an object. The selected objects control table is displayed by selecting an accessible object with the current state information retrieved through the objects database. The Manager/Controller pushes the Objects Control GUI's status button, and submits his/her status command. By submitting the status button the status window, the frame C on Figure 4.4, begins to request status information from the messaging system at every predefined interval. So The status is displayed on the status window.

One thing has to be clearly identified in the "Get Status" transaction. The system does not send any status request to the system, i.e. to the DCOM-Proxy/Application Server. It just loads the frame C shown in Figure 4.4 with an ASP page capable of requesting itself at the specified intervals. The requested ASP page will get the relevant information from the messaging system, and will display the current status of the selected object. That means physical communication will be established only at this specified intervals, or at the send command transaction intervals.

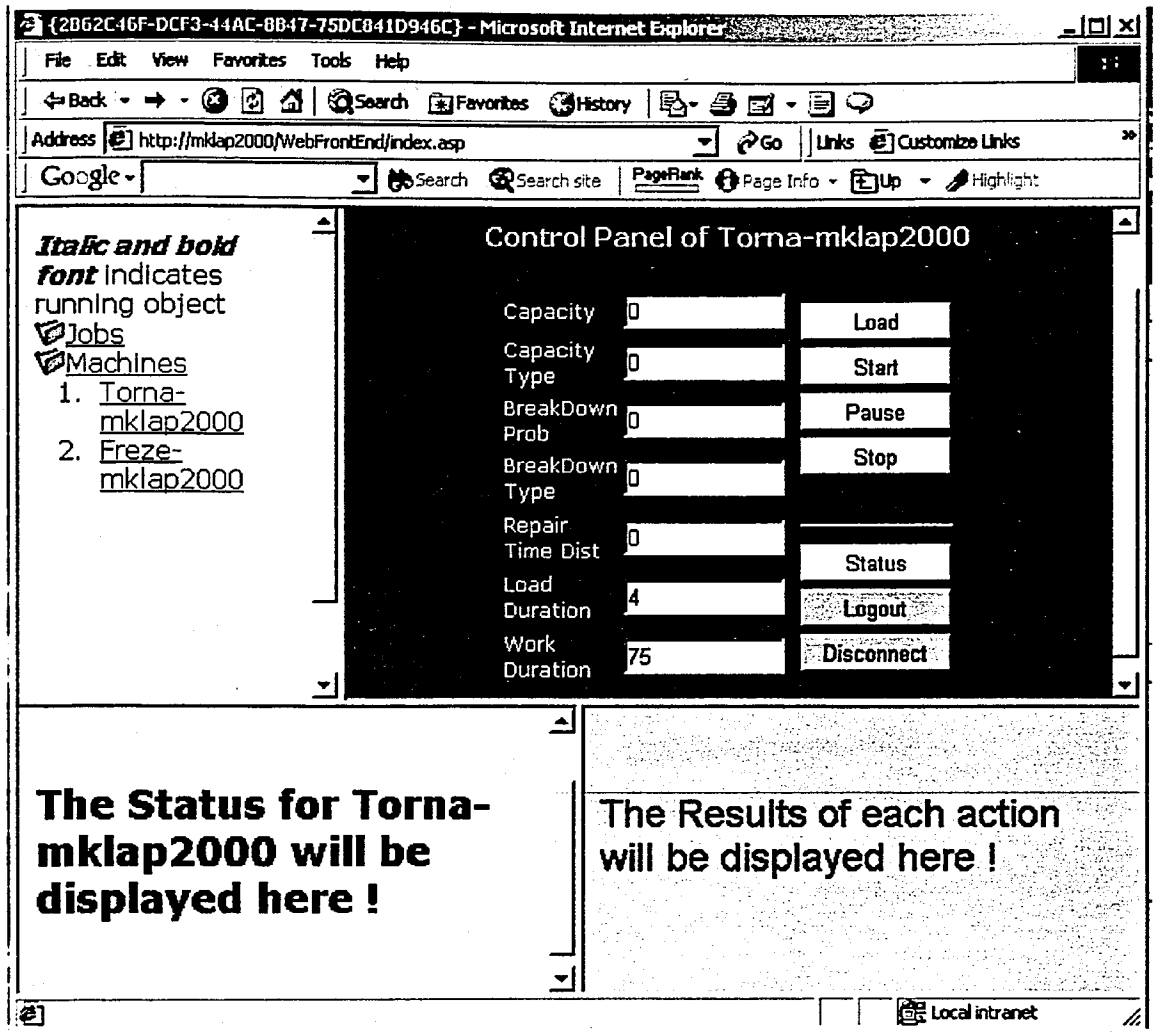

Figure 4.7. The main control screen

We want to show instances of the system to clarify the execution of the system. Therefore a sample process is initiated. "Torna-mklap2000" is selected as the virtual manufacturing machine. The machine is loaded first with its default variables. Then a start command is sent to the object. After a while a snapshot of the user interface is taken when a stop command was sent to the object and displayed on Figure 4.9. It can be seen that the left menu has updated itself and displayed in frame A in Figure 4.9. The selected machine, or the machines those are operating at that instant are listed in bold and italic font.

The frame B updates itself according to the current state of the virtual manufacturing machine, and displays only the start button. The control interface is a living interface, which updates itself as the objects state changes. At this instance no other command than "Start" can be sent to the object. This builds up the initial first layer of the sanity check

routines, and implemented at the User Interface level to minimize the interaction cost of sending and receiving meaningless control commands to the system. But it may also be possible that another browser located somewhere in the world is monitoring the status of the object, but it has selected that object prior to the previously mentioned browser, and neither requested the status of the object nor sent a control command. At this instance that browser will not be aware of the current status, and may send a meaningless command to that object. But this should not bother the proposed system, since it has an object level secondary sanity check, which will prevent the object to misbehave.

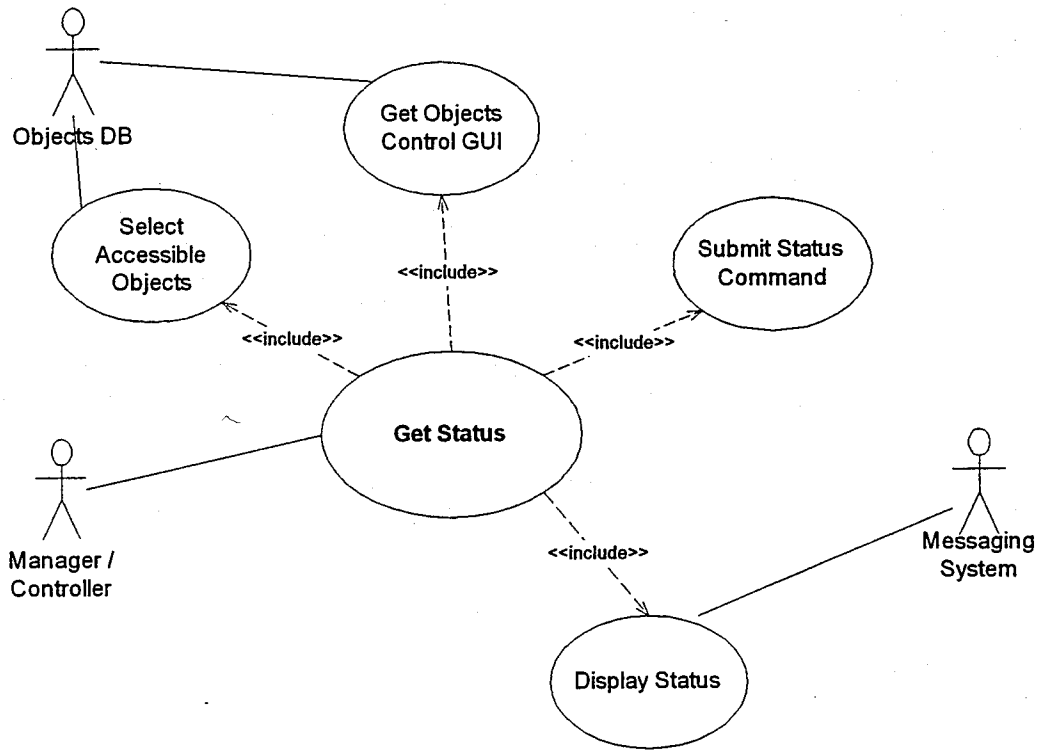

Figure 4.8. The get status use case

The status of the selected object is displayed as stated previously on the frame C in Figure 4.9. It displays the current status of the object with a horizontal bar. Tricky coding techniques implemented especially in "Status.asp" the file that displays this status information.

In the creation of this graphic neither an applet, nor an ActiveX object is used for the sake of cross-platform operability. This graphic is just the composition of two images' orientation in a proper manner in an html table. The details of the status page cannot be

seen on this graphic, therefore the frames are adjusted, and another snapshot is taken and presented in Figure 4.10.

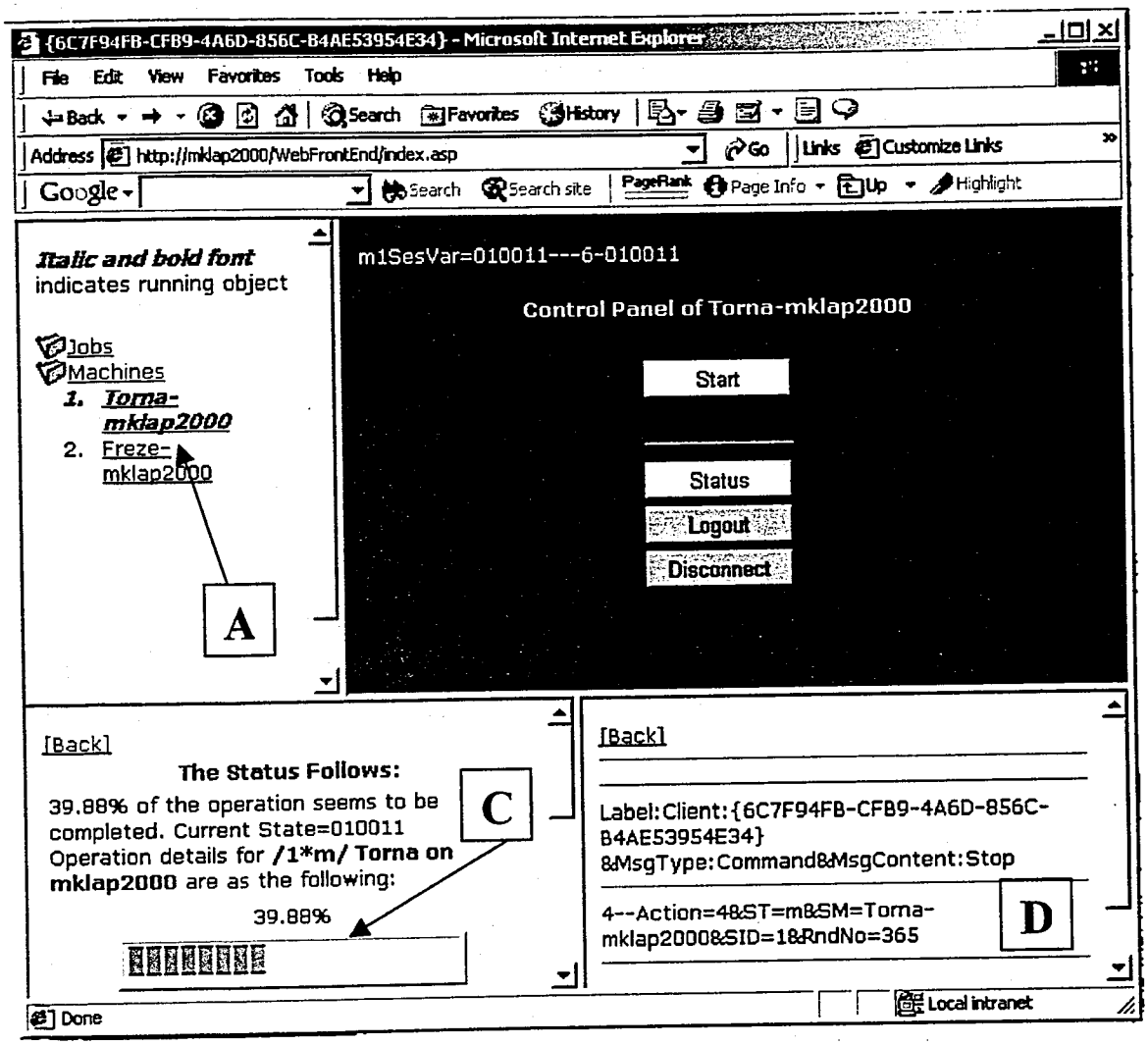

Figure 4.9. An instance of the running system

As it can be seen on the status frame of Figure 4.10, the virtual machines operation log is also displayed below the status graph. The operation log contains the timing and the actors of the operation. It also contains the behavioral events of the virtual machine.

The User Based View of the system is completed with these last snapshots of the system.

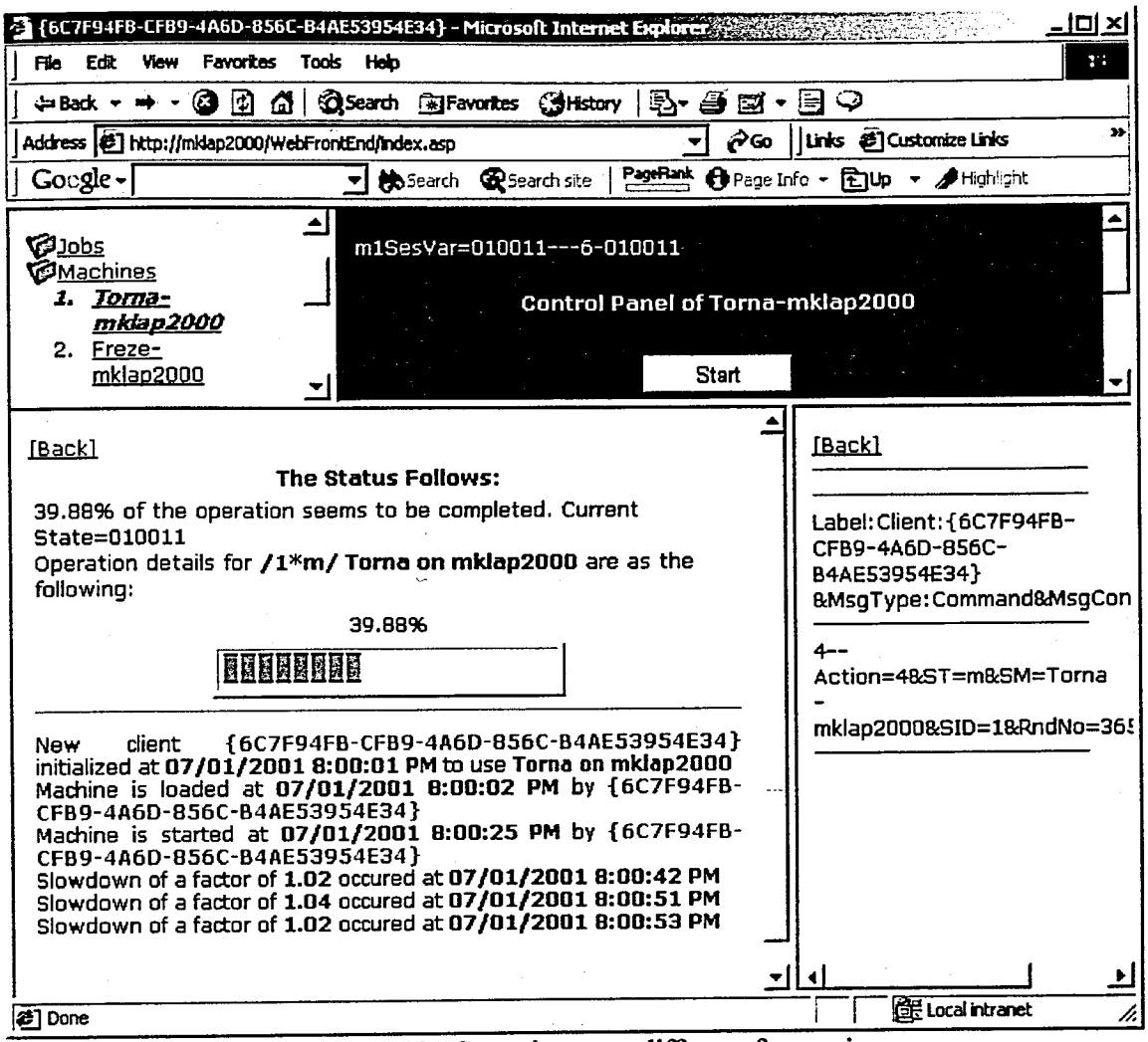

Figure 4.10. Same instance different frame size

4.3. The Internal Structure of the System

In this section we will study the internal structure of the system with the aid of UML sequence diagrams, and will try to clarify the messaging sequence between the systems internal objects. First a brief description about UML sequence diagrams will be given, and then the functioning of the system will be presented.

A sequence diagram has two dimensions; the vertical dimension represents time and the horizontal dimension represents different objects. Normally time proceeds down the page. Usually only time sequences are important. There is no significance to the horizontal ordering of the objects. Objects can be grouped into "swimlanes" on a diagram [15]. In a sequence diagram, vertical lines represent objects with the identification boxes above

them. On these boxes the first name is the object name, and the second one is the class name which the object belongs. The horizontal directed lines are messages. Each message line starts at the originator object and ends at the target object. Each of them carries its identification on it [9].

Again we will start from the very beginning as it is done in the previous section, but this time we will go into details of the object, database, and messaging system communication.

The Access Control System transaction is revisited with the sequence diagram represented in Figure 4.11. There is nothing special in this transaction with respect to the forthcoming transactions. The "WebFrontEnd" application enables the user authentication and login to the system. Here and in the whole of the system database communication is made through ODBC protocol. One thing that has to mentioned is that ODBC connection pooling approach is used especially for the "WebFrontEnd" application, to enable memory efficient database connection handling.

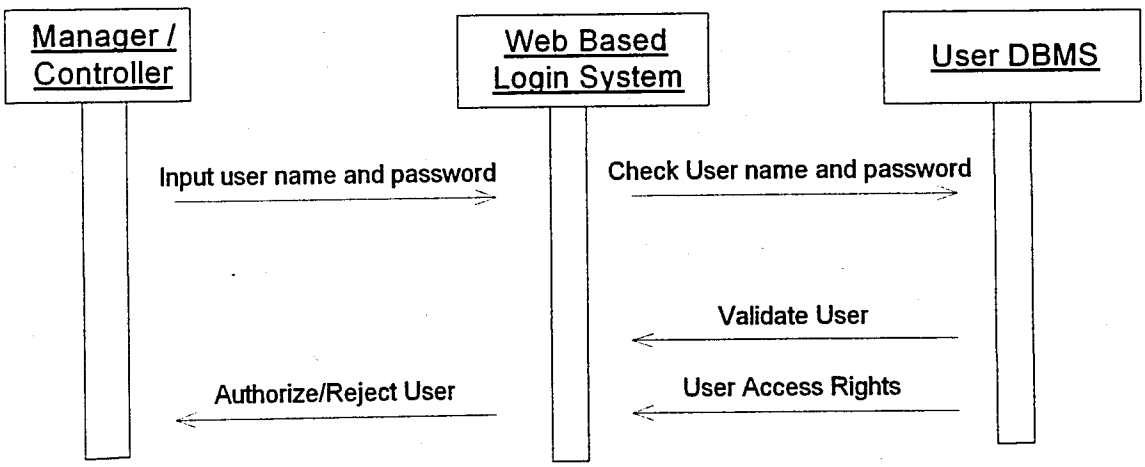

Figure 4.11 . Access control system sequence diagram

The "Select Accessible Objects" transaction is not very much different from the "Access Control System" transaction in terms of the internal structure. A thing to note is the listing of the objects in a tree-view menu style. A javascript driven menu system implemented for the cross-browser implementation. The code of this menu system is borrowed from Dan Steinman's [16] GPL (General Public Licence) DynaLayer library.

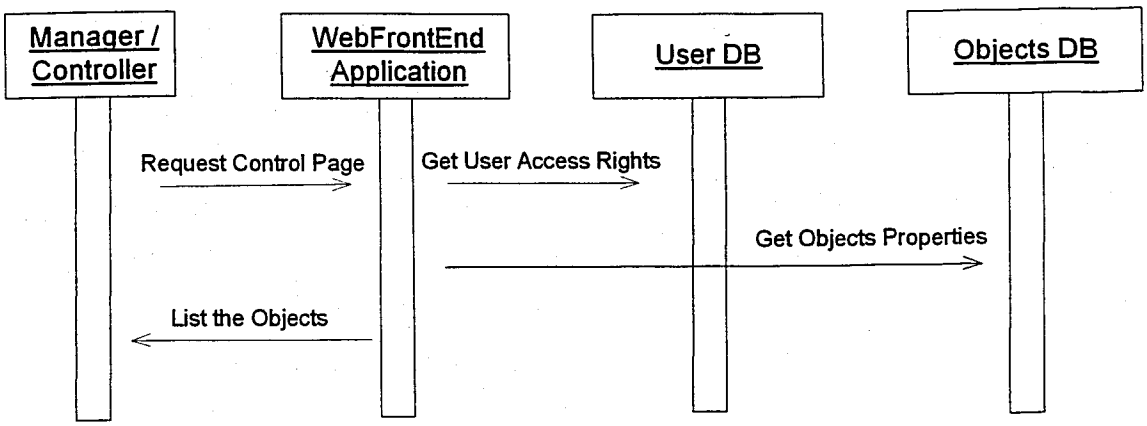

Figure 4.12. Select accessible objects sequence diagram

In Figure 4.13 the "Send Control Command" sequence diagram is represented. The messaging between the transaction entities are explained already in detail, but there is one thing that is not explained in detail for one purpose is the messaging system, and the internal message flow beyond this point until to the invocation of the virtual machine object and the execution of the control command.

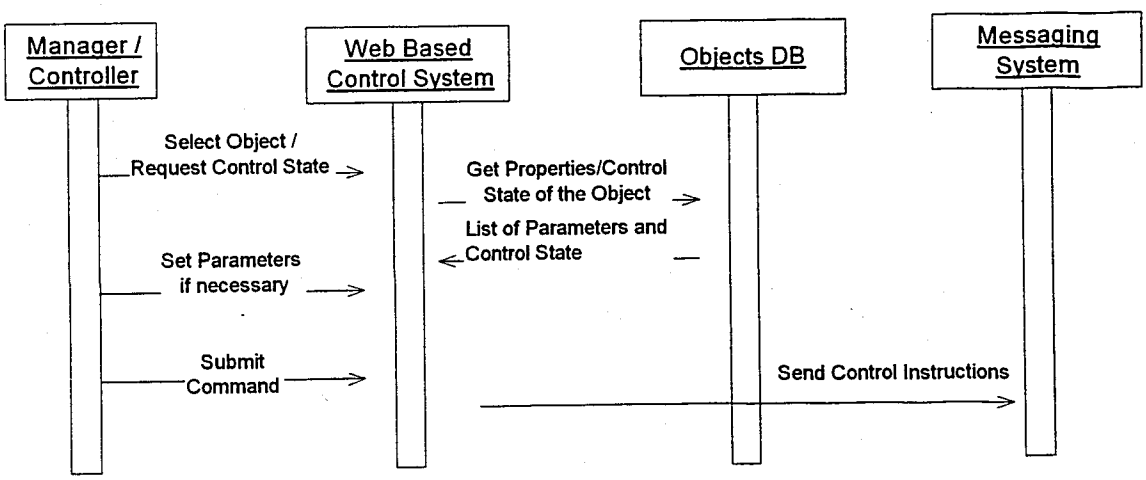

Figure 4.13. Send control command sequence diagram

The "Send Control Command" transaction generates its output as a message that is sent to the system to be interpreted and executed. The message structure and messaging system that is used in the communication between the "WebFrontEnd" application and the DCOM-Proxy/Appserver application of the proposed system is explained in great detail at the end of this section.

- ***Study 1*:** Now let's assume a case, where a manager/controller wants to control a specific virtual machine object with the aid of this system. The manager/controller first points its browser to the internet address of the system, he will log on the system as the predefined user named *kursun*. After the system verifies the manager/controller, and authorizes him/her to logon the system, the manager/controllers connecting browser sends an initialization message to the system. The explanation of the message is done at the Message Structure Section of this study, and the message follows;
    - ***Message Label**: Client={* A4C8E880-41A9-4A04-BAC4-F3694E05B4A9*}&MsgType=INIT&IP=194.1.1.100&UserAgent=Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)&User=kursun*
    - ***Message Body**: Empty*

When this message or any other message falls into the Controller queue, an event will be raised in the DCOM/Proxy Appserver application, and the incoming message is passed immediately for the interpretation to the "InterpretTheMessage" function.

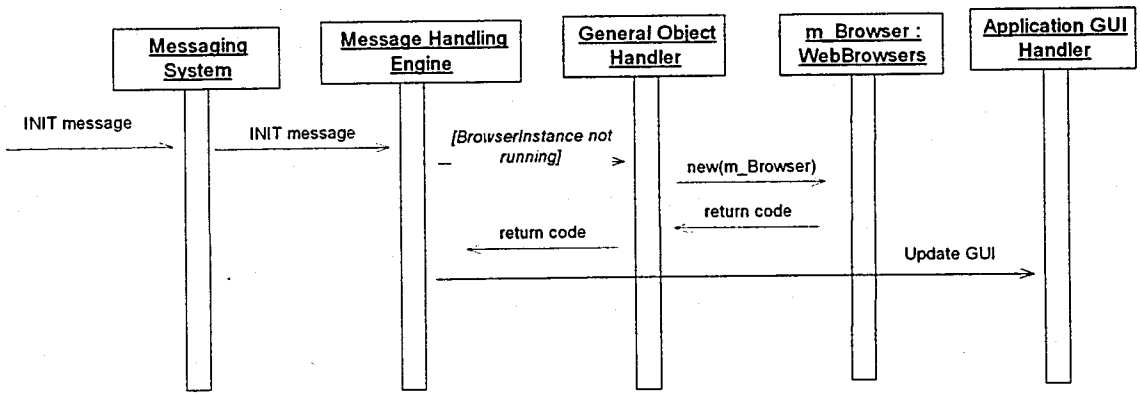

Figure 4.14. Init type of message flow sequence

"InterpretTheMessage()" function will first parse the message label, and message body. It will identify from the message label, whether the incoming message is an Init type, or Command type message. In our case it is an Init type of message, and the sequence of message flow between objects will occur as it is showed in Figure 4.14. "InterpretTheMessage()" function has been identified as the Message Handling Engine on this figure. The message label will be passed to "BrowserInit()" function, which can be seen as the General Object Handler. This handler will first check whether the browser

instance is running or not. It will set a new m_Browser object which stems from Browser class, will initialize the properties of this object, and it will also initialize a m_Browser.Controllers collection which will store the VirtualController classes. Then the newly created m_Browser object will be added to the WebBrowsers collection. It is now the time to update the GUI of DCOM/Proxy AppServer to reflect the newly connected browser. The manager/controller sees on his browser the same snapshot as in Figure 4.4, but at the same instance the DCOM/Proxy AppServer has updated its GUI. The snapshot covering this instance is shown in Figure 4.15.

There are four information boxes on the DCOM/Proxy AppServer's graphical user interface. The information box labeled A lists the processed messages. The information box B lists the connected browsers and their controller interfaces in a tree view. The information box C lists the current pool of running virtual manufacturing components, the real life machine objects. And the last information box labeled D represents the list of DCOM specific logs, i.e. the commands that are sent to the DCOM objects.

- ***Study 2***: Now the manager/controller logged on to the system. He/She selects the virtual machine object as "Torna-mklap2000" with the aid of the menu of the web browser. Now he/she is confronted with the control menu of that object. He/she has to load first this machine, so he pushes the Load control button on the objects control interface. By sending the Load command the system sends the following message to the DCOM/Proxy AppServer;
    - ***Message Label:*** *Client={A4C8E880-41A9-4A04-BAC4-F3694E05B4A9}&MsgType=Command&MsgContent=Load&SM=Torna-mklap2000&ST=m&SID=1*
    - ***Message Body:*** *Action=1&Cap=50&CapType=21&BProb=4&BDT=2& RepTD=12&LoadD=4 &WorkD=75&ST=m&SM=Torna-mklap2000&SID=1&RndNo=318*

When this message falls into the Controller queue, an event will be raised in the DCOM/Proxy Appserver application, and the incoming message is passed immediately for the interpretation to the "InterpretTheMessage" function as it happens in every received message case.

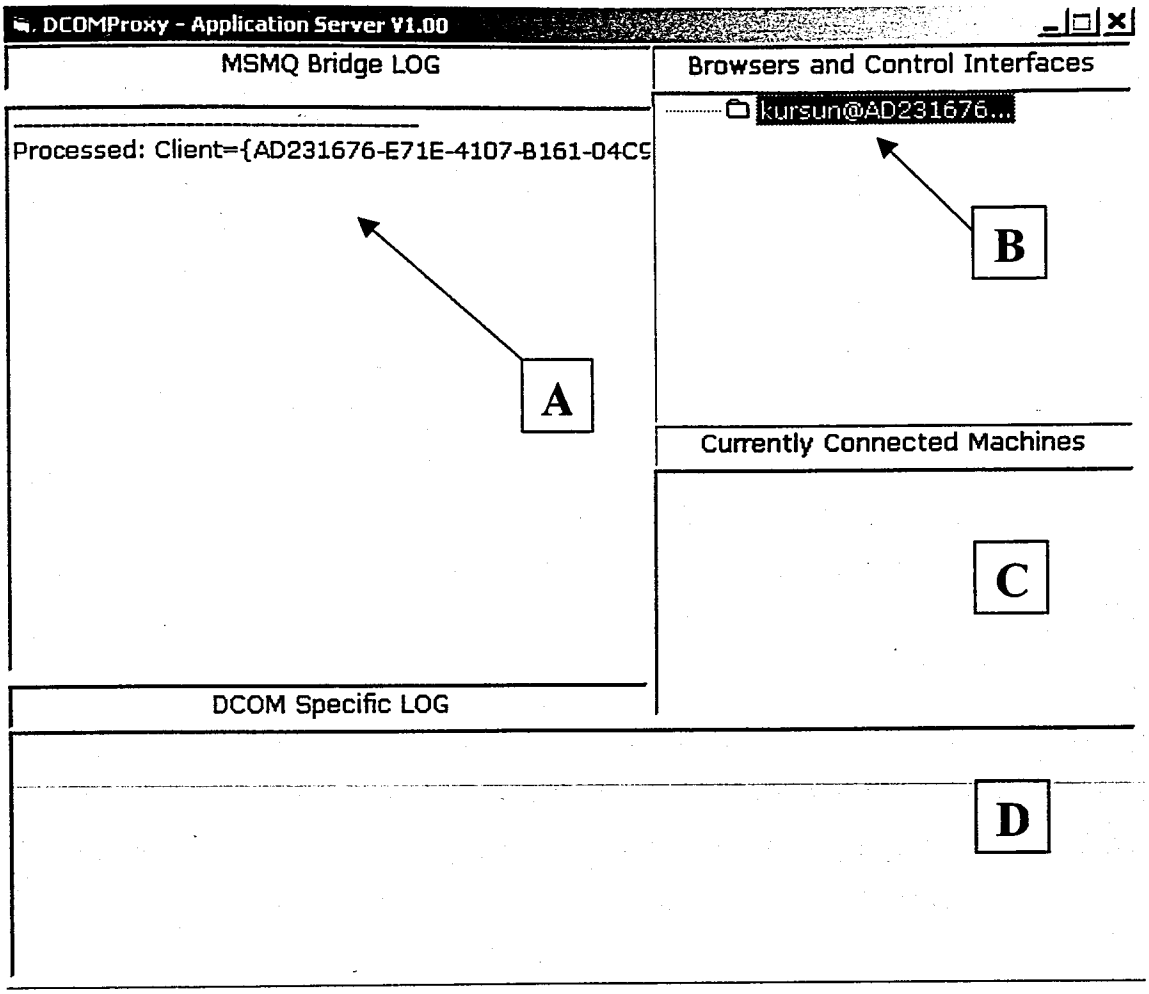

Figure 4.15. An instance from DCOM/Proxy Appserver

"InterpretTheMessage()" function will determine that the incoming message is a Command type message, and the sequence of message flow between objects will occur as it is showed in Figure 4.16. "InterpretTheMessage()" function has been identified as the Message Handling Engine on this figure. The message label and body will be passed to "ProcessCommands()" function, which can be seen as the General Object Handler.

General Object Handler will first check whether the browser instance is running or not, although the browser instance has to run. After determining that the browser instance m_Browser is running, it will further check, if there is a controller interface VC for the selected machine in the m_Browser.Controllers collection. If the controller instance is running, then there is nothing to worry, so the requested command will be passed to the DCOM Handler, to be executed on the oServer object, which stems from Real Life Objects class.

If there is not a controller interface for the selected machine in the m_Browser.Controllers collection, then a new interface has to be created. Therefore the object handler, in this case "ProcessCommands()" function creates a new VrtualController object denoted by VC, adds it to the current browsers controllers collection, and to initialize this newly created control interface, it calls the "Init()" method of the newly created VC object.

The VC.init method has to bind the VC object with the selected machine object. It first loops through the DcomObjects collection to check whether an instance of the selected machine is currently invoked and running. If it finds a running objects corresponding MachBag object in the DcomObjects collection, than it will get the reference from that MachBag object, binds the VC object with the running instance of the selected machine, and increases the corresponding MachBag objects "GUICount property" by one. And with this binding the requested command will be passed to the DCOM Handler, to be executed on the oServer object, which stems from Real Life Objects class. One thing is worth to note here. Low level object reference handling is not allowed normally, so we had to use advance coding techniques described in Appendix D.

If there is not a representing MachBag object for the selected machine in the DcomObjects collection, then the VC object has to create both a new MachBag object, and a oServer Real Life Machine object. It first creates the new MachBag object, sets its properties, and then creates a new Virtual Manufacturing Component object instance on the previously specified physical location, i.e. on the computer specified in the objects database. The newly created MachBag object will be added to the DcomObjects collection, and a initial zero status message will be sent to the connecting browsers receiving queue. It is now the time to update the GUI of DCOM/Proxy AppServer to reflect the newly created controller interface as the subcomponent of the browser. Finally the requested command will be passed to the DCOM Handler, to be executed on the oServer object, which stems from Real Life Objects class.

- ***Study 3***: Now let's assume the user kursun first starts the machine that he has loaded before, and then selects another machine called "Freze on mklap2000" to load. He then sends start command this machine. At the same instance another

manager/controller logs on to the system as the predefined user named test. This new user has an access right value of five, which is less than the minimum level of access right value of "Freze on mklap2000", therefore he can only access to the machine called "Torna on mklap2000". The previously described Init type message flow will occur again for the newly connected browser. Now he selects the only accessible virtual machine object for him, which is "Torna-mklap2000". He gets the control interface for the machine, and immediately he sees the current state of this machine, and for some reason he wants to stop the operation of this machine, so he pushes the stop control button on the objects control interface. By sending the stop command the system sends the following message to the DCOM/Proxy AppServer, and his browser reflects the snapshot shown in Figure 4.17.

- ***Message Label:*** *Client={EE33B531-0894-4DAF-9BA5-6EEC05F8EF54}&MsgType=Command&MsgContent=Stop&SM=Torna-mklap2000&ST=m&SID=1*
- ***Message Body:*** *Action=4&ST=m&SM=Torna-mklap2000&SID=1&RndNo=207*

When this message falls into the Controller queue, the incoming message is passed again immediately for the interpretation to the "InterpretTheMessage" function.

"InterpretTheMessage()" function will determine that the incoming message is a Command type message. The message label and body will be passed to "ProcessCommands()" function, which can be seen as the General Object Handler.

General Object Handler will first check whether the browser instance is running or not, although the browser instance has to run. After determining that the browser instance m_Browser is running, it will further check, if there is a controller interface VC for the selected machine in the m_Browser.Controllers collection. In our case, there is already a controller interface and therefore the requested command will be passed to the DCOM Handler, to be executed on the oServer object, which stems from Real Life Objects class.

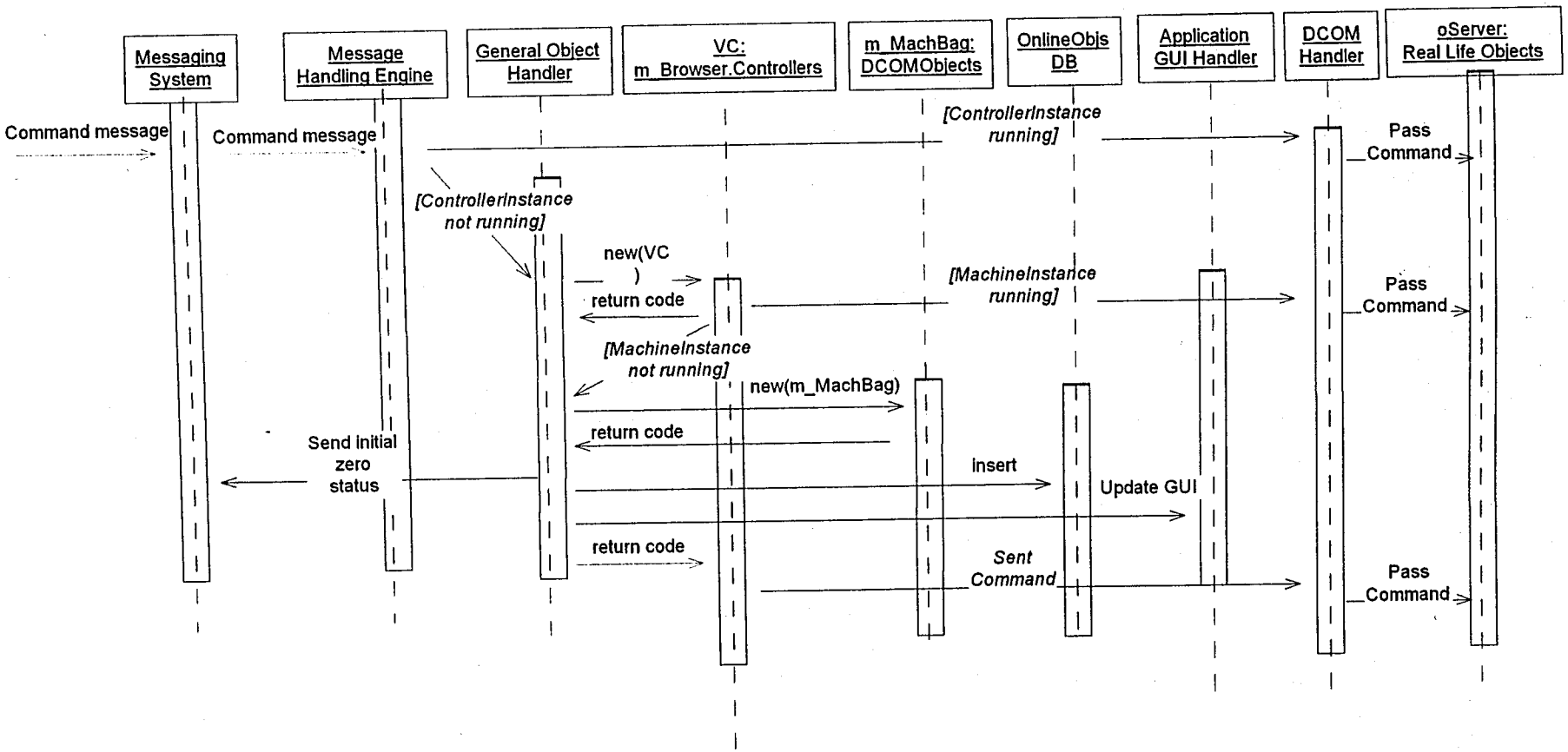

Figure 4.16. The Command type message flow sequence diagram

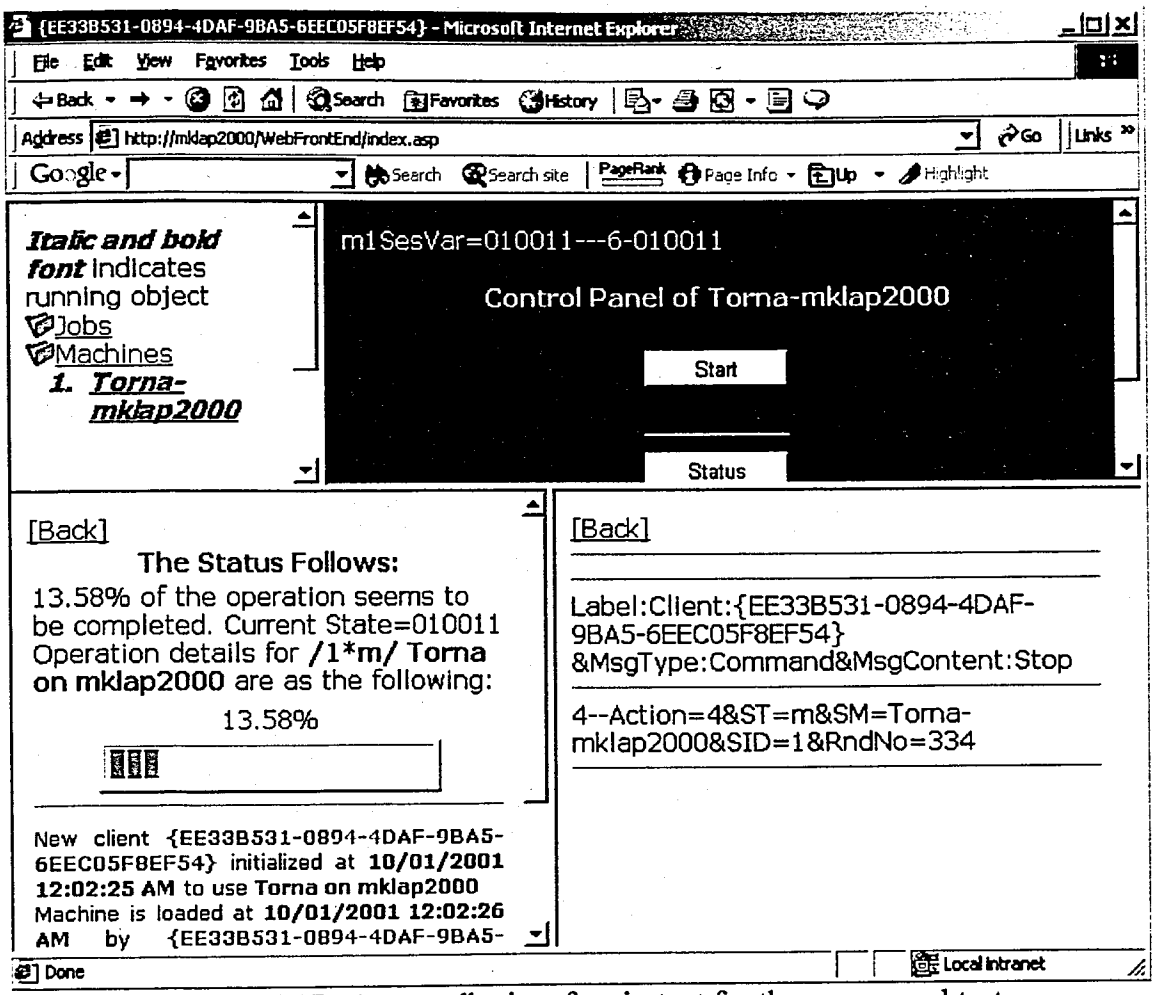

Figure 4.17. A controller interface instant for the user named test

The representation of all the message flow realized by the user kursun and test is shown in Figure 4.18 at the DCOM/Proxy AppServer viewpoint. From the Figure 4.18 one can easily see all previously discussed things in action. All of the messages sent from the WebFrontEnd to the DCOM/Proxy AppServer are listed on the "MSMQ Bridge Log" section of Figure 4.18. The connected browsers, their users, and the machines they control can be easily identified from the "Browsers and Control Interfaces" section. The pool of objects, i.e. currently running, or connected machines are listed on the "Currently Connected Machines" section. And the commands passed from DCOM/Proxy AppServer to the Real Life Machine Server objects are presented on the "DCOM Specific Log" section of Figure 4.18.

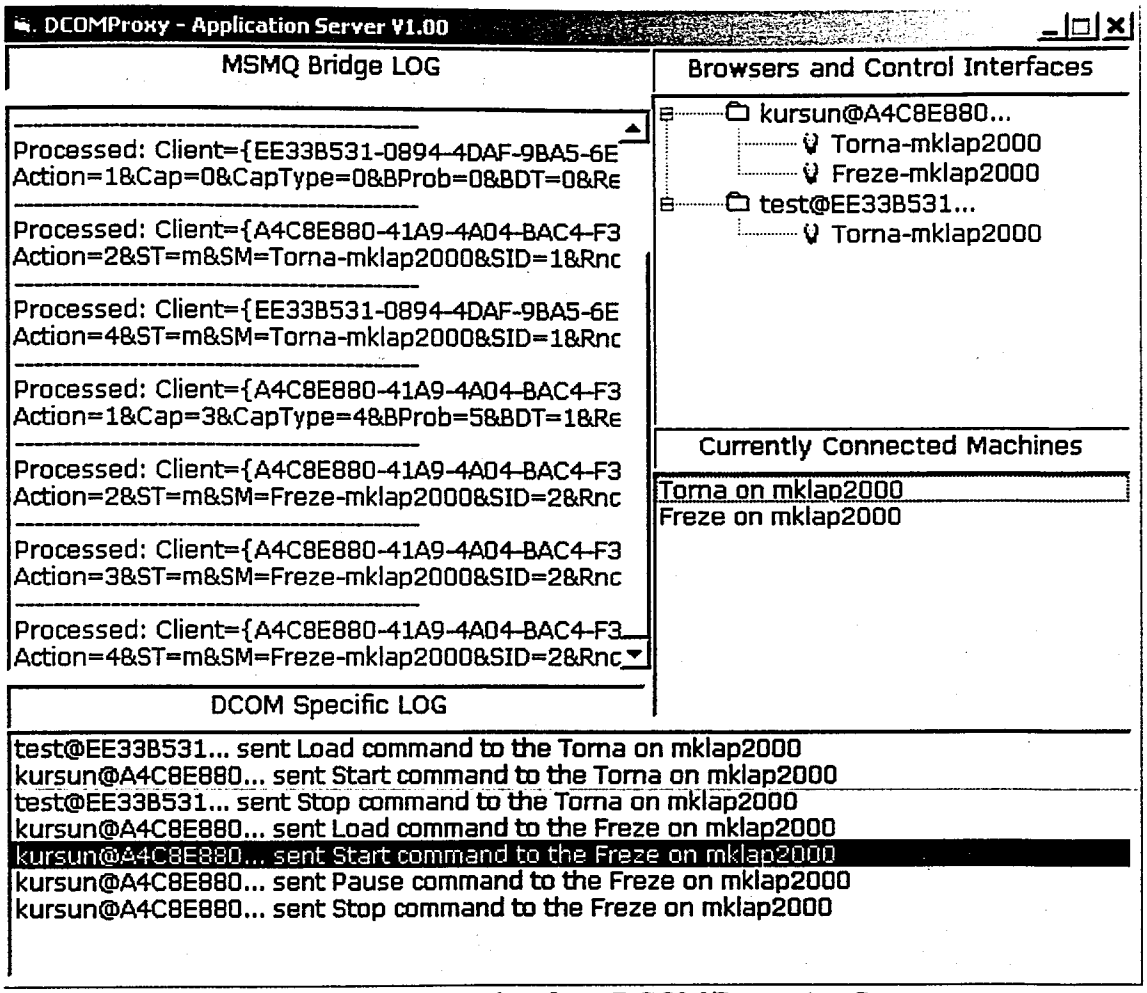

Figure 4.18. A snapshot from DCOM/Proxy AppServer

Until now, we first examined the User Based of the system, and discussed the interactions between the user and WebFrontEnd application. Then we went one step further and examined the detailed interaction between the WebFrontEnd application and the DCOM/Proxy Application Server. In the following paragraphs we will examine how the DCOM/Proxy Application Server communicates with the DCOM objects of Real Life Machine Server class.

The DCOM/Proxy AppServer has a function called "PassCmdDCOM" which is denoted as DCOM Handler in Figure 4.16. All requests to control a virtual manufacturing component, i.e. a DCOM object, will be passed to the DCOM Handler, i.e. to the "PassCmdDCOM" function, after the setting up the controller interfaces for the execution. The DCOM Handler just invokes the method relevant to the requested command on the VirtController, the VirtualController class of the connected browser, and since the

VirtController has the reference of the indicated object, it simply passes this call to the relevant method of oServer, which is a Real Life Machine Server class.

In the third study discussed previously, the situation was as the following. User "*kursun*" is monitoring two machines named, "Torna on mklap2000" and "Freze on mklap2000" respectively. At the same instance user "*test*" is monitoring just the "Torna on mklap2000" machine. The running objects graphical user interface representing "Torna on mklap2000" is shown in Figure 4.19. The title of the window is "2684 – Torna @ mklap2000" as indicated with label A. The number 2684 is the thread ID of the current object, and Torna @ mklap2000 is the name of the running object. The machine has completed 13.6 per cent of its operation, and this information is indicated with label B in Figure 4.19. The users "*kursun*", and "*test*" both connected to this machine, and so there are two clients connected currently, which is indicated with label C. And the operation log of this machine is listed in the textbox labeled with D.

4.3.1. Real Life Machine Server Objects

These objects form the heart of the system. All of the previously studied applications are there, just to enable the communication of a remote client with one of these objects. These objects represent the components in the manufacturing process. They have the primitive methods, which can be found in every type of machine. A machine may require a loading process before it can start to its operation. It will have start and stop functionality, and it may also have a pause functionality. We give our objects these four primitive functionality. The class representation of the Real Life Machine Server object is shown in Figure 4.20. There are basic events which a real life machine will represent, in case of a breakdown, slowdown or process finish. The object has several properties, by which the object can be modeled to a desired real life machine. The system requires that an object has to have the following properties mandatory, which are pControlState, pCurrentStatus, pLoadDuration, pState and pWorkDuration. All the other properties especially related to the loading process, or working process may change according to the desired object type.

4.3.1.1. Simulation Case. If we want to use this proposed system as a simulating environment for a real factory, then these objects will represent the counterparts of the

manufacturing components. These objects may represent the CNC machines, or any type of manufacturing machine, or they may represent for example jobs, since the jobs can be identified as a component, which takes part in the manufacturing process.

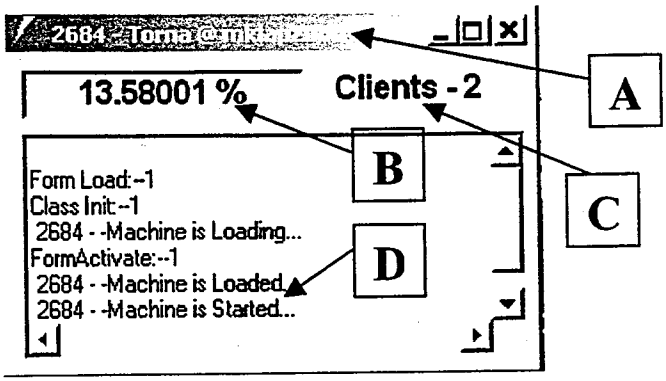

Figure 4.19. An instance of a running Real Life Machine Server object

Back again to our initial MultiNat example, revisit the case, where the multinational company MultiNat Inc. wants to replace its current scheduling software in its headquarter, but before the implementation the managers want to be sure that the new system will fit their system perfectly. To test whether the new system will fit or not, they have to run the software on the virtually designed environment. With our proposed infrastructure and object model, it is possible to handle such a situation. There will be two types of objects, which can be identified as jobs, and machines. These objects have to have the previously discussed mandatory properties, and all the events, and methods described in Figure 4.20.

The flexible and object oriented structure of the system, a third-party's software, the scheduling software in the mentioned case, can easily run these components for simulation purposes. The only requirement is, that this third-party software can connect to MSMQ servers and ODBC enabled database. By connecting to the MSMQ server(s) this third-party software will be able to send commands to and receive status information from these objects. A complete message reference is given in Appendix A.

4.3.1.2. Control Case. If we want to use our proposed system as a control environment for a real factory, then these objects will represent the counterparts of CNC Machines, or any other machine that is involved in the manufacturing process. On the contrary to the simulation case, these objects will stand as an interface to the real manufacturing

machines, i.e. invoking the start method of the object will cause to start the related CNC machine physically. By the design nature of these DCOM objects one can define, where the object will run, i.e. on which physical computer the machine will run. This feature is one of the enablers of physical control of the indicated real life manufacturing machine.

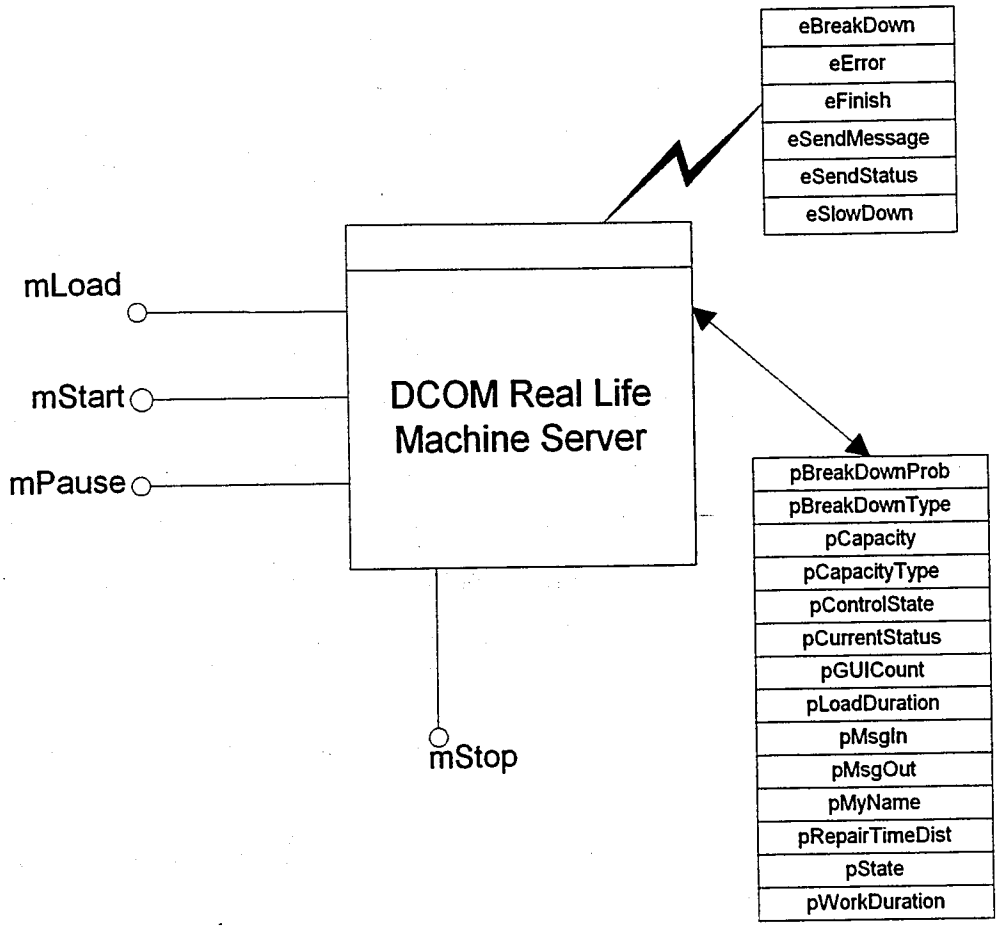

Figure 4.20. DCOM Real Life Machine Server class

If there is an appropriate digital connection port to the indicated CNC machine, executing the requested operation on such a machine may be realized without the need of an extra software. But if the real machine has no direct digital connection port, then a connector software may be easily programmed, and by the help of Analog to Digital Convertors (ADC) and Digital to Analog Convertors (DAC) this software can act as a connector between our DCOM Object and the indicated manufacturing machine.

4.3.2. The Messaging System

The messaging system is built upon the Microsoft Message Queue infrastructure. In the following paragraphs we will briefly explain what MSMQ is, and why we choose MSMQ.

With the trend toward distributed computing in enterprise environments, it is important to have flexible and reliable communication among applications. Businesses often require independent applications that are running on different systems to communicate with each other and exchange messages even though the applications may not be running at the same time [17].

Microsoft® Message Queue Server (MSMQ) technology enables applications running at different times to communicate across heterogeneous networks and systems that may be temporarily offline. Within an MSMQ enterprise, applications send messages to queues and read messages from queues. Figure 4.14 shows how queues hold the messages used by both the sending and receiving applications [17].

MSMQ ensures that all messages eventually reach their destination, whether a message is sent to a queue or a message is read from a queue. MSMQ provides guaranteed message delivery, efficient routing, security, and priority-based messaging [17].

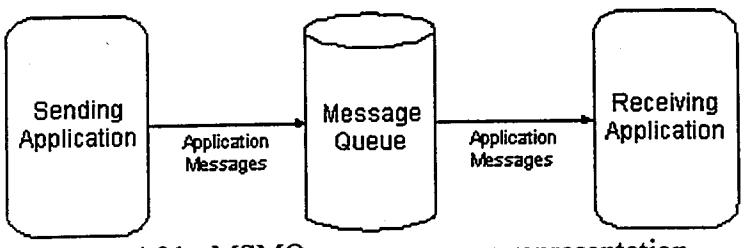

Figure 4.21. MSMQ message queue representation

MSMQ is different from remote procedure calls (RPC), Windows Sockets, and messaging API (MAPI). Because MSMQ is a connectionless message service where applications do not need to maintain a session, it is different from RPC where applications are required to maintain sessions. And although Windows Sockets provides low-level

functions for writing applications, Windows Sockets does not allow applications to run at different times in the way that MSMQ does. MSMQ is also different from MAPI (an e-mail oriented service) in that it uses a more general-purpose message queuing model than MAPI [17].

In the proposed system two types of MSMQ queues will be used. The one and the most important queue is the queue, which is used as the receiving queue by the DCOM-Proxy/Appserver. We will name this queue as the Controller queue from now on in this study. It is a transactional queue, and this guarantees for the sender that the message is sent one and only one time, and that the message is sent really. Otherwise the transaction will be rolled backed, and an error will be raised. The same is true for the receiving application.

The following illustration shows how transactions are used by the sending and receiving applications. In this model, MSMQ uses two transactions: one to send messages to the queue and the other to retrieve messages from the queue [17].

In this model, the sending transaction can commit to sending the messages to the queue, and the receiving application can commit to retrieving the messages; MSMQ provides its own confirmation process to notify the sending application that either the messages were retrieved from the queue or why the receiving application failed to retrieve them [17].

MSMQ provides several ways to send and receive messages through transactions. Transactions can be called either explicitly by providing pointers to a transaction object, or implicitly, directing MSMQ to find the appropriate transaction object. Transactions that can be explicitly called include MSMQ internal transactions and the Microsoft® Distributed Transaction Coordinator (MS DTC) external transactions [17].

The Controller queue or Controller queues, if scalability issues arise, will receive all the control requests from the connecting clients, i.e. web browsers. On the other side the system has also to send some data to the connecting browsers. Therefore new queues will be assigned to all connecting browsers. The only need of a client is the information of the current status of the virtual machine(s) that the client want to control. This data is also an

important data, but any duplication of this data, or any corruption of the data in a pass is not important, since the status of the machine change in time, and the client will get info in a the next time interval. And if one thinks about the physical manufacturing process where the operations are in magnitude of hours, and seldom in multiples of ten minutes, the criticality of this repeated data is not so important, as the control command messages criticality. Therefore, the clients receiving queues do not need to be transactional.

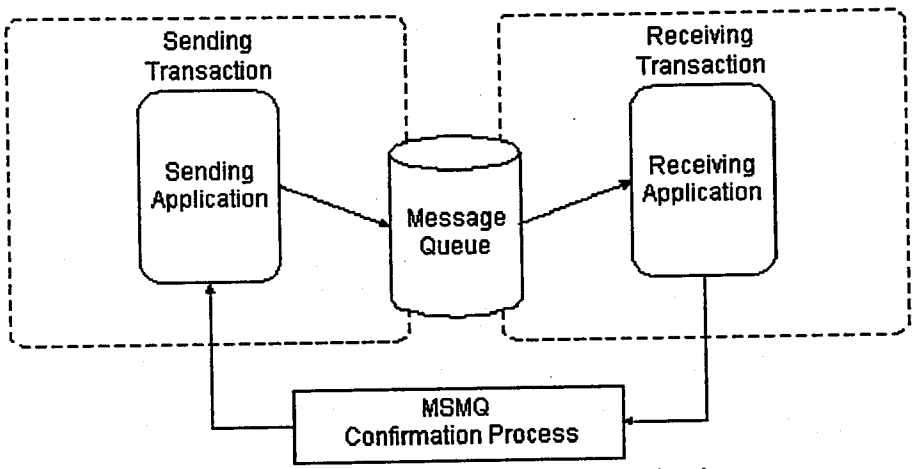

Figure 4.22. MSMQ transaction mechanism

Another problem is the uniqueness of queue names. The queue name of each connecting client has to be unique. To provide uniqueness we first assigned IP (Internet Protocol) numbers to each connecting client, since it is clear that every machine that is directly connected to the internet has to have a unique number called IP number to represent itself in the universe. Unfortunately this is true for most of the cases, actually true for the computers that are connected to the internet via dial-up connection, or to the computers that have a C-class IP. But what will happen, if the client is coming from a corporate proxy, or simply if the client is behind the proxy. In this case, all clients coming from that proxy will have the same IP as of their proxy servers. It is obvious that this naming convention will not work 100 per cent. Therefore a 128 bit GUID (Globally Unique Identifier) will be generated for each client, and this GUID will be assigned as their queue names.

4.3.3. The Message Structure

We want to briefly mention about our systems message structure. The messages will be transferred from sender to receiver as MSMQ message objects. A MSMQ message object is capable to transfer messages that are at most four MB capacity. The message object provides label and body properties for a message. The messages are indexed according to their labels in a MSMQ queue.

We had various possibilities in determining the right and optimized message structure. First of all it has to be stated that the body of the message may be set to any type of object. This make it possible to send recordsets, Microsoft Word Documents as objects in the message body. There is no limitation, and one may send whatever he/she wants to send to the other party with only the four MB size constraint. We thought about several alternatives, which are XML (Extended Markup Language) type messages, Property Bag type messages, or a simple system wide defined string messages. We choose to adopt a custom messaging structure, which will be just a line of string containing key and values separated by a seperation character from the other key-value pairs in the string. There is only one reason to do this, the size of the messages are really too small compared to the other alternatives, and the message size may matter in case of heavy load. The reduction in the size of the message reduces the bandwidth requirements, and speeds the transfer time of the messages between the sender and receiver applications.

We are aware what XML is, and that XML will be the industry standard in the data exchange structures, we left it out of the box for our system, since there are really small set of parameters to be defined and passed to the receiving application in the case of our system. But a very clear code is implemented for the interpretation of the messages, and the parsing of the messages is coded as a function called "ParseString()" in the DCOM-Proxy/Appserver program. That means, if we decide to use XML for the composition of messages in a complex implementation, the system will easily adopt to the new XML structure by just changing the "ParseString" function in to a XML parser function. At the time of writing of this study, Microsoft's XML Parser was still in Beta edition, which worried us about a semi-stable system if implemented.

There are two types of message used in the system, Command type and Init type of messages respectively. We will briefly identify this two types of messages in this section, but the reader will see the detailed discussion about these messages in the following pages.

When the manager/controller first connects to the system after the login process in the "Access Control Transaction", the connecting browser will send an "Init" type of message to the system. This can be seen as the handshaking of the two parties, with this initial message the browser introduces itself to the system. The initialization message contains only a label, the body of this message is empty. The sample message follows;

***Message Label**: Client={98CA1AA9-6FF6-4C92-AFD3-EE694E05B4A9}&MsgType=INIT&IP=194.1.1.100&UserAgent=Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)&User=kursun*
***Message Body**: Empty*

The message label contains Key, Value pairs which are separated by the seperation character "&". The message interpretation algorithm parses this message in to the corresponding key, value pairs. In this type of messages the message body contains nothing since this messages are used for triggering the DCOM-Proxy/Appserver for the newly connecting browser.

The keys and their description follows:

- **Client.** It identifies the name of the connecting browser, the GUID number.
- **MsgType.** It identifies the type of message, possible values are INIT, or Command.
- **IP.** It indicates the IP number of the connecting browser.
- **UserAgent.** It identifies the type of browser and the platform of the connecting computer.
- **User.** It identifies the registered user, who is using the connecting browser.

Another examples to identify the "Command" type messages two of them are presented. The first one is a sample Load command sent by controller/manager to the system.

Message Label: *Client={98CA1AA9-6FF6-4C92-AFD3-EE694E05B4A9}&MsgType=Command&MsgContent=Load&SM=Torna-mklap2000&ST=m&SID=1*

Message Body: *Action=1&Cap=50&CapType=21&BProb=4&BDT=2& RepTD=12&LoadD=4 &WorkD=75&ST=m&SM=Torna-mklap2000&SID=1&RndNo=318*

The keys and their description for the message label follows:

- **Client.** It identifies the name of the connecting browser, it is the generated GUID number.
- **MsgType.** It identifies the type of message, possible values are INIT, or Command.
- **MsgContent.** It identifies the content of the message, possible values are Load, Start, Pause, Stop, Exit and Disconnect.
- **SM.** It stands for selected machine, and it identifies the name of the selected machine. All string values are possible, in this case it is equal to "Torna-mklap2000"
- **ST.** It stands for selected type of the object. All one character values are possible, in this case it is equal to "m"
- **SID.** It stands for selected ID of the object. All number values are possible, in this case it is equal to "1".

The Load command may contain different number of parameters for different kind of objects. For the case where the object represents virtual manufacturing machine, we defined seven parameters for the loading process of this kind of object.

The parameters are:

- **Cap.** It represents the Capacity of the machine.
- **CapType.** It represents the Capacity type of the machine; i.e. in our system it is used to identify the unit of the capacity stated with Cap parameter with predefined numbers each representing a metric unit.
- **BProp.** It represents the breakdown probability model to be obeyed, with predefined numbers each representing a probability model.

- **BDT.** It represents the breakdown type, i.e. whether the resulting breakdown is repairable, replaceble, or etc., with predefined numbers each representing a breakdown type.
- **RepTD.** It represents the repair time distribution, with predefined numbers each representing a distribution.
- **LoadD.** It represent the load duration in seconds.
- **WorkD.** It represents the work duration in seconds.

All the parameters that need to be send in the predefined order, in this examples case the parameters, are described in the previous paragraphs. The message body for a Command will contain the following keys:

- **Action.** It identifies the type of action, and must be consistent with the message labels MsgContent key. Possible values are, 1-4, indicating Load, Start, Pause, Stop commands.
- **SM.** It stands for selected machine, and it identifies the name of the selected machine. All string values are possible, in this case it is equal to "Torna-mklap2000"
- **ST.** It stands for selected type of the object. All one character values are possible, in this case it is equal to "m"
- **SID.** It stands for selected ID of the object. All number values are possible, in this case it is equal to "1".
- **RndNo.** It is the dummy variable introduced by the web based application. Since the parsing operation at the server side scripting level is costly, the variable introduced as is, but neglected by the DCOM-Proxy/Appserver.

And as a last example to further clarify the messaging structure, a stop command sample follows:

Message Label: *Client={98CA1AA9-6FF6-4C92-AFD3-EE694E05B4A9}&MsgType=Command&MsgContent=Stop&SM=Torna-mklap2000&ST=m&SID=1*

Message Body: *Action=4&ST=m&SM=Torna-mklap2000&SID=1&RndNo=72*

4.3.4. Classes of DCOM/Proxy AppServer

DCOM/Proxy AppServer is the vital part of the system. It contains several classes which enables program to act as a bridge between the outer world, i.e. manager/controller, and the in-house machines or virtual machines. There are three types main classes, Browser, VirtualController and MachBag respectively, and two types of collections, DcomObjects and WebBrowsers respectively.

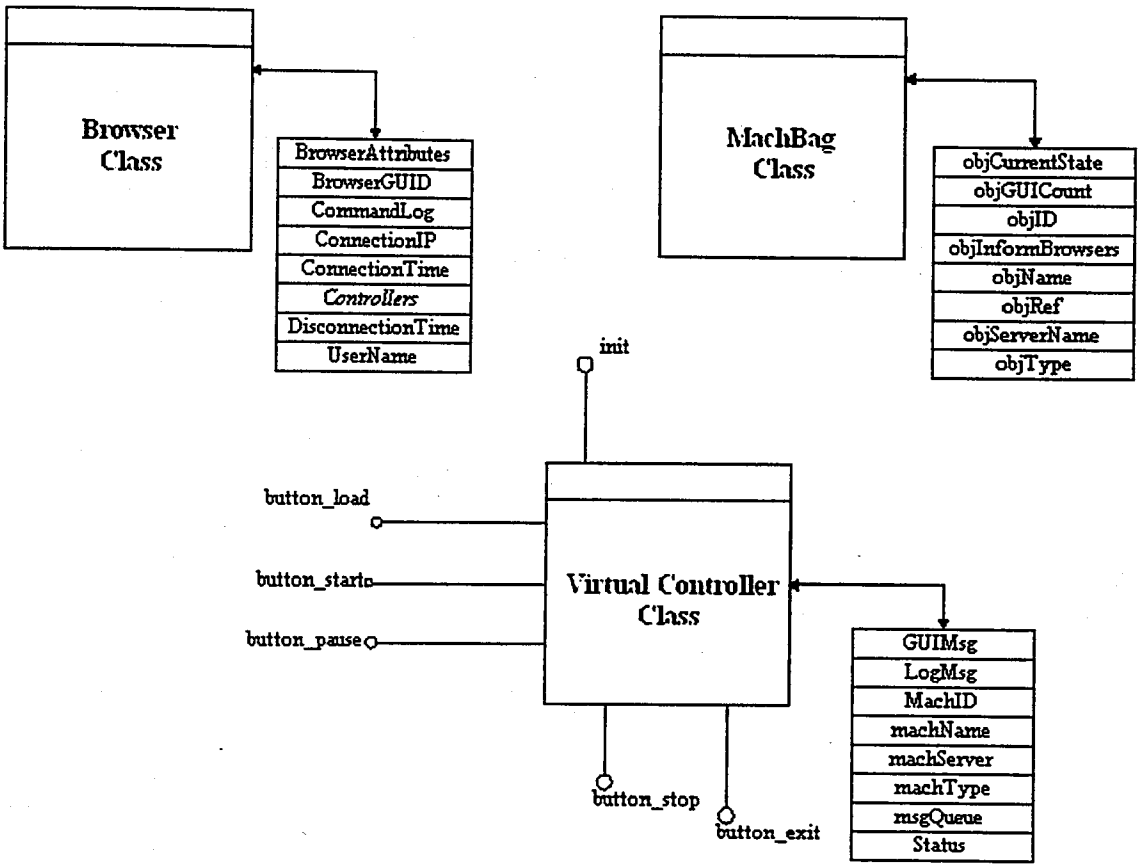

Figure 4.23. DCOM/Proxy AppServer Classes

Each new connecting browser, which is controlled by the controller/manager, will be represented by the Browser class shown in Figure 4.23, and all these classes will be strored in the WebBrowsers collection. A Browser may control many virtual manufacturing components. To control these components the browser needs a controller interface. The controller interface is represented by the VirtualController class shown in Figure 4.23. The controller interface will facilitate the control process by communicating with the desired virtual manufacturing component. Since many manager/controllers (browsers) may want to

monitor or control the same virtual manufacturing component (Real Life Machine object) at the same time, there has to be a pooling mechanism to enable this situation. Therefore the virtual manufacturing components properties and object reference will be represented by the MachBag class shown in Figure 4.23. And to enable a pool with all running virtual manufacturing components (objects), the MachBag classes will be stored in DcomObjects collection. To sum up every browser may have many controllers, but a Virtual Controller can control only one and only one RealLife Machine object with the help of MachBag object. To further clarify this situation let's examine our famous MultiNat example.

Both Manager A and Manager B want to control the turret lathe in Istanbul plant. Manager A connects to the system interface of Istanbul plant, logins to the system, and selects the turret lathe from the menu, to be able to control or monitor this machine. Now a virtual manufacturing component, namely the Real Life Machine object is created to represent the turret lathe. In the mean time the Manager B logins to the system and selects the turret lathe also. But since another manager/controller Manager A requested this current lathe before, there is a running object which represents this turret lathe. The Manager B's Browser class will now create a virtual controller class, which will hold exactly the same reference of the object that is representing the current lathe, and will add it to the Browser.Controllers collection. So with this infrastructure both Manager A and B and many other will easily control or monitor every virtual manufacturing component.

5. SOFTWARE DOCUMENTATION

5.1. Real Life Machine Server

5.1.1. Methods

Real Life Machine Server object has the following methods:

- **Mload.** To be able to operate any kind of machine, there is an initialization process. This function is named as loading since some real life machines also require loading before beginning any operation. To sum up, this function includes initialization and loading together.
- **MStart.** This function aims to start the operation of the machine.
- **MStop.** This function aims to stop the operation of the machine.
- **MPause.** This function aims to pause the operation of the machine.
- **CheckError.** This function will represent the probability function. Here a probability algorithm will be performed to determine whether an error occured or not in the simulation of the machine.
- **ProcessTimer.** This subroutine is the heart of the simulation. There is a private class variable called TimerType. TimerType is used as an indicator of loading process or other machinery processes. When the machine has started or loaded, the process(timer) is enabled, and process_timer subroutine will be invoked every "CheckInterval" seconds. And so the behavior of a real life machine is simulated.
- **Class_Initialize.** This subroutine is the initialization module of the server object.
- **Class_Terminate.** This subroutine is the termination module of the server object.

5.1.2. Properties

Real Life Machine Server object has the following properties:

- **PCurrentStatus.** It indicates the difference between current time and starting time.

- **PControlState.** It represents the current state of the machine. It shows the functions that can be executed in the current state. Since there may be many controllers, which want to control this machine, it is essential to make the sanity check in this level. It is a 6 digit binary number.The digits 0 and 1 represent whether the function can be executed or not. From left to right the digits correspond, mLoad, mStart, mPause, mStop functions. And the last 2 digit is reserved for Logout and Status functions for the web interface. It is a sanity check parameter, and used for dynamically generating the web based controller interface for this machine.
- **PState.** It represents the log information of all the commands sent to this machine from various GUI's. Since it is used for the status information on the web interface, HTML tags are allowed in this string.
- **PMyName.** It represents the thread and name of the current machine.
- **PLoadDuration.** It represents the total time required to load the machine in seconds.
- **PWorkDuration.** It represents the total time required to finish the process in seconds for the machine in normal conditions.
- **PBreakDownType.** It represents the type of Breakdown.
- **PRepairTimeDist.** It represents the probability distribution of RepairTime.
- **PBreakDownProb.** It represents the probability distribution of breakdown.
- **PCapacity.** It represents the capacity of the machine.
- **PCapacityType.** It represents the type of capacity of the machine.

5.1.3. Events

Real Life Machine Server object has the following events:

- **ESendMessage.** This event will be raised, when the server object needs to send a message.
- **ESendStatus.** This event will be raised to send the status of the object.
- **EBreakDown.** This event will be raised when the machine breaks down.
- **ESlowDown.** This event will be raised when the machine slows down.

- **EFinish.** This event will be raised when the machine finishes its operation.
- **EError.** This event will be raised when the machine encounters an error.

5.1.4. Constants

The CheckInterval constant represents the process_timer interval in miliseconds.

5.2. DCOMProxy / AppServer

5.2.1. FrmDCOMProxy Form

Form frmDCOMProxy is the user interface of this project, and it is named as DCOMProxy Monitor.

This form has the following methods:

- **Form_Load.** MSMQ event receiving system is initialized and enabled. WebBrowsers and DcomObjects collections are initialized, and the `Available Machines` sub section on the DCOMProxy Monitor is initialized.
- **FillBrowserGUID.** This function fills the `Available Machines` sub section on the DCOMProxy Monitor with all connected browsers and the machines that are controlled by that browser.
- **QRecEvents_Arrived.** If a message arrives to the receiving queue (qRequestPath), then qRecEvents_Arrived event is fired. This function first interprets the message by calling InterpretTheMessage function. Then it logs the message to MSMQ Bridge LOG on the DCOMProxy Monitor.

FrmDCOMProxy form has the following variables:

- **QRecEvents.** This variable is one of the most important object defined in this project. It is declared as `Private WithEvents`, and this declaration enables that every received message raises an event.

- **QRequestPath.** It is the main path, where all the requests from browsers are sent. The current application server will use this queue as the receiving queue.

5.2.2. DCOMProxy_Main Module

This module has the following methods:

- **DCOMOK.** This function checks whether the machine is capable of making DCOM connections.
- **InterpretTheMessage.** If qRecEvents_Arrived event is fired, the received message is passed to this function to be interpreted. The body and label of the received message is first parsed by calling ParseString function, and then the message is processed according to the Message Type. If message is a `Command` then it is passed to ProcessCommands function to be processed. But if it is a `INIT` command it is passed to the BrowserInit function.
- **SendMessage.** This function takes queue name, message body and message label as arguments, and sends the indicated messge to the indicated queue.
- **InformAll.** If virtual controllers machine timer is enabled, this timer generates a tick event at every StatSendInterval. This function checks for all browser and their respecrtive controllers, if a browsers controller of the currend DcomObject (machine) is present, than a status messages should be send to that browser. In this manner every browser that is connected to the specified DCOM Object (machine, ..) receives the status message.
- **Main.** This subroutine is the main subroutine. The database connection is initialized in this subroutine, and the user interface is loaded.

This module has the following variables:

- **DcomObjects.** It represents the collection of MachBag objects. It contains all currently available and initialized DCOM objects. Each new DCOM object is represented in a MachBag class as a collection item.

- **WebBrowsers.** It represents the collection of Browser objects. It contains all currently available and initialized Browser objects. Each new Browser object is represented in a Browser class as a collection item with a unique BrowserGUID.

5.2.3. SimContModule Module

This module has the following methods:

- **BrowserInit.** This function initializes a new Browser object and register it to the WebBrowsers collection.
- **ParseString.** This function parses an input string into a collection of keys and values. strDelimiter indicates the character which is used to seperate key, value pairs from other key, value pairs. And strKeyValSeperator indicates the character which is used to seperate key and its value.
- **ProcessCommands.** This function first checks whether the current Browser is registered to the WebBrowsers collection, and whether it has the controller that is responsible for controlling the specified object. If there is a controller responsible for the specified object, it passes the coming command to the DCOM Object by calling the PassCmdDCOM function. If there is no controller responsible for the specified object registered to this Browser.Controllers collection, it first initializes a new Virtual Controller, and register it to current Browser.Controllers collection. After this initialization process, the DCOMProxy Monitor's `Available Machines` sub section is updated, and the received command is passed to the DCOM Object by calling the PassCmdDCOM function.
- **PassCmdDCOM.** With the help of this function the received messages are sent to their corresponding Virtual Controllers according to the Message Content. Load, Start, Stop, Pause, Exit and Disconnect request are sent to the appropriate DCOM Objects (machines, jobs...) with the help of a virtual controller interface. And after every processed request, the request is logged to DCOM Specific LOG on the DCOMProxy Monitor.
- **DisconnectBrowser.** This function is used to disconnect the browser from the application server.

StatSendInterval constant indicates the status sending interval time in miliseconds.

5.2.4. VirtualController Class

This class is used to represent controller interface of a browser to a DcomObject(machine, job ...). A browser may control many machines, and therefore may have many virtual controllers. A virtual controller will connect to the specfied DCOM Object if it is previously initialized, if not it will first initialize the specified DCOM Object and then connect to it.

VirtualController class has the following methods:

- **CopyMemory.** This function enables to pass the reference of an object to another object. Although it is not possible to pass objects by reference in Visual Basic, this API call enables it.
- **Init.** This function is one of the core functions of the DCOMProxy/AppServer. It is executed in every initialization of a virtual controller. If a new browser connects to the system, and if it wants to control an object, a new virtual controller has to be initialized to control the specified object. This function checks whether the specified object (machine) is currently working. If it finds a working object (machine), then it gets the running objects properties from DcomObjects collections MachBag object. It creates a new instance of the object through the objects reference. With the help of this mechanism, any object can be controlled through many controllers. If the specified object is not running, then it initializes a new object, and registers it into the DcomObjects collection as a MachBag object and uses it.
- **Button_load.** It sends a load command to the controlled object. It emulates a desktop applications load button. With the help of this function the load request of the GUI (Web Browser, Controlling Client) is passed to the object with all the parameters.
- **Button_start.** It sends a start command to the controlled object. It emulates a desktop applications start button. With the help of this function the start request of the GUI (Web Browser, Controlling Client) is passed to the object.

- **Button_pause.** It sends a pause command to the controlled object. It emulates a desktop applications pause button. With the help of this function the pause request of the GUI (Web Browser, Controlling Client) is passed to the object.
- **Button_stop.** It sends a stop command to the controlled object. It emulates a desktop applications stop button. With the help of this function the stop request of the GUI (Web Browser, Controlling Client) is passed to the object.
- **Button_exit.** It sends an exit command to the controlled object. It emulates a desktop applications exit button. With the help of this function the exit request of the GUI (Web Browser, Controlling Client) is passed to the object.
- **MachineTimer_Tick.** This subroutine is very similar to the DCOM objects process subroutine. If the machineTimer object is enabled, then at every specified StatSendInterval, this subroutine is called. It checks whether the objInformBrowsers flag is set to true, and if it is set to true, it sends its status information to all connected machines through the InformAll function.
- **OServer_eBreakDown**. If the object breaks down it sends a BreakDown event. In such a case this function will be invoked.
- **OServer_eFinish.** If the object completes its job it sends a Finish event. In such a case this function will be invoked.
- **OServer_eSendStatus.** If the object raises SendStatus event, this function will be invoked. And the current status of the object is acquired.
- **OServer_eSlowDown.** If the object slows down it sends a SlowDown event. In such a case this function will be invoked.

VirtualController class has the following properties:

- **MsgQueue.** It represents the name of the corresponding MSMQ Queue, which is used for the communication of the DCOM object. Actually it is a GUID, and is equal to the BrowserGUID.
- **MachServer.** It represents the server name on which the DCOM object resides.
- **MachType.** It represents the type of the object. Currently there are two types of objects, which are jobs and machines.
- **MachName.** It represents the real life name of the object.

- **MachID.** It represents the ID of current object. It is used as an index for objects.
- **Status.** Status indicates the completion percentage of object's current job.
- **LogMsg.** It holds the last status message that is generated by the controller.

5.2.5. MachBag Class

This class is used to represent the connected DcomObjects (Machines, Jobs ...). Each newly connected object (machine, job) will be represented by this class in `DcomObjects` collection.

MachBag class has the following properties:

- **ObjInformBrowsers.** It is a flag, which determines, whether the object informs the controllers that are connected to this object, or not. If this flag is set to true, the events and status of this object will be sent to every virtual controller that are connected to this object.
- **ObjCurrentState.** It represents the current state of the machine. It shows the functions that can be executed in the current state. Since there may be many controllers, which want to control this machine, it is essential to make the sanity check in this level. It is a 6 digit binary number. The digits 0 and 1 represent whether the function can be executed or not. From left to right the digits correspond, mLoad, mStart, mPause, mStop functions. And the last 2 digit is reserved for Logout and Status functions for the web interface. It is a sanity check parameter, and used for dynamically generating the web based controller interface for this machine.
- **ObjServerName.** It represents the server name on which the DCOM object runs.
- **ObjType.** It represents the type of the object. Currently there are two types of objects, which are jobs and machines.
- **ObjName.** It represents the real life name of the object.
- **ObjID.** It represents the ID of current object. It is used as an index for objects.
- **ObjGUICount.** It represents the number of DcomObjects that are currently connected to this object.

- **ObjRef.** It is string variable which holds the reference to the object. With the aid of this reference any virtual controller, which need to control this object will get the reference of this object.

5.2.6. Browser Class

This class is used to represent the connecting clients (Browsers, other GUIs). Each newly connected browser (GUI) will be represented by this class in `WebBrowsers` collection, with a unique BrowserGUID.

Browser class has the following properties:

- **UserName.** It represents the name of the user, who is currently logged in to control the system.
- **BrowserAttributes.** It is used for logging purposes. It represents the attributes of the current browser that is logged in. It can be used to determine, the web browser (Netscape, IE, ...) in use, the platform in use etc.
- **Controllers.** Controllers property represents the collection of VirtualController class. A browser may control many machines, and therefore may have many virtual controllers.
- **ConnectionTime.** It represents the connection time of the browser for logging purposes.
- **ConnectionIP.** It represents, as the name implies, the IP of the connected browser. If the browser is connected through a firewall or proxy, the determined IP will be the proxy servers IP.
- **DisconnectionTime.** It represents the disconnection time of the browser from the system.
- **CommandLog.** It is used for logging purposes. All commands sent through this browser will be stored in this variable.
- **BrowserGUID.** It is used to uniquely identify each browser. It is a Globally Unique Identifier (GUID). It is used instead of IP, since IP numbers may not be unique in case of a proxy connection. GUID stands for Globally Unique IDentifier, a 128-bit (16-byte) number generated by an algorithm designed to ensure its uniqueness. This

algorithm is part of the Open Software Foundation (OSF) Distributed Computing Environment (DCE), a set of standards for distributed computing.

5.3. WebFrontEnd Project

5.3.1. Global.asa

Global.asa file is a special file residing in the WebFrontEnd project directory, where one can define session and application specific functions. It runs in every request to the index file of this directory, and also enables the session object. In this file application wide database settings are initialized through Application("VarName") variables, and MSMQ settings through session("VarName") variables.

Global.asa has the following subroutines:

- **Session_OnStart.** When a new session starts, this subroutine will be invoked automatically. The Appserver's receiving queue name, current user's authentication status variables are initialized. The "WebFrontEnd_GenLib.GUIDGen" ActiveX DLL is called and a new GUID is generated and assigned as current browsers receiving queue name.
- **Session_OnEnd.** When the current session ends, this subroutine will be invoked automatically. The browsers receiving queue will be deleted from the MSMQ Server.
- **Application_OnStart.** When the application starts, this subroutine will be invoked automatically. The database related connection parameters are initialized in this subroutine.

5.3.2. Index.asp

Index.asp is the startup file of the directory containing the WebFrontEnd project. This file is a transaction enabled asp file. It first checks whether the current user is authenticated or not, and if it is not authenticated, this asp file redirects itself to login.asp. Otherwise it creates first its receiving queue, and then acts as a simple html file to arrange the frames that will be displayed to the user as shown in the Figure 4.4.

The indicated frames will be displayed by the following html, asp files:

a. LeftMenu frame with LeftMenu.asp
b. Controller frame with welcome.htm, controller.asp
c. StatusWin frame with Statuswin.asp, ShowStatus.asp
d. ResultSet frame with Resultset.asp

Index.asp has the following subroutines:

- **InitializeMSMQ.** This subroutine first checks whether a queue with its receiving queue name exists, and if not, it creates its receiving queue on the MSMQ Server defined in Session("mname") variable. After the creation of the queue this function sends "INIT" message to AppServer's receiving queue, to announce itself as a new browser.
- **Session_OnEnd.** When the current session ends, this subroutine will be invoked automatically. The browsers receiving queue will be deleted from the MSMQ Server.

5.3.3. LeftMenu.asp

LeftMenu.asp is displayed on the leftmost frame and it presents the available object tree. To make the application a cross-browser (ie. Compatible with IE and Netscape) application Dan Steinman's [16] "DynAPI", an open source javascript programming library is used. This asp file retrieves the relevant information from the database, and displays it in a tree view menu format.

LeftMenu.asp has the following functions:

- **Init.** This function is used to initialize the tree view menu.
- **RedirectAll.** This function will be invoked by on_click event when a link is clicked. Controller frame will be redirected to ContUrl parameter, and StatusWin frame will be redirected to StatUrl parameter. A random variable is attached to each Url parameter to force the browser not to cache the contents of the corresponding files.

5.3.4. Controller.asp

Controller.asp is the heart of the simulation/control process. It first gets all the parameters passed through the Request.QueryString object, and session('VarName') variables. It will display the corresponding control buttons according to the state of the object that is currently simulated or controlled. The DCOM objects pControlState property determines which control buttons will be displayed. After clicking any of the control buttons appropriate client side javascript function will be invoked. These functions will do the rest of the job.

Controller.asp has the following functions:

- **Load.** This function gets the parameters that need to be passed to the DCOM object. All these variables will be passed as a querystring to sendcmd.asp file. Sendcmd.asp will be displayed in ResulSet frame. Again to avoid caching of the browser a random variable is attached to the QueryString as a new parameter.
- **SendCommand.** This function gets an input variable called "No". No variable is an integer variable, and the numbers two to six represent the Start, Pause, Stop, Logout and Disconnect buttons respectively. Start, Pause, Stop buttons cause the appropriate QueryString to be passed to ResultSet frame with sendcmd.asp file. Logout button will send the appropriate QueryString to the Controller frame with logout.asp file. And Disconnect button will send the appropriate QueryString to a new page by deleting all the frames with disconnect.asp file. Again to avoid caching of the browser a random variable is attached to the QueryString as a new parameter.
- **ShowStatus.** Status button will send the appropriate QueryString to the StatusWin frame with Showstatus.asp file.
- **Logout.** Logout button will redirect the browser to the logout.asp to terminate the session, and connection.

5.3.5. Sendcmd.asp

This file is a transaction enabled asp file. It first gets all the parameters passed through the Request.QueryString object, and session('VarName') variables. It creates

appropriate MSMQ objects for sending a transactional message. It sets the label of the message, according to the message type, and sets all the parameters passed to the message.body. So the requested commands will be sent to the MSMQ servers, to be processed by the DCOM/Proxy AppServer. In case of a failure of sending the message, the OnTransactionAbort function will be called to indicate the failure status in the resultset frame.

5.3.6. ShowStatus.asp

This file requests itself in every five seconds, if it is active, i.e. displayed on the status frame. This asp file first connects to its receiving queue, and retrieves the messages that is sent to the current browsers queue about the status of the connected machines. It loops through the messages of the queue to find the last message containing information about the current selected machine. By including progressbar.inc.asp, it has included the graphing function. It passes the currentstatus value to the DrawPrgBar() function, to graphically represent the status of the selected nachine, then it lists the log messages of the selected machine below this image.

5.3.7. Login.asp

This file is used for the control of access of the users. A user must login to the system before it can use the controller. It gets the user name and password variables from the user, validates it against to the database, and according to the result, either lets the user go in, or refreshes itself to block the user.

5.3.8. Logout.asp

This file is a transaction enabled asp file. It sends "Exit" command to the MSMQ servers to be processed by the DCOM/Proxy AppServer. After successfully sending the EXIT command, the statuswin frame is initialized, i.e. refreshed to its initial content.

5.3.9. Disconnect.asp

This file is a transaction enabled asp file. It sends "Disconnect" command to the MSMQ servers to be processed by the DCOM/Proxy AppServer. Then it deletes its receiving queue from the MSMQ servers, and displays the disconnected message on the whole browser window.

5.3.10. Progressbar.inc.asp

This file includes the DrawPrgBar() function. It can be called from any page. This function represents the current status of any object graphically.

5.4. WebFrontEnd_GenLib

5.4.1. GUIDGen Class

This class and project is used to create an ActiveX DLL, which can be called from the WebFrontEnd Project, to create unique GUIDs in ASP pages.

GetGUIDString function creates a GUID string by making an API call, and returns this GUId string.

6. DISCUSSIONS AND CONCLUSIONS

In this study we developed an object oriented web-based distributed simulation and control system using a technology that is built upon a set of common standards.

The resulting infrastructure introduces virtual manufacturing component objects, which is tried to be extended from virtual manufacturing devices (VMD) [13] concept, and which have the primitive methods, those can be found in their real-life manufacturing component counterparts. With the aid of these objects any manufacturing component in a factory or a manufacturing environment as whole can be represented and modeled. These objects may act as counterpart of a real-life manufacturing components in a simulation environment, or may act as a real-time execution interface of the real-life machines in a control environment. The virtual components are developed as DCOM objects, and these objects can be reused even in the binary level, besides source code level. Since the specification is at the binary level, it allows integration of binary components possibly written in different programming languages such as C++, Java and Visual Basic.

The resulting architecture has the ability to minimize the execution load and time of a complex simulation by distributing the simulation to many physical computers, and can also simulate third-party simulator objects designed according to the system standards. The developed architecture also enables the control of any manufacturing machine of a factory from any source, which can either be the web based clients, or a third-party enterprise-wide software running anywhere.

Although the distributed component object model (DCOM) enables the communication between objects via Internet, we did rather prefer to use a messaging layer with a queuing mechanism and a DCOM/Proxy Application Server. The DCOM/Proxy Application Server enables seamless communication between in-house virtual manufacturing component objects and web based clients, or third party enterprise-wide anywhere running software. The developed architecture enables a secure, asynchronous communication with the virtual manufacturing components without a need of a thin client, i.e. even a simple javascript enabled web-browser can communicate with the components,

through the use of Microsoft Message Queue servers. The implemented queue mechanism aims to recover any downtime of the DCOMProxy Application Server, and when the application server comes back alive, all queued commands can be processed immediately according to the order they arrived. In other words the developed system enables 24x7 working through its scalable and reliable architecture.

Any communication need between any of the players in the manufacturing environment, i.e, the manufacturing components, or the enterprise-wide legacy system, is satisfied with the proposed architectures messaging system. Any request for object to object communication is routed to the messaging system, and the messaging system enables the communication. In this way the objects or the controller of the objects does not care which low level interface the object provides, and if it is alive, i.e. running, so that a message can be sent.

It may seem contradictory to our approach not to use a standard like HLA in our distributed simulation system, but we think it is not. First of all, with our developed architecture we provide a very simple object programming environment compared HLA's complex structure. Anyone who has ever dealt with Microsoft Word's or Excel's macro language (namely Visual Basic scripting language), has knowledge about Visual Basic to some extend, and can easily implement its custom virtual manufacturing component model to our Real Life Machine Server object template. Of course in addition to Visual Basic other languages like C++, Java, and Pascal can also be used to implement these object templates.

On the other hand, since the HLA is developed mainly for military simulations by the US Department of Defense, the applications in the civil fields are rare. Access to HLA tools requires a complex registration process, which includes the verification of the registrars address via postal mail, although it was allowed to download the software in the beginning of our study. The usage statistics of HLA is rather interesting. As of October 1999 the breakdown of the number of RTI (Run Time Infrastructure) and Tools downloads presents, that the most downloaded country was Germany with 99 downloads, and the least downloading country was Israel with 7 downloads, and Turkey was in this foreign countries list with 12 downloads.[11] As we came to September 2000, according to the the

HLA update by Phil Zimmerman, total number of RTI downloads reached to 3863, of which only 34 per cent is international downloads.

Besides the simulation of a system we also wanted to reuse the developed system for controlling purposes, and this combined with the above concerns leaded us to develop a custom made system for our purposes. But we inherited several key concepts from HLA like reusability, interoperability, and component based approach for our developed system.

When we come to the application areas of our developed system, one will see that there are many options in implementing this system.

The developed architecture may be used as an execution layer for the legacy systems in a plant with its message driven architecture. This will enable adding execution capability to any third-party enterprise wide legacy system (ERP, MRP, APS), which means for example, operation plans developed by the legacy system can be released to the physical entity by the help of our developed architecture. The only requirement is that the third-party software has to be able to communicate with our MSMQ servers to send commands to the execution interfaces of real life manufacturing components.

With the help of our system's web based parametric message passing interface, which is utilized by the WebFrontEnd application, the above mentioned execution layer capability can even be extended to internet. That means any plant located worldwide can be controlled from the headquarter by using our developed system integrated to their enterprise-wide software and to the plant which is desired to be controlled.

Above applications show the usage of our system in the control area. But our developed system is also capable to simulate real world manufacturing components for testing purposes. This also includes testing of the performance of a scheduling software. Consider a situation where a plant is running, and the managers are proposed to replace the existing scheduling system. Before replacing the system, the managers has to be convinced about the performance of the proposed system, so they can run our simulation environment to test the proposed scheduling system.

Also our developed system can be used in the education area, especially for experimentation purposes. With the help of our developed system, students can easily create virtual manufacturing environments for their specific needs. And the need for a costly, hard and time-consuming setup of physical real manufacturing environments can be eliminated by the developed system.

Up to this point we stated that the system is capable of representing any manufacturing environment, and that it can be used for both simulation and control purposes. Now we will focus how one may define other possible objects as AGV (Automated Guided Vehicle), a robot, a CNC machine, a job, or a plant, i.e. the manufacturing environment itself.

We first need to define an object template for the real-life manufacturing component, which we want to describe in terms of our virtual manufacturing component objects. To define any of the object model templates, one has first to define the generic methods and properties to be implemented on the objects. The methods and properties defined in this manner do not have to be the same methods or properties like in the Real Life Machine Server objects. After defining these methods and properties, then the message structure has to be defined and adapted to the system's message structure.

One can define an object template for AGV's in the system. One has to define first the generic methods and properties of AGV's. For example routing information, collection and delivery point information may be the general input to the object template representing the AGV. In our case the Real_Life_Machine Server objects had primitive methods like load, start, pause and stop, and several parameters for the loading process. In the case of an AGV template the loading process may involve all the initialization parameters like the routing information, and delivery and collection point location information. The start method may start the AGV to go to the desired locations defined in the initialization process. The pause and stop methods may pause and stop the AGV in motion respectively. Two other new methods may be implemented for the "Get Part" and "Release Part" processes. The process function in this case will represent the traveling of AGV. The breakdown error may represent a collision of the AGV with another AGV or a real breakdown. A slowdown event may represent the slowdown of the AGV. The status

information may represent the location of the AGV, and the part carrying status of the AGV.

One may even represent the overall factory operation as a virtual manufacturing component in a factory object template, and so one can define many factories located worldwide as system objects. These objects may be used as the monitoring objects, which will get information from the factories and represent them in real time, or they may for example be used in a supply chain simulation to represent the behavior of the factory in a supply chain. In this case, the objects load function may represent a process, which will set the preconditions, and start function may represent the beginning of the processes in the factory. And the status of such a system may represent the total number of end products that are needed to be delivered to the other party of the supply chain as semi-finished products.

As it can be seen with the aid of the proposed architecture, there are numerous design and setup possibilities, which may be used in a simulation or control environment, or in both of them. With the aid of the proposed architecture virtual environments composed of real machines, and virtual components can be designed and let to interact each other.

We believe that the implemented technology and the designed architecture is a very extensible structure, and the mechanism can be implemented to many real life scenarios.

The starting point of this study was the simulation need for the testing of an object oriented distributed scheduling system from a previous M.S. thesis by Oguz [9]. Throughout the study, the implementation of technologies brought us new perspectives, and the control facility is arised from the technological capability of the system. From that point on, we tried to generalize the coding structure especially for the handling of the virtual manufacturing components. More generalization in the template structure will ease the use of virtual manufacturing components, stemming from an object template, which have different number of initialization parameters, or different number of methods.

Set of virtual manufacturing object templates may be created to further promote the reuse of these objects. And an infrastructure to classify and store these templates may be created.

Currently since we insisted on the use of dummy clients, the status information is delivered to requestors with the help of both the messaging mechanism, and the database system, although the command execution do not require database system. This may be completed with only messaging mechanism.

APPENDIX A: COMPLETE MESSAGE REFERENCE

The messages, that will be sent by the WebFrondEnd application to the DCOM/Proxy AppServer application, are listed below with appropriate samples.

A.1. Init Message Sample

Message Label: Client={98CA1AA9-6FF6-4C92-AFD3-EE694E05B4A9}&MsgType=INIT&IP=194.1.1.100&UserAgent=Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)&User=kursun
Message Body: Empty

A.2. Load Command Message Sample

Message Label: Client={98CA1AA9-6FF6-4C92-AFD3-EE694E05B4A9}&MsgType=Command&MsgContent=Load&SM=Torna-mklap2000&ST=m&SID=1
Message Body: Action=1&Cap=50&CapType=21&BProb=4& BDT=2&RepTD=12&LoadD=4& WorkD=75&ST=m&SM=Torna-mklap2000&SID=1&RndNo=318

A.3. Start Command Message Sample

Message Label: Client={98CA1AA9-6FF6-4C92-AFD3-EE694E05B4A9}&MsgType=Command&MsgContent=Start&SM=Torna-mklap2000&ST=m&SID=1
Message Body: Action=2&ST=m&SM=Torna-mklap2000&SID=1&RndNo=207

A.4. Pause Command Message Sample

Message Label: Client={98CA1AA9-6FF6-4C92-AFD3-EE694E05B4A9} &MsgType=Command&MsgContent=Pause&SM=Torna-mklap2000&ST=m&SID=1

Message Body: Action=3&ST=m&SM=Torna-mklap2000&SID=1&RndNo=138

A.5. Stop Command Message Sample

Message Label: Client={98CA1AA9-6FF6-4C92-AFD3-EE694E05B4A9}&MsgType=Command&MsgContent=Stop&SM=Torna-mklap2000&ST=m&SID=1

Message Body: Action=4&ST=m&SM=Torna-mklap2000&SID=1&RndNo=72

A.6. Logout Command Message Sample

Message Label: Client={98CA1AA9-6FF6-4C92-AFD3-EE694E05B4A9}&MsgType=Command&MsgContent=Exit&SM=Torna-mklap2000&ST=m&SID=1

Message Body: Empty

A.7. Disconnect Command Message Sample

Message Label: Client={98CA1AA9-6FF6-4C92-AFD3-EE694E05B4A9}&MsgType=Command&MsgContent=Disconnect&SM=Torna-mklap2000&ST=m&SID=1

Message Body: Empty

A.8. Status Message Sample

Message Label: 38.85% of the operation seems to be completed. Current State=010011

Operation details for **/1*m/ Torna on mklap2000** are as the following:

Message Body: New client {A9799601-3387-43E4-BC10-583D674E9C49} initialized at 02/01/2001 8:56:30 PM to use Torna on mklap2000

Machine is loaded at 02/01/2001 8:56:31 PM by {A9799601-3387-43E4-BC10-583D674E9C49}

Machine is started at 02/01/2001 8:56:37 PM by {A9799601-3387-43E4-BC10-583D674E9C49}

Machine is paused at 02/01/2001 8:56:49 PM by {A9799601-3387-43E4-BC10-

583D674E9C49}

Machine is restarted at 02/01/2001 8:56:52 PM by {A9799601-3387-43E4-BC10-583D674E9C49}

Slowdown of a factor of 1.06 occured at 02/01/2001 8:56:54 PM

.

.

.

APPENDIX B: SETTING UP THE SYSTEM

The system is composed of several applications, i.e. a web based application called "WebFrontEnd" , a desktop application called DCOM/Proxy AppServer, and several DCOM objects. A sample setup is shown on Figure B.1.

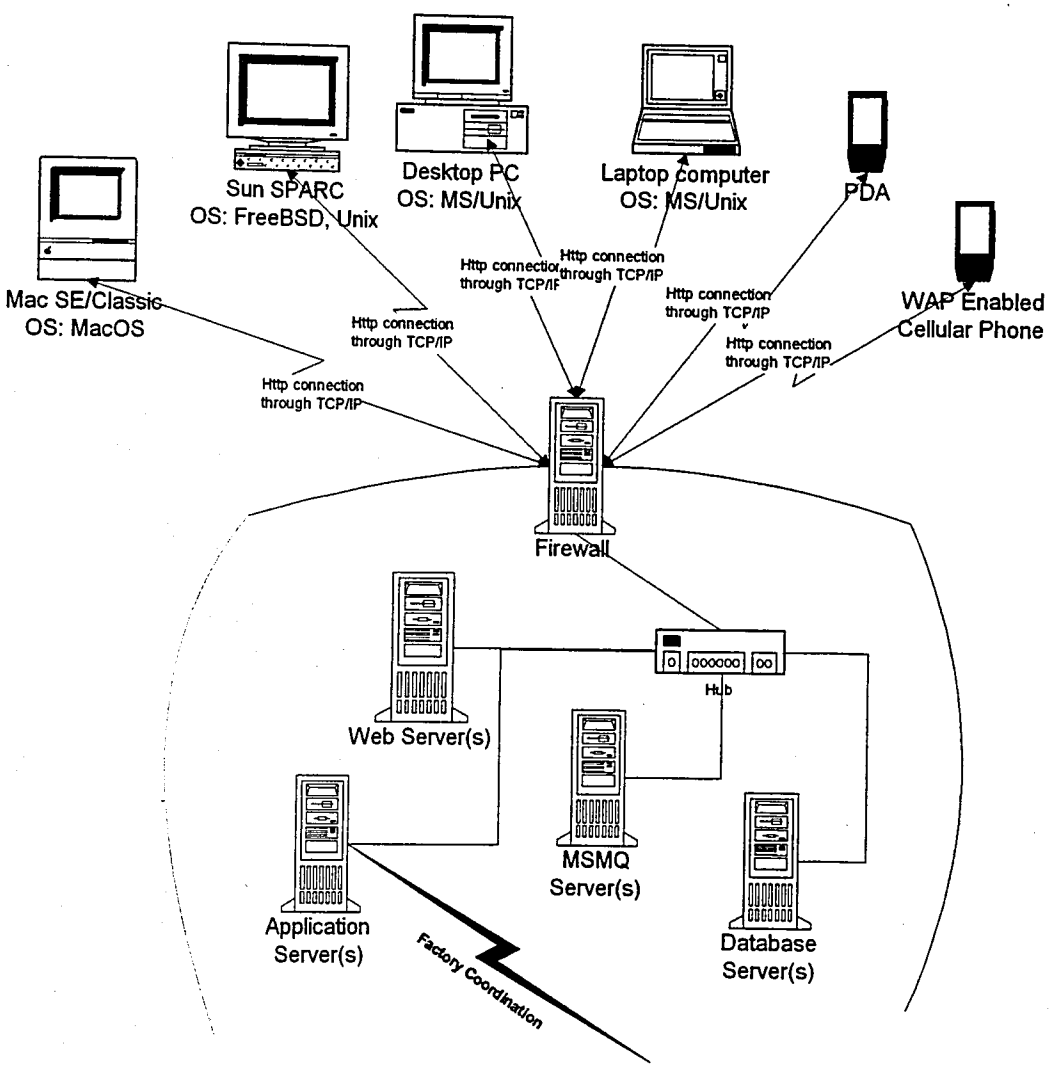

Figure B.1. A sample setup overview of the system

And the interaction between the remote client and the DCOM objects is presented on Figure B.2.

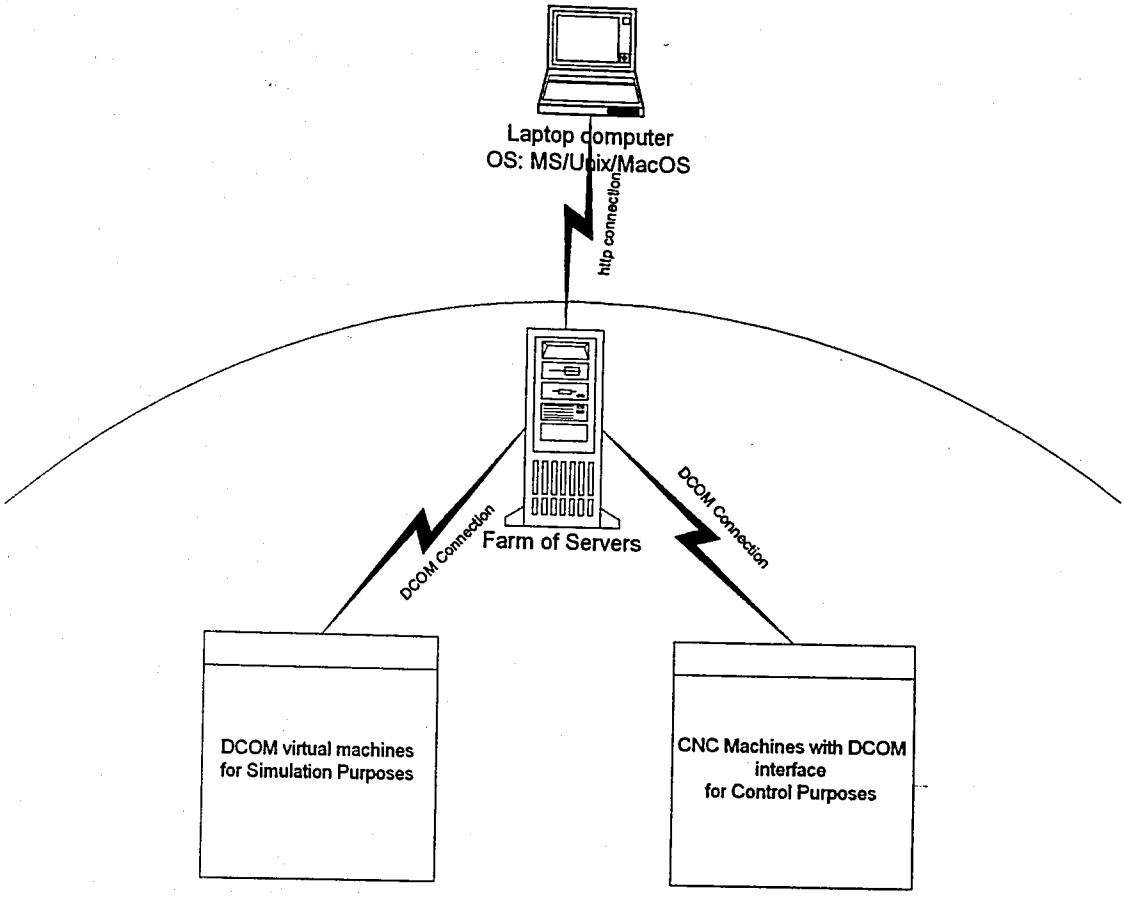

Figure B.2. Interaction between the remote client and DCOM objects

To provide a secure system, a firewall may setup between the internal network, and internet connection. A web server, or farm of web servers, preferably MS Internet Information Server(s) has to set-up to serve the "WebFrontEnd" application on the Internet. The web server has to be configured, either for a new site setup, or a virtual directory setup, to serve the web based application. For the messaging system MSMQ Server(s) has to set-up. To run the provided software a public queue called "Controller" has to be created as a transactional queue. After creating this queue proper security settings has to be performed, in order to allow internet visitors be able send messages to this queue. The database file called "cc.mdb" has to be copied in an arbitrary directory, on the machine, that will act as a database server. An ODBC DSN (Data Source Name) record has to be created with the name "ControllerME", and Microsoft Access Driver has to be chosen as the engine, and the physical location of the file has to defined. Now the last part is setting the DCOM Objects. To run the sample system the "MachineServer.exe" file that can be found on the diskettes provided, has to be registered on every computer, that will

serve as virtual machines. In order to register these objects, the machine on which these objects will run has to be DCOM enabled. The setup program provided will ease the setup of the objects. To register them manually, simply enter the following command on the command prompt of the windows machine, that will run these objects:

- *"regsvr32 c:\location\to\MachineServer.exe"*

More than one object may be registered on one physical computer. After registering the components, the components has to defined in the Access database. The machines table represent the machine names and their running computers name. The current setups values are already on this table. One may either define new values to this table, or edit the predefined values, and set the MachineServerName attribute to the name of the machine on which the DCOM object will run. It is now time to setup up the DCOM/Proxy AppServer program. The setup program for this application server is also provided with the CD. Although the previously defined configuration may reside on several computers, it is also possible to have all in one setup, so that every application will run on the same computer. Now point your browser to the newly defined virtual directory of the web server, and that's all.

APPENDIX C: DATABASE STRUCTURE

In the proposed system every ODBC database can be used. In our case we used MS Access 2000 as the database server. The structure of the database is shown in Figure A.1.

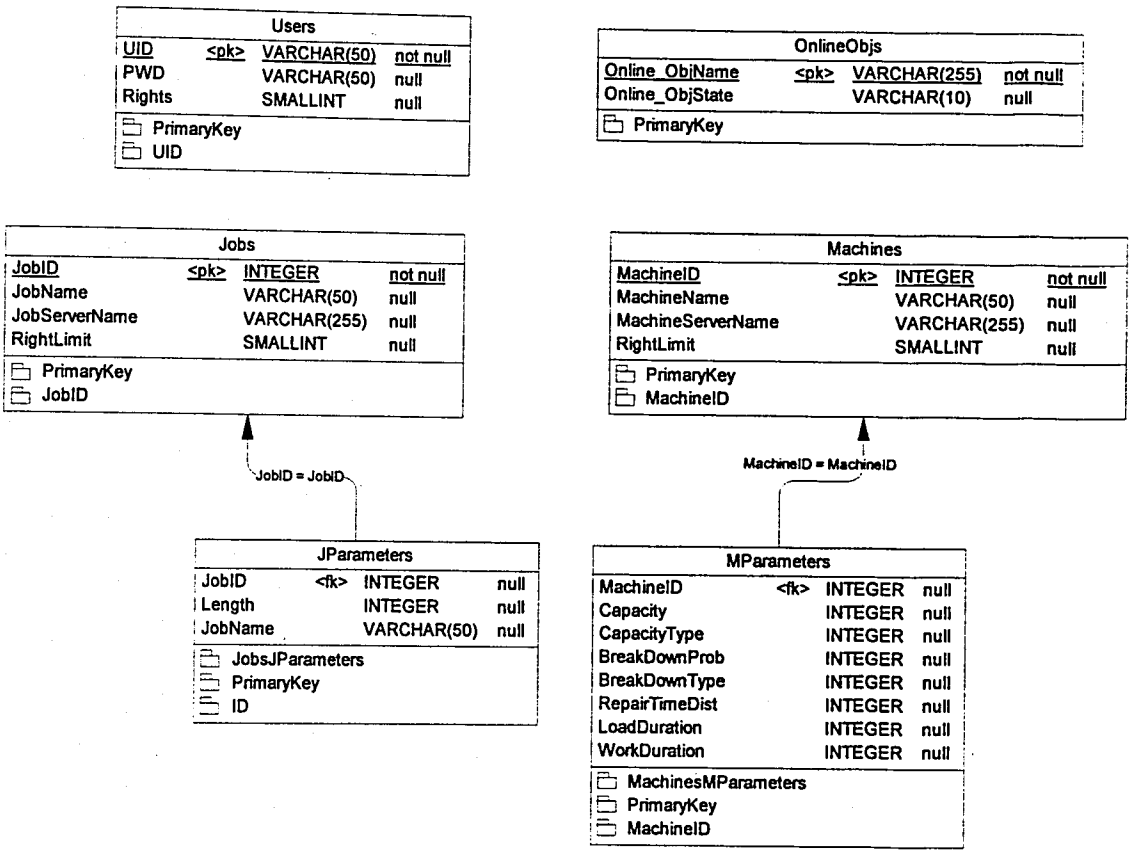

Figure C.1. The database model

C.1. SQL Code to Generate the Database

```
create table OnlineObjs
(
   Online_ObjName    VARCHAR(255)        not null,
   Online_ObjState   VARCHAR(10)                ,
   primary key (Online_ObjName)
);
```

```
create unique index PrimaryKey on OnlineObjs (Online_ObjName asc);

create table Users
(
   UID            VARCHAR(50)      not null,
   PWD            VARCHAR(50)            ,
   Rights         SMALLINT               ,
   primary key (UID)
);

create unique index PrimaryKey on Users (UID asc);
create unique index UID on Users (UID asc);

create table Jobs
(
   JobID              INTEGER          not null,
   JobName            VARCHAR(50)            ,
   JobServerName      VARCHAR(255)           ,
   RightLimit         SMALLINT               ,
   primary key (JobID)
);
create unique index PrimaryKey on Jobs (JobID asc);
create unique index JobID on Jobs (JobID asc);

create table Machines
(
   MachineID          INTEGER          not null,
   MachineName        VARCHAR(50)            ,
   MachineServerName  VARCHAR(255)           ,
   RightLimit         SMALLINT               ,
   primary key (MachineID)
);
```

```
create unique index PrimaryKey on Machines (MachineID asc);
create unique index MachineID on Machines (MachineID asc);
create table JParameters
(
   JobID          INTEGER          ,
   Length         INTEGER          ,
   JobName        VARCHAR(50)      ,
   foreign key  (JobID)
   references Jobs (JobID)
);
create unique index JobsJParameters on JParameters (JobID asc);
create unique index PrimaryKey on JParameters (JobID asc);
create index ID on JParameters (JobID asc);

create table MParameters
(
   MachineID          INTEGER          ,
   Capacity           INTEGER          ,
   CapacityType       INTEGER          ,
   BreakDownProb      INTEGER          ,
   BreakDownType      INTEGER          ,
   RepairTimeDist     INTEGER          ,
   LoadDuration       INTEGER          ,
   WorkDuration       INTEGER          ,
   foreign key  (MachineID)
     references Machines (MachineID)
);
create unique index MachinesMParameters on MParameters (MachineID asc);
create unique index PrimaryKey on MParameters (MachineID asc);
create index MachineID on MParameters (MachineID asc);
```

APPENDIX D: LOW LEVEL OBJECT HANDLING WITH VB6

The DCOM/Proxy AppServer program is written with Visual Basic 6. Since it is rapid application tool (RAD) everything went smoothly, but at some point we realized that we cannot pass the objects references, and so can not control an instance of an object with more than one other controlling object. We made an extensive research o this area, and found the real problem, running object table (ROT) registration problem.

When an object gets created based on a class from an ActiveX Exe it gets created in its own memory area (Multiple objects based on the same class cannot share their data). In C++ that object registers itself with the ROT (Running Object Table) so that anybody who needs to, can access that instance of the object. For instance Excel. When Excel is running you can refer to the running instance from VBA/VB, and access any data that's present in the spread sheet. However VB based objects lack that functionality. Activex EXE created in VB will NOT get registered in the ROT. That blocks the multiuse of the same thread of an object.

Thanks to Matthew Curland's article on Visual Basic Programmer's Journal on August 1997 called "Give Your Classes GetObject Support" [18]. With the aid of this articles sample code, we were able to handle all kind of object reference passing.

One more thing to note is that a function, ObjPtr(), used in the programming of DCOM/Proxy AppServer application for the handling of DCOM objects is an undocumented Visual Basic function. Therefore we will give a brief description about this function. But unfortunately Microsoft does not support this function and a series of low-level functions, which are VarPtr, VarPtrArray, VarPtrStringArray, StrPtr, ObjPtr. Although Microsoft states that it does not guarantee that they will be available in future releases of Visual Basic, it is available since Visual Basic 1.0.

D.1. ObjPtr Function

ObjPtr returns the pointer to the interface referenced by an object variable. ObjPtr takes an object variable name as a parameter and obtains the address of the interface referenced by this object variable. One scenario of using this function is when you need to do a collection of objects. By indexing the object using its address as the key, you can get faster access to the object than walking the collection and using the Is operator. In many cases, the address of an object is the only reliable thing to use as a key. [17]

APPENDIX E: DISTRIBUTED OBJECT TECHNOLOGY

The explosive growth of the Web, the increasing popularity of PCs and the advances in high-speed network access have brought distributed computing into the main stream. To simplify network programming and to realize component-based software architecture, two distributed object models have emerged as standards, namely, DCOM (Distributed Component Object Model) and CORBA (Common Object Request Broker Architecture) [19].

DCOM is the distributed extension to COM (Component Object Model) that builds an object remote procedure call (ORPC) layer on top of DCE RPC to support remote objects. A *COM server* can create object instances of multiple *object classes*. A COM object can support multiple interfaces, each representing a different view or behavior of the object. An interface consists of a set of functionally related methods. A COM client interacts with a COM object by acquiring a pointer to one of the object's interfaces and invoking methods through that pointer, as if the object resides in the client's address space. COM specifies that any interface must follow a standard memory layout, which is the same as the C++ virtual function table. Since the specification is at the binary level, it allows integration of binary components possibly written in different programming languages such as C++, Java and Visual Basic [19].

CORBA is a distributed object framework proposed by a consortium of 700+ companies called the Object Management Group (OMG). The core of the CORBA architecture is the Object Request Broker (ORB) that acts as the object bus over which objects transparently interact with other objects located locally or remotely. A CORBA object is represented to the outside world by an interface with a set of methods. A particular instance of an object is identified by an object reference. The client of a CORBA object acquires its object reference and uses it as a handle to make method calls, as if the object is located in the client's address space. The ORB is responsible for all the mechanisms required to find the object's implementation, prepare it to receive the request, communicate the request to it, and carry the reply (if any) back to the client. The object

implementation interacts with the ORB through either an *Object Adapter (OA)* or through the ORB interface [19].

Currently, the IDL (Interface Definition Language) used by CORBA is quite different from the one used with DCOM, which causes severe interoperability problems. DCOM is based on a proprietary format and is more limiting than CORBA, e.g. DCOM has no naming or trading services, and its objects are not persistent [20]. On the other hand, if a system mainly uses Win32 platforms, it may be simpler and more practical to implement DCOM than CORBA. DCOM also has the added advantage of being a standard part of the Win32 operating systems. The fact is there are no clear winners in the standards for distributed objects. However, bridges are currently being established between these two standards. (CORBA 3.0 will support interoperability between the two standards and Microsoft, working with Iona Technologies and Visual Edge Software, is in the process of doing the same.) The bottom line is that both CORBA and DCOM are here to stay and, if necessary, these two sets of standards for distributed objects and environments can co-exist within the extended enterprise [20].

APPENDIX F: FORMAT OF CD CONTAINING COMPUTER SOFTWARE

The CD contains all the source files and setup programs. There are two main directories on the CD, which are "sources", and "setups". The "sources" directory contains five other directories, called ActiveXLib, DCOMProxy-AppServer, reallife_machines, WebFrontEnd, and db, with respective applications source files in them. The "setups" directory contains the setup programs, and contains two directories, called DcomProxy-AppServer, and reallife_machines, with respective applications setup programs in them.

REFERENCES

1. Hibino, H., Y. Fukuda, S. Fujii, F. Kojima, K. Mitsuyuki, and Y. Yura, "The development of an object-oriented simulation system based on the thought process of the manufacturing system design", *International Journal of Production Economics*, Vol. 60-61, pp. 343-351, 1999

2. Klein, U., "Simulation-based distributed systems: serving multiple purposes through the composition of components", *Safety Science*, Vol. 35, pp. 29-39, 2000

3. Fujii, S., T. Kaihara, and H. Morita, " A distributed virtual factory in agile manufacturing environment", *International Journal of Production Research*, Vol. 38, pp. 4113-4128, 2000

4. Joshi, S. B., E. G. Mettala and J S. Smith, "Formal models for control of flexible manufacturing cells: Physical and system model", *IEEE Transactions on Robotics and Automation*, Vol. 11, pp. 558-570, 1995.

5. Kim, J., J. Park, and J. Park, "A generic event control framework for modular flexible manufacturing system", *Computers & Industrial Engineering*, Vol.38, pp. 107-123, 2000

6. Naylor, A. W. and R.A. Volz, "Design of integrated manufacturing system control software", *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 17, pp.881-897, 1987

7. Smith, J. S., W.C. Hoberecht and S. B. Joshi, "A shop floor control architecture for computer integrated manufacturing", *IIE Transactions*, Vol. 28(10), pp. 783-794, 1996

8. Smith, J. S. and S. B. Joshi, *Message-based part state graphs (MPSG): a formal model for shop floor control*, Technical report, Department of Industrial Engineering, Texas A and M University., 1994

9. Oğuz, S., *Object Oriented Design of a Distributed Scheduling System*, M.S. Thesis, Boğaziçi University, 1998

10. Gan, B. P. and S. J. Turner, "An asynchronous protocol for virtual factory simulation on shared memory multiprocessor systems", *Journal of the Operational Research Society*, Vol. 51, pp. 413-422, 2000

11. DMSO, *The Defense Modeling and Simulation Office's (DMSO) Website*, 2000, http://www.dmso.mil

12. Gan, B.P., L.L.S. Jain, S.J., Turner, W. Cai, and W-J. Hsu, "Distributed supply chain simulation across the Enterprise boundaries", *Journal of the Operational Research Society*, Vol. 52, pp. 484-493, 2000

13. Kovacs, G.L., S. Kopacsi, J. Nacsa, G. Haidegger, and P. Groumpos, "Application of software reuse and object-oriented methodologies for the modeling and control of manufacturing systems", *Computers in Industry*, Vol. 39, pp. 177-189, 1999

14. Gramm, U., and M.Brill, "MMS:MAP application services for the manufacturing industry", *Computer Networks and ISDN Systems*, Vol. 21, pp. 357-380, 1991

15. UML, *OMG Unified Modeling Language Specification V1.3*, 1999

16. Steinman, D., *DynApi Project – an open source javascript library*, 2000, http:// www.dansteinman.com/dynduo

17. MSDN, *Microsoft Developer Network Library*, 1998

18. Curland, M., "Give Your Classes GetObject Support", Visual Basic Programmer's Journal, Vol:8, pp. 109-112, 1997

19. Chung, P.E., Y. Huang, S. Yajnik, D. Liang, J. C. Shih, C. Y. Wang and Y. M. Wang, *DCOM and Corba Side by Side, Step by Step, and Layer by Layer*, 1998, http://www.cs.wustl.edu/~schmidt/submit/Paper.html

20. Reyazat. M., "The Enterprise-Web portal for life-cycle support", *Computer-Aided Design, Vol.32*, pp. 85-96, 2000

REFERENCES NOT CITED

Ahn, G-J, "Role-based access control in DCOM", *Journal of System Architecture*, Vol. 46, pp. 1175-1184, 2000

Alfieri, A. and P. Brandimarte, "Object-oriented modeling and simulation of integrated production/distribution systems", *Computer Integrated Manufacturing Systems*, Vol. 10, pp. 261-266, 1997

Goldfinger, A., D. Silberberg, J. Gersh, J. Hunt, F. Weiskopf, T. Spisz, Z.G. Mou, G. Rogers and R. Semmel, "A knowledge-based approach to spacecraft distributed modeling and simulation", *Advances in Engineering Software*, Vol.31, pp. 669-677, 2000

Miller, J. A., A. F. Seila and X. Xiang, "The JSIM web-based simulation environment", *Future Generation Computer Systems*, Vol. 17, pp. 119-133, 2000

Page, H. E. and J. M. Opper, "Investigating the application of web-based simulation principles within the architecture for a next-generation computer generated forces model", *Future Generation Computer Systems*, Vol. 17, pp. 159-169, 2000

Rembold, U., W. Reithofer and B. Janusz, "The role of models in future enterprises", *Annual Reviews in Control*, Vol. 22, pp. 73-83, 1998

Sheremetov, L.B. and A.V. Smirnov, "Component integration framework for manufacturing systems re-engineering: agent and object approach", *Robotics and Autonomous Systems*, Vol. 27, pp. 77-89, 1999

Wong, A.K.Y., and Dillon, T.S., "A fault tolerant model to attin reliability and high performance for distributed computing on the Internet", *Computer Communications*, Vol. 23, pp. 1747-1762, 2000