TAMPERE UNIVERSITY OF TECHNOLOGY

JOSE ROBERTO RODRIGUEZ ALVARADO

DISTRIBUTED SIMULATION IN MANUFACTURING USING HIGH LEVEL ARCHITECTURE

Master of Science Thesis

Examiner: Professor Reijo Tuokko
Examiner and topic approved in the Faculty of Automation, Mechanical and Materials Council meeting on 8 September 2010

PREFACE

This Master's Thesis was carried out at Tampere University of Technology, in the Department of Production Engineering. The thesis work was performed as part of *MS2Value – Modeling and Simulation of Manufacturing Systems for Value Networks* project. This project was funded by TEKES - National Technology Agency of Finland and a number of Industrial partners.

I would like to thank to my supervisors, Prof. Reijo Tuokko and M.Sc. Ricardo Velez, for the trust that they deposited in me for the realization of this research work. I would like also to thank especially to Dr. Minna Lanz and M.Sc. Eeva Jarvenpää for their advices and support they showed me during the writing process of this work.

Especially, I would like to thank my family, without their support it would have been impossible for me to achieve this goal. They had been always my source of inspiration and a backup of strength when mine have weakened.

In addition, I would like to thank my friends for encouraging me. You and only you are the ones who make my days better, without all of you my life couldn't be fulfilled.

Tampere 22th of September of 2010

Jose Roberto Rodriguez Alvarado
Hämeenpuisto 25 A 3
33210 Tampere
gsm : +358440162487
email: roberto.rodriguez@tut.fi

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Factory Automation
RODRIGUEZ ALVARADO, JOSE ROBERTO: Distributed Simulation in Manufacturing using High Level Architecture.
Master of Science Thesis, 77 pages, 3 Appendix pages
September 2010
Major: Factory Automation
Examiner: Professor Reijo Tuokko
Keywords: distributed simulation, HLA,

Manufacturing is a critical industry for all major economies. Every individual and industry depends on manufactured goods, which makes manufacturing crucial to the national economies. Competition is increasingly hard and globalization is leading to worldwide distribution of production, products and services, affecting all countries and economical regions. At the same time, markets are changing. Customers call for faster product changes and demand products, which are increasingly targeted to individual needs. Mass production is therefore replaced by customised and personalised production of individual products.

Distributed simulation has the potential to become widely applicable for geographically-dispersed manufacturing environments, as is the case with desktop manufacturing or rapidly deployable micro-assembly stations. This thesis focuses on creating a generic framework that permits the distribution of manufacturing simulations, which was one of the goals of the MS2Value (Modeling and Simulation of Manufacturing Systems for Value Chains) project.

Companies, nowadays, normally have their activities and resources geographically dispersed, which represents a challenge for the reusability and interconnection of their manufacturing simulation models. Different approaches have been taken by different communities like the research and military community, but no solution has been presented yet in the manufacturing field.

The thesis work presented here proposes the use of the HLA (High Level Architecture) in combination with a simulation software as a solution to these problems. This proposal is demonstrated by an implementation of a distributed simulation using 3DCreate and an open source RTI (Runtime Infrastructure).

CONTENTS

ABBREVIATIONS AND ACRONYMS

AFRL	Air Force Research Laboratory
ALSP	Aggregate Level Simulation Protocol
API	Application Programming Interface
BPR	Business Process Reengineering
COM	Component Object Model
COTS	Commercial of the Shelf
DARPA	Defense Advanced Research Projects Agency
DDM	Data Distribution Management
DIS	Distributed Interactive Simulation
DMSO	Defense Modeling and Simulation Office
DoD	Department of Defense
DVE	Distributed Virtual Environment
FOM	Federate Object Model
GUI	Graphic User Interface
HLA	High Level Architecture
IEEE	Institute of Electrical and Electronics Engineers
IMTI	Integrated Manufacturing Technology Initiative
LAN	Local Area Network
MOM	Management Object Model
OMT	Object Model Template
PDU	Protocol Data Unit
PLC	Programmable Logic Controller
RTI	Runtime Infrastructure
SME	Small and Medium Enterprises
SOM	Simulation Object Model
TBRP	Technology Blue Ribbon Panel
WAN	Wide Area Network
XML	eXtensible Mark-up Language

LIST OF FIGURES AND TABLES

1. INTRODUCTION

Manufacturing is a critical industry for all major economies. Every individual and industry depends on manufactured goods, which makes manufacturing crucial to the national economies. Competition is increasingly hard and globalization is leading to worldwide distribution of production, products and services, affecting all countries and economical regions. At the same time, markets are changing, customers call for faster product changes and demand products which are increasingly targeted to individual needs. Mass production is therefore replaced by customized and personalized production of individual products.

Manufacturing companies are now living the time where fast prototyping is the rule and products should be assembled fast and accurately. Moreover, the life cycle of products is decreasing constantly on the market and costs have to be cut everywhere. One way to keep up with all those changes has been shrinking of the value chain with the main objective of producing high quality and fast deployable customized products.

Constructing a prototype may be costly, infeasible, and/or dangerous [Fujimoto 2003]. Using 3-D models, designers can study and refine assembly sequences for ease of execution, and identify problems that otherwise might not be detected until significant resources were already committed to production. [IMTI 2003]

MS2Value project

The project MS2Value - Modeling and Simulation of Manufacturing Systems for Value Networks aims at developing methodologies and tools for the concurrent development of manufacturing operations and value networks. The main goal of the project was to develop a generic modeling and simulation framework that would enable the modeling and analysis of manufacturing operations in value networks. The framework includes tools for optimization and analysis on different levels of abstraction, as well as filtering of information from one tier to the next, going from machine level to supply chain.

The modeling and simulation infrastructure was developed/enhanced to support distributed modeling and analysis. Through the automatic linking of several models, the end-user can execute “what-if” scenarios of local manufacturing models and see the changes on a global scale. The modeling infrastructure also

supports model validation to test the compatibility and usability of the created models. Case studies with industrial partners provided valuable feedback on the creation of the models, testing them with real-life scenarios and benchmarking them against best-practices from leading Finnish manufacturing companies.

The research done during this thesis work focused on interconnecting different 3D manufacturing models into a single distributed simulation. The theoretical part of the thesis offers background information about simulation and distributed simulation architectures. Furthermore, it proposes a method to distribute a manufacturing simulation. The practical part of this research analyses a set of software tools for implementing distributed simulation as well as describes the implementation done.

Industrial trends affecting the simulation industry are presented in Chapter 2, as well as the problem presentation and the scope of this thesis work. Chapter 3 contains the theoretical background, the foundation for all this research. The design of the infrastructure needed for a distributed simulation is discussed in Chapter 4, while the practical implementation is explained in detail in Chapter 5. Finally, the conclusions and results are presented in the Chapter 6.

2. TRENDS AND PROBLEM PRESENTATION

The present chapter introduces the forces behind the work performed under the scope of this thesis. It introduces some of the actual challenges that simulation faces on the industry and manufacturing as a driver of change.

2.1. Industrial Trends Affecting the Simulation Industry

Modeling and simulation is the key to optimizing the total product and system design before production; for optimizing the design for speed, quality, and affordability in production; and for optimizing the production processes so that they are in place and ready to execute upon production go-ahead. Maturation of the enabling technologies will enable system developers to slash months and years of development time and reduce costs by 50% or better from current design/build/test/fix practices. [IMTI 2003]

Designing a product in the past has been a continuous iteration between prototyping and test-evaluate-modify those prototypes. With the implementation of M&S (Modeling and Simulation), the time spent on those iterations could be decreased, reducing the overall cost of development of product.

Figure 1 shows the relative cost of the development of a product through time. As shown in the figure below, the major amount of expenses are going directly to testing, evaluating and modify a product.

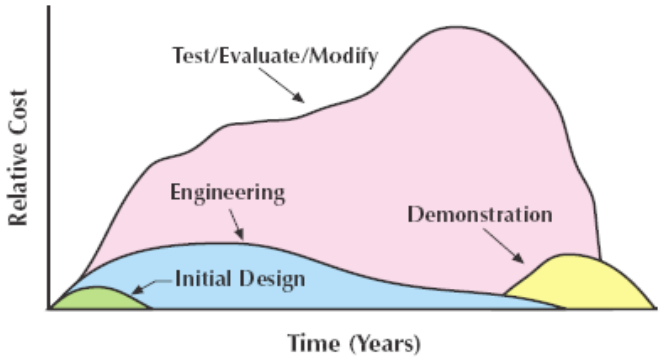

Figure 1 *Iterative Prototyping consumes billion of dollars and years of development for complex products. M&S can drastically reduce those costs [IMTI 2000].*

Simulation, as just mentioned, has proved once and again its abilities as a tool of change, but still its real power hasn't been yet fully exploited. Simulation is often used as a side development in projects or a requisite to fulfill a set of requirements and its results are not taken as a serious approach or are commonly misunderstood. In order to avoid the situations mentioned above and take advantage of simulation a list of goals was set by the industry.

In March 2000, the AFRL (Air Force Research Laboratory) convened a (TBRP) Technology Blue Ribbon Panel to address issues and challenges related to M&S for manufacturing in the defense community. The TBRP effort conducted an extensive research of published studies and conference and workshop proceedings to identify manufacturing M&S technology voids and barriers to implementation. In addition, the team conducted several one-day visits to various prime contractors, government organizations, and software vendors to identify and validate technology voids and gain insight into each company's needs and current information technology modernization plans. At a high level, the TBRP identified five technology voids it considered critical [IMTI 2003]:

- Physical representation
- New and improved tools
- Database integration
- Ease of use
- Training.

Based on these gaps, the IMTI established a plan to fill the holes known as the Modeling and Simulation for Affordable Manufacturing Roadmap.

This roadmap defines more than 75 top-level goals and 250 supporting requirements for research, development, and implementation of M&S technologies and capabilities. Subsequent processing by the workshop participants distilled these needs into four focused, high-level goals [IMTI 2003]:

- Automated Model Generation – Develop techniques to enable automated generation and management of models at various levels of abstraction for multiple domains.
- Automated Model-Based Process Planning – Provide the capability to automatically generate manufacturing process plans based on product, process, and enterprise models, with integrated tools to evaluate producibility of features, resources, and repeatability.

- Interoperable Unit Process Models – Develop a shared base of robust, validated models for all materials and manufacturing processes to enable fast, accurate modeling simulation of any combination of processing steps.

- Scalable Life-Cycle Models – Provide the capability to create and apply scalable product life-cycle models in every phase of the life cycle and across all tiers of the supply chain.

In addition to those goals introduced by the IMTI, Fowler [Fowler, 2003] mentions four grand challenges that should be accomplished by simulations:

- Grandest Challenge #1 – An Order-of-Magnitude Reduction in Problem-Solving Cycles.

- Emerging Grand Challenge #2 – Development of Real-Time Simulation-Based Problem-Solving Capability.

- Emerging Grand Challenge #3 – True Plug-and-Play Interoperability of Simulations and Supporting Software within a Specific Application Domain.

- Big Challenge #4 – Greater Acceptance of Modeling and Simulation within Industry.

This set of challenges should be accomplished by the companies developing simulation software and designers to position the simulation software as an improvement approach in the near future.

Fowler [2003] calls the greater acceptance of Modeling and Simulation within the industry as the simulations grandest challenge. As he mentions, this challenge is more a social one and might be the most difficult.

While the use of modeling and simulation in manufacturing is steadily gaining acceptance for certain applications, there is a still a long way to go before it is commonly applied for a multitude of other applications. Currently, modelers often spend much of their time convincing management of the need for these services. [Fowler 2003]

Simulation software companies are committed, by following those goals, to make the use of simulation the rule and not the exception in the industry. Today, simulations are used for the engineering of logistics, machines and kinematics – and partly for process. Future engineers will need multi-scale simulation, with high-performance computing and the ability to adapt to real or forecast system behavior. New basic models of processes and simulation

techniques should be developed, extended by automated planning and programming and, possibly, by provision for cognition and learning features – as well as the integration into unified models of diverse simulation aspects such as mechanics, control and process physics. [Manufuture 2006]

2.2. Simulation in the Defence Industry

The Defence Industry has been really interested since first developments on the field of simulation, because it foresaw the potential use of simulation. Much of the work in distributed simulation for virtual environments began in the 1980s. A key factor driving the development and adoption of distributed simulations for synthetic environments has been the need for the military to develop more effective and economical means of training personnel prior to deployment. Field exercises are extremely costly. [Fujimoto 2003]

Moving troops and vehicles for war exercises is quite expensive and might involve extra risks, like accidents resulting in loss of human lives, which simulations can avoid. These situations would make the simulation of battles a perfect tool for avoiding any risk and unnecessary cost. In addition, simulation software offers the possibility of engage armies in battles and scenarios that might be impossible to be executed in real life, offering commanders multiple forecasts depending on the actual situations. Figure 2 shows an U.S. Army soldier training on combat simulation.

Figure 2 *By training with simulations soldiers are better prepared for the real battlefield. [U.S. Army 2007]*

An example of those trainings it is described by Macedonia (Macedonia 2002): “Weeks before U.S. pilots took to the skies above Afghanistan last October, they had a pretty good idea what they would see there. Already they had logged many hours doing virtual fly-throughs over the rugged mountain terrain, using a mission rehearsal system known as Topscene (Tactical Operational Scene). Built for the U.S. Department of Defense by Anteon Corp., Fairfax, Va., Topscene combines aerial photos, satellite images, and intelligence data to create high-resolution three-dimensional databases of a region.”

Seated at computer consoles running on Silicon Graphics Onyx processors, pilots could visualize flying from ground level up to 12 000 meters, at speeds up to 2250 km/h. The detailed renderings, showing roads, buildings, and even vehicles, helped them plot the best approach, scout for landmarks, and identify designated targets. [Macedonia 2002]

Early work in distributed simulation for virtual environments for the military began with the SIMNET (SIMulator NETWorking) project that extended from 1983 to 1989. The success of the SIMNET experiment has had far-reaching effects throughout the defense modeling and simulation community in the United States. SIMNET was replaced by what came to be known as DIS (Distributed Interactive Simulation). A second major development springing from SIMNET was the ASLP (Aggregate Level Simulation Protocol) work that applied the SIMNET concept of interoperability to war game simulation. [Fujimoto 2003]

The next major milestone in the evolution of this technology was the development of the HLA (High Level Architecture). HLA was mandated in September 1996 as the standard architecture for all modeling and simulation activities in the Department of Defense in the United States [Fujimoto 2003].

HLA was welcomed by the open community in 2000 under the standard 1516. After this several commercial and free RTIs have been developed. Many countries have followed the development of HLA and supported different projects. This is the case of the Department of the Defence of Australia that in May 2007 started supporting the Portico Project. [Portico 2007]

Detailed information about DIS and HLA will be introduced to the reader in the following chapter.

2.3. Simulation in Game Industry

A second main thread of activities in DVEs (Distributed Virtual Environments) for nonmilitary applications grew from the interactive gaming and Internet communities [Fujimoto 2003]. Modern games can submerge players into a high detailed 3D graphics world, for example simulating a real-time cockpit.

***Figure** 3 Microsoft's Flight Simulator X immerses players into realistic airplanes' cockpits [Microsoft 2010].*

An example of this is shown in Figure 3 where the player gets immerse in the pilot seat of a commercial airliner. In flight simulators games users can choose between different airplanes to use, take off or land from different airports, different visualizations of the surroundings of the airplane, offering the user a realistic experience of being the pilot of their favorite airplane.

First-person games are similar to DVE where the player sees the simulated world as if she were immersed in the game. Many of those first-person games are shooting games, where the player faces a direct combat experience against bots controlled by the game or other human players. Examples of those games are Quake 3, Counter Strike and Unreal Tournament. These games place sessions of tens of players per server to increase the responsiveness of game. The main purpose of those games is that players play against each other or align as teammates and fight against opponent teams.

Even though it is difficult to compare these simulations to other approaches, first-person approach can be used to teach people how to react to different situations. An example of this could be a first-person simulator used to teach people to drive a car.

Figure 4 shows a snapshot of the game Counter Strike, where several factors can be observed: Number of weapons, health indicator, radar, bullets available, etc. These factors, in addition to the reality scenery and sounds, which immerse the players into the virtual world, have increased the popularity of these games amongst young players.

Figure 4 *First-person shooter games have gathered a high number of players*

Another kind of game simulations is the third-person games. In this kind of games the player looks the environment from an outside point of view. These games can help to train players for commanding decisions in economics, construction or battle simulations. As example of these games: The Sims and Age of Empires.

Games use, mostly, a client-server architecture approach. This kind of architecture will be explained in the next chapter.

2.4. Simulation in Manufacturing

Nowadays manufacturing enterprises are intensively using simulation in the different areas of application. Areas like factory design, PLC (Programmable Logic Controller) validation and simulation of robotic work cells where simulation has been used thoroughly. In contrast, the area of human simulation is one of the fields where simulation has just started to be used in recent years. Some of those areas are presented in detail next:

2.4.1. Factory Design and Layout

Using simulation as a sales tool is nothing new. A few forward thinking companies with R&D budgets embraced the promise 3D simulation offered as a sales tool in the early 90's. 3D simulation enables theses companies to demonstrate to the customer their current facility, how could be improved with the next round of equipment purchase and, most important, the lasting vision of what their manufacturing facility could be capable of in the future. Unfortunately the cost and skills required to implement simulation as a sales tool was a prohibitive for an SME (Small and Medium Enterprises) to consider. [Visual Components 2007]

Recent advances in simulation technologies have drastically reduced, and in many cases eliminated the challenges faced by the early adopters. Thus, allowing 3D simulation to be deployed on a large scale by equipment manufacturers. These advancements include [Visual Components 2007]:

1. Applying component software techniques to simulation data enabling the equipment to be easily connected and to drastically simplify the task of line layouts.
2. Encapsulation of the complexity within a component based equipment model to facilitate the reuse and streamline maintainability.
3. User focused simulation products designed for different skills level throughout the organization.
4. The pricing model, whereby each simulation license does not cost tens of thousands of euros.
5. Advancements in computer and software performance enabling interactive performance on laptops PC's.

Virtual reality factory models enable to move through factory mock-ups, walk through, inspect, and animate motion in a rendered 3D-factory model. This design and communication technology also provides design collaboration activities in order to view, measure, analysis, and inspect for clearance in a 3D-virtual factory model. [Kühn 2006]

These tools help the designers and engineers in charge of the development of the factory floor, since they provide a clear idea of how the machines could be arranged to improve the use of space in the factory floor.

2.4.2. Optimizing the Factory Flow

In a company, an efficient flow of materials in the factory floor is crucial and directly reflected in production. Simulation can help to compare different layouts of the factory floor and choose the best layout for the task.

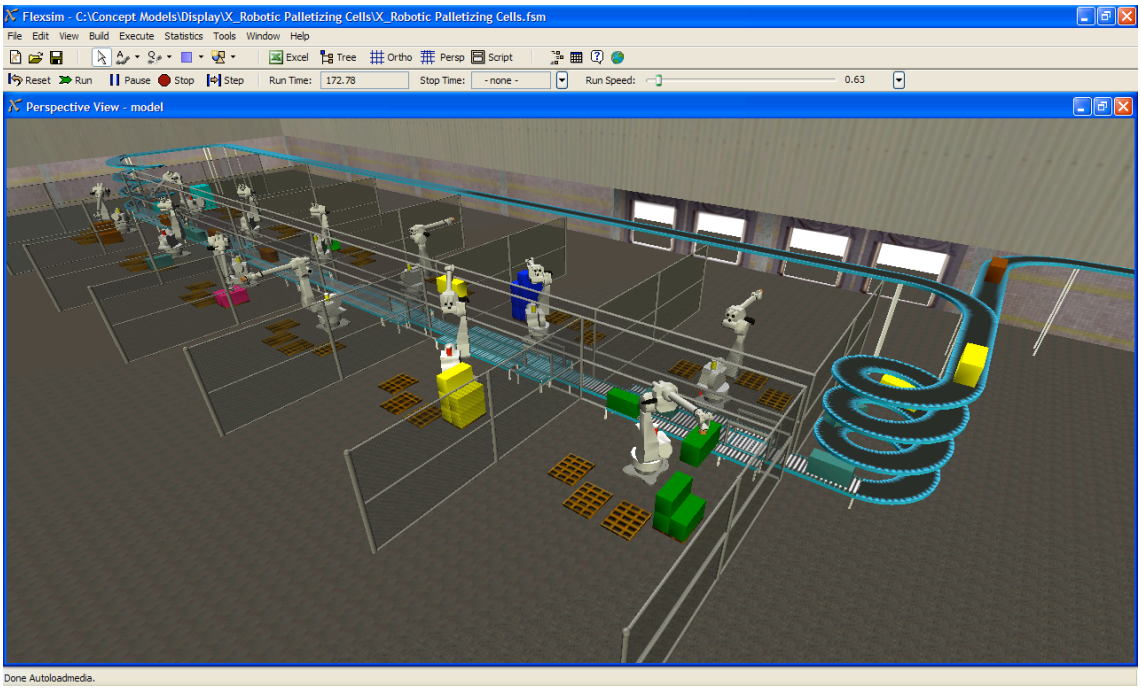

Figure 5 *Flow of products can be analyzed and optimized by using simulation.*

Enhancing the factory layout based on material flow distances, frequency and cost is a first step towards more efficient factory layouts, which directly result in reduced material handling and improved production outputs. [Kühn 2006] Figure 5 shows an example of how the flow of a product can be reproduced and analyzed in order to improve the production.

In addition, simulation tools also can exemplify the paths that humans and machines could use. This is important due the fact that, if human models move less, as same with machines and robots, then people become more efficient and less time is lost moving parts from one side to the other.

2.4.3. Simulation of Robotic Workcells

Robotic workcells are important elements in automated manufacturing systems for delivering required manufacturing materials and operations with industrial robots and associated peripheral devices. Rapid design and deployment of a

robotic workcell require the successful applications of concepts, tools, and methods for fast product design, manufacturing process planning, and plant floor/cell control support. An important technology for achieving this goal is robotic workcell simulation. [Cheng 2000]

Simulation of robotic workcells focuses on the design, simulation, optimization, analysis and offline programming of robotic workcells and automated manufacturing processes in the context of product and production resource information.

Motion simulation and synchronization of several robots and mechanism including 3D path definition is required to perform reachability checks, collision detections and optimization of cycle time. [Kühn 2006]

Models, which are going to be used for offline programming, have to implement physical and control characteristics of robots and other automated devices. Robot offline programming requires accurate simulations of robot motion sequences in order to download machine programs to the real controller on the shop floor. [Eberst et al 2004.]

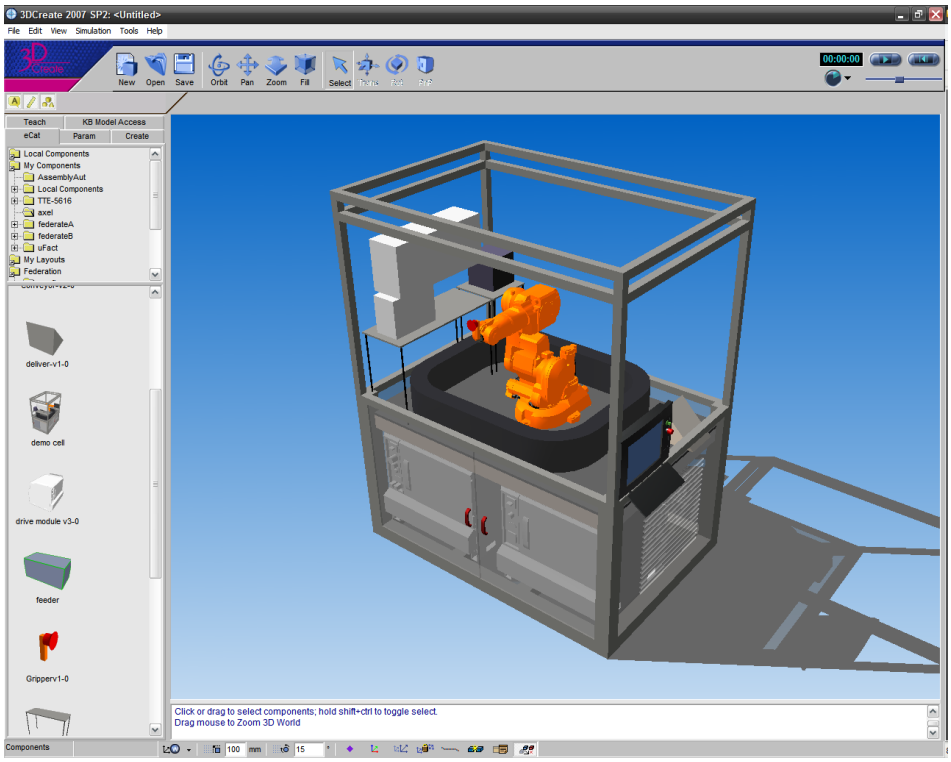

Figure 6 *Simulation helps not only to design a robotic cell, but also to improve its functionality.*

Figure 6 shows an example of a robotic cell that was designed for a fair. From this model is possible to get features like the size of the cage and height, to

improve the transportation and the space required to transport it and place it on the stand. The robot placed in the cell was programmed with different tasks accordingly to the requirements of the client, exemplifying the whole pick and place process. Some aspects as the cycle time were observed and improved during the development of the simulation. This simulation was the foundation of a project, which culminated as a real life robotic cell.

2.4.4. Model Based PLC Offline Programming

With time and cost pressure on introducing new products and production changes, the PLC programming shall not be handled as an isolated, independent function on the shop floor level. The PLC program generation integration in a 3D-integrated virtual environment allows working in parallel and sharing information from both mechanical design and control departments. This enables an automatic generation of PLC programming directly from the virtual manufacturing model and allows for the virtual commissioning prior to building the equipment on the shop floor. [Kühn 2006]

These days, offline programming of PLCs and robots is possible since many simulation suites offer the possibility to export the code directly to their hardware counterparts. In that case, only small code changes are needed to setup the PLC or robot correctly and place it to work. All these changes minimize the downtime in a work cell and also in a production line, which makes offline programming a really desired feature. This would speed up the setup of the process, reducing the waiting times that come when installing or making changes to a production line.

2.4.5. Human Resource Simulation

An accurate modeling, simulation and analysis of manual assembly designs, manual work places and human operations with detailed 3D virtual human models can optimize execution times and prevent work-related health problems. Human resources simulation focuses on [Kühn 2006]:

- Detailed design of manual operations
- Checking the feasibility of tasks
- Ergonomic analysis

- Time analysis
- Generating work instructions

Figure 7 shows a human model handling a piece inside of an airplane structure. This model shows how spaces in the layout can be used by a worker, how long it will take for a human to handle the parts and to visualize any security measure needed to perform the desired task.

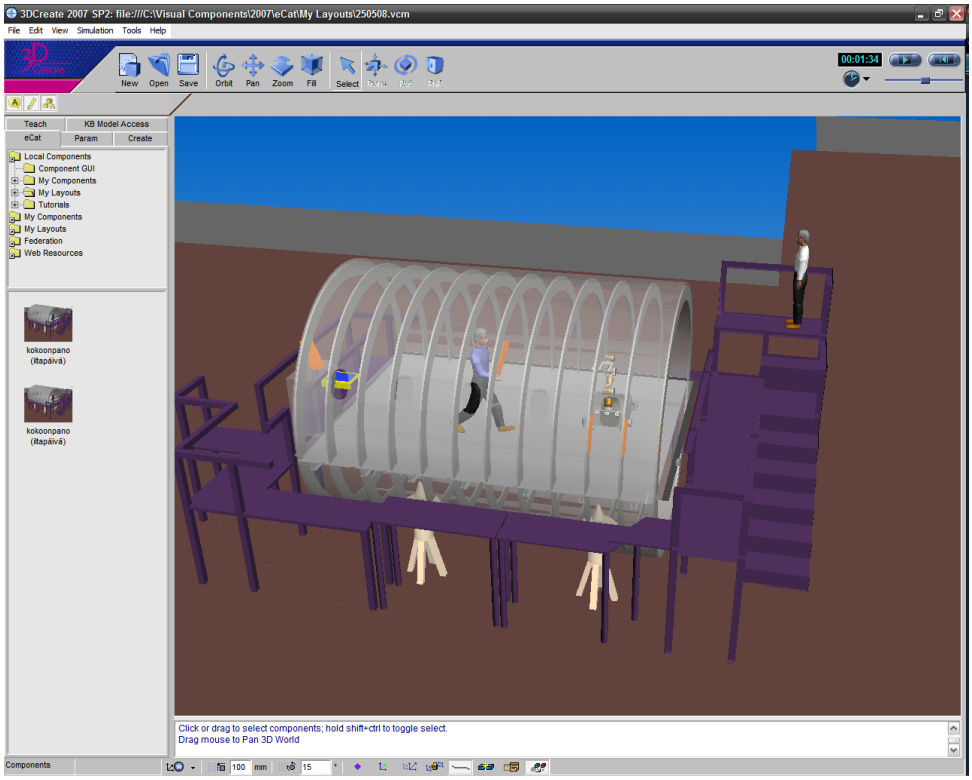

Figure 7 *Ergonomic analysis is possible by the inclusion of human models to the simulation environment.*

Although simulation software allows simulating human behaviors at present time, this type of simulations are still in early faces of development and should be developed further to increment the ease of use of these simulation models. These advances will allow non-specialized users to create prototypes of simulated environments including human models in no time.

2.4.6. Summary

The applications mentioned above are some of the most common uses of simulation in manufacturing. Advances in technology had made possible that, nowadays, complex simulations can be simulated in laptop machines, which it was impossible almost a decade ago. In the near future, advances in technology surely will make possible the application of simulation technologies in additional tasks that at this moment are thought of as impossible.

In the next section, the problem that this thesis work focuses on solve, is presented.

2.5. Problem Presentation

One of the main problems that companies faces with modeling and simulation of their activities is that resources are geographically dispersed. Interconnectivity between manufacturing simulations has used limited approaches if any at all.

The game industry, defense and research community have used different architectures to solve this problem and allow to link simulations. In the Manufacturing Industry, simulation is a new field where most of the models and resources are spread not only in departments but also across continents.

The present section introduces the domain in which this work is developed as the scope of this thesis. In the last section, the problem to be solved is presented.

2.5.1. Domain and Scope

The domain of this thesis is the Simulation of Manufacturing Systems, in particular the area of production.

The work done in this thesis is focused on the use of the High Level Architecture to implement a distributed simulation network along resources spread across the world.

2.5.2. Problem Definition

As was shown before, companies have been using Modeling and Simulation to increase their advantage in the markets today. Departments inside of companies usually develop their own models and simulations that are rarely shared. As result, plenty of models and simulations are isolated in different locations without any interaction.

Another common problem is that models required for simulation are not in the same physical location. This makes more difficult to reuse models and simulations. In consequence, frequently new simulation models are design from scratch, resulting in additional time to develop new models. The additional time taken in developing the whole simulation often impacts on the budget of the project.

The problem solved by the work of the present thesis is the interconnectivity issues between simulation software. In order to allow those applications to interact, an architecture is proposed that should allow the following:

- Applications connected to the architecture should be able to publish or request data.
- Multiple applications should be able to interconnect independently of their geographical location.
- The architecture proposed should be flexible enough to allow new applications to integrate to it.

Figure 8 shows a geographically dispersed scenario targeted by the pilot implementation developed during this thesis.

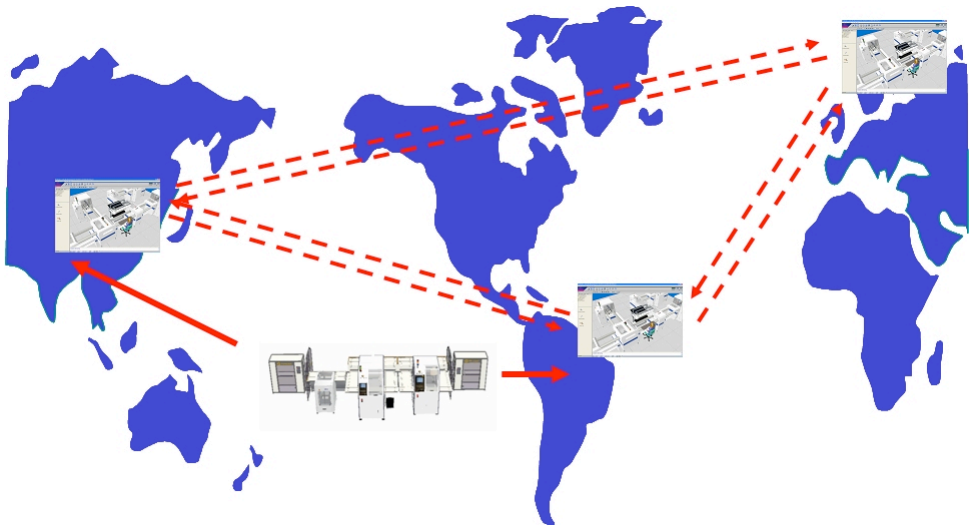

Figure 8 *Geographically dispersed simulations.*

These and other issues will be approached by the work in this thesis. The next chapters will present different COTS (Commercial of the Shelf) simulation software, an analysis of their main characteristics and the architecture proposal for distributing simulations using the High Level Architecture. These areas would give the reader an overall view of the functionality of the simulation software and an in-depth view of the MS2Value architecture and its usage.

3. THEORETICAL BACKGROUND

The present chapter will introduce essential information about Distributed Simulation and High Level Architecture. Initially a definition of simulation and its elements is needed, as they are the foundation of Distributed Simulation. Subsequently some distributed architectures, in which HLA was based, are defined. At the end of this chapter the HLA and its elements are defined. This chapter contains the core knowledge in which the entire project is based on.

3.1. Simulation

A simulation is a system that represents or emulates the behavior of another system over time. In a *computer simulation* the system doing the emulating is a computer program. The system being emulated is called the *physical system*. The physical system may be an actual, realized system, or it may only be a hypothetical one, for example, one of several possible design alternatives that only exist in the mind of its inventor. [Fujimoto, 2003]

Simulations can be classified in two groups:

- Continuous. The state of the simulation can change in any time.
- Discrete. Changes are only reflected at discrete times. Discrete simulations are also divided in two.
- Time stepped. Every certain period of time the simulation is updated.
- Event driven. When an event is triggered the simulation is updated.

In the work handled in this thesis only event driven simulation approach was used.

3.1.1. Elements of a simulation

Model

A model is a representation of a system or a process. A simulation model is a representation that incorporates time and the changes that occur over time. A

discrete model is one whose state changes only at discrete points in time, not continuously [Carson 2005]. In computer, a model of a system or process is represented in a programming language where inputs and outputs are represented in variables that adjust their value to the changes of the simulation. Although models are representations of a system, the level of detail of these representations may differ widely from one model to other, since two models of a system can focus in different aspects of the real system.

State

A model state is a list of values that are sufficient to define the complete state of the system at any point in time. In practice, a model's state is defined implicitly by the internal status of all the entities used in the simulation software package [Carson 2005]. A state is usually represented as a vector of values, in which each value represents certain state in the simulation. In case that several faults

Event

An event is an instantaneous occurrence that changes the model's state [Carson, 2005]. For example, when a conveyor transports a box, and the edge of the box reaches a sensor, this sensor then triggers a signal to the controller meaning that a box is present. The change on the signal value is an event. Based on the events received and the model state, the controller will make a decision on how to proceed.

Activity

An activity is a duration of time that is initiated by an event in conjunction with the model being in a certain state. [Carson 2005]

Entities
An entity is an object in the model that represents some real-world object that moves through a system [Carson 2005]. Commonly denominated virtual entity to enhance its non-physical nature.

Resource
A resource is an entity that provides a service to entities [Carson 2005]. In a manufacturing line, a resource can be a robotic arm, a lifter, etc.

These are the definitions of the basic elements of the simulation and should be taken into account in the future chapters to avoid misunderstandings and ease the lecture.

3.2. Distributed Simulation

Distributed simulation refers to distributing the execution of a single “run” of a simulation program across multiple processors. [Fujimoto 2003]

Parallel and distributed technologies for analytic simulation applications originated largely from basic research in the late 1970 and throughout the 1980s. This research has flourished in the 1990s. Work in this field began with the development of synchronization algorithms to ensure that the simulation is distributed across multiple computers, the same result are produced as when the simulation is executed on a single machine.

There are several benefits from executing a simulation across multiple computers [Fujimoto 2003]:

- First motivation for distributing the executions is to reduce the length of time to execute the simulation. In principal, by distributing the execution of a computation across N processors, one can complete the computation up to N times faster that if it were executed on a single processor. When confined to a single computer system there may not be enough memory to perform the simulations. Distributing the execution across multiple machines allows the memory of many computers system to be utilized.

- The second motivation concerns the desire to integrate several different simulators into a single simulation environment.

- The third motivation is the geographical extent over which the simulation executes. Often distributed simulations are executed over broad geographic areas. This is particularly useful when personnel and/or resources (e.g., databases or specialized facilities) are included in the distributed simulation exercise. Distributed execution eliminates the need for these personnel and resources to be physically collocated, representing an enormous cost savings.

Distributed simulation has been present mostly using two common architectures in networking [Fujimoto 2003]:

- Client-Server

- Peer-to-Peer.

In the Client-Server architecture, clients request services and servers provide those services. A variety of servers exist in today's Internet -- Web servers, mail servers, FTP servers, and so on. The Client-Server architecture is an example of a centralized architecture, where the whole network depends on central points, namely servers, to provide services. [Krishnan 2001]

The Client-Server architecture, as mentioned before, depends totally on the server availability. In case of server failure, the entire simulation would be impossible to run. Additionally, this architecture also presents the problem that the server becomes a bottleneck when the number of clients starts growing up.

In the simulation field, a client-server approach would require that most of the operations are run in the server. The clients would log into the server and interact with the simulation. Figure 9 shows the client-server architecture. This approach offers more security, since only clients that are allowed to log in the server can participate in the simulation.

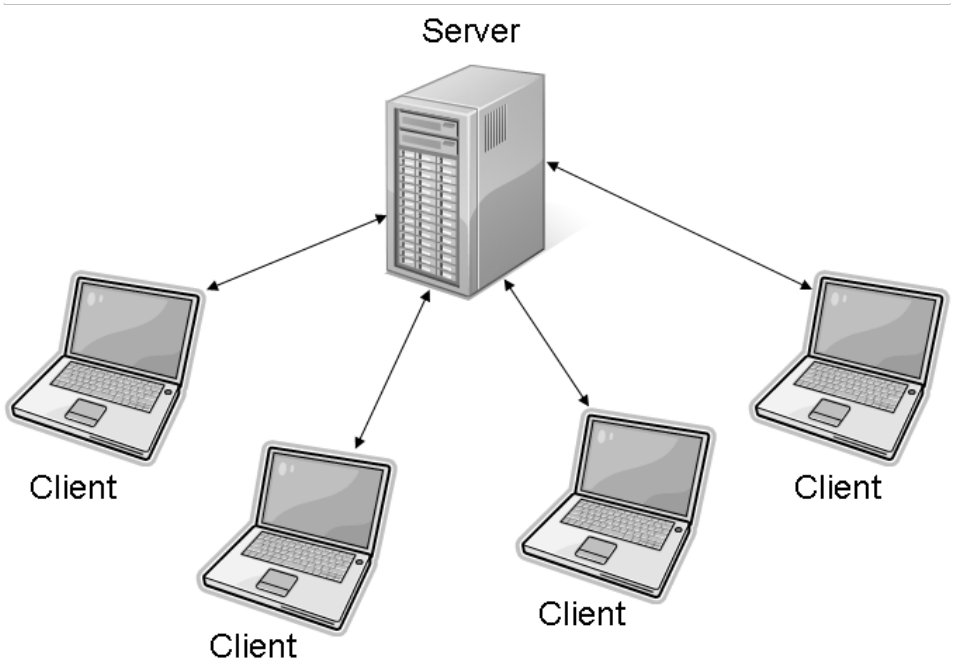

Figure 9 *The traditional approach of Client-Server allows more control of clients.*

The P2P (Peer-to-Peer) architecture overlay networks are distributed systems in nature, without any hierarchical organization or centralized control. Peers form self-organizing networks that are overlayed on the IP (Internet Protocol) networks, offering a mix of various features such as robust wide-area routing architecture, efficient search of data items, selection of nearby peers, redundant storage, permanence, hierarchical naming, trust and authentication, anonymity,

massive scalability, and fault tolerance. It allows access to its resources by other systems and supports resource sharing, which requires fault-tolerance, self-organization, and massive scalability properties. [Lua 2004]

In a simulation that uses the P2P approach a small part of the simulation is run in every computer connected to the network. Each computer shares data with other computers as needed to perform the simulation task until the simulation is finished. As a drawback of the P2P networks, a small part of the code containing the management of the connections and the simulation has to be programmed in each of the clients. This complicates the process of administrating the whole mesh of computers and the simulation itself.

Figure 10 shows the possible connections in a P2P approach. Peer to peer approach is preferred in several systems since it can scale up easily without bottlenecks.

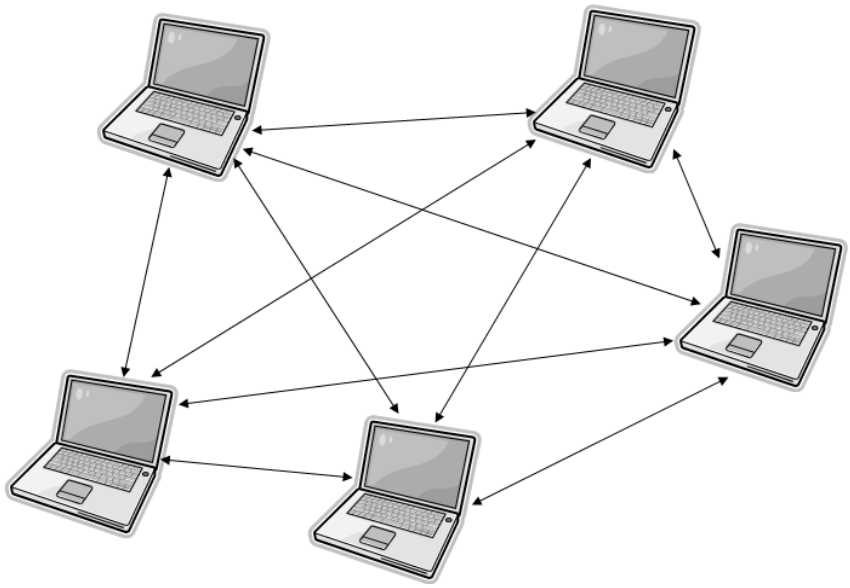

Figure 10 *In a Peer-to-Peer architecture each computer is connected to several computers without a centralized server.*

LANs (Local Area Networks) carry messages at relatively high speeds between computers connected to a single communication medium, such as twisted copper wire, coaxial cable or optical cable. WANs (Wide Area Networks) carry messages at lower speeds between nodes that are often in different organizations and may be separated by large distances. [Colouris 2004]

Mostly these definitions had been done in direct relationship to the geographical area that the networks cover, being the WANs the ones that serve as media to interconnect several LANs.

The management of the distributed system is greatly simplified by the centralized management of the simulation computation [Fujimoto 2003]. All the synchronization points are handled by the server, which simplifies the control of the overall simulation. In a P2P approach, some of the synchronization management has to be implemented in each of one of the peers. This increases the complexity of programming simulation software using the later approach.

3.3. Distributed Interactive Simulation

DIS (Distributed Interactive Simulation) is an infrastructure that enables heterogeneous simulators to interoperate in a time-and-space-coherent environment. In DIS, the virtual world is modeled as a set of entities that interact with each other by means of events that they trigger. Simulator nodes independently simulate the activities of one or more entities in the simulation and report their attributes and actions of interest to other simulator nodes'. The simulator nodes are linked by a communication network and communicate entity. [Cheung 1994]

The initial focus of DIS has been on linking human-in-the-loop simulations, such as simulators used for training operators of tanks, aircraft and ships, in exercises for training forces that may include elements form more than one military service (Army, Navy, air Force, and Marina Corps) and multinational forces.

DIS was defined by the IEEE (Institute of Electrical and Electronics Engineers) as the series of standards 1278 (IEEE Std. 1278, 1993). The Standard for Distributed Interactive simulation - Communication Services and Profiles (IEEE Std. 1278.2, 1995) specifies the requirements for the underlying network (see Figure 11) in a DIS. The most notable requirement is real-time delivery (100 to 300 milliseconds) of the protocol messages to the simulation nodes on the network.

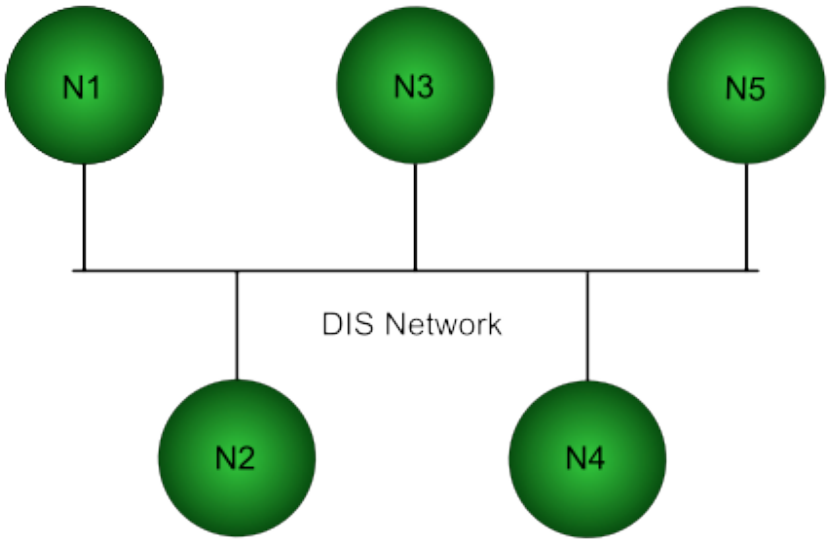

Figure 11 *DIS Network topology.*

In DIS, there is no central computer. Instead, a number of computers are interconnected via a network (such as the one shown in Figure 11). [Cheung 1994]

DIS has been used extensively in building DVEs for training in the defense community. A principal objective of DIS (and subsequently the High Level Architecture effort) is to enable interoperability among separately developed simulators. [Fujimoto 2003]

3.3.1. Overview of DIS

DIS utilizes the following design principles [DIS Steering Committee, 1994]:

- *Autonomous simulation nodes.* Each node is only responsible for the entity or entities it is simulating, and does not have to calculate what other nodes are interested in. Receiving simulations are responsible for determining the effects of an event on the entities it is simulating. The autonomy principle enables nodes to join or leave an exercise in progress without disrupting the simulation.

- *Transmission of "ground truth information"*. Each node transmits the absolute truth about the state of the entity/entities it simulates. The receiving nodes are solely responsible for determining whether their objects can perceive an event and whether they are affected by it.

Degradation of information (essential for realistic portrayal of system behavior) is performed by the receiving nodes.

- *Transmission of state change information only.* Simulations will only transmit changes in the behavior of the entities they represent, in order to reduce unnecessary information exchange.

- *Dead-Reckoning Mechanisms.* The objective of dead-reckoning (a term borrowed from navigation) is to determine new states based on previous ones, i.e. by extrapolation. Only when the ground truth data differs enough from the extrapolated data (by a predetermined threshold) is a new state issued.

- *Simulation Time Constraints.* Current DIS standards primarily support human-in-the-loop simulations. The simulation time constraints (100 – 300 milliseconds) were obtained based on human factors. Other types of simulations (such as wargames) operate faster or slower than real time. In order for these types of simulations to interact with real time simulations, interfaces to the constructive simulation need to be capable of issuing data at real time rates.

The existing DIS protocols work extremely well for certain applications particularly between manned 'virtual' simulators. However, the United States DoD has defined as one of its future goals for Advanced Distributed Simulation development, increased interoperability among simulations for which the current DIS protocols are not well suited such as interactions with war games. As a result of that, HLA has been established, incorporating additional functionality and increased flexibility to allow this interoperability to occur. The HLA will support DIS-like exercises just as well as it supports detailed engineering or analysis modeling. The HLA also provides a growth path that the current generation DIS protocols may not have available. [Perry, 1998]

The DIS Product Support Group (PSG) is a permanent support group chartered by the SISO Standards Activities Committee to support DIS products such as the IEEE 1278 series of standards. The DIS PSG publishes, maintains, and updates a series of reference documents related to DIS that are helpful to users and developers. [SISO 2010]

3.4. High Level Architecture

The High Level Architecture is an architecture for reuse and interoperation of simulations. The HLA is based on the premise that no simulation can satisfy all uses and users. An individual simulation or a set of simulations developed for one purpose can be applied to another application under the HLA concept of the federations: a composable set of interacting simulations. [Dahmann 1998]

The roots for the HLA stem from DIS aimed primarily at training simulations and the ALSP (Aggregate Level Simulation Protocol) which applied the concept of simulation interoperability to war gaming simulations. The HLA development began in October 1993 when the DARPA (Defense Advanced Research Projects Agency) awarded three industrial contracts to develop a common architecture that could encompass the DoD modeling and simulation community. On September 10, 1996 the undersecretary of defense designated that the HLA become the standard high-level technical architecture for all modeling and simulation activities in the U.S. Department of Defense. [Fujimoto, 2003]

It came to public use as a standard by the IEEE with the name of IEEE Std. 1516-2000. [IEEE Std. 1516 2000]

Like DIS, the principal goal of the High Level Architecture is to support interoperability and reuse of simulations. Unlike DIS, HLA provides explicit support for simulations other than training. [Fujimoto 2003]

In the next sections, the main actors of HLA are presented.

3.4.1. Federate

Federate is an individual object that shares data with other federate(s). Federates can be of different types: pure software simulators such as computer generated forces, human-in-the-loop simulators such as virtual simulators, or live components such as instrumented weapon systems [Perumalla, 2006]. In addition, a passive logging application can act as a federate to record and report data of interest.

Federate objects can be put together to interact with each other. In these interactions a federate doesn't need to behave the same in each simulation. In other words, the behavior of a federate relies more in the design of the distributed simulation than in its own.

In a manufacturing simulation, a federate can be a model of a whole company, a manufacturing line, a robot cell or as simple as a gripper. This gives flexibility to the designer to define the level of detail desired on the simulation. For example, if the designer wants to focus on a machine then the sensors and motors could be defined as federates and each of those would respond to messages and share information about the machine status.

3.4.2. Federation

The Federation is a common simulation between systems that interact and are part of that simulation. These federates don't need to be from the same type. A common approach would be to have federates which represent a machine in a simulation and are hosted in a computer; whereas other federates can be the real machines sending and receiving data to other simulated instances in the simulation.

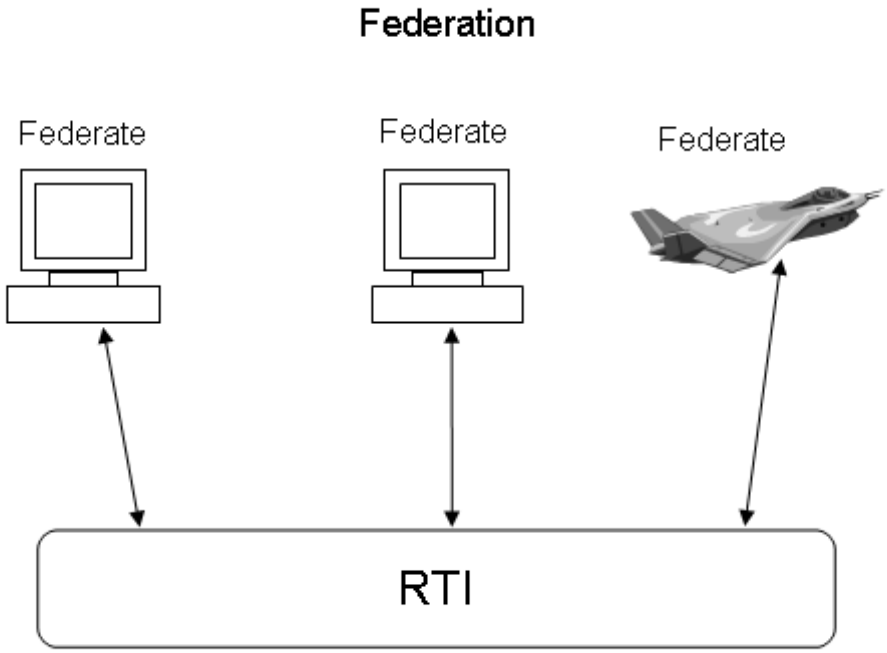

Figure 12 *An example of a Federation integrated by various types of federates.*

Figure 12 shows and example of a Federation where two computers are interacting with a flight simulator. In this simulation, several clients interact with each other without caring if they are the actual machines or simulated environments.

3.4.3. Run-Time Infrastructure (RTI)

The RTI is a collection of software that provides commonly required services to simulation systems. The overall goal has been to keep the RTI “lean and mean.” However, this goal is occasionally moderated when it is clear that an additional service could be used across multiple simulation domains. The RTI is also intended to provide a measure of portability (across computing platforms, operating systems, and communication systems) and simulation interoperability. Of course, interoperability requires commonality between the Federation Object Models (FOM) of the simulations involved. The services of the RTI are described by the HLA Interface Specification [Calvin, 1996]. This interface has rules to coordinate and manage the federation.

The RTI Interface Specification is composed by:

- Federation Management
- Object management
- Time Management
- Declaration Management
- Ownership Management
- Data Distribution Management

3.4.4. Object Model Template (OMT)

The Object Model Template specifies the HLA objects that provide a commonly understood mechanism for the exchange of data, coordination between federates and a description of the capabilities for possible federates.

3.4.5. HLA Rules

The rules for the High Level Architecture can be divided in two: From 1 to 5 to regulate the Federation and from 6 to 10 for the Federates.

Rule 1 *Federations shall have an HLA FOM, documented in accordance with the HLA OMT.*

All data to be exchanged in accordance with the HLA shall be documented in a FOM. A FOM shall document the agreement among federates on data to be exchanged using the HLA services during federation execution and the minimal set of conditions of the data exchange. [IEEE Std. 1516, 2000]

Rule 2 *In a federation, all simulation-associated object instance representation shall be in the federates, not in the RTI.*

In the HLA, the responsibility for maintaining the values of HLA object instance attributes shall take place in the joined federate. In an HLA federation, all joined federate-associated instance attributes shall be owned by federates, not by the RTI. However, the RTI may own instance attributes associated with the federation Management Object Model. The RTI may use data about instance attributes and interactions to support RTI services (e.g., Declaration Management), but these data are merely used by the RTI, not changed. [IEEE Std. 1516, 2000]

Rule 3 During a federation execution, all exchange of FOM data among joined federates shall occur via the RTI.

The HLA federate interface specification (see IEEE Std 1516.1-2000) specifies a set of interfaces to services in the RTI to support coordinated exchange of instance attribute values and interactions in accordance with a federations FOM. Under the HLA, intercommunication of FOM data among joined federates participating in a given federation execution shall be executed by the exchange of data via the RTI services. Based on the FOM, joined federates shall identify to the RTI what information they will provide and require, along with instance attribute and interaction data corresponding to the changing state of object instances in the joined federate. The RTI shall then provide the coordination, synchronization, and data exchange among the joined federates to permit a coherent execution of the federation. [IEEE Std. 1516, 2000]

Rule 4 *During a federation execution, joined federates shall interact with the RTI in accordance with the HLA interface specification.*

The HLA provides a specification for a standard interface between the federate application and the RTI. Joined federates shall use this standard interface to access RTI services (see IEEE Std 1516.1-2000). The specification shall define how federate applications interact with the infrastructure. However, because the interface and the RTI will be used for a wide variety of applications requiring data exchange of diverse characteristics, the interface specification says nothing about the specific federate data to be exchanged over the interface. [IEEE Std. 1516, 2000]

Rule 5 *During a federation execution, an instance attribute shall be owned by, at most, one joined federate at any given time.*

The HLA allows for different joined federates to own different attributes of the same object instance (e.g., a simulation of an aircraft might own the location of the airborne sensor, whereas a sensor system model might own other instance attributes of the sensor). To ensure data coherency across the federation, at most, one joined federate may own any given instance attribute of an object instance at any given time. Joined federates may request that the ownership of instance attributes be acquired or divested, dynamically, during federation execution. Thus, ownership can be transferred, dynamically during execution, from one joined federate to another. [IEEE Std. 1516, 2000]

Rule 6 *Federates shall have an HLA SOM, documented in accordance with the HLA OMT.*

The HLA SOM shall include those object classes, class attributes, and interaction classes of the federate that can be made public in a federation. The HLA does not prescribe which data are included in the SOM; this shall be the responsibility of the federate developer. SOMs shall be documented in the format prescribed in IEEE Std 1516.2-2000. [IEEE Std. 1516, 2000]

Rule 7 *Federates shall be able to update and/or reflect any instance attributes and send and/or receive interactions, as specified in their SOMs.*

The HLA allows for joined federates to make internal object representations and interactions available for external use as part of federation executions. These capabilities for external interaction shall be documented in the SOM for the federate. If documented in the SOM, these federate capabilities shall include the obligation to export updated values of instance attributes that are calculated internally in the federate and the obligation to be able to exercise interactions represented externally (i.e., by other federates in a federation). [IEEE Std. 1516, 2000]

Rule 8 *Federates shall be able to transfer and/or accept ownership of instance attributes dynamically during a federation execution, as specified in their SOMs.*

The HLA allows ownership of instance attributes of an object instance to be transferred dynamically during a federation execution. The instance attributes of a federate that can be either owned or reflected, and whose ownership can be dynamically acquired or divested during execution, shall be documented in the SOM for that federate. [IEEE Std. 1516, 2000]

Rule 9 *Federates shall be able to vary the conditions (e.g., thresholds) under which they provide updates of instance attributes, as specified in their SOMs.*

The HLA permits federates to own (i.e., provides the privilege to produce updated values for) instance attributes of object instances represented in the federate and to then make those values available to other federates through the RTI. Different federations may specify different conditions under which instance attributes will be updated [e.g., at some specified rate, or when the amount of change in value exceeds a specified threshold (such as altitude changes of more than 1000 ft, etc.). The conditions applicable to the update of specific instance attributes owned by a federate shall be documented in the SOM for that federate. [IEEE Std. 1516, 2000]

Rule 10 *Federates shall be able to manage local time in a way that will allow them to coordinate data exchange with other members of a federation.*

Federation designers will identify their time management approach as part of their implementation design. Federates shall adhere to the time management approach of the federation. [IEEE Std. 1516, 2000]

The rules mentioned above are the corner stone of the HLA, any simulation should follow them to be HLA compliant.

3.4.6. HLA Summary

Federates are objects represented in HLA, those objects can be a model of several machines, a single feeder or a simulation of a process. An object has is unique identity, some attributes like parts per minute, actual state of the process called and its association to other objects.

The HLA includes two components: a run time component and a non-runtime component.

The runtime component, RTI, offers services that allow federates to interact between them and manages those interactions. As a rule on HLA, the values of the attributes of the objects are stored in federates, not in the RTI. In order to fulfill a general usability, the RTI doesn't have knowledge of the information transmitted; it can be seen as a media utilized to transport information from one point to other.

The non-runtime component, OMT, specifies the object model used by the federation, meaning the objects, attributes and associations possible.

3.5. Strategy to distribute a simulation using HLA

In order to distribute a simulation across different machines a set of technologies have to work simultaneously. It is important to a deep research of the tools available to avoid issues of compatibility in the long run. To assure an optimal result certain of steps that have to be followed are named below:

- Choose an HLA RTI
- Choose a Simulation Suite
- Create a Middleware, if needed
- Create the Simulation Manager
- Create the simulation models

In general, these steps have deal with the overall architecture, available technologies and software packages, type of communications and expected interactions, etc. In addition, these steps will also present the needs of the project. A detailed guide of steps is explained next.

Step 1 – Choose an HLA RTI

The first step that has to be done is to select the HLA RTI since it will be the foundation of the project. The designer has to be careful at this step since the RTI has a direct impact on the behavior of the system and the services offered to connected systems.

There are several commercial and open source RTI's, which will give several options to the designer. This is an advantage since a lack of options would have imposed restrictions to the designers.

In principle the designer should focus in:

- Expected number of federates in the simulation. The number of federates running on a simulation in a RTI has impact in two different areas. The first impact goes to the efficiency of the RTI. Several RTI's are better working in small environments whereas others are better suited for a large number of federates. The second aspect is the economic: commercial RTI's regularly charge per-federate license, which would definitely bring repercussions to the overall price of the project.

- Amount of interactions. The number of interactions in the simulation is correlated with the number of federates, but is not directly proportional to it. As was mentioned before, some RTI's handle better small federations than large. A similar circumstance happens with the interactions, some

RTI's are developed to handle a huge number of interactions per second, which also is reflects in the price of the software. Plan in advance which interactions will be needed and make a rough estimation about the expected number of interactions.

- Hardware Requirements. RTI's have different hardware requirements; most of them do not require expensive hardware to operate, but also state that best performance is achieved with high-end configured equipment. If the equipment that is intended to host the RTI is already owned, those specifications would be a constraint when deciding which RTI to choose. Otherwise, when a RTI is already is chosen, the host hardware should be acquired accordingly to the needs of the RTI.

- Development Language: RTI's are developed mostly on C++ and Java languages. It is important that the RTI offers a variety of interconnectivity solutions, so simulations can interconnect as federates easily, without requiring middleware, which would increase the development time.

Step 2 – Choose a Simulation Suite

The decision of what simulation suite to use will not be an easy one, since the suite should fulfill the needs of the designers and is also dependant on the HLA RTI chosen before. The designer of the system should focus on the following points to succeed in the implementation of the system:

- Simulation requirements: Every project in a company has different needs; some of them just require a numeric result, whereas others will be in need of a 3D visualization. In the case, a 3D simulation software is needed to visualize and inspect the simulation process of a manufacturing line. In addition, the availability of predefined models, easiness to learn and use of the simulation suite should be taken into account when choosing it.

- Hardware requirements: Depending on the type of simulation used is the type of hardware that will be needed. For example, 3D simulation software commonly needs a 3D Graphics card as a minimal requirement. These days simulation software should work almost in any computer without dedicated hardware. However, if a better performance of the simulation software is expected, the system designer must choose a hardware system above the minimal specs needed.

Step 3 – Create a Middleware if needed

According to Linthicum et al. (2004) Middleware is a term that is used to mean different things in different contexts. In the context of distributed computing, the following definition can be used: “Middleware is any type of software that facilitates communication between two or more software systems.” [see Lindqvist 2007]

After the decision for choosing the simulation suite and the HLA RTI is done, there is a possibility that the systems chosen cannot be connected directly. If that situation arises, a middleware will be needed.

This middleware will have the main tasks of passing the messages between the simulation suite and the RTI. It should be done in some platform that allows both software suites to interact and preferably with the minimal delay possible.

The use of a middleware should be avoided, if possible, since generally it increases the delays on communications and complexity of the project. It is preferable to select a simulation suite and RTI that can communicate directly to avoid any kind of overhead.

Step 4 – Create the Simulation Manager

The simulation manager is a federate that has the sole purpose of controlling the behavior of the Federation. The main tasks of the manager are to start and finish the execution of the Federation.

The manager’s first task is to connect to the RTI and create a new Federation. The creation of a new Federation requires a unique name that the manager must provide. After the federation is created, the manager subscribes to the federation and then waits for new federates to join.

After all federates have joined to the Federation, the manager can start the execution of the simulation. Federates are allowed to interact with each other, until the execution has started.

The manager is also in charge of terminating the execution of the distributed simulation. At this step the manager tells federates to finish their interactions and resign from the Federation. The final step is to destroy the Federation, freeing resources from the RTI.

There is the possibility that the management behavior is integrated as another federate, which would eliminate the need of having a federate for the sole process of administrating the distributed simulation. Additionally, other tasks as

logging or supervising some messages could be part of the tasks of the simulation manager.

Step 5 – Create the simulation models

In this step the designers of the simulation must focus on the main needs of the simulation and decide the representation that they want to give to the desired process. Some simulations may contain very detailed models while others might only need basic features to accomplish their goals. Defining those needs and creating the models for fulfilling them is an important step before moving forward to the next step.

Step 6 – Create Federates models

The creation of federate models is a necessary step to link the simulation models to the distributed environment. These federates are also simulation models which have some programming interface that would allow them to interact with other federates through the RTI.

The interactions between simulators are done by sending information coming from the local simulation to the federation, for example, sending updates every time a product is created. The simulation models also retrieve information for the local simulation models from the RTI. An example of this situation is when a product is created a simulator can send this information to another simulator which triggers the process for creating the box for packing that product.

It is desirable that federates models are as generic as possible so that they can be reused without needing a lot of changes.

4. DISTRIBUTED SIMULATION INFRASTRUCTURE DESIGN

In order to distribute a simulation, two fields were observed. First step was to analyze which runtime infrastructure was the one that could fit better to the needs of a distributed manufacturing simulation. The second and final step was to analyze the simulation software to discover which simulation software was the best for the experiment.

4.1. Analysis of High Level Architecture Infrastructures

4.1.1. Commercial RTI's

MÄK High Performance RTI

MÄK Technologies is a software company that developed their own HLA RTI. The MÄK RTI implements the full HLA Interface Specification, and has been verified by DMSO as fully compliant with both HLA 1.3 and IEEE 1516. As advantages, MÄK Technologies mentions [MÄK Technologies 2009]:

- Verified by DMSO as Fully HLA compliant (HLA 1.3 AND IEEE 1516)
- Fast and efficient
- Lightweight Mode — No rtiexec required
- Sender-Side Filtering for Efficient WAN Operation
- Network and Shared Memory Communication
- Fault tolerant
- Web-based RTIspy Diagnostic GUI (Graphic User Interface)
- Plug-in API (Application Programming Interface) for user customization

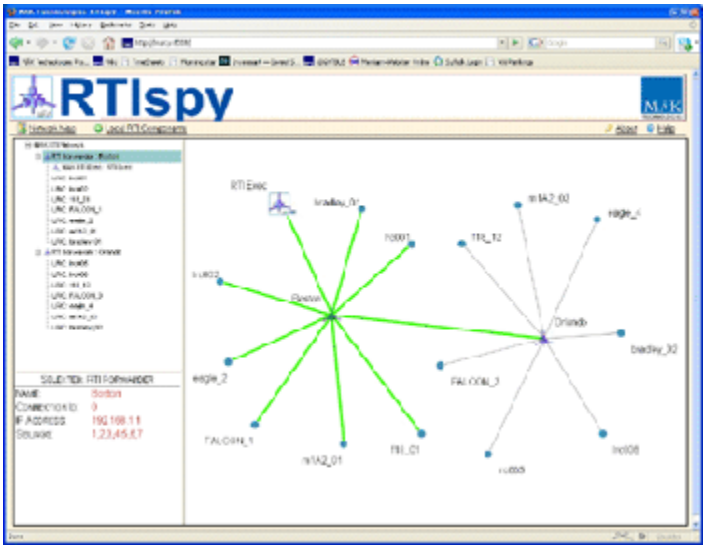

Figure 13 *MÄK RTI shows the federation data by using of plug-in applications [MÄK Technologies 2009]*

According to Burks et al (2001), the MÄK RTI has exceptionally low latencies, but is limited both in RTI functionality, and attribute size. While larger numbers of Data attributes could be moved by updating each attribute separately, no single attribute can exceed the maximum size of a UDP packet using best effort transport. As an advantage MÄK RTI permits the use of extension plug-ins, like the RTIspy, shown in Figure 13 to extend the usability of the RTI. RTIspy permits the user to have access to the information of the federation. MÄK Technologies provides a fully functional RTI, free of charge, for up to two federates.

PITCH pRTI

In February 2000, pRTI™ 1.3 became the first commercial RTI to be certified by DMSO. As the HLA standard was transferred to IEEE it was seen the potential for a broader market. It was decided to develop pRTI™ for this standard and to get it certified. The development of pRTI™ 1516 has been driven by many different needs, several of them seemingly in conflict with each other - flexibility versus ease-of-use, performance versus complexity, etc. It was decided at the outset that should be certified. This meant that pRTI™ 1516 would be a complete RTI, and that no functionality would be sacrificed. This forced PITCH to look for solutions that achieved good performance while still providing full functionality [Karlsson et al. 2001].

The performance of pRTI™ 1516 was one of the most important parameters. The objective when creating pRTI™ 1516 was to assure that the performance

provided was good enough for the targeted market segment. After some research the following numbers were obtained [Karlsson et at. 2001]:

- 100 federates
- 100K object instances
- < 3ms latency
- 50 Hz time management
- Object turnover 2000/minute = 35/second

The installation of pRTI™ 1516 consists of simple steps. The number of options, settings and flags available to users are reduced to avoid overwhelming users with excessive menus. Mostly, people don't take the time to learn about configuration options, they run things as-is out of the box. As a result of that, the default settings have to be the most conservative.

Another aspect of ease-of-use is robustness. If the RTI is fragile and needs to be restarted whenever a developer makes a mistake, when a federate crashes, or when a network connection is lost then that puts an unnecessary burden on the user. pRTI™ 1516 is able to handle the following situations without ill effects [Karlsson et at. 2001]:

- Graceful termination of federate without resign.
- Crashing of federate without resign.
- Crashing of federate after resign.
- Invalid parameters.

The graphical user interface allows users to inspect federations and federates in detail. Additionally, Pitch pRTI™ also supports the HLA Web Services API. As mentioned before now it is possible to use Web Services to connect to the RTI with full HLA functionality. From the previous facts it is possible to summarize the main advantages of pRTI as:

- Interface accessible locally or over the network using a web browser.
- High number of updates per second allowed.
- Web services integrated, known also as HLA Evolved.
- Graphical interface allows inspecting the federation.

4.1.2. Open source RTI's

Portico Project (formerly known as JaRTI Project)

Portico is a fully supported, open source, cross-platform HLA RTI implementation. Designed with modularity and flexibility in mind, Portico is intended to provide a production grade RTI implementation and an environment that can support continued research and development. Portico is licensed under the terms of the Common Developer and Distribution License (CDDL) and is actively developed and maintained by its team of core contributors. [Portico 2008]

One of the primary purposes of the JaRTI project was to develop an RTI implementation that could function both as an RTI for general use, as well as a flexible environment in which extensions could be quickly developed, and easily deployed (be they for research or other purposes). As such, a significant amount of effort has been put into the creation of an underlying architecture to support this objective [Pokorny 2007].

The JaRTI project provides various Java interfaces implementing the core HLA standards: 1.3 and IEEE 1516-2000. Additionally, a compatibility library is provided to mimic the interface often used with the older DMSO RTI-NG (in the `hla.rti13.java1` package). Although a number of interfaces are provided, not all of the HLA services are implemented. The set of HLA features implemented at the beginning of 2007 included the following features:

- Basic Services (create/destroy)
- Synchronization Services
- Publish and Subscribe Support
- Object Creation, update and Removal
- Interaction Sending
- Time Management (not including optimistic)

Figure 14 shows the console of Portico's main window after the RTI has been launched. On this simple but functional interface, allows clearing the console or quitting the application, by simple clicking the appropriated button. In the console all the activities, form interactions to data exchange, are logged to give the user a deep inside view of the operations done through the RTI.

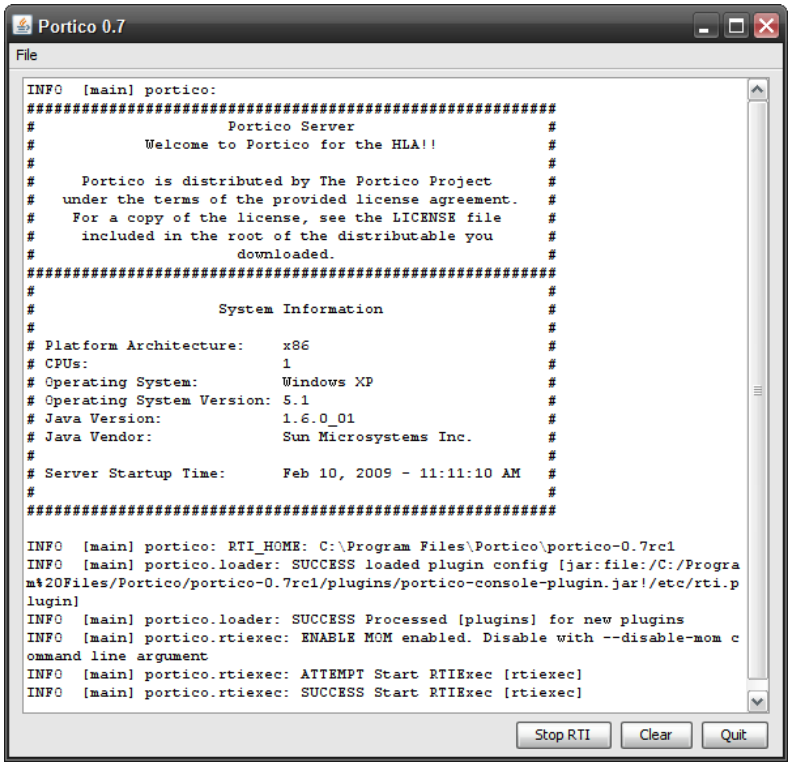

Figure 14 *Console of Portico Project RTI.*

The key features missing from this list are support for Data Distribution Management (DDM), Ownership Management and Save/Restore support. However, as with all open source projects, JaRTI is a work-in-progress.

Open HLA

Open HLA (oh-la) provides an open-source implementation of the HLA RTI spec 1.3 and IEEE 1516. It also provides a framework to wrap the standard RTI classes and FOM to code generation to make life simpler [Open HLA, 2009]. The project development was stopped in the year 2007, there are not updated versions or any information about future development to this day.

4.1.3. Analysis results

Accordingly to the characteristics mentioned before the commercial RTIs had a superior support, development and options. As a downside, commercial RTIs offer packages for 5, 10, 15 or more federates which are directly reflected in their licensing prices. It also incurs in maintenance costs for future developments and plug-ins that have to be bought independently of the RTI. For analyzing the RTIs presented above, the following properties were taken into account:

- Performance
- Platform of development
- Add-ons
- Price
- Compliance with standards
- Support

Table 1 shows the evaluation of these properties based on the information presented in the previous sections and the experiences of the author with those RTIs.

Table 1 *Comparison of different RTI's properties.*

Runtime Infrastructure	Performance	Plataform of Development	Add-ons	Price	Compliant HLA 1.3 / 1516-200	Support
MÄK High Performance RTI	√√√*	C++	RTISPy API	High / Price per federate	Yes	Yes
PITCH pRTI	√√√*	Java	C++**	High / Price per federate	Yes	Yes
Portico Project	√√√	Java	C++	Free	Yes	Yes
Open HLA	Not tested	Java	-	Free	Partially	None

*According to information of the product.
**Incurs on extra expenses.

In this case the main factor to don't use a commercial RTI is the price per federates license terms, since that would have a direct impact on the budget of

the project. While some commercials RTIs offer the possibility to run up to two federates per federation, without having to pay a license fee, it is impractical for this experiment to have that few amount of federates.

The open source RTIs represented a good option for the development of the project, since the software was provided without a charge. Two different options were analyzed: Open HLA and JaRTI. JaRTI was chosen as the best option since Open HLA had little support and documentation. JaRTI is constantly developed and supported by its core team of developers. The support is done via forums or IRC, which is free of charge and usually takes less than 2 days to have a satisfactory answer.

In addition to the advantages mentioned above, the project has been very well documented. In the website of the project information for users and developers is available. Information for users consists on documents about how-to install the software, example federates, configuration of the software and tools.

JaRTI is actually known with the name of PORTICO project. The change in the name was to give a more professional approach to maintain the long-term growth of the project. It was officially changed in May 2007 from JaRTI to Portico project with the founding from the ADSO (Australian Defence Simulation Office).

After observing the advantages and disadvantages of the commercial and open source RTI's, JaRTI was a good option since it is well documented, supported, developed and it was free of charge. It also offers the possibility to develop Java and C++ federates whereas commercial counter parts would incur in an extra charge.

4.2. Analysis of Simulation Software

Simulation software has been widely developed in the last 20 years that companies have now different options to model a desired process.

These days simulation software can be created using so different options that is not an easy task to decide how to develop a simulation. From these options we can enclose the simulation software in three main areas:

- Programming languages.
- General Purpose Simulation Software.
- Application Oriented Simulation Software.

All of them have their own advantages and drawbacks. The person on charge of choosing the simulation software must focus on the requirements of the simulation to model and the impact of the characteristics of the software suite on those requierments.

4.2.1. Programming languages

In the defense related applications the use of general programming languages on simulations is wide. Many “old” simulations are written in C or C++ language. Programming languages can create high efficient simulation software that would be executed using fewer resources on a computer. Another advantage is that the software can be programmed accordingly to specific needs.

A major drawback of programming simulation software is that the development of an application from scratch is usually time consuming, which would have a direct impact on the budget of the project. In addition, maintaining the software can get difficult since these applications generally are coded in thousands of lines.

4.2.2. General Purpose Simulation Software

A key part in deciding to use general-purpose simulation software is flexibility to model the desired process. General purpose software simulation usually tends to develop simulation in less time since it provides most of the features needed to build a simulation model. Also the simulations are easier to modify and maintain when the model is build by General Purpose simulation software [Law, 2000].

The drawback of use a general purpose simulation software is that the learning curve at the beginning is big if the simulation developers are not familiarized with the application. Also sometimes these applications may be too generic for modeling complex or application oriented processes.

4.2.3. Application Oriented Simulation Packages

An application oriented simulation package is designed to be used for a certain class of application such as manufacturing, health care, or call centers [Law, 2000]. Thus the main advantage of this kind of packages is that they are

focused on a special application, containing tools which would help the user to model the processes accurately.

The research of simulation suites did not explore all the existing applications, but it explored only the most common applications on the market, which are listed next:

- Applied Materials - AutoMod
- Imagine That - Extend
- Lanner Group - Witness
- Rockwell Automation - Arena
- Visual Components - 3DCreate

A description of their features will be described in the following sections.

Applied Materials - AutoMod

AutoMod is a suite of simulation tools that provides an environment for building models for analysis and development, as well as for control system emulation.

The AutoMod simulation system differs from other systems because of its ability to deal with the physical elements of a system in physical (graphical) terms and the logical elements of a system in logical terms. AutoMod also offers advanced features to allow users to simulate complex movement (kinematics and velocity) of equipment such as robots, machine tools, transfer lines, and special machinery. All graphics are represented in 3-D space with viewing control, including: translation, rotation, scale, light-sourced solids, perspective, and continuous motion viewing. [Phillips 1998]

AutoMod consists of two working environments. The build environment is for the physical and logical model definition. After the user has defined the physical and logical components of the model, it is then compiled into an executable model, where the simulation and animation run concurrently. The executable model is fully interactive; it can be stopped at any instant in simulated time to view statistics and model status. It also provides the user a set of material handling system templates like conveyors, path based movement, cranes and kinematics for controlling the movement of robots and turntables. [Phillips 1998]

Imagine That - Extend

The Extend simulation environment provides the tools for all the levels of modelers to efficiently create accurate and credible models. Extend's design facilitates every phase of the simulation project from creating, validating, and verifying the model, to the construction of a user interface which allows others to analyze the system. Extend's blocks can be easily configured and combined to model very complex systems [Krahl 2003]. Extend offers different options depending on the needs of the modelers:

- Extend CP. Designed for modeling continuous processes.
- Extend OR. Adds the discrete event architecture and capabilities to the Extend CP.
- Extend Industry. This option strength on the industrial approach
- Extend Suite. This suite bundles leading analysis and animation software to the Extend Industry product.

Lanner Group - Witness

Witness, provides the means to model a working environment, simulate the implications of different business decisions and understand any process, however complex.

The keys of this software are:

- Simple and powerful building block design.
- Modular and hierarchical structure.
- Extremely interactive.
- Powerful range of logic and control options.
- Elements for discrete manufacture, process industries, BPR (Business Process Reengineering), e-commerce, call centers, health, finance and government.
- Comprehensive statistical input and reports.
- Quality graphical displays.
- Great linkage-databases.

Witness is available in two versions: Manufacturing Performance Edition and Service and Process Performance Edition. In addition to those characteristics optional modules are offered to increase functions like CAD and Visio linkeage, Virtual Reality and a developer's version with COM and ActiveX. WitnessVR incorporates 3D displays to any Witness simulation model. It allows viewing models from any angle, setting up fly pasts and adding additional animation effects. It can be also used separately to any simulation model. [Lanner 2008]

Rockwell Automation - Arena

Arena is a general purpose simulation suite property of Rockwell Automation, which comes in different versions to fulfill the needs of different customers.

It is offered in five versions:

- Basic. The introductory package offers customer service and internal business processes modeling.
- Professional. Focusing on complex and large-scale project related to manufacturing, logistics and supply chain.
- Enterprise. A solution for the organization facing a wide range of modeling problems.
- Contact center. Application focused on call centers simulation.
- Factory analyzer. Package focused on manufacturing, process and packaging.

The simulations are numerical, but in case the designer wants to see a 3D animation, the Arena 3DPlayer offers the ability to create and view animations. The animation speed can be controlled and it is possible to record AVI files. Also it can import VRML shapes and DXF files. It targets users who need a more realistic perspective of their simulation animations. 3DPlayer is offered as an option to the any of the versions mentioned above

Visual Components - 3DCreate

Visual Components' 3DCreate is the authoring environment for developing interactive 3D simulation software and reusable visual component libraries. With 3DCreate, users of all experience levels can generate high fidelity digital

replicas of current and developmental production equipment from existing design engineering information. Once created a visual component is published for general access via a web page or a local library. [3DCreate]

Bringing 3D CAD data to life as a simulation component supports an organization and its trading partners on many levels. The life-like simulation behavior is a tool for engineers to analyze system performance in fine detail, and it's a tool for customers to reduce their risk in proposal selection [3DCreate].

The software uses a graphical approach for simulating different scenarios. It also provides an extensible behavior by using a COM interface and a python script executor. The application can be embedded in other programs and programs can be connected through COM. Those characteristics are very useful to extract statistics, display the data on charts or save it on databases.

4.2.4. Comparison of Simulation Software

This comparison was done based in the characteristics offered by the software that would fit better to the project needs. The requirements for the project were the follow:

- Interface with Java or .NET.
- Simulation models and execution in 3D.
- Possibility to extend the functionality of the software via plug-ins or scripts.
- Library of models available.
- Easy to learn and use.

Table 2 shows a comparison of the different simulation software presented on the previous section. This table is based on the information recollected from the product vendors and the experience of the author with the software.

Table 2 *Comparison of Simulation Software*

Simulation Software	Java/.NET Interface	3D Simulation	Scripts/ Plug-ins	Models availability	Ease to use / Drag and Drop
Programming Languages	Yes	No	-	-	No
General purpose software	Yes	No	-	Yes	No
Automod	C++	No, uses a 3D viewer	N/A	Yes	Yes
Arena	VB	No, uses a 3D viewer	Yes	Yes	Yes
Extend	.NET	Only on ExendtSim Suite	Yes	Yes	Yes
3DCreate	.NET	Yes	Yes	Yes	Yes
Witness	.NET	No, uses a 3D viewer	Yes	Yes	Yes

In order to develop federates to connect to the federation the simulation package has to be able to have an interface for Java or COM (Component Object Model). This interface should allow to the developer to program the federate within the simulation objects or using a way to communicate from the simulation to an external federate (Web services, sockets, etc).

Most of the simulation packages compared in here are based on models that are 2D, and they use external plug-ins to make the rendering of the model in 3D. This was a drawback for such packages, since it is easier to work with a program where the models are already in 3D, easier to connect, move and rotate incrementing the ease of use for the end user. 3D simulation software is also easier to learn for first time users, resulting in the fact that designers don't need to code or script any behavior.

The ease of use is also increased when the software includes model libraries. Users can take advantage of those libraries and start designing their simulations as simple as drag-and-drop components to the scenario. The following table shows a comparison of the properties of different simulation suites.

As a result of the observations it was decided that the software that would fit better the needs of the project was Visual Components – 3DCreate. The decision was based in the fact the package complies with the needs of the project such as:

- It has a COM Interface, indeed no Java supporting.
- Based completely in 3D Models.
- Scripting by using Python, which is a fast and powerful feature.
- 3D Models library already available.
- Drag and drop components to layouts.

These characteristics were consistent also with our analysis of the HLA software. After this analysis the next step was to plan the test case and develop the tool that would allow us to distribute a manufacturing simulation.

5. HLA-DS PILOT IMPLEMENTATION

This chapter describes the pilot implementation of the HLA-DS tool. The description goes from the implementation to the results obtained to allow the reader to fully understand its principles.

The tool was named HLA-DS, as coming from High Level Architecture and Distributed Simulation conjunction. This tool uses different technologies that allow distributing simulations across the world. Figure 15 illustrates the intended connection between the simulation package and the HLA RTI software.

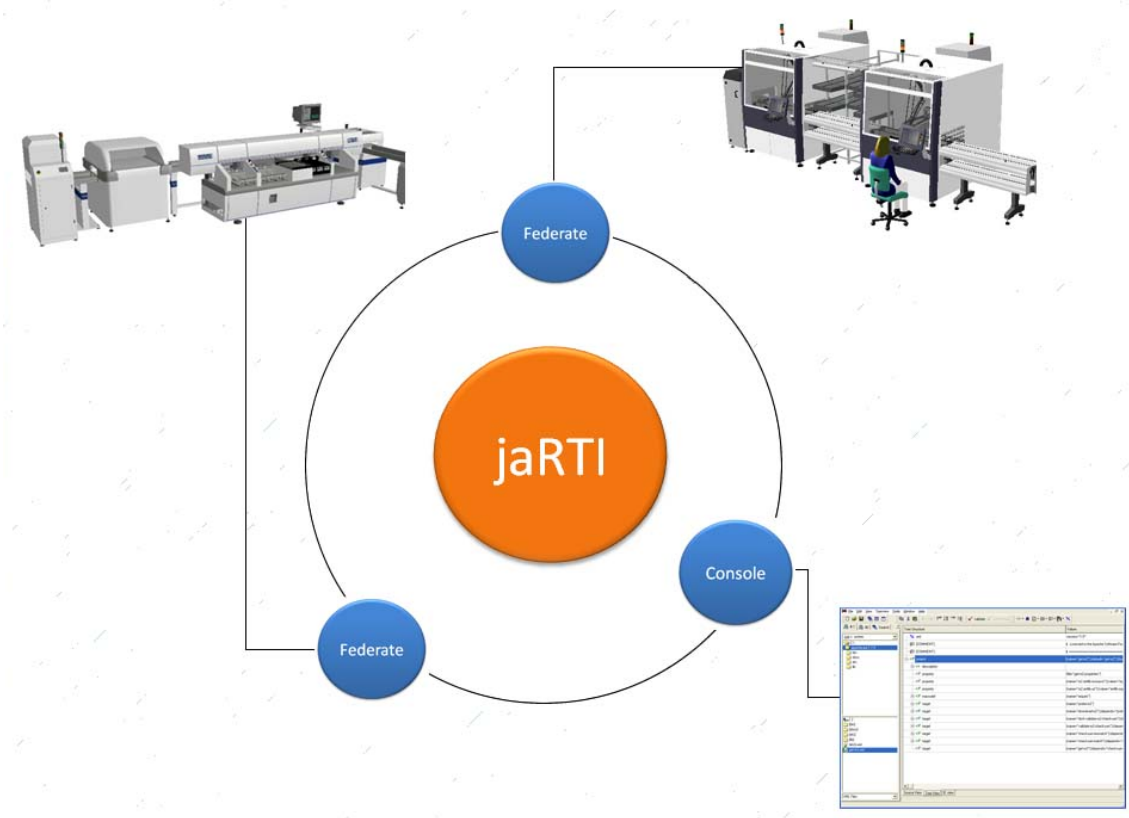

Figure 15 *Proposed connection between simulations and RTI.*

In order to accomplish this schema, an architecture had to be designed. The follow section describes the architecture and its parts in details.

5.1. Architecture

The architecture was designed to allow several Visual Components 3DCreate simulations to communicate within each other by using HLA.

With that goal in mind, different layers were designed to allow the flow of information from the simulation package to the HLA and backwards.

Figure 16 shows the architecture design for the HLA-DS tool. The architecture consists of five layers, on the top layer resides the distributed software and in the bottom layer HLA. Each layer is explained next.

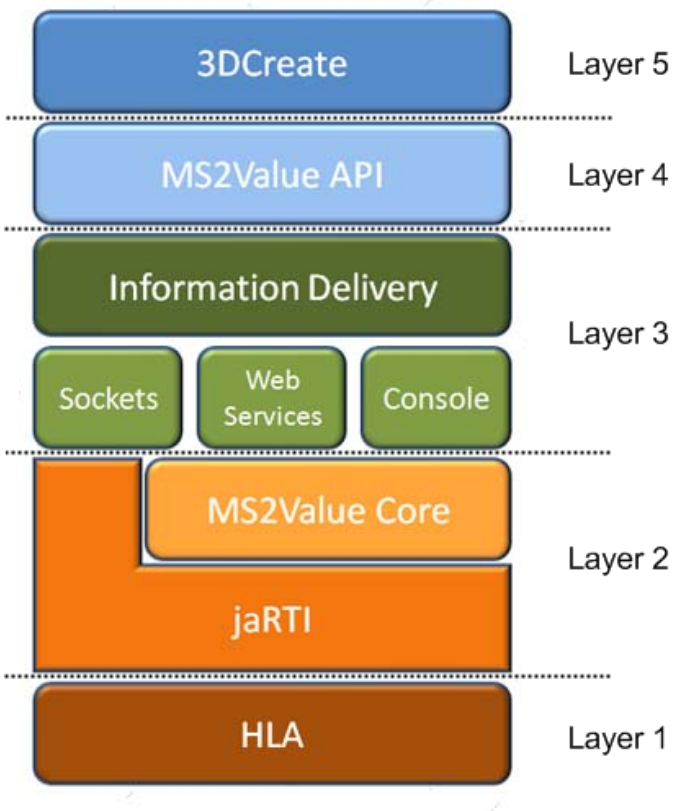

Figure 16 *Architecture for the HLA-DS tool.*

5.1.1. Layer one: HLA

This layer is the mainframe where all the communications between simulations take place. All these interactions are done according to the High Level Architecture standard.

5.1.2. Layer two: JaRTI

The second layer is completely handled by the RTI, in our case the JaRTI. It receives all the data from simulations in the upper layers and accordingly to the OMT it delivers this data to the next layer. This layer hosts all federates. These federates are in direct contact with the simulations through the next layer.

Federates act as gateways for the simulations and components connected to them and their task is to pass the information that they received to the RTI. Based on the functionality needed from federates, two types were designed:

- Management federates.
- Simulation federates.

The management federate is in charge of creating the federation, starting and stopping the execution of the federation, sending synchronization points and destroying the federation.

The simulation federate is in charge of publishing or/and subscribing to information shared by other simulation federates. In addition, the simulation federates should warn to other federates when is about to leave the federation. This is important when other federates are subscribed to the data shared by the resigning federate, so this subscribers can take the necessary steps towards this event.

Federates can obtain information from the Information Delivery layer in two different ways:

- Directly, by using sockets.
- Connecting though the MS2Value Core via Web Services.

Simulation federates receive and send information to the application through sockets, which are described in the next layer.

MS2Value Core:

Federates that manage the federation can connect through web services. This part of the project is explained in deep in the Web services part. The console was not part of this thesis project.

5.1.3. Layer three: Information Delivery

This layer handles all them movement of data between the MS2Value API and the JaRTI layers. There are three forms of communication available under this architecture:

- Sockets
- Web services
- Console

A socket is an abstraction, which provides an end point for communication between processes. Sockets originate from BSD UNIX, but are also present in most version of UNIX, including Linux as well as Windows NT and Macintosh OS. For a process to receive messages, its socket must be bound to a local port and one of the Internet addresses of the computer on which it runs. Messages sent to a particular Internet address and port number can be received only by a process whose socket is associated with that Internet address and port number. Processes may use the same socket for sending and receiving messages [Colouris et al. 2001]

The information flow between the MS2Value API and JaRTI is mainly handled through sockets. Each layer opens a socket port to listen and send information to the other layer. This information should be commonly understood by the receiver and sender.

The console was planed to be a graphical application in which the user can connect to a federation and interact with it. By using this application the user can visualize federates that are connected and their shared attributes. In addition the console would also have some management tasks, e.g. disconnect federates, create and destroy federations. The web services communication is covered in detail in a following section of this chapter.

5.1.4. Layer four: MS2Value API

This layer is the interface between the 3DCreate software and the Information Delivery layer. The layer uses one of the advantages of 3DCreate, scripting languages and COM API. These allow creating programs that can extract information, modify the simulation environment and receive data from other simulations.

This layer was programmed by the use of two different tools:

- Python Scripts
- 3D Create COM API
- The 3DCreate suite comes bundled with the 2.3 version of Python Script.

Python is a dynamic object-oriented programming language that can be used for many kinds of software development. It offers strong support for integration with other languages and tools, comes with extensive standard libraries, and can be learned in a few days. [Python 2007]

The python scripts are used to customize a machine model. 3DCreate offers to developers and designers complete access to the properties, methods and events of the components, the simulation environment and the application. This is very useful to create different behaviors and supervise them by using a python script, in case some event is triggered, e.g. a sensor; a response can be executed by the script.

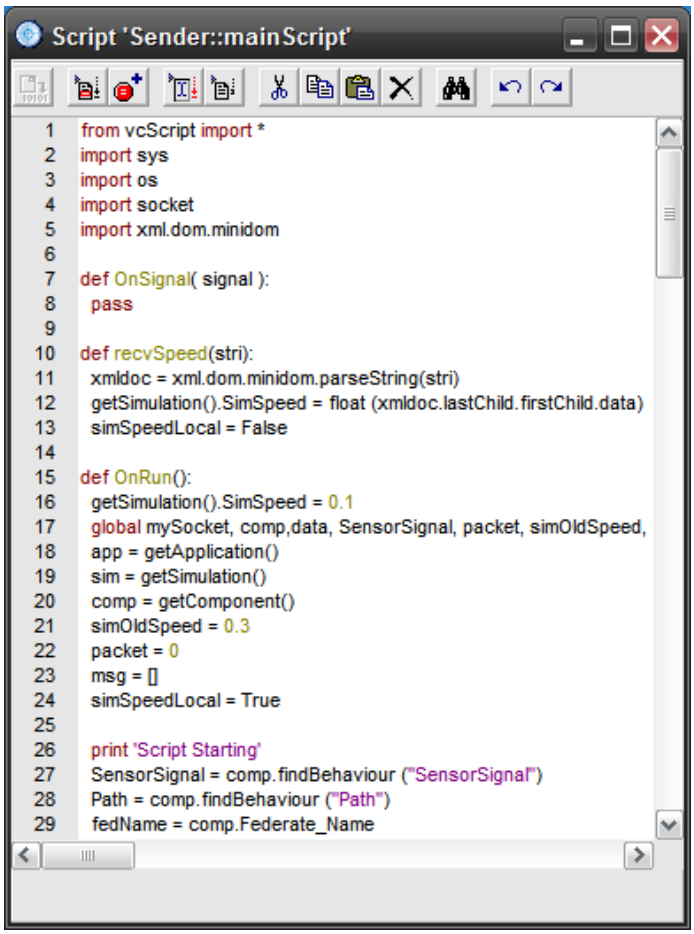

Figure 17 *An example of a python code to get actions from buttons and menus in the 3DCreate suite.*

Figure 17 shows an example of a script that allows capturing events from a button and menus in the program. This simple script allows different values to be printed out every time the value of a menu was changed.

This offers infinite possibilities on how to use python scripts to improve the realism of the simulation. A component can contain multiple scripts. In addition to the advantages presented before, the scripts are lightweight executed by the software resulting in a detailed simulation that doesn't overwhelm the processor when is executed.

Python scripts embedded in components developed in this layer were executed when the simulation was run; at this point the connections between the previous layer and the next layer are done. After the connection is established the information starts flowing between these two layers.

The 3DCreate COM interface provides all the operations available in the GUI as well as access to the layout model. It is designed primarily to allow client applications to create and manipulate components and layouts. It consists of the following parts [3DCreate User's manual]:

- Dynamic property interface
- Application object
- Component list and node tree
- Command and selection lists
- Feature tree interface
- Behavior interface and descendants
- Geometry interface

The COM interface provides stronger control over the application. Applications can be created and embedded in the 3DCreate as a tabs or can run independently of the software. This interface allows programming software for the simulation suite in different programming languages, for example:

- Visual Basic
- Visual C++
- Visual C#

In this layer, a tab for the HLA-DS application was designed for the 3DCreate environment. This tab was programmed on C# language using Microsoft® Visual Studio® 2003.

5.1.5. Layer five: 3DCreate

The 3DCreate layer is completely in charge of the simulation task. This is done completely by the 3DCreate simulation software.

In 3DCreate users can create components, or use components from the local library and drag them to the simulation environment. After several components are dragged to the environment, these components can be snap with each other and/or arrange them in different layouts. An example of basic layout is shown in Figure 18.

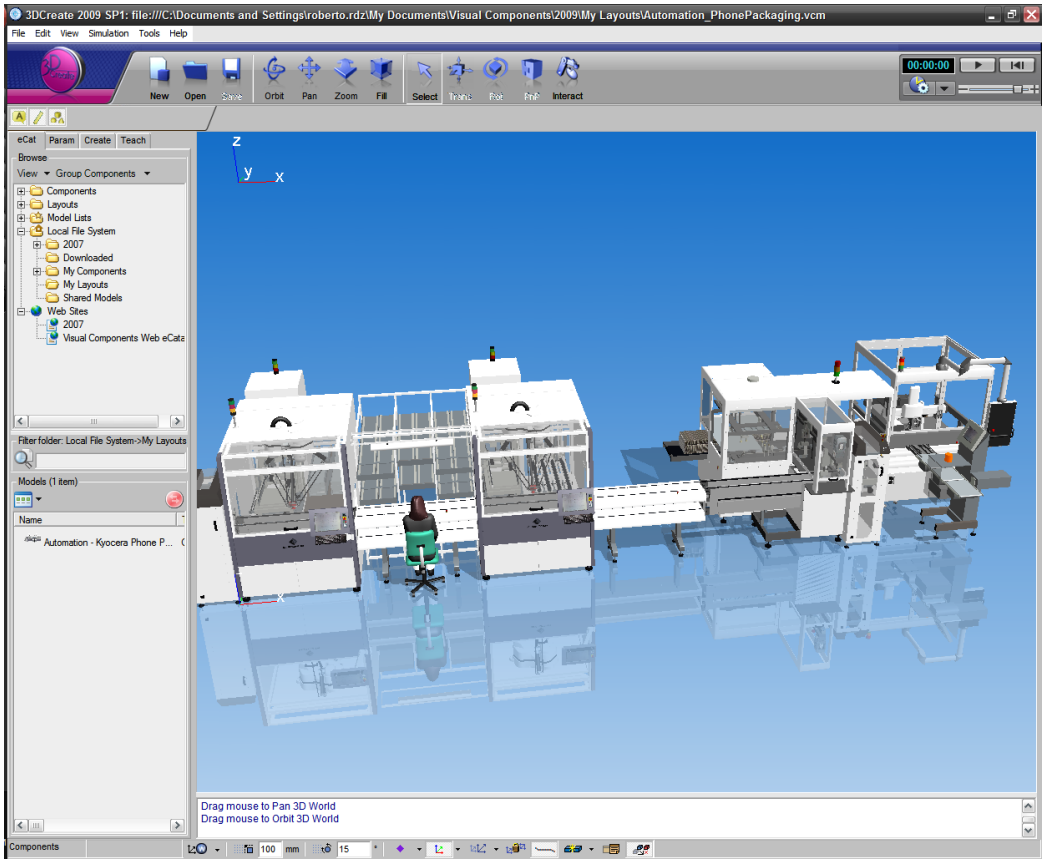

Figure 18 *Example of a simulation running on the 3DCreate software.*

The simulation is executed by clicking on the “Run” button on the top-right corner. Following that action the simulation starts to be executed where

machines and robots start their preprogrammed tasks until the user ends the simulation by clicking the “Pause” button.

The use of layers during the definition of the architecture of the system permitted to develop different areas that were implemented individually during the project. Layers allow to isolated areas, which eases finding problems and fixing them, with little or no repercussions at all in other areas of the system.

5.2. Web services

By definition “a Web Service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards”. [Earl 2004]

Using Web Services gives the following advantages:

- Invocation of applications around the Internet and across companies.
- Standardization of applications.
- Scalable and extensible application.

Service-Oriented Architecture (SOA) is defined in the W3C Glossary as “A set of components which can be invoked, and whose interface descriptions can be published and discovered.”[W3, 2008]

A basis of SOA is the concept of service as a functional representation of a real-world business activity that is meaningful to the end user and encapsulated in a software solution. Using the analogy between the concept of service and business process, SOA provides for loosely coupled services to be orchestrated into business processes that support business goals. [Stojanovic, 2005]

In addition, SOA extends the use of Web Services for applications by embedding these services in the application. As a result, the applications are based in Web Services, not just applications with Web Services as an extension.

SOA is based on Extensible Markup Language (XML), defined later in the document, using the XML Layers, with focus on exposing existing application logic as a loosely coupled application. [Earl 2004]

Applications using SOA integrate a new tier to the architecture of the application; this new tier is called the integration tier. This tier is in charge of sending data to the next tier of the application, as is shown in Figure 19. This makes possible that the next tier is not in the same system but anywhere in the world, being the principal advantage of using SOA.

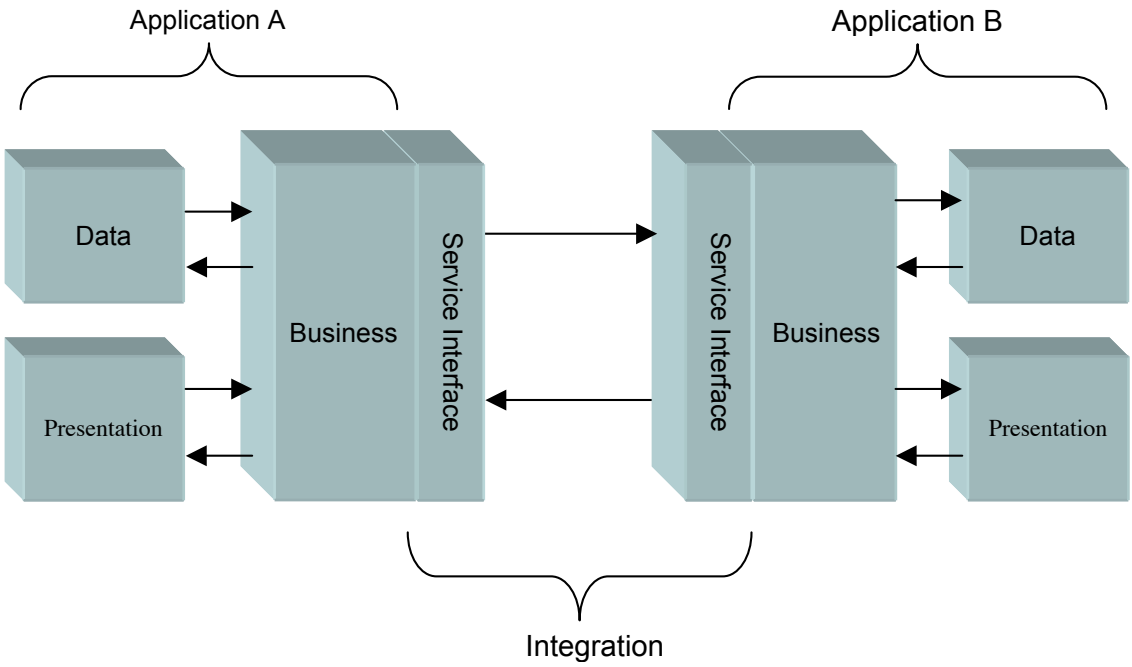

Figure 19 *Logical Representation of a Service Oriented Integration Architecture [adapted from Earl 2004].*

All of these advantages offered by Web Services and SOA were taken into account when planning and selecting the best technologies for the architecture of the HLA-DS application. Specifically these technologies were applied when developing the Federation Manager, which allowed creating a federation from a client without implementing the application in every client.

Developed Web services

As it is mentioned above, the second layer of the architecture specifies that the Manager federate would make use of web services technology for communicating with the federation. This allows invoking and controlling remotely the behavior of the federation via the manager federate.

The following services where created to manage the federation:

- Create federation. As it name points out, it creates a new federation and sets the federation to wait for clients to connect.

- Start Simulation. After all the clients have connected to the federations, the manager sets the simulation to start. This action will allow interactions from one client to other(s).

- Stop Simulation. The stop simulation web service sends a message through the federation to all federates subscribed to it to stop their interactions in order to stop the simulation and resign from the federation. After all of them have resigned, it destroys the Federation.

With the use of those three web services, the Manager federate facilitates the remote control of the federation. The following code is an extract of the web service implemented to create the federation:

```
   public String createFederation (String fedname )
{
     ref = SingletonObject.getSingletonObject(fedname);
     String result = ref.createFederation(fedname);

     return ((String) result);
}
```

This code shows that after getting a unique reference of the Manager federate, this instance calls to the method that creates a federation instance. The parameter passed to the method is the desired name for the federation. The full code programmed to create the Manager federate web services can be found in Appendix 1.

There were several considerations when designing and implementing the federation Manager. First of all, there should be only one instance of the Manager running at the moment. When an application is invoked by a web service, a new instance of the application is created, giving the possibilities that several instances of the application can be created and run separately. In the case of the federation manager this behavior is not acceptable and undesirable since it will create problems if the service to create a new federation is called several times. This will cause that the web service tries to create several federation managers, which will result in different managers trying to start or stop the federation causing an erratic behavior in the Federation.

In order to avoid this problem the service was programmed to only create a new instance of the federation manager if no instance of it exists already. This kind of programming pattern is called “singleton” and is represented with following code:

```
  public static SingletonObject getSingletonObject()
{
     try {
          if (ref == null) {
               ref = new SingletonObject();
          }
     }
     catch (Exception e){
          System.out.println("ERROR: " + e);
     }
     return ref;
  }
```

In case that an instance of a federation manager exists, every request would be served by the existing federation manager. This ensures that no errors are created in the federation, for example two federations trying to create a federation at the same time, since an error might disrupt its execution and result in erroneous behaviors or crash of the system. The full implementation of the Singleton that calls the federate manager is showed in the Appendix 2.

The web services created were hosted in an Apache Tomcat server with Axis 2 web services framework. The setup used for the implementation of this experiment was based totally on the instructions that came with the software. The explanation of how to install the web server is not covered in this thesis work. A very detailed explanation of how to install and configure the Apache Tomcat server can be found in the following address http://tomcat.apache.org. Additionally, a guide for installing the framework for supporting web services can be found in this address http://ws.apache.org/axis.

For this implementation the version 6.0.13 of the Apache Server was used along the Axis2 1.2 build. These versions of the software are recommended and were tested during the implementation of the project. Newer versions of the software are available at the moment of writing, but due to time constraints the author was not able to test them.

5.3. 3DCreate

The following step was to define the federates that will connect to the simulation. These federates are implemented in the simulation suite and pass the information data from the simulation environment to the federation.

As proof of concept, an implementation of two models with different behaviors were created and programmed in 3DCreate. The first model was defined as sender of information to the federation; whereas the second model focused on receiving data sent by the first federate. Both models have the ability to change the simulation speed. Figure 20 shows the first created model, mentioned above, which presents a simulator where mobile telephone parts are packed. The machines in this simulator pick the parts from the feeders' trays and place them on pallets that are transported by a conveyor. The receiver federate, shown in red on the right side of Figure 20, is placed at the end of the conveyor. This model connects this simulator to a federate, sending information every time a package arrives to it. The model uses a python script that opens a socket to communicate with its Federate counterpart and share information with the Federation.

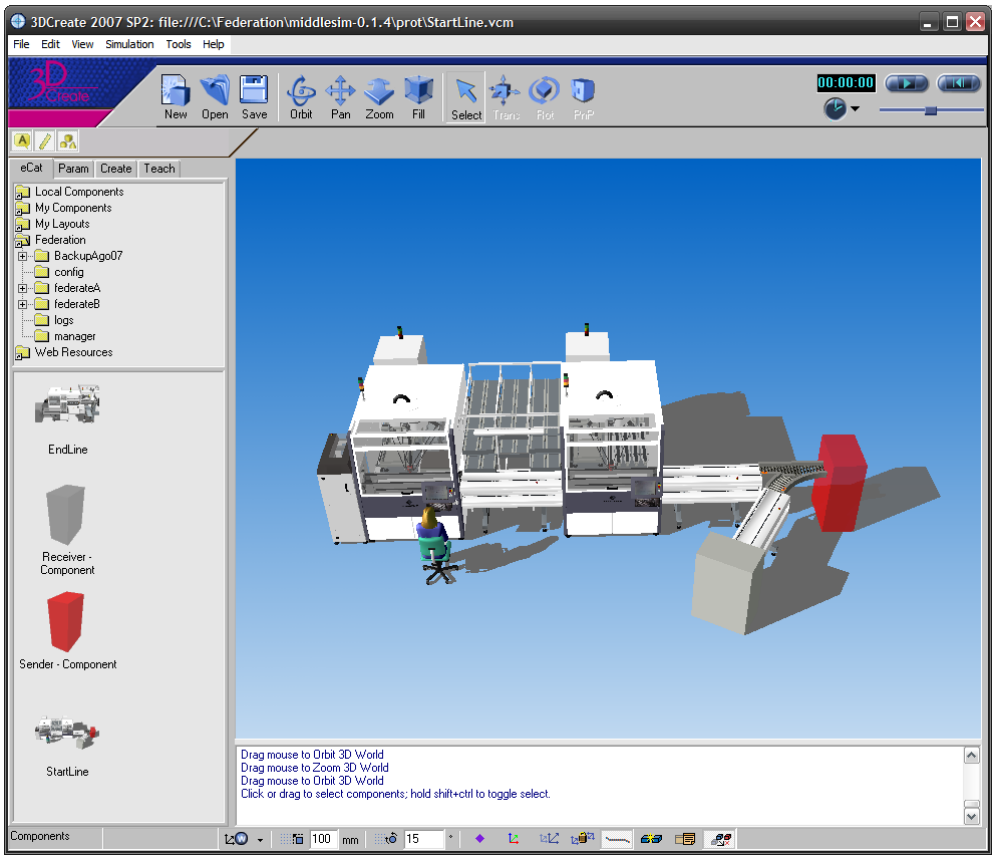

Figure 20 *3D Simulation model with the federate implementation (shown in red).*

The information that each federate shares with others was defined previously in the OMT of the federation. The full OMT definition can be found in Appendix 3. For this experiment the OMT only allowed interactions of two types of products the changes on simulation speed by the simulators, as shown by the following code:

```
(class ManufacturedPackages reliable timestamp
     (parameter packageA)
     (parameter packageB)
)
(class SimulationSpeed reliable timestamp
        (parameter simSpeed)
 )
```

During this experiment two types of products were defined to be sent from one simulator to the next one: one was only an empty container and the other one was a similar box containing a mobile phone, its cover and a battery. Figure 21 shows one of the latter packages where all the mobile telephone components can be appreciated.

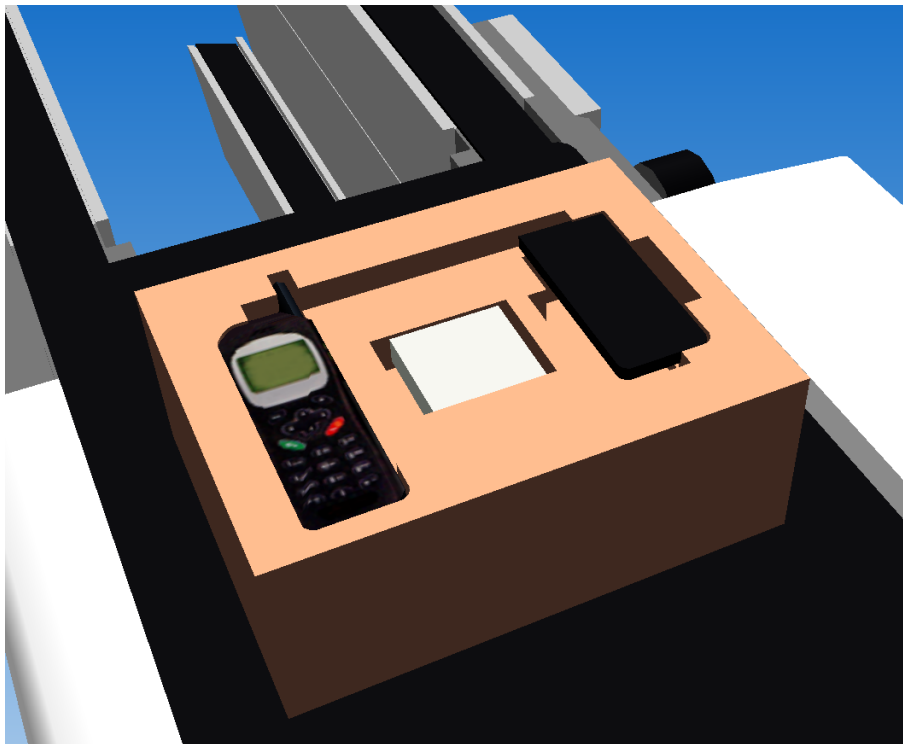

Figure 21 *Container sent by the Simulator "A".*

The data corresponding to the box and its contents are sent by one simulator via the federate to the federation; from there other subscribed interface federates can pass the message to their simulators. In this specific experiment

the boxes were partly processed in the Simulator “A” and finally packed in Simulator “B”.

To send all this information between simulations a message format was designed and implemented in the federates. The format chosen was XML since it is easy to use, adapt and extend to individual needs. The messages format used was similar to the following example:

```
<objectP>
     <Name>string</Name>
     <VCID>string</VCID>
     <objectC>
          <Name>string</Name>
          <VCID>string</VCID>
          <Center.X>float</Center.X>
          <Center.Y>float</Center.Y>
          <Center.Z>float</Center.Z>
     </objectC>
</objectP>
```

The simulation experiment was focused on sending packages from one simulator to another. As it was described above, a package is a board that contains several parts of a mobile phone. In the message the container board was designed as “objectP”, or parent object. The objects contained by this cardboard were designed as “objectC” or child objects. Name and VCID (Visual Componets Identity) are parameters that help to identify the type of product is being analyzed and shared by the federate. In Figure 21, the package show that in the container board has pockets specially designed for the placement for each of the parts of the mobile phone. In order to place the parts in the right position, additional information about the coordinates where the parts of the mobile phone have to be placed, are provided. XML permits that if several parts are provided on the container the “objectC” section will repeat as many times as needed by the container, describing each of the parts individually.

Figure 22 shows the console where Simulator “A” and its federate, Federate “A” subscribe to the Federation. It also shows that the step where the federate publishes/subscribes to the desired information.

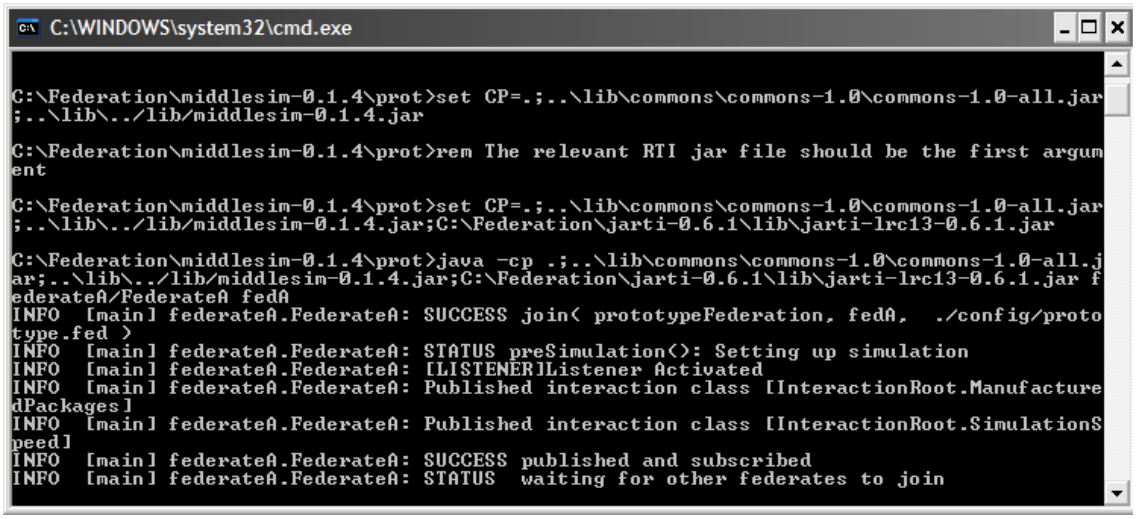

Figure 22 *Screenshot of the console output of a Federation.*

The lines shown in the bottom of Figure 22 describe that the federate starts waiting for other federates to join to the Federation to start the simulation. This was a condition in this experiment, since the Federate Manager was the only federate able to start the simulation until all federates were signed in.

5.4. Summary

During this chapter the test implementation was presented and described. A detailed description of the architecture was shown as well as each of the parts composing the proposed architecture. Additionally the functionality of the test implementation and some algorithms used were described. It is important to note that the experiences of the author are included in this chapter.

The conclusion and results of the experiment done during this thesis work are shown in the following chapter.

6. CONCLUSIONS AND RESULTS

This thesis presented the advantages of using simulation as a tool in different industries, in particular, a distributed simulation for manufacturing processes. The requirements for implementing a distributed simulation and a method to distribute it along different clients were explained in detail.

6.1. Conclusions

The technological trends presented at the beginning of this thesis work have showed that until now, manufacturing companies haven't taking real advantage of the real power of simulation for their own benefit. Game and defense industries have been using simulations for several years by now and have learned that simulation is a tool that will give them advantages over their competitors. Some of the fields where simulation has showed unquestionably its potential are designing of products and simulation of virtual environments, among others.

Companies, which use simulation as a tool at the present time, still develop complex models that are commonly stored and forgotten in insolated computers. Additionally, in recent years companies have started to move their offices and production facilities closer to their customers in geographical dispersed locations. This latter fact has led to the situation where models are housed in those facilities without interaction with the outside world, becoming isolated and rarely reused. By using a distributed simulation approach, those problems can be solved.

The implementation of the HLA-DS tool showed that the concept of integrating different tools like a 3DCreate simulation and a HLA RTI is feasible. This integration allowed common models designed in the simulation suite to communicate with other simulations by using plug-and-play components resulting in a set of simulations interconnected as a whole. 3DCreate allows using drag-and-drop components that are easily plugged to existing models. By using the components designed in the implementation of this thesis work, a previously isolated model was able to communicate with others models through the HLA RTI.

Distributed simulation has the potential to become an important tool for widely distributed manufacturing organizations. A very promising field for the effective

use of distributed simulation is Desktop factories. Desktop factories are a push towards more efficient manufacturing operations, and because of their very nature, distributed operations are a must. This increases the complexity of overall capacity calculations and overall system simulation.

The HLA-DS application developed as part of the MS2Value serves as a theoretical proof of distributed simulation for manufacturing systems. A pilot case study has been developed, but no industrial application has been foreseen in the near future. Development on the HLA-DS will continue still as an open source project, partly funded and developed by Visual Components and research partners, such as Tampere University of Technology.

6.2. Future Work

During the implementation of this thesis work there were new ideas of what could be a future development of this project to solve some of the problems found during the implementation phase. This is the case of the JaRTI framework, which didn't allow a direct implementation with the 3DCreate suite. Federates in the future should be develop by using the feature of C++ bindings, which was not present at the time of the implementation. The use of those features would allow direct development of federates in connection with the simulation suite by means of the COM interface. That will solve any connection problem that could arise in the actual implementation and decrease the actual time of response of the simulations.

The testing part of the federation was done in a limited scale. The amount of federates interacting concurrently during the tests done in the implementation phase was five at its maximum. A set of tests with a larger number of federates must be planned and implemented to show that the system reliability is also good in a larger scenarios. Additionally the information shared by federates must be extended assure that other study cases could take advantage of this approach. At present time, federates exchange limited amount of information, based principally on the study case.

Finally, the implementation of the visual console client should be done in the future for an easy exploration of the federation and the information exchanged by federates. The features mentioned above are not obligatory, but recommended by the author as a way to satisfy better the needs of future projects.

REFERENCES

Burks, T., Alexander T., Lessman, K. & LeSueur ,K. (2001). *Latency Performance of Various HLA RTI Implementations*. Simulation Interoperability Workshop, paper 015, 01S-SIW-014, Spring 2001

Calvin, J.O.,& Weatherly, R. (1996). *An introduction to the high level architecture (HLA) runtime infrastructure (RTI)*. In: Proceedings of the 14th Workshop on Standards for the Interoperability of Defence Simulations, Orlando, FL, March 1996, pp. 705–715

Carson, J. (2005), *Introduction to Modeling and Simulation*, Proceedings of the 2005 Winter Simulation Conference.

Cheng, F. (2000). *A methodology for developing robotic workcell simulation models*. In: Simulation Conference Proceedings, 2000. Winter, Orlando, FL, December, pp. 1265–1271

Cheung, S.,& Loper, M. (1994). *Synchronizing simulations in distributed interactive simulation*. Proceedings of the 26th conference on Winter simulation, 1316 - 1323, 1994.

Colouris, D., Dollimore, C.,& Kindberg, T. (2005). *Distributed Systems Concepts and Designs*. pp. 70-71, 129 ISBN 0201-61918-0

Dahmann, J., Morse, K. (1998). *High Level Architecture for Simulation: An Update*. Distributed Interactive Simulation and Real-Time Applications, International Workshop on, p. 32, Second International Workshop on Distributed Interactive Simulation and Real-Time Applications, 1998.

DIS Steering Committee, (1994). The DIS Vision, A Map to the Future of Distributed Simulation (Version I). IST-SP-94-01.

Earl, T. (2004). *Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services*. Prentice Hall, Upper Saddle River, NJ, 2004. ISBN-13: 978-0131428980

Eberst, C., Bauer, H., Minichberger J., Nöhmayer, H. & Pichler, A. (2004). *Self-programming robotized cells for flexible paint-jobs*. MechRob, Mechatronics and Robotics, (2004) p. 1089-1093.

Figure 2. By training with simulations soldiers are better prepared for the real battlefield. From: *PEO STRI Expands Support to Soldiers*. U.S. Army News, April 2007. Photo by Doug Schaub. [Online, viewed on 03/24/2010]. Available: http://www.defense.gov/transformation/images/photos/2007-04/hi-res/army_mil-2007-04-16-095532.jpg

Figure 3. Microsoft's Flight Simulator X immerses players into realistic airplanes' cockpits. From: *Screenshots, Microsoft Flight Simulator X*. [Online, viewed on 01/12/2010]. Available: http://www.microsoft.com/games/flightsimulatorx/screenshots/image_21.html

Figure 13. *MÄK RTI shows the federation data by the using of plug-in applications.* [Online, viewed on 04/04/2010]. Available: http://www.mak.com/products/rti.php

Fowler, J.W. & Rose, O. (2004). *Grand Challenges in Modeling and Simulation of Complex Manufacturing Systems*. SIMULATION, Vol. 80, No. 9, 2004, pp. 469–476.

Fujimoto, R.M. (2003). *Parallel simulation: distributed simulation systems*. Proceedings of the 35th Conference on Winter Simulation: driving innovation, December 07-10, 2003, New Orleans, Louisiana

IEEE Standard For Distributed Interactive Simulation, IEEE Std. 1278, IEEE.

IEEE Standard For Modeling And Simulation (M&S), IEEE Std. 1516-2000, IEEE.

IMTI (Integrated Manufacturing Technology Initiative) (2003). *Modeling & Simulation for Affordable Manufacturing – Technology RoadMapping Initiative.* Version 3.2, 18th of January 2003. Oak Ridge, TN: IMTI Inc 80p

Karlsson, M., & Olsson, L. (2001). *pRTI™ 1516 – Rationale and Design*. In Proceedings of the Fall 2001 Simulation Interoperability Workshop. Orlando, Florida. 01F-SIW-038. 2001.

Krahl, D. (2003). *Extend: An Interactive Simulation Tool.* Proceedings of the 35th Conference on Winter Simulation: Driving Innovation, December 07-10, 2003, New Orleans, Louisiana

Krishnan, N. (2001). *The JXTA solution to P2P.* JavaWorld.com. October 2001. [Online, viewed on 01/05/2010]. Available: http://www.javaworld.com/javaworld/jw-10-2001/jw-1019-jxta.html

Kühn, W. (2006) Digital Factory-Integration of simulation enchanging the product and production process towards operative control and optimization, International Journal of Simulation, 7(7), 27–39, 2006.

Lanner (2008). *Witness Simulation.* Lanner Group Limited. [Online, viewed on 04/03/2008]. Available: http://www.lanner.com/en/media/witness/overview.cfm

Law, A. M. & Kelton, W.D. (2000). *Simulation Modeling and Analysis.* McGraw Hill Higher Education; 3rd edition (April 1, 2000) ISBN-13**:** 978-0071165372

Lindqvist, L. (2007) *Unified Infrastructure for Simulation, Communication and Execution of Robotic Systems.* Msc. Thesis, Helsinki University of Technology. p. 36

Lua, E. K., Crowcroft, J., Pias, M., Sharma, R., & Lim, S. (2004). *A Survey and Comparison of Peer-to-Peer Overlay Network Schemes*. IEEE Communications Survey and Tutorial, March 2004.

Macedonia, M., (2002). *Games soldiers play*. IEEE Spectrum. March 2002*,* 32-37 Page 2265.

Manufuture, (2006). *Manufuture Strategic Research Agenda – Assuring the Future of Manufacturing in Europe*. Report of the High-Level Group 2006. Luxemburg: European Commission. 102 p. ISBN 92-79-01026-3.

MÄK Technologies (2009). *MÄK High Performance RTI.* [Online, viewed on 01/08/2009]. Available: hhttp://www.mak.com/products/rti.php

Open HLA (2009), *Open HLA Project Description*. [Online, viewed on 01/03/2009]. Available: http://ohla.sourceforge.net/

Perry N., Ryan P. & Zalcman L., (1998). *Provision of DIS/HLA Gateways for Legacy Training Simulators*, SimTecT 98 Conference Proceedings, Adelaide, South Australia, March 1998, pp. 227-232.

Perumalla, K.S. (2006). *Parallel and Distributed Simulation: Traditional Techniques and Recent Advances*. Proceedings of the 2006 Winter Simulation Conference. Monterey, California, December 3–6, 2006. Pp. 84–95. ISBN 1-4244-0501-7.

Phillips T. (1998). *Automod^{TM} by Auto Simulations.* Proceedings of the 2006 Winter Simulation Conference. Society for Computer Simulation, San Diego, CA, 213-8

Pokorny, T. (2007). *Anatomy of a Open Source RTI.* SimTecT 2007 Simulation Conference: Simulation - Improving Capability and Competitiveness (SimTecT 2007) Brisbane, Queensland, Australia, June 4 - 7, 2007

SISO (2010). *Simulation interoperability Standards Organizations - DIS Product Support Groups.* [Online, viewed on 04/12/2010]. Available: http://www.sisostds.org/index.php?tg=articles&idx=More&article=449&topics=110

Stojanovic, Z., (2005), *Service-Oriented Software System Engineering: Challenges and Practices.* Hershey, PA, USA: Idea Group Publishing, 2005. p ix. ISBN-13: 978-1591404279

Thomas, E, (2004). *Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services*. Prentice Hall, Upper Saddle River, NJ, 2004. ISBN-13: 978-0131428980

Visual Components (2007). Affordable plan layout and process simulation software. White Paper, 2007.

W3 (World Wide Web Consortium), *Service-Oriented Definition*. W3 Glossary. [Online, viewed on 03/24/2010]. Available: http://www.w3.org/2003/glossary/keyword/All/?keywords=service-oriented%20architecture

APPENDIX 1

```
package ems;

import com.lbf.middlesim.MSProxy;
import com.lbf.middlesim.fom.ICMetadata;
import java.util.Hashtable;
import org.apache.log4j.Logger;
import org.apache.log4j.PropertyConfigurator;

public class SingletonObject
{
      private MSProxy proxy;
    private ICMetadata stopMetadata;
    private static Logger logger = null;
      private String fedfile;
      private String iClass;
      private Hashtable HT;

  private static SingletonObject ref = null;

  private SingletonObject(String fedname) throws Exception
  {
       HT = new Hashtable();
     this.fedfile = "C:/Federation/middlesim-
0.1.5/prot/config/prototype.fed";
       this.iClass = "InteractionRoot.Management.StopSimulation";

       // Specify the binding and create the proxy
       System.setProperty( "msim.binding", "portico" );

       //This line went to the end, to use the Hash Table
       //this.proxy = new MSProxy( "executionManager" );
       this.logger =
Logger.getLogger(SingletonObject.class.getName());
       PropertyConfigurator.configure("C:/Federation/middlesim-
0.1.5/prot/config/log4j-prototype.properties");


  }

  public static SingletonObject getSingletonObject(String fedname)
  {
      try {
      if (ref == null) {
          ref = new SingletonObject(fedname);
          }

      }
      catch (Exception e){
            e.printStackTrace();
      }

          return ref;
  }

  public Object clone()throws CloneNotSupportedException
  {
```

```
  throw new CloneNotSupportedException();
  // that'll teach 'em
}

public String createFederation(String fedname)
{
      if (! HT.containsKey("EX"+fedname)){
            try {
                        this.proxy = new MSProxy( "EX"+fedname );
                        HT.put("EX"+fedname, proxy);

                        proxy = (MSProxy)HT.get("EX"+fedname);

                  // 1. create the federation //
                  logger.info( "Create federation with fed file [" +
                  fedfile + "]" );
                  proxy.create( fedname , fedfile );

                  // 2. join the federation //
                  proxy.join( fedname, "EX"+fedname, fedfile );

                  // 3. announce JOINED sync point //
                  proxy.announce( "JOINED" );

    // 3.1 set up ability to send "StopSimulation" interaction //
                  this.prepare();
                  return "Created";
            }

            catch (Exception ex){

                  HT.remove("EX"+fedname);
                  return "Error - "+ ex.getMessage();
            }
    }
    else{

            return "Error - Federation already exists";
    }
}

public void prepare()
{
          // fetch the metadata for the interaction //
this.stopMetadata = proxy.getFOM().getInteractionClass( iClass );

          // publish the interaction class //
          proxy.publishIC( iClass );
    logger.info( "Published interaction class [" + iClass + "]" );
}

public String startSimulation(String fedname)
{
      try{
                  proxy = (MSProxy)HT.get("EX"+fedname);

                  logger.info( "Starting Simulation" );

                  // 5. achieve the JOINED sync point //
                  proxy.achieveAndWait( "JOINED" );
```

```
                  // 6. announce and wait on the REGISTERED_OBJECTS
sync point //
                  proxy.announce( "REGISTERED_OBJECTS" );
                  proxy.achieveAndWait( "REGISTERED_OBJECTS" );

                  // 7. announce the READY sync point //
                  // after this point has been achieved, the
simulation will actually start //
                  proxy.announce( "READY" );

                  // 8. achieve READY sync point //
                  proxy.achieveAndWait( "READY" );

                  return "Started";

      }
            catch (Exception ex){

                  return "Error - "+ ex.getMessage();

      }

  }
  public String stopSimulation(String fedname)
  {
        try{

            proxy = (MSProxy)HT.get("EX"+fedname);
            // send an instance of the stop simulation interaction //
            proxy.sendRO( this.stopMetadata.createInstance() );
            logger.info( "Sent RO instance of [" + iClass + "]" );

            // 11. register FINISHED sync point and wait for it //
            proxy.announce( "FINISHED" );
            logger.info( "Waiting for other federates to finish" );
            proxy.achieveAndWait( "FINISHED" );

            // 12. register DESTROY point //
            proxy.announce( "DESTROY" );
                  logger.info( "Waiting for other federates to resign"
);
                  proxy.achieveAndWait( "DESTROY" );

                  // 13. resign from the federation //
                  logger.info( "All other federates resigned,
resigning and destroying" );
                  proxy.resign();

                  // 14. destroy the federation //
                  proxy.destroy( fedname );
                  logger.info( "Execution over" );
                  HT.remove("EX"+fedname);
                  return "Stopped";
            }
            catch (Exception ex){

                  return "Error - "+ ex.getMessage();

            }
  }
}
```

APPENDIX 2

```
package ems;

import ems.SingletonObject;

public class ExecutionManagerService{
      private static SingletonObject ref = null;

      public String startSimulation (String fedname)
      {
            ref = SingletonObject.getSingletonObject(fedname);
            String result = ref.startSimulation(fedname);

            return ( (String) result);
      }

      public String stopSimulation (String fedname)
      {
            ref = SingletonObject.getSingletonObject(fedname);
            String result = ref.stopSimulation(fedname);

            return ((String) result);
      }

      public String createFederation (String fedname )
      {
            ref = SingletonObject.getSingletonObject(fedname);
            String result =   ref.createFederation(fedname);

            return ((String) result);
      }

}
```

APPENDIX 3

```
(FED
   (Federation Prototype)
   (FEDversion v1.3)
   (spaces)
   (objects
    (class ObjectRoot
       (attribute privilegeToDelete reliable timestamp)
       (class RTIprivate)
     (class Data
                  (attribute speed reliable timestamp)
                       (attribute time reliable timestamp)
                       (attribute size reliable timestamp)
     )
    )
   )
   (interactions
     (class InteractionRoot reliable timestamp
       (class ManufacturedPackages reliable timestamp
           (parameter packageA)
           (parameter packageB)
       )
       (class SimulationSpeed reliable timestamp
           (parameter simSpeed)
       )
       (class Management reliable timestamp
          (class AdjustRatio reliable timestamp
               (parameter newRatio)
         )
       (class StopSimulation reliable timestamp)
       )
     )
   )
 )
```