A DISTRIBUTED TIME STEPPED SIMULATION APPROACH FOR ANALYSIS AND COMPARISON OF SHOP FLOOR CONTROL ARCHITECTURES

by

Umut Beşikci

B.S., Industrial Engineering, İstanbul Technical University, 2004

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

Graduate Program in Industrial Engineering

Boğaziçi University

2006

ACKNOWLEDGEMENTS

I am grateful to my thesis supervisor, Assoc. Prof. Ümit Bilge for giving me a chance to work with her and her guidance and effort in this study. This work would not be completed without her suggestions and advices.

I would like to thank to Prof. H. Levent Akın and Assoc. Prof. Taner Bilgiç for their interest and participation in the thesis defense.

I am grateful to Gönen for helping me whenever he can, with his knowledge and experience. In addition to him I would like to thank other members of BUFAIM Laboratory Team, Erinç and Salim for their support and friendship. I also want to thank to Deniz and Burak who participated this study with their great efforts.

I would like to thank to Sezen for her hearty patience and unlimited support during my thesis study. I am also grateful to my friend Kamil for his help.

Finally I want to thank to my family for their encouragement and support which was my source of energy.

The work reported in this thesis is supported by Bogaziçi University Research Fund under Grant No: 01S107 and 05A301, and by State Planning Organization under Grant No: 01K120310

ABSTRACT

A DISTRIBUTED TIME STEPPED SIMULATION APPROACH FOR ANALYSIS AND COMPARISON OF SHOP FLOOR CONTROL ARCHITECTURES

Different shop floor control architectures (SFCAs) are trying to resolve control problem of Flexible Manufacturing Systems (FMSs) by proposing different paradigms. Hierarchical SFCAs enable a layered hierarchy and try to overcome the complexity of the system by aggregating the problem. Heterarchical approaches see the system as a whole that consists of autonomous and cooperative agents. To develop efficient applications of SFCAs, simulation is an appropriate approach which lets a limitless configuration and test possibility. But the methodology of the simulation application is very important and must be developed with considering the system at hand. When modeling a system composed of entities, which interchange messages, consideration of messaging structure becomes essential. In this respect, Parallel and Distributed Simulation (P/DS) is the most appropriate methodology for these systems. In this thesis, simulation applications for both hierarchical and heterarchical SFCAs are built with a distributed and parallel simulation approach. Thus a test bed for comparing the performances of different architectures and for developing new approaches to decision making, information sharing, or communication within each architecture is obtained. In developing P/DS applications, existing real-time SFCA applications in Boğaziçi University Flexible Automation and Intelligent Manufacturing (BUFAIM) laboratory are taken as basis. Applications are developed in a modular and object oriented fashion, to enable easy progression from simulation to real-time control and vice versa. With running each distributed module on a different computer and keeping the messaging real it is aimed to catch the dynamics of the communication structures which are the defining characteristics of the SFCAs.

ÖZET

ÜRETİM SAHASI KONTROL MİMARİLERİNİN ZAMAN ADIMLI DAĞINIK BENZETİM YÖNTEMİYLE KARŞILAŞTIRILMASI

Atölye Tipi İmalat Ortamı Kontrol Mimarileri (ATİOKM) Esnek Üretim Sistemlerinin (EÜS) kontrolü için sunulmuş çözümlerdir. Hiyerarşik ATİOKM sorumluluklara göre atelyeyi seviyelendirerek problemin karmaşıklığını indirgemeyi amaçlamaktadır. Dağınık ATİOKM ise sistemi bağımsız ama birlikte çalışan bir ajanlar topluluğu olarak görmektedir. İşlevsel ve etkili kontrol mimarileri geliştirebilmek için benzetim yöntemi konfigürasyon üstünlüğü ile uygun bir yöntembilmi olarak ortaya çıkmak- tadır. Birbiriyle mesajlaşan birimlerden oluşan bir sistemin modellenmesi iletişim sisteminin göz önüne alınması gibi ek önem noktaları doğurmaktadır. Bu sebepten dolayı Paralel ve Dağınık Benzetim (P/DB) yöntemi yukarıda belirtilen sistemler için uygun bir modelleme aracı haline gelmektedir. Bu tez çalımasında, dağınık ve hiyerarşik kontrol mimarileri için zaman adımlı, P/DB benzetim uygulamaları geliştirilmiştir. Böylelikle değişik mimariler için yeni karar yaklaşımlarını, bilgi paylaşım yapılarını, ve ya iletişim altyapılarını deneyebilmek için bir test yatağı geliştirilmiştir. Bu uygulamaları geliştirmek için Boğaziçi Üniversitesi Esnek Otomasyon ve Zeki İmalat Laboratuvarında geliştirilen kontrol mimarileri temel alınmıştır. Uygulamalar nesne tabanlı ve modüler bir metodoloji izlenerek geliştirilmiştir, böylelikle benzetim ve gerçek zamanlı uygulamalar arsında kolay bir geçiş sağlanmıştır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AGV	Automated Guided Vehicle
AFMCS	Advanced Factory Management and Control System
AMRF	Advanced Manufacturing Research Facility
AS/RS	Automated Storage and Retrieval System
ATC	Apparent Tardiness Cost
BUFAIM	Boğaziçi University Flexible Automation and Intelligent Manufacturing Laboratory
CC	Cell Controller
CE	Causality Error
CIM/OSA	Open System Architecture
CNC	Computer Numerical Control
CIM	Computer Integrated Manufacturing
FMS	Flexible Manufacturing System
IT	Information Technology
HMS	Holonic Manufacturing System
MHS	Material Handling System
MSMQ	Microsoft Message Queuing
NC	Numerical Control
P/DS	Parallel and Distributed Simulation
PROSA	Product Resource Order Staff Architecture
SC	Shop Controller
SFC	Shop Floor Control
SFCS	Shop Floor Control System
SH	Synchronization Holon
SMC	SimulationMessageController
UML	Unified Modeling Language
XML	Extensible Markup Language

1. INTRODUCTION

A Flexible Manufacturing System (FMS) is an automated, computer-controlled manufacturing system, developed in order to address production of mid-volume, mid-variety parts. Mid-volume, mid-variety part production constitutes approximately 75 percent of discrete parts manufacturing. In this category of manufacturing, both flexibility and production volume are of prime importance, whereas flexibility to process a high variety of products is more important in a low volume-high variety system and production volume is more important in a high volume-low variety system (Singh 1996). Thus there is need to reduce the manufacturing cycle time and permit small batch production simultaneously to gain economic advantages similar to those of transfer lines while retaining the flexibility of stand alone numerical control (NC) machines. In an FMS much importance is put on integration of the physical components of the manufacturing system. The physical sub-system in a FMS includes the following:

1. A set of computer numerical control (CNC) machines augmented by a part buffer, a tool changer and a pallet changer. A FMS may consist of a number of workstations (flexible manufacturing cells) each having a number of such CNCs sometimes served by a robot.
2. Storage and Retrieval System for part input and output to the system.
3. Material handling systems composed of automated guided vehicles (AGV), conveyors and other systems to carry parts between workstations.

While these automated hardware should be compatible to work in an integrated fashion, much of the integration is to be achieved by a control sub-system that will run the FMS.

The control of an FMS is enabled by a network of control hardware which includes computers, programmable logic controllers, sensors and other devices. The control software is responsible for monitoring, controlling and real time management of parts and physical components in the FMS shop floor with respect to a set of rules and algorithms, and collecting data (Buzacott and Yao 1986).

Real-time management of orders and resources on the shop floor to execute a short term production plan is called Shop Floor Control (SFC). A SFC system receives production orders and necessary information such as quantity, due date, priority from a higher level production planning system, and returns shop floor status information to be used in updating the production plan for the next period. The SFC system is responsible for selecting specific process routings (in case of flexible process plans), allocating resources, dispatching parts, controlling part flow among various cells and workstations, downloading instructions to equipment controllers, monitoring the progress of activities, detecting and recovering from unexpected events and preparing reports on system status. The control system should be designed in a way to ensure optimum performance of the FMS, i.e. producing the given product range in minimum time and cost. Furthermore, the capabilities of the FMS control system determines, whether potential flexibility inherent in the FMS configuration can be effectively used to increase efficiency of the system.

There are alternative ways of dividing the SFC responsibilities among the shop floor entities leading to different control architectures, each with its own advantages and disadvantages. Research on appropriateness of different SFC architectures (SFCA) in creating an integrated, flexible and responsive manufacturing system therefore gains importance. The main aim of this thesis is to compare performance of an FMS under different SFCA.

As far as we are aware, there is almost no reported work in literature that investigates the effect of the control architecture on the performance of the system; all comparisons made are on theoretical and/or descriptive basis. In this thesis two main SFCAs, namely hierarchical architecture and heterarchical architecture, will be compared by means of parallel and distributed simulation experiments. In SFC there are many tasks that are processed simultaneously yet in a synchronized fashion by several different entities, and the communication structure among them has significant importance. An architecture's main aim is, in fact, to formalize and describe this behavior. Therefore the test bed for modeling and comparing the SFCAs should be able to capture this complex dynamic behavior. A time stepped simulation application is

developed in a distributed manner to realistically model the complexity and dynamics of a shop floor. Time stepped parallel and distributed simulation methodology is used because of the distributed viewpoint of the approach which correlates with the structure of the real time control applications. In the experiments, Boğaziçi University Flexible Automation and Intelligent Manufacturing Laboratory (BUFAIM) model factory is used as the sample case.

In the next three chapters, a review is given for the necessary concepts about the thesis; first SFCAs in the literature are introduced, then distributed simulation concept is explained and last BUFAIM model factory is described with respect to hardware configuration and SFCA applications. In the fifth chapter, the proposed distributed simulation applications of the two SFCA implementations in BUFAIM model factory is given. In chapter six performance based comparisons of SFCAs are reported which are gathered by using the distributed simulation application. In the last chapter conclusions of the study and the future research areas are given.

2. SHOP FLOOR CONTROL ARCHITECTURES

Researchers, both in academia and industry, give great interest to improve performance of manufacturing systems and their robustness to the changing environment. In addition to these requirements new challenges had arisen such as real-time responsiveness to disturbances, fault tolerance, easy hardware and software reconfigurability in order to control manufacturing environment efficiently. Information technology (IT) area offered new solutions for these challenges in the manufacturing with integrating all the activities of a manufacturing system which is known as the Computer Integrated Manufacturing (CIM) concept. Most of the CIM implementations in the industry could not answer the challenges because of the disunity between organizational structure and CIM environment, which can be thought as architectural problems (Babiceanu and Chen 2005).

An architecture is defined as "a framework or a set of rules and guidelines for managing the development and operation of complex systems". In a manufacturing environment architecture becomes a control tool, usually a software, where the aim is to constitute a framework and a set of rules to operate and manage a manufacturing system. In the literature related with the control architectures in the manufacturing systems, four basic types can be seen: centralized, hierarchic, modified hierarchic and heterarchic (Babiceanu and Chen 2005). In centralized control architectures, a central unit controls all the activities of resources and equipments whereas the idea behind hierarchical control architecture is decomposing responsibility into layers each of which is commanded by its supervisor. The modified hierarchical control architecture tries to increase the domain related responsibilities of lower level entities. In heterarchical control architecture all entities are seen as autonomous and co-operating agents (Yang and Lin 1998, Kotak *et. al.* 2003). For the sake of this study centralized control architectures will not be covered in this chapter. In the next sections hierarchical and heterarchical control architecture are explained in detail.

2.1. Hierarchical Control Architectures

Since the complex manufacturing systems already show the characteristics of a hierarchy, as far as the relationship between decisional and operational layers is concerned, a hierarchically structured shop floor control system seems to fit their nature. The complexity in manufacturing system can be handled with giving authority to layers of control system to manage the production. With that rationale, a hierarchical control architecture can be formulated as a pyramid with control modules in all the layers, each of which has its own functions and goals. This basic structure is used as a control system in the hierarchy. In a control system like this, commands flow up from higher to lower layers and feedback information flows up from lower to higher layers. Each command goes down through the layers and the response for the command follows the same path (Figure 2.1). The communication between layers is limited to master-slave relationships between parent and child nodes and the response time for a command or generating decisions according to a response are limited to technological and logistical considerations of the system (Balasubramanian *et. al.* 2001).

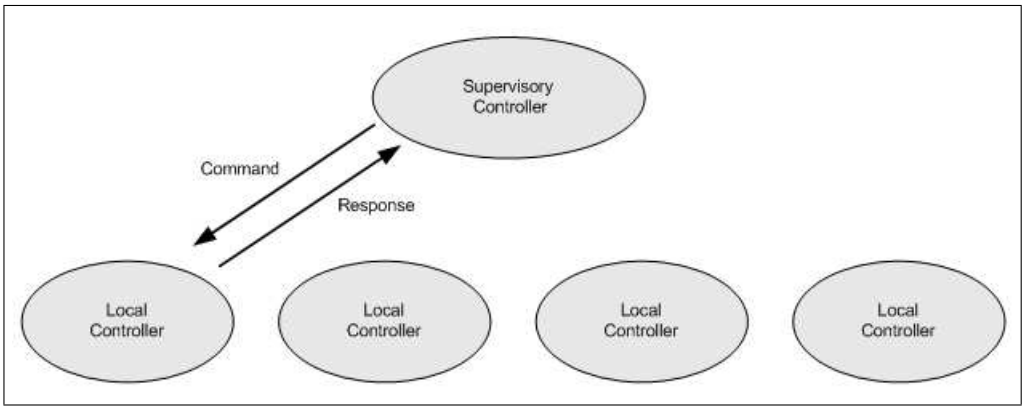

Figure 2.1. Communication between layers in a hierarchical system

Hierarchical control architectures are widely used in industry and there are various models for implementation. One of the most known hierarchical control architecture is Advanced Manufacturing Research Facility (AMRF) which is composed of five levels, namely: facility, shop, cell, work station and equipment. Another hierarchical architecture, Advanced Factory Management and Control System (AFMCS), is focused on the decision processes of a CIM environment and has four levels: factory, job shop, work

center and resource. The open system architecture (CIM/OSA)is also an important model which offers a library for building an enterprise architecture and an operation environment containing implementation tools and resources for the defined operations of the enterprise (Yang and Lin 1998).

The pure hierarchical control architectures have lots of drawbacks due to the rigid statues of layers and problems related information flow up between layers. Also operationally these drawbacks caused problems with reactivity to disturbances. Some of these problems are stated as below by Brussel *et. al.* (1998);

- Changing the structure of the system is difficult. Even a little change in a low level entity, for example modifying a resource, needs a system shut down and a complete update of data structures of all high level entities.
- An unpredictable modification is practically impossible
- Resource allocation is done by prediction at higher levels. Sometimes the generated schedule becomes invalid before it reaches to operational levels because of disturbances.
- And for last hierarchical control architectures introduce a top down development methodology which brings more constraints to the system.

Because of these problems hierarchical control developed in various ways. The new hierarchical control systems are known as modified hierarchical systems where the rigid master-slave relationship is more relaxed and modules can communicate peer-to-peer. Through usage of these systems low level entities can communicate and synchronize their processes with a certain degree. But this improvement did not cancel communication problems since response times to disturbances are still not sufficient in modified hierarchical systems (Balasubramanian *et. al.* 2001).

2.2. Heterarchical Control Architectures

With recent developments in manufacturing systems, today's manufacturing environments become a whole consists of loosely coupled operation units which are dis-

tributed, autonomous and experts of their operations and capable of handling their problems. Thus centralized architectures are not seen as valid approaches for control (Anussornnitisarn *et. al.* 2005). To cope with these requirements and problems of the distributed manufacturing environment heterarchical control systems or so called distributed control architectures are developed. In heterarchical shop-floor control systems entities, which are called agents, are highly distributed in other words there is not a central controller in the system. These agents are autonomous and co-operating entities unaffected from central or outside control. In distributed systems system goals are reached with co-operation and communication which is free between agents. Introducing heterarchy in shop floor control brought improvement in response times to disturbances. But heterarchical control systems tend to have their own problems such as being unpredictable and lack of global optimum because of their distributed nature (Balasubramanian *et. al.* 2001).

The rationale behind the co-operation approach used in heterarchical control is realizing the uncontrollable nature of complex systems. In stead of directly controlling the system activities in the manufacturing environment, these activities are carried out by co-operation between interacting sub-systems which can have different motives and objectives (Yuh *et. al.* 1992).

The distribution of entities in a heterarchical architecture is enabled with agents. An agent can be thought as a software program that can perform specific tasks and interact with its environment. Agents have autonomy and intelligence to a degree. The interaction can be defined as communication between other agents, data collection from environment; whereas intelligence can be thought as making decisions using the information gathered by interaction and ability to change environment. Entities in a manufacturing environment are involved with physical components which executes manufacturing tasks. Thus agents in a manufacturing environment also control a physical device (Kotak *et. al.* 2003, Usher 2003).

There are various new heterarchical control architectures like genetic manufacturing, fractal manufacturing or holonic manufacturing. In this study holonic man-

ufacturing is used as heterarchical control system, because of its agent based and object oriented structure. The Contract-Net Protocol is the general method used in distributed systems for resource allocation and communication. Below these two important concepts are explained.

2.2.1. Holonic Manufacturing System

Holonic Manufacturing System (HMS) is a distributed shop-floor control system. HMS is one of the major projects of Intelligent Manufacturing System Program which is based on the concept of holons. The word holon is first used by Arthur Koestler, a Hungarian author and philosopher. It is constructed with the Greek words holo (means whole) and suffix -on (means part). Thus holon means a whole that is a part of a bigger whole. In an HMS holons are self-contained wholes to their sub-wholes and are dependent parts when looked from reverse angle simultaneously (Bomgaerts *et. al.* 2000).

It will be appropriate to define some basic concepts in HMS (Brussel *et. al.* 1998);

- *Holon:* An autonomous and cooperative building block of a manufacturing system for transforming, transporting, storing and/or validating information and physical objects. A holon consists of an information processing part and often a physical processing part. A holon can be part of another holon.
- *Autonomy:* The capability of an entity to create and control the execution of its own plans and/or strategies.
- *Cooperation:* A process where by a set of entities develops mutually acceptable plans and executes them.
- *Holarchy:* A system of holons that can cooperate to achieve a goal or an objective. The holarchy defines the basic rules for cooperation of the holons thereby limits their autonomy.

The definition of HMS is built on the concept of holarchy; HMS is a holarchy that integrates the entire range of manufacturing activities from order booking through

design, production and marketing; to realize the agile manufacturing enterprise. Holons in a holarchy do not loose their basic features thus, holons do not rely on proper operation of each holon in the holarchy to get their work done. In addition to this holons can belong to multiple holarchies and they can form temporary holarchies (Brussel *et. al.* 1998).

Each individual holons has at least two parts. One of them is the functional component which corresponds to the physical entity of the holon. It can be pure software or a physical device with its information component (driver). The information component of the functional component of a holon has mechanisms for managing the physical component. Communication and coordination component of the holon is usually called agent software. The agent software has decision making capabilities, data base modules and mechanisms for communication with other holons. Also driver and agent software can be thought as information component of the holon. In Figure.3 a general components of a holon can be seen (Brussel *et. al.* 1998, Gou *et.al* 1998).

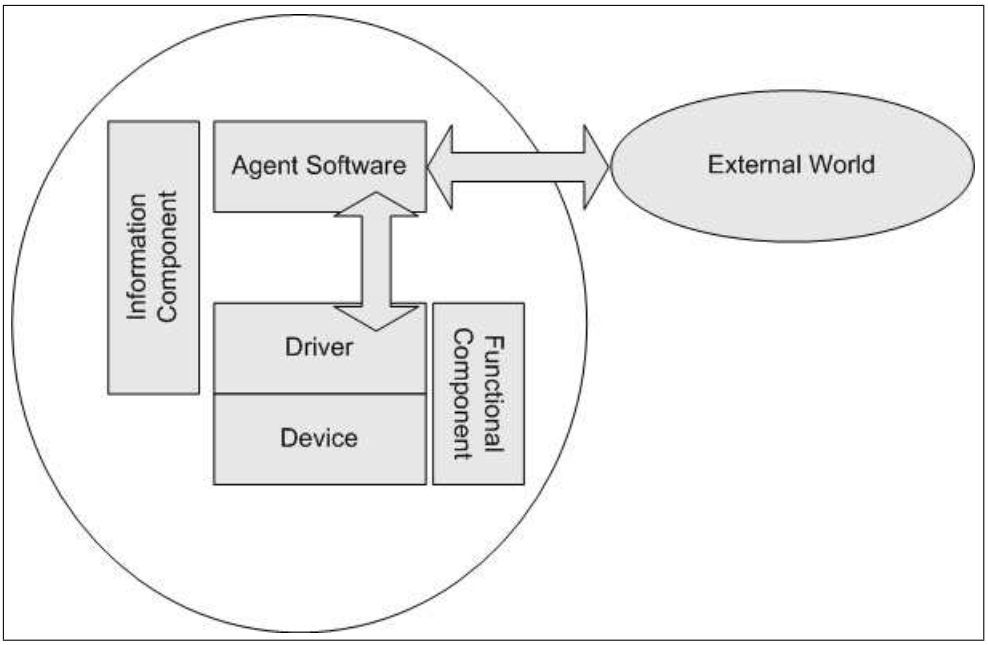

Figure 2.2. Basic structure of a holon

HMS is a new concept in heterarchical control. Thus as Babiceanu and Chen (2005) stated, there are not many studies about HMS which can be called control architectures. Product-Resource-Order-Staff Architecture (PROSA) is mentioned as an

applicable architecture in literature. The basic idea of PROSA comes from categorizing relatively independent manufacturing activities. These are; product and process related concerns which can be thought as product design, process planning and quality aspects; resource related concerns such as proper allocation of the resources, driving the production and logistical aspects such as due dates. All these three main and relatively independent manufacturing activities form the basic holons of HMS according to PROSA namely; product holon, resource holon and order holon (Figure 2.3). According to the developers of PROSA these basic holons forms a necessary and sufficient set to control a flexible manufacturing system (Brussel *et. al.* 1998).

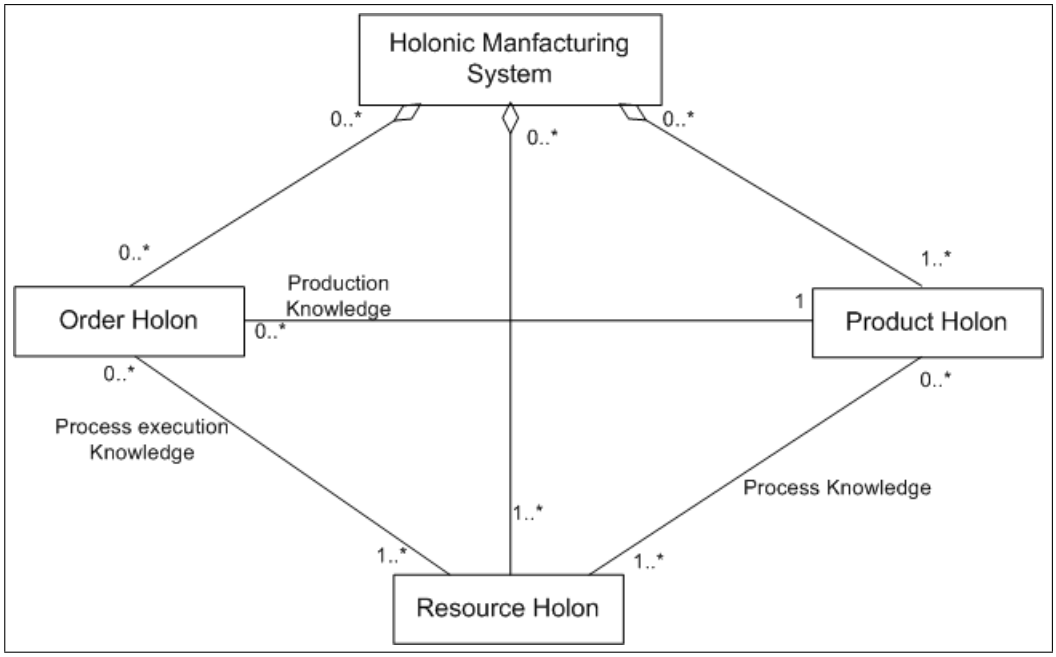

Figure 2.3. Basic blocks of HMS according to PROSA and their relations

2.2.2. Negotiation Concept

In distributed systems there are some approaches for driving production in the manufacturing system. All these approaches are based on Smith's Contract-Net Protocol which is an infra-structure for communication and co-operation between individual agents. The protocol has a set of agents that negotiate with each other to execute certain tasks. Each agent has one of the roles that are necessary to execute a task which are the manager or the contractor. Contractor executes the task and manager controls execution of the task and interprets the results of the execution. Here contract

is the agreement between manager that gives task and contractor that offers processing for the task (Veeramani *et. al.* 1998).

These messaging structures are generally called auction-based protocols and are the most used distributed control approaches. An auction-based protocol can be part initiated or resource initiated. In part initiated protocols parts have the role of the manager and they choose the winner contractor from resources whereas in resource initiated protocols resource choose which task to allocate. Steps of a part initiated auction-based protocol are given below (Figure 2.4) (Siwamogsarham and Saygın 2004, Veeramani *et. al.* 1998).

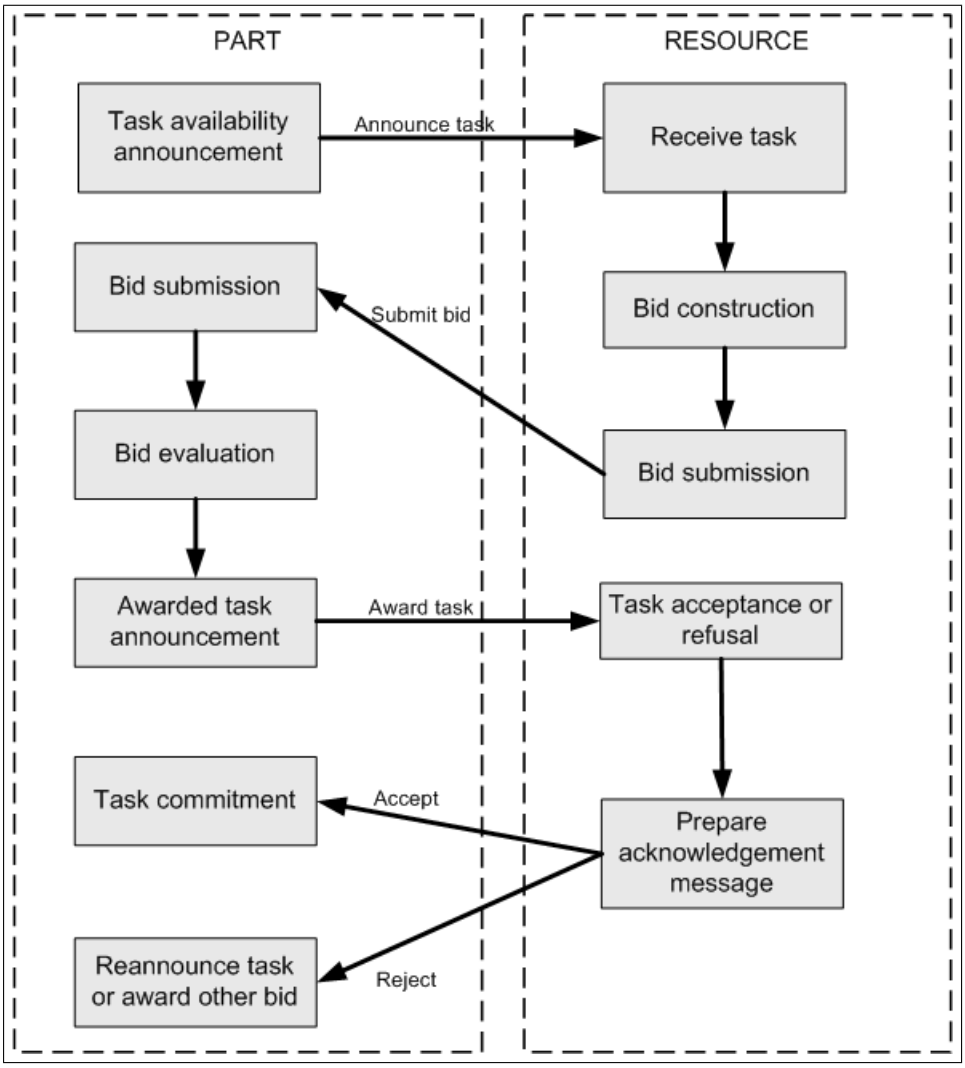

Figure 2.4. Basic steps of a negotiation procedure

- *Task Announcement:* When an order in the system has a task to be executed it announces the task to all resources.
- *Bid Submission:* Each resource that can execute the task prepares a bid and submits it to part
- *Bid Evaluation:* The part evaluates the bids and allocates the winner resource.
- *Contract Confirmation:* Winner resource gets the award and sends a confirmation message to part.

Bid and offer submission and evaluation issues are important concepts of auction-based protocols. The algorithms of bid preparation and evaluation are the key elements for the orchestrating the distributed system and directly affect system performance. There are various formulations in literature to have an efficient distributed control algorithm. Some examples of these algorithms are given below.

Siwamogsarham and Saygın (2004) proposed an auction based shop floor control model similar to model developed by MacChialori and Riemma (2002). According to their base model the auction is structured as a multidimensional auction model. The auction does not have to conclude to an agreement in the first auction round; in a case of disagreement the model iterates for an agreement in the following auction rounds. The proposed auction protocol is a resource initiated model. The basic structure of the protocol is given below in Figure 2.5 (Siwamogsarham and Saygın 2004);

Proposal formulation of the model for part i to resource j is built as follows;

$$Prop_{ij} = D_{ij} \times (1 + \alpha \times N_{ij}) \tag{2.1}$$

where

N_{ij}: Negotiation round, initial value is 0 and has a limit of N_{max}.

α: The increment ratio of $Prop_{ij}$ when previous negotiation rounds did not end up with an agreement.

D_j: Negotiation coefficient which assumes values between 0 and 1.

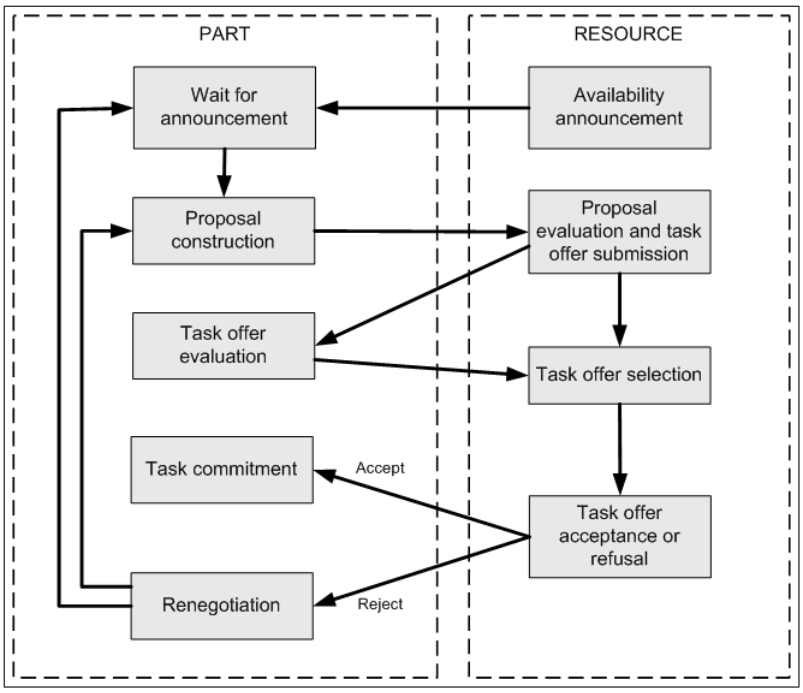

Figure 2.5. Basic structure of the proposed negotiation protocol

Offer formulation is similar to proposal formulation; it is iterative and parametric. Offer formulation depends on the number of proposals, number of resources and the availability of resources(Siwamogsarham and Saygın 2004).

After part agents evaluate offers received from resources and made the selection of the best resource according to the due date constraint; two possible outcomes can occur. First, more resources in the system can complete the operation within the due date. Then the part agent will select the resource which has the lowest offer value, if the offer does not exceeds the threshold value of accepting an offer without a renegotiation. On the other hand, if a resource offer is accepted by only one part a task agreement occurs. If there is not an acceptance to offer then a re-negotiation occurs until maximum number of re-negotiation rounds is reached. In the case of acceptance of a resource offer from more than one parts, then resource selects the part which has shortest completion time to become available as fast as possible (Siwamogsarham and Saygın 2004).

Siwamogsarham and Saygın (2004) used a combination of routing selection rules and part selection rules for benchmarking the proposed model and constructed a cen-

tralized simulation model in a single processor and tested the proposed model in a specific layout and processing - routing data. The results showed that proposed auction model outperformed other algorithms in most of the cases according to these measures; average tardiness, average lateness, average due date deviation, utilization balance, average throughput and average wait time.

McDonnel *et. al.* (1999) proposed an auction protocol to integrate process planning and distributed shop floor control. In their paper in stead of proposing a biding algorithm, authors introduced a framework for integrating process planning with auction based heterarchical shop floor control. According to authors with technological developments in manufacturing systems, an independent process planning would not be sufficient. The problem definition shifts from determining process routes for jobs to dynamically configuring the process plans by considering technological requirements. In the proposed infrastructure there are three agents. First one is part manager agents which coordinate resource agents to execute production tasks by allocating necessary resource agents and making feasible process plans. The second one is resource agents which schedule production on the machine and manage upkeep and support of the resource. The last type of agents are secondary resource agents which are responsible for supporting other agents. These agents can be material handling agents or tool manager agents.

The proposed auction protocol includes a precondition idea to every manufacturing task. As an example consider a task announcement of a certain part agent at location A for a specific operation. To this announcement a machine agent M submits a conditional bid with a pre-condition of delivery of part from A to C. The part agent accepts this conditional bid and announces a task for precondition. Let's assume a conditional bid for this task is proposed by handling agent H with a pre-condition of delivery of part from A to B and part agent accepts it. Then in the following auction round the part agent announces a task for precondition of the previous task which is delivery from A to B. And for this task, assume that a proposal is submitted by robot agent R and part agent accepted this. With this final round, all preconditions of first

task is fulfilled. With this cascading structure the resource agent decomposes original task to sub-tasks based on requirements of task (McDonnel *et. al.* 1999).

Dewan and Joshi (2002) developed an integer programming formulation for the job shop and used this formulation for bid construction and task submission. The objective of the formulation is minimizing the penalty for completing the last operation for all jobs in period k. The formulation has constraints to force a job to be completed in one period. Addition to this, operations of a job has a set of knapsack constraint to attain technological precedence constraints. Also there are constraints to ensure that machines can have only one operation during a period k.

The new formulation set does not give strait forward structures for bid construction. To achieve this, authors used Lagrangian Relaxation method for machine capacity constraints. The obtained Lagrangian Dual is decomposable into sub-problems for each job and machine. With the relaxed formulation, the objective and constraint sets related with jobs of the problem becomes separable for all jobs. When the Lagrangian multipliers are given, sub-problems can be solved independently. Thus the optimal solution for the problem can be achieved by choosing the best Lagrangien multipliers (Dewan and Joshi 2002).

In the proposed model the goal of each agent is maximizing their utilities. For a job agent, utility is minimizing the cost of resources and for a resource agent, utility is maximizing the revenue earned from the job agents. The bid submitted by the job agent is composed of the time slots desired from a resource and the amount that job is willing to pay to a resource, which is the minimum cost computed as the objective of the part sub-problem. The Lagrangien multiplier is adjusted at each auction round by machines according to the desire of jobs to resources. The auction helps to tune the Lagrangien multipliers; not too high to cause all parts to avoid the resource or too low to prevent maximizing the revenue of resources (Dewan and Joshi 2002).

To test the proposed structure authors coded algorithm in Borland C++ and used a single computer configuration and a multi-computer configuration to run the

scenarios. For comparison they considered a problem set with 12 different configurations. The results showed that proposed algorithm outperformed classical algorithms from three to 118 percent (Dewan and Joshi 2002).

MacChialori and Riemma (2002) proposed a classical negotiation structure which has a different point of view in proposal construction and evaluation. In their proposed structure, authors used a resource initiated and resource finalized methodology. The process advances as follows; first when a resource agent becomes available it announces its availability to all resources. Then Part agents answer this announcement with by preparing and sending proposals. After this, resource agent evaluates proposals and send task offer to related part agents. Then part agents answer this task offer and a commitment between the part and the resource agent is made by the judgment of the resource agent.

The proposed negotiation structure has the following issues to be considered. First, a budget is assigned to all part agents. This budget is constructed with consideration of the routing flexibility of the manufacturing layout. Thus the budget of a part agent is the maximum cost value it can take which occurs with selection of the most costly resources in the process route of the job. This value then increased according to the slack time, the penalty and the number of operations of the job up to twice of initial budget with consideration of other parts agents' budget attributes. In addition to this, proposed negotiation structure has a different proposal formulation. The formulation of a proposal consists of standard unit cost per time, number of negotiation rounds with no success, the position of the operation in the process plan (if the operation at consideration is in the early stages of the process plan then the part can be subject to critical issues thus is more eager for a commitment) and the slack time of the part and the production resource efficiency (MacChialori and Riemma 2002).

The next issue is the offer formulation of the negotiation structure which is developed according to condition based rules as given in the following content. First if only one proposal received by an idle resource then the offer is equal to proposal and if the resource is busy then offer is equal to maximum of resource cost and proposal values.

Second if several proposals are received by an idle resource then offer is constructed by a formulation which will force the part agents to increase their bids. Thus only urgent part agents will agree to increase their bids. And for the last possibility, if several proposals are received by a busy resource then the offer formulation will give smaller offers to attract parts that have longer slack times (MacChialori and Riemma 2002).

The last issue is the offer evaluation and selection where a selection by part agents for resources is occurred according the due date constraint. Here there are two possibilities. First if a part agent receives offers from resources that can not complete operation within the due date, then the part agent will accept the offer of the resource which can complete the job with the lowest completion time. If more resources are able to guarantee the completion of the operation within the due date, then the lowest offer will be accepted with the consideration of a threshold value which a part accepts an offer with no renegotiation (MacChialori and Riemma 2002).

To test the proposed model authors used a layout with job routing flexibility and a certain number of machines with processing flexibility. The test bed for the comparison is not mentioned. The assumptions of the test are; deterministic process times, negligible transportation times and stacked process and set-up times. For comparison shortest processing time, earliest due date, minimum slack time and critical ratio dispatching rules are used. The performance measures for comparison are mean tardiness, number of late jobs, mean lateness, maximum tardiness and penalty. According to the test results, proposed negotiation structure gave good overall performance and is very competitive. The most significant result is in penalty measure where negotiation structure outperformed other algorithms. Note that penalty measure is more effective then late jobs measure because it includes the importance of the jobs with respect to a given parameter (MacChialori and Riemma 2002).

3. PARALLEL AND DISTRIBUTED SIMULATION

3.1. Basic Concepts in Simulation

Simulation is an efficient tool used for modeling, analyzing and design of systems which enables testing different issues of the system. Classification of simulation models can change from point of view of the analyzer. But for the sake of our study, it is convenient to divide simulation models into two main categories according to the changes in the system states, namely discrete event simulation and continuous simulation. Manufacturing systems are usually modeled as discrete event systems which will be the approach in our study. A further classification in discrete event simulation can be done according to the progress of simulation, time-stepped and event driven. In event driven simulation entities executes a chain of events that are stored in an event list. In event driven simulation, system can and will advance, with different time intervals. In time-stepped simulation, system time advances with constant steps and states of variables changes if there is an event that changes the state at current simulation time (Cho 2005).The constant time step, in time stepped simulation is called time scale factor and it is an important parameter in discrete event time stepped simulation. In Figure 3.1 a general classification of simulation approaches is given.

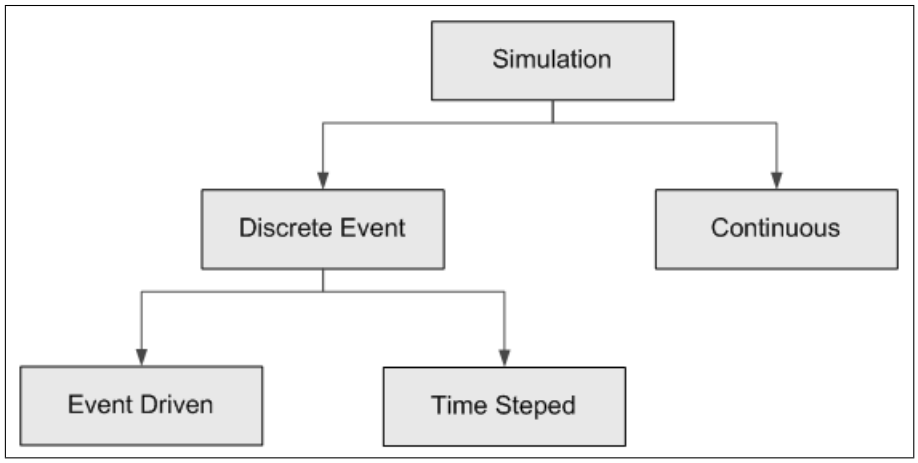

Figure 3.1. General classification of simulation approaches

As mentioned above, time concept is essential for time driven discrete event simulation models. First there are several different notions of time that must be clarified. Simulation time is an instant of time that is modeled in a simulation run. Physical time is the modeled period of time in the simulation software which can advance in real-time or a scaled factor of the real time. Wall clock time is a clock that simply represents the real-time. To give an example consider of a simulation of a manufacturing system which will be used for making production plan of the next week and the current date is 01.01.2006, 09:30. Then the simulation time extends from 01.01.2006 to 08.01.2006. Simulation time might be represented in the simulation software by a double precision floating point number with each unit corresponding to an hour. In this case, simulation time advances by an hour with each time tick of the program. If the simulation run lasts for 10 minutes then the wall clock time will extend from 09:30 to 09:40 in that day. As it can be seen in the example there is a relationship between wall clock time and simulation time via usage of time scales (Fujimato 2000).

With the time scale factor, simulation time can advance slower or faster than the wall clock time. For example the scaling factor can be arranged in such a way that the simulation advances ten minutes in every second of wall clock time. Similarly simulation can be slowed with an appropriate scale factor. Real time simulations are a special case of time scaled simulations with a scale factor of one. The relation between simulation time and wall clock time can be expressed by scaling factor with this general function (Fujimato 2000).

$$TS = TS(0) + Scale \times (TW - TW(0))$$

where

TS: Simulation time,

TS(0): Simulation time at which the simulation starts,

TW: Wall clock time,

TW(0): Wall clock time at which simulation starts.

3.2. Parallel and Distributed Simulation Approach

A computer simulation can be executed in a centralized computer or in parallel/distributed computers. This distinction gives us another classification for simulation models, independent from others, namely centralized or parallel and distributed simulation (P/DS) approaches. A formal definition for a distributed simulation can be given as; a technology that simulation can be executed in different computers which are connected with a communication infra-structure. There are certain benefits of distributed simulation and parallel simulation, which can be stated as follows (Fujimato 2000).

- *Reduced execution time:* By simply subdividing computational workload to different processors execution time of the simulation can be reduced. However some processors can have very low utilization in simulation run which can be seen as a problem.
- *Geographic distribution:* With network technologies, the simulation can be executed in geographically distributed computers, this provides opportunities to create virtual worlds.
- *Integrating simulators that executes on machines from different manufacturers:* Different simulators can be coordinated with convenient communication mechanisms thus existing simulators can be used in a distributed way.
- *Fault tolerance:* If a processor goes down in a parallel/distributed simulation other processors can take the workload of that processor and the system can execute normally.

In a distributed and parallel simulation environment the physical system is viewed as a set of physical processors that somehow interact. These physical processors are modeled as logical processor (LP) units in parallel and distributed simulation. The interaction between LPs is achieved by exchanging time-stamped messages between themselves. The processes performed by LPs can be stated as follows; computation executions that change state variables or generate new events for itself or for other LPs in the system and communication with other LPs in the system. This modeling approach of a system would perfectly match with the core logic of parallel and distributed

simulation with defining LPs for every physical process in the system and enabling each LP to execute its events according to the event list and exchange messages with other LPs. But the application of these paradigms brings some problems(Fujimato 2000). In the simulation model, each LP must execute all of its events that are generated by itself and other LPs in a time-stamped order. If this simple rule is violated, then an event can effect another event in its past, which is absolutely unwanted. The error in the system resulting from this behavior is known as *causality error* (CE), and the general problem of forcing the events to be executed in the time-stamped order is called *synchronization problem* in parallel and distributed simulation literature.

There are two general approaches for overcoming synchronization problem in distributed and parallel simulation, each of which has a completely different logic. The first method is known as conservative approach. In conservative algorithms the aim is to prevent occurrence of any causality errors in the system. The other method is called optimistic approach. Different from conservative approach, in optimistic algorithms, the system does not try to prevent occurrence of causality errors but terminates their effects on the system with a mechanism of system roll back, after they are detected (Fujimato 2000). For the sake of this study, conservative approaches will be explained in further detail.

Fujimato (2000) stated two basic conservative synchronization algorithms. The first one enables synchronized execution by exchanging time-stamped safe guard messages between LPs to ensure each LP that, it is safe to execute its computation. This algorithm is suitable for event driven simulations where the exchanged messages include next event time-stamps of the LPs.

The other conservative algorithm uses a barrier to synchronize LPs. The barrier is defined as a specific point in time when all LPs in the system stops and waits to reach other LPs to this barrier. As it can be seen in Figure 3.2 when an LP reaches the barrier primitive it stops the execution and waits for the other LPs to reach the barrier. When all LPs in the system reach the barrier the execution is resumed from the barrier primitive point of time (Fujimato 2000).

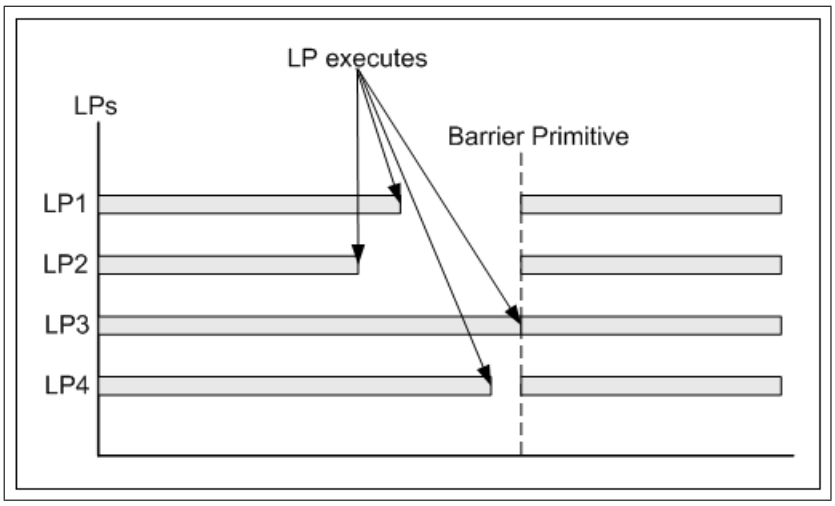

Figure 3.2. Synchronization using barrier primitive

There are two main problems that must be considered for using barrier primitive for synchronous execution. The first one is enabling the barrier logic with a messaging mechanism for every LP in the system. In other words there must be a barrier controller to block LPs in the barrier primitive and release them when all LPs have reached the barrier primitive. The other issue that must be resolved is preventing occurrence of any transient messages in the network when LPs are released from barrier primitive (Fujimato 2000).

Fujimato (2000) gives a simple solution for barrier primitive application by creating a centralized LP that controls barrier primitive. When one of the LPs reaches to barrier primitive, it sends a synchronization message to barrier controller and stops its execution. The barrier controller collects all these synchronization messages and sends a release messages to all LPs when it can confirm that all the LPs reached the barrier primitive. Then all LPs resume their execution until they reach the next barrier primitive. In this approach the apparent drawback is the computation load of the barrier controller which must send *N-1* messages in every barrier primitive.

Another approach for implementing barrier primitive stated by Fujimato (2000) is the tree barrier. In tree barrier LPs are organized as a balanced tree where each node represents an LP in the system. When a leaf LP reaches the barrier primitive, it sends a synchronization message to its parent. After all child nodes of an interior node

reach the barrier, the interior node sends a message to its parent, as soon as it reaches to barrier primitive. Finally when the root element receives synchronization messages from all its child nodes it sends a release message to all LPs in the system by reversing the follow of messages used in reaching barrier primitive. This procedure needs *2(N-1)* messages to be sent and received. In Figure 3.3 an organization for a tree barrier can be seen.

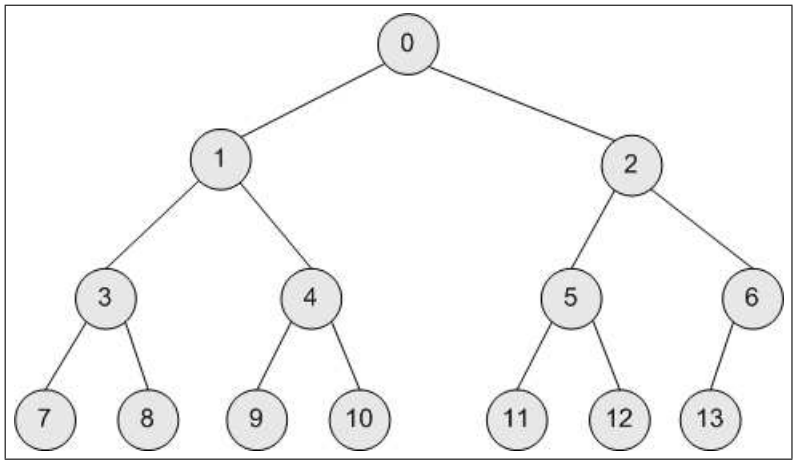

Figure 3.3. A binary tree organization of LPs to implement barrier primitive

The last algorithm for barrier primitive approach is the butterfly barrier. Butterfly barrier has a similar structure as tree barrier such as a series of messages are interchanged between LPs to notify that all of the LPs reached the barrier primitive. The methodology is applied as follows. In the first round only the LPs whose identity numbers only differ in the last significant digit of the binary representation. For example LP_0 *(000)* and LP_1 *(001)* perform a pair wise barrier. In the second round the procedure is applied with the same rule except between LPs whose identity numbers only differ in the second last significant digit. Broadly speaking in round k LP_i synchronize with all the LPs whose addresses differ in only the k_{th} bit. The barrier is released when all $logN$ pair wise barriers are completed. In Figure 3.4 an example of butterfly barrier is given. The highlighted nodes and arches in the figure show the progress of an LP in the algorithm. Note that in the last round the example LP exchanged synchronization messages with all other LPs (Fujimato 2000).

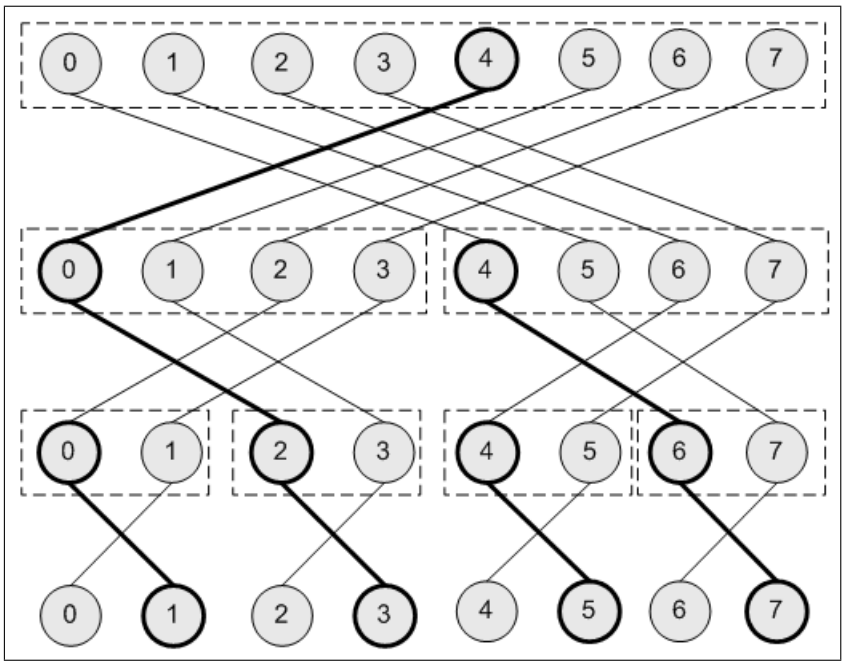

Figure 3.4. A butterfly barrier application example to implement barrier primitive

3.3. Advantages of Distributed and Parallel Simulation

The benefits of a distributed simulation model, as far as the problem domain in this thesis is concerned, can be summarized as follows. First distributed simulation approach is suitable for modular model building such that the distributed LPs in a simulation model, even LPs in different models can share modules. To give an example, think of a manufacturing cell. The coding of a virtual cell would be different for different architectures. But with a modular approach, the shared entities of cells i.e. a robot can be used by two different implementation when they are designed as modules. This will have various advantages. First, shared modules will be coded only once. Second modular approach will provide an easy progression from simulation software to SFC software by replacing virtual modules with physical ones and also real time software of entities would be used for building virtual modules. Third, simulation software will be a plug and play tool for testing different systems by changing configuration of modules. Again thinking of a manufacturing system, this approach will enable adding multiple AGVs or additional cells with different structures to the system by only adjusting parameters of the modules. And last advantage is that, model complexity will be reduced because the system will be divided into separate modules.

In addition, with P/DS approach, some aspects of distributed systems would be taken into account easily which can be very difficult in a conventional simulation. For example modeling of a communication system between system entities would not be sufficient with respect to communication architecture in a centralized simulation. With distributed simulation, the real communication system can be used without modeling whereas other aspects of the entities are virtually reflected in to the model. This enables to capture dynamics of communication in the system which could be an important cause of complexity in the system.

4. BUFAIM MODEL FACTORY

Boğaziçi University Flexible Automation and Intelligent Manufacturing Laboratory (BUFAIM) model factory is founded in 2001 to research new paradigms in flexible and intelligent manufacturing systems. Research activities of BUFAIM are concentrated on manufacturing planning and control, flexible automation concepts and their integration issues (Bilge *et. al.* 2002). In this chapter general information about BUFAIM model factory will be given. This description will in fact constitute the problem definition for this thesis as it provides the basics for the FMS under consideration on both hardware and software aspects. First model factory layout will be introduced. Then the infrastructure of the implementations in BUFAIM model factory, FMS.NET will be briefly explained. Lastly the control architectures implemented in the model factory will be descried since they are the basis for the simulation applications developed in this thesis.

4.1. Model Factory Layout and Hardware

BUFAIM Model Factory (Figure 4.1) is configured to be an infrastructure for the research topics of flexible manufacturing system. In Model Factory there are three cells and an automated storage and retrieval system (AS/RS). Various part types each with alternative process plans can be processed in the system. Parts enter the system from the AS/RS and processed in the cells according to their specific process requirements. The CNC modules of the cells are virtually modeled by a movie. The material handling between the cells is enable with two automated material handling devices. The first material handling device is a free-ranging automated guided vehicle (AGV) with multiple loading capacity. The AGV travels on the pre-determined routes between cells. The other material handling device is a PLC controlled closed loop conveyor which serves two cells in the model factory. The loading and unloading tasks and intra-cell material handling tasks of the cells are handled by a robot in each cell. With this integrated automated material system different modes of material transfer is enabled in the model factory (Bilge *et. al.* 2002).

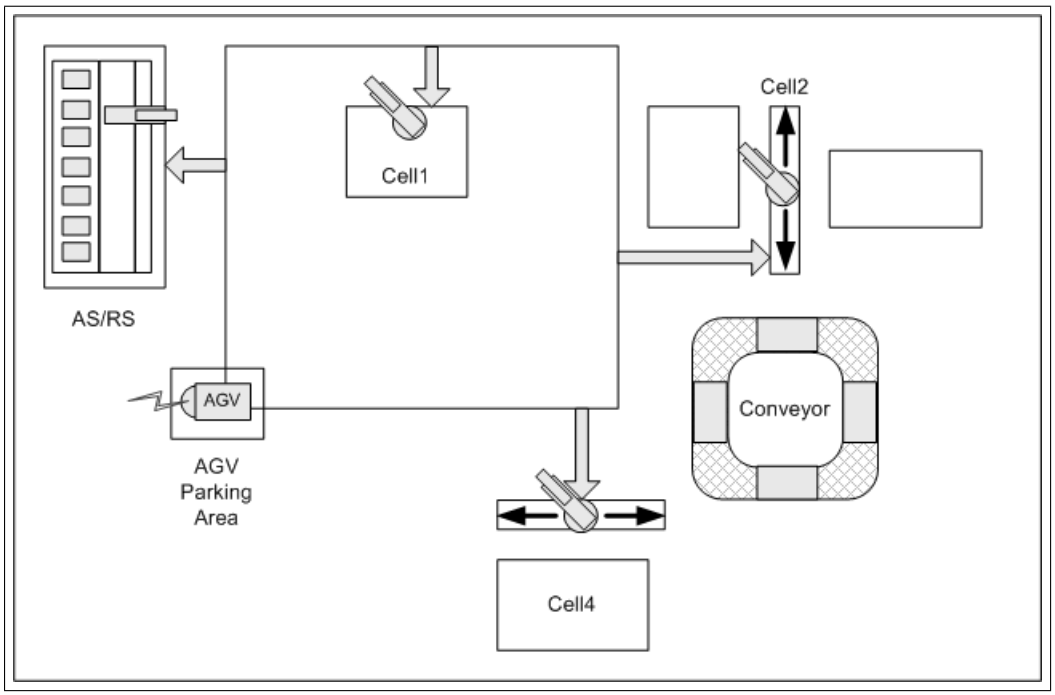

Figure 4.1. BUFAIM Model Factory layout

Entities in the Model Factory are controlled by inter-connected computers via a local area network. The network is closed to outside to increase connection speed between cell controllers. Cell controllers exchange messages through this network infrastructure to enable shop floor control. AGV is controlled by a controller with a different structure in the model factory. The messages are exchanged between AGV and AGV controller via a radio frequency receiver and transmitter located at both of them. Every other device controller in the Model Factory uses RS232 protocol for communication.

4.2. FMS.NET: A Complete Infrastructure for Modeling Flexible Manufacturing Systems

Gönen (2005) developed an object oriented infrastructure for developing FMS software for BUFAIM which is called FMS.NET. FMS.NET is built using BUILD.NET, an application generator for object oriented software. FMS.NET is a namespace using C♯ software environment, consists of all the necessary operational and decisional classes for an FMS. The software is built in an object oriented fashion, thus it can be used to develop any software application related with FMS. In Table 4.1 the layers of the

FMS.NET which are structured as different namespaces and their brief explanations are given.

Table 4.1. FMS.NET namespaces

Namespace Name	Explanation
FMS.NET	Contains basic object definitions
FMS.NET.Decision	Contains operation decision implementations
FMS.NET.Layout	Contains static model data objects
FMS.NET.Operation	Contains dynamic model data objects and simulation related objects
FMS.NET.Random	Contains random variate generators

4.2.1. FMS.NET.Random Namespace

FMS.NET.Random namespace enables various random number generators for the application developer. The generators use streams that are far apart from each other to generate independent random numbers in every seed (Gönen 2005).

4.2.2. FMS.NET Namespace

FMS.NET namespace has the necessary classes for a FMS simulation. FMS.NET namespace consists of the *Statistics, FMSObject, StaticObject, MovableObject* classes. *Statistics* object enables collecting necessary performance measures from the system such as mean, minimum value, maximum value or standard deviation. In *Statistics* object three different data types can be stored. These are count based statistics (submitted job count, completed job count, etc), average based statistics (average flow time, average weighted tardiness) and time weighted average based statistics (average queue length, average waiting time in queue). *FMSObject* is the base class of all FMS.NET objects. It has main properties like *Name* and *Statistics* dictionary. *StaticObject* and *MovableObject* classes are derived from *FMSObject* and represents static objects in FMS like work centers, work center buffers or lanes and AGVs or unit loads respectively (Gönen 2005).

4.2.3. FMS.NET.Layout Namespace

FMS.NET.Layout namespace represents the layout of the FMS with storing static and dynamic information about the layout of the FMS. Classes in the FMS.NET.Layout namespace is given in Table 4.2. *Lane* object represents flow paths in the layout and has necessary data like length and starting and ending nodes whereas *Node* object stands for the intersection points of lanes. Transfer nodes in the layout are the intersection points of AGVs and work center buffers. *Operation* class has the alternate work centers list and processing time information of the operations in specific work centers. *Queue* class represents the basic input and output buffer in a work center. *WorkCenter* object stands for a work center in a FMS. *WorkCenter* class has some static data like number and type of the *Processors* and dynamic data like *unitloads* (Gönen 2005).

Table 4.2. FMS.NET.Layout namespaces classes

Class Name	Base Class
Agv	MovableObject
Lane	StaticObject
Layout	FMSObject
Node	StaticObject
Operation	FMSObject
Queue	StaticObject
Route	FMSObject
WorkCenter	StaticObject

4.2.4. FMS.NET.Decision Namespace

FMS.NET.Decision namespace is stands for all the necessary decision algorithms in FMS. There are seven classes in the namespace each of which represents a decision algorithm set and has various algorithms. In Table 4.3 decision algorithms and their corresponding classes are given (Gönen 2005).

Table 4.3. Algorithms in FMS.NET.Decision namespace

Decision Class	Explanation
BlockageSolving (No Action, Output Buffer Availability)	Decides what to do when an AGV becomes blocked in front of a full input buffer
Matching (Nearest Pickup, Minimum Remaining Output Queue Space, Random)	Assignment of eligible AGVs to parts waiting at output buffer
Dispatching (Nearest Active Station, Random)	Giving AGV its next destination
Routing (Shortest Path, Shortest Path Without Blockage, Random)	Selecting the route AGV should take to reach its next destination
TrafficManagement (First Come First Served, Random)	Prioritizing the AGVs which jammed at a junction
OperationDecision (Earliest Expected Finish Time, Smallest Queue Workload, Random)	For a part having a flexible process plan, when an operation is completed a work station/operation pair is selected as the next processing step
ProcessOrder (First Come First Served, Shortest Process Time, Random)	Selects a part waiting in the input buffer to be processed next by a work center

4.2.5. FMS.NET.Operation Namespace

FMS.NET.Operation namespace has static and dynamic data about a job like routes, definitions and unit loads in the system respectively. The classes of namespace is given in Table 4.4. *Job* object stands for orders submitted to the system which can be consist of multiple unit loads. *Job* class has also static and dynamic data like *JobDefinition* which is a data structure to define a job type in the system and *Unitloads*. *JobMix* object represents the collection of the jobs in the system that can be produced. *JobRoute* class is a data structure which stores information about alternative routings for a job. *Unitload* class stands for the single entities in the system that are processed (Gönen 2005).

Table 4.4. Job-related classes in FMS.NET.Operation namespace

Class Name	Base Class
Job	FMSObject
JobDefinition	FMSObject
JobMix	FMSObject
JobRoute	FMSObject
Unitload	MovableObject

4.3. Control Architectures in BUFAIM

Real-time management of orders and resources on the shop floor to execute a short term production plan is called Shop Floor Control (SFC). A SFC system should be developed such a way that, a range of products (with relatively low quantities) can be processed in a responsive manner with considering some performance measures like lead times or weighted tardiness cost. In BUFAIM model factory two different Shop Floor Control Architectures (SFCA) are implemented in a modular, reusable and extensible fashion to allow future developments (Bilge *et. al.* 2002). In the next sections these implementations, namely, layered and distributed SFC systems are explained and compared based on implementation aspects.

4.3.1. Layered SFCA Application in BUFAIM

The layered control architecture implementation in BUFAIM Model Factory is based on a modified hierarchical control architecture and developed in C♯ programing language with using FMS.NET infrastructure. The proposed control architecture has three levels; shop, cell and equipment (Figure 4.2)(Bilge *et. al.* 2002).

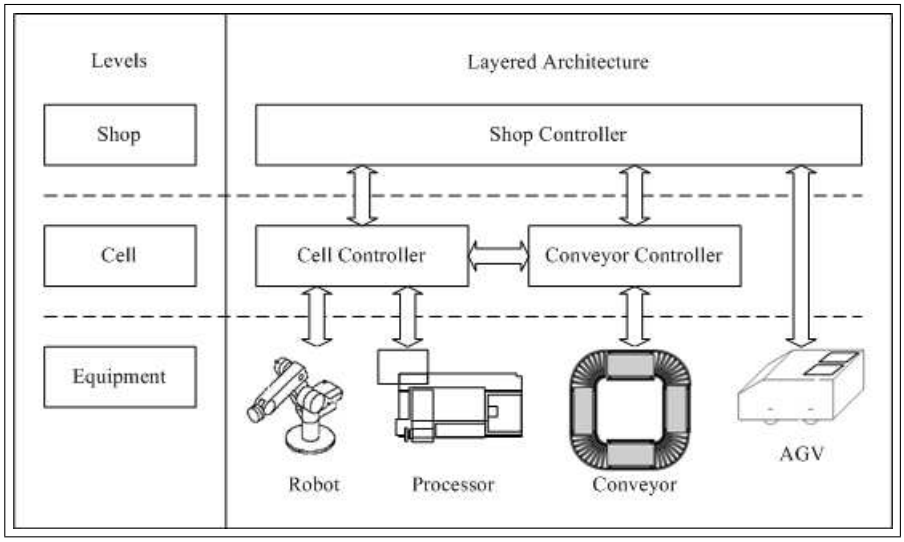

Figure 4.2. Structure of the layered SFCA implementation

In the proposed implementation all resource allocation (i.e. dispatching jobs to alternate cells, matching AGVs and decisions for transfer tasks) and AGV management decisions (i. e. dispatching jobs to AGVs, traffic management) fall into the domain of the Shop Controller (SC). All decisions in SC is made according to the selected algorithm set from the decision algorithm library. The operations of a job and material handling tasks between cells are executed according to these selected algorithms. The Cell Controllers (CC) are responsible of managing operational activities within the cell (i.e. robot transfer tasks, part sequencing) independent from the SC. The communication between shop floor entities is enabled via an TCP/IP network except AGV which communicate by using radio frequency signals. The layout of the manufacturing system and job mix is loaded to the system in initialization and the layout configuration and job mix can be changed easily for experimental purposes (Bilge *et. al.* 2002).

4.3.2. Distributed SFCA Application in BUFAIM

The distributed SFCA application in BUFAIM Model Factory is based on the PROSA architecture. In the proposed application all the cell controllers which are called holons, are at the same level and there is not a higher level controller in the system which forces commands to cell controllers. The real-time control and resource allocation is achieved by bidding procedures between holons in the system (Bilge *et. al.* 2002). In Figure 4.3 the general responsibilities and functions of a resource holon can be seen.

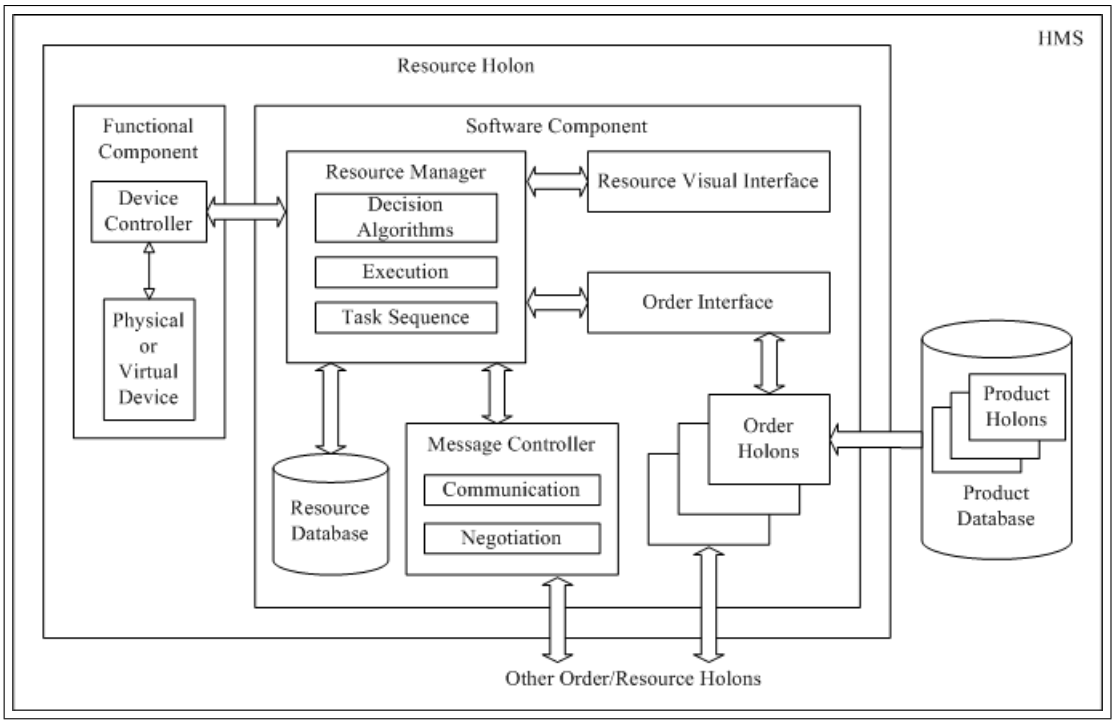

Figure 4.3. Holonic system implementation in BUFAIM

In the proposed architecture, product holon holds the product and process knowledge. It is associated with a product type not a specific product instance on the shop floor. It holds necessary information about product attributes such as type, receival and shipment stations, and alternative routes in the flexible process plans available to order holons when requested. Thus, product holons act as information servers and they are implemented in a shared Product Database. Hence, the communication with a product holon is mainly via database connections. An order having finished or about to finish an operation reads its alternatives for the next operation from the relevant

product holon. Resource holon is a physical resource in the manufacturing system like conveyor or cell which has knowledge about capabilities and state of the resource it corresponds. Note that every resource holon runs in a separate computer. The main responsibilities of the resource holon are contributing production and managing the physical devises of the resource like robot. In the proposed system there are two types of resource holons; workstation holons (ASRS Holon and Cell Holon) and material handling holons(AGV Holon and Conveyor Holon). Order holons are software agents that represent individual orders. Every order holon has necessary information about the job like receival time, due date or job type. Order holons are also have functionalities to achieve bidding with resources (Bilge *et. al.* 2002).

The communication structure of the proposed system is based on the Microsoft Message Queuing (MSMQ) technology which is a standard component of the Windows operating systems. There are two main types of messages in the system; bidding messages and transfer messages. To enable communication for these messaging tasks each resource has a dedicated message queue and four bidding message queues in the server. The bidding procedure in this study is a part initiated one and follows the general rules of the Contract-Net Protocol. The basic steps of the procedure is as follows. Order holon prepares a task announcement message for the candidate operation and sends it to the "Task Announcement Queue". The resource holons, which are able to execute the announced operation prepare a proposal and send a proposal message to "Order Proposals Queue". Then, order evaluates the bids and sends a task offer message that announces selected resource to "Task Offer Announcement Queue". After this, the awarded resource evaluates the task offer and sends an accept-reject message which includes its decision about the task, to "Orders Accept-Reject Queue". If order is accepted by the resource it sets its next operation, otherwise it re-announces the task (Bilge *et. al.* 2002).

4.3.3. Implementation Based Comparison of the Two Different SFCAs of BUFAIM Model Factory

According to Bilge *et. al.* (2002) the main advantage of the distributed SFCA to layered SFCA is the reduced complexity achieved by decentralization. The software development stage started with developing smaller components that are independent of each other. These components then introduced to system in suitable convenient development stages thus making debugging easier. In addition to this, with using the product/resource based architecture the system is more open to re-configuration. On the other hand, in layered architecture, a change in the system brings a burden of modification in all the higher levels.

In layered architecture response time to disturbances is longer because of the flow up of the feedback information about the disturbance to the higher level controllers. But in distributed application the planning actions does not need any assumptions about the active resource that must be included to the decision process. But in both systems the re-arrangement of the system for the disturbance need roll back procedures (Bilge *et. al.* 2002).

Two SFCA applications use different communication infrastructures. The layered system uses TCP/IP system which is a lower level communication structure compared to the MSMQ protocol which is used by distributed system. The TCP/IP protocol needs handling mechanisms for unsuccessful send operations. But MSMQ technology guarantees delivery without exceptions[1] . When two applications are compared according to the messaging load, distributed system needs much more messages than layered system because of the additional bidding messages(Bilge *et. al.* 2002).

[1]In this study it is seen that although the MSMQ technology provides a more rubost and applicable messaging structure it is much more slower than the TCP/IP protocol. For a simulation application a carefully applied TCP/IP protocol is seen more efficient than MSMQ technology.

5. DISTRIBUTED SIMULATION IMPLEMENTATION OF SFCAs

This chapter introduces the time stepped and distributed simulation implementations of the layered and holonic architectures of BUFAIM Model Factory. The virtual modules of layered simulation implementation and holonic simulation implementation are developed re-using the real-time implementations of the two architectures by using the FMS.NET infrastructure. The simulation modules corresponding to the physical components of the system are shared by the two control architectures. Thus virtual mimics of these physical components are only developed for once in a modular fashion. This enabled to use the shared modules like virtual robots and virtual AGV in both simulation implementations. In the following two sections simulation implementations of holonic and layered architectures are explained in detail. After these sections simulation and layout related databases of the simulation implementations are given. In the last section data collection structure of two different simulation implementations are mentioned.

5.1. Simulation Implementation for the Distributed Control Architecture

The simulation implementation of the distributed control architecture is based on the real-time distributed control architecture of BUFAIM Model Factory. To implement the simulation model there had been various changes in the original control architecture, but the main design issues are protected and used for the simulation implementation and real-time control architecture is re-coded using FMS.NET infrastructure in C♯ programming language. As it is mentioned above there are various changes in the simulation implementation from communication structure to bidding mechanisms, all these changes will be mentioned in this chapter. In addition to this, shared virtual modules, data structures of the two simulation implementation will be explained in detail. First the general structure of the system will be given with the explanation of the basic holons of the simulation implementation. After this LPs of

the system and virtual modules will be identified. Finally communication structure of the proposed simulation implementation will be clarified which is completely different from distributed control architecture of BUFAIM model factory.

5.1.1. General Structure of the System

The proposed simulation implementation uses the real-time distributed control architecture as a basis. But for simulation and architectural concerns there had been various changes in the former real-time application. Also it is needless to say that all physical components in the real-time application are virtual modules in the simulation application. The simulation implementation has the same design structure with the real-time application in identifying the manufacturing holons. To summarize, in the proposed simulation implementation there are three types of holons; order, product and resource[2] . In addition to this in the simulation implementation a synchronization holon is added to enable P/DS. All resource holons and the synchronization holon are modeled as LPs running on different computers in the simulation implementation. As it was mentioned in Section 4.3.2, proposed system does not have a higher level controller, every resource in the proposed system is at the same level. In this section basic structure of resource and order holons will be explained. Product holon has the same structure with the real time application. To summarize product holon serves to other holons as a database and the communication between product holon and other holons are achieved via database relations.

5.1.1.1. Resource Holons. Resource holons stand for all the manufacturing resources in the system. Every resource holon in the system has a corresponding LP in the simulation implementation. In this section the structure of the resource holons will be explained. Detailed information about the implementation of individual resource holons will be given in sections about the LPs. In Figure 5.1 specialization of the resource holons can be seen.

[2]For detailed information about design issues of the distributed architecture of BUFAIM model factory refer to Tunç (2005)

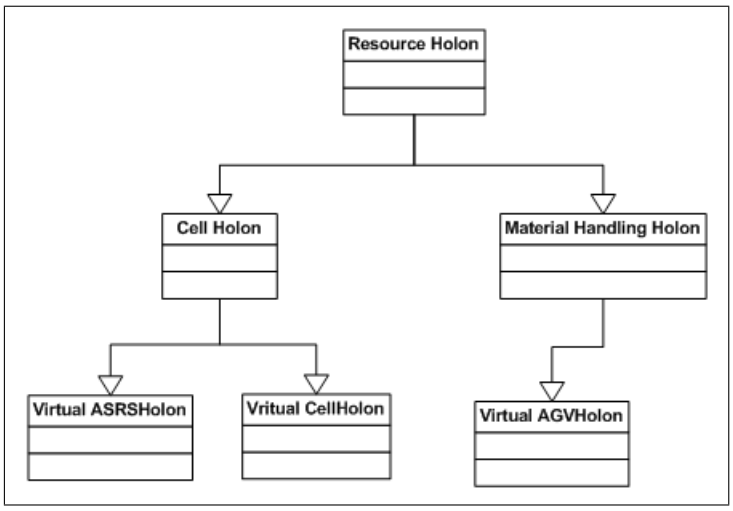

Figure 5.1. Class specialization diagram for simulation implementation of distributed SFCA

As it can be seen in the figure there are three types of resource holons in the proposed system. These are Virtual ASRSHolon, Virtual CellHolon and Virtual AGVHolon. All these three holons share a general structure in several components, but they differ according to their specialization in the manufacturing system. In addition to this they are identical to holons of Holonic Architecture of BUFAIM model factory if they are considered according to the resource activities and responsibilities. To summarize Virtual ASRSHolon enables order receival and shipment, Virtual CellHolon enables processing of the parts and Virtual AGVHolon enables material handling between various cells. It is needless to say that all the physical parts in the Holonic Architecture of BUFAIM model factory are virtual in the simulation implementation. In Figure 5.2 and at Appendix A.1 general structure and class relations of the simulation implementation of a resource holon can be seen respectively.

As it can be seen in the Figure 5.2, resource holon has two main components; resource control component which is related with the execution of the production and virtual functional component which are virtual modules that represents physical devices. In general a resource holon co-operates with other holons to drive production with bidding mechanisms and interact with its virtual modules to execute some tasks. Remember that there is not a higher level controller in the system, every manufacturing

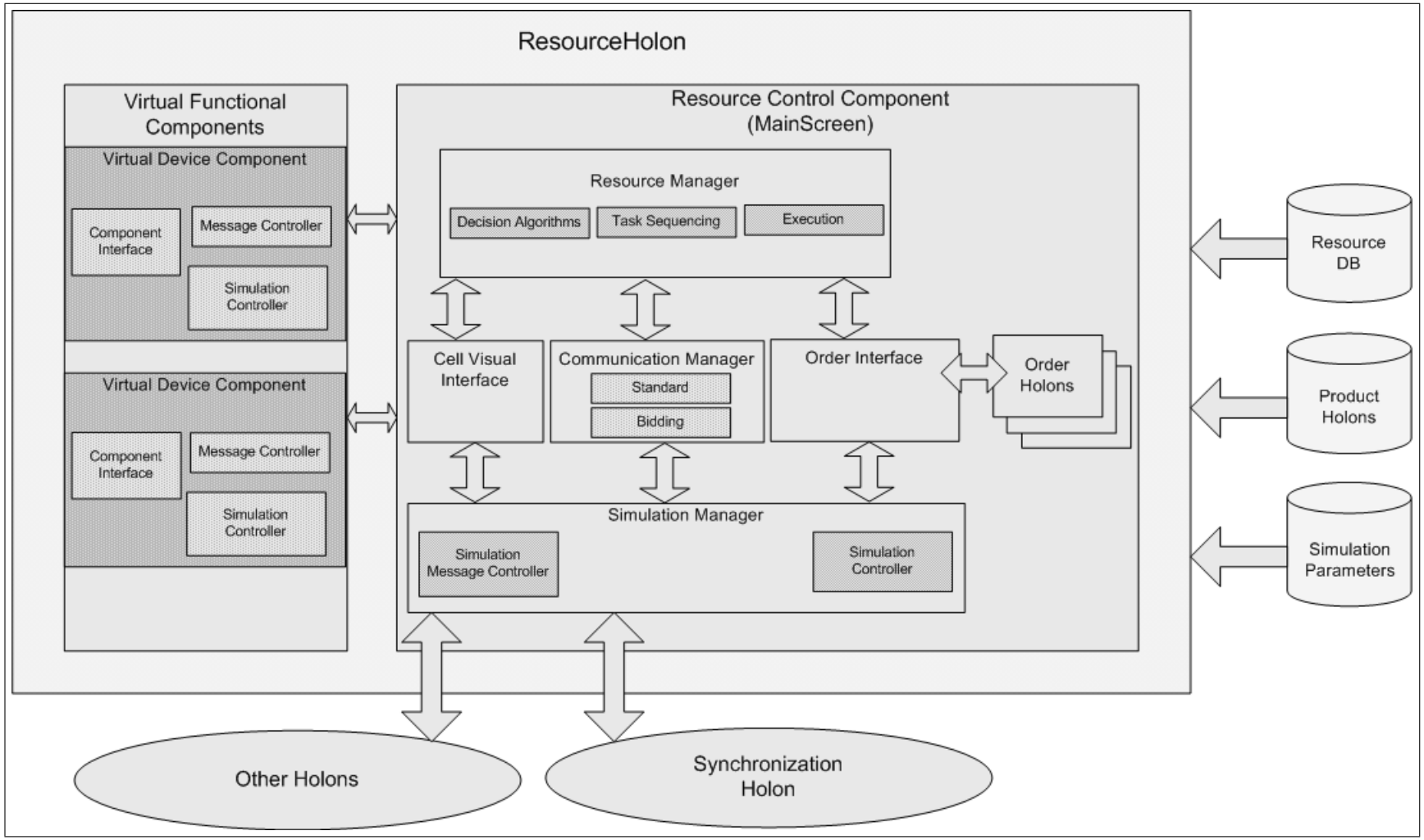

Figure 5.2. General structure of a resource holon

task is accomplished by interchanging messages between holons in the manufacturing system. In Table 5.1 resource related components and their basic responsibilities are given.

Table 5.1. Resource related components of a resource holon

Component	Basic Responsibilities
ResourceManager	Operational decisions within the cell
Resource Visual Interface	Status output for user
CommunicationManager	Prepares and interprets bidding messages Interprets standard transfer messages
Order Visual Interface	Homes order holons in the resource Gives visual outputs of orders
SimulationManager	Manages P/DS related issues Enables and controls communication with other holons

MainScreen provides the visual interface for cell activities, controls programming tools like serial port management systems and houses all other components like virtual modules, ResourceManager, CommunicationManager and OrderScreen user control objects and SimulationManager. MainScreen initializes all the components of the resource holon and coordinates all the resource, virtual device activities and simulation related execution in the cell holon. Class diagram of MainScreen component can be seen in Appendix A.2. With a timer programming tool, which has a pre-determined timer tick interval, MainScreen enables a time stepped simulation progress. In every timer tick all the components in the virtual cell executes their computations; resource components executes resource related computations and virtual devices simulates virtual device events according to the current time of the system.In addition to these activities in every timer tick P/DS related events are controlled by SimulationManager i.e. which are barrier primitive, batch event and simulation termination. After execution of all events current time is increased by one time unit. In Appendix A.3 the UML sequence diagram of the clock timer tick event of the MainScreen and in Appendix A.4 the simulation related executions are shown.

ResouceManager controls cell activities like storing location status of elements, and has the layout and product information of the current configuration of the manufacturing system. In initialization ResourceManager reads necessary databases and creates necessary data structures. Other components use these data structures in various ways. Thus ResourceManager has relations with almost every autonomous entity within a cell. Class diagram of ResourceManager is given in Appendix A.5.

CommunicationManager component manages communication with the outside and enables bidding with order holons. Bidding structure of the proposed simulation implementation will be given in detail in the following sections. The communication does not only consist of bidding messages, there are also some standard messages that are interchanged between resource holons which are messages that trigger transfer of a part from AGV or sending a call message to AGV for the scheduled part[3] which are also managed by CommunicationManager. CommunicationManager component is visually represented in the MainForm as a user control. From this user controller all the messaging (standart or bidding) can be tracked. Class diagram of CommunicationManager can be seen in Appendix A.5

OrderScreen visual interface stores order agents, gives basic information about their statuses and shows messages interchanged between orders and other resource holons. OrderScreen is embedded to MainScreen as a user control. The class diagram of OrderScreen is given in Appendix A.6.

SimulationManager component enables communication of the resource holon with other holons and handles P/DS related issues. As it is mentioned above communication structure of the proposed simulation implementation is totally changed. *MSMQ* technology used by real-time implementation was first tried in the simulation implementation. Because of the high level structure of *MSMQ*, sending and receiving messages with using this structure takes time. Thus time step value of the simulation could not be decreased efficiently. Because of this burden, communication structure is totally changed to *TCP/IP* which is a low level communication structure compared with *MSMQ* and

[3]For detailed information about the standard messaging structure refer to Tunç (2005)

much more faster. The drawbacks of using *TCP/IP* comes with exception handling which is more difficult compared with *MSMQ*. Briefly SimulationMessageController component of SimulationManager receives all messages and frees them with concerning message time and current simulation time for execution, and sends all messages of the components of the resource. Detailed information about messaging structure will be given in Section 5.1.5. P/DS responsibilities of SimulationManager is handled by SimController component. SimController component controls P/DS events which are barrier primitive, batch, and simulation termination. In addition to this in a batch event SimController collects necessary statistics for the resource and reports them with a convenient format. In Appendix A.7 class diagrams for SimulationManager and SimController are given.

5.1.1.2. Order Holon. Order holon in simulation implementation is similar to real-time application of the order holon. It has the same procedures but there are some important differences related with bidding which will be explained in Section 5.1.2.1. Order holon can be defined as a software agent which represents a part in the system. Order holons are created in the resource holons when they enter and are deleted after they leave. Order holon has all the necessary attributes to store its necessary data like due date, weight or remaining processing time. Order holon also has functionalities to enter and accomplish bidding with other resources[4] . The bidding mechanism of the simulation implementation will be explained in detail in Section 5.1.2.1. The class diagram of order holon is given in Appendix A.6.

5.1.2. Resource Allocation and Dispatching Structure

In this thesis, resource allocation and dispatching structure formerly used in the real-time holonic SFC implementation is replaced by an enhanced one. There are two important phases of the proposed approach related with the progress of the production. First phase is resource allocation which is accomplished through a bidding mechanism ending with a contract between a resource and order. The second phase is

[4]For design issues about order holon refer to Tunç (2005)

the dispatching, in other terms summon of an allocated part by a resource when there is a free capacity. These procedures will be explained in detail in this section.

The first phase is completed with the bidding between operation resources for operation tasks and material handling resources for transfer tasks. After these two contracts, an order has made all the necessary decisions for its next operation; the operation resource which will execute the operation and the material handling resource which will transfer the order from its current location to the next location are determined.

5.1.2.1. Bidding Structure. As it is mentioned in the previous sections operational decisions are taken with a co-operation mechanism between manufacturing holons. In other words, the production is progressed by bidding sessions between resources and orders. The main structure of the bidding mechanism in simulation implementation is similar to real-time control architecture. The bidding procedure is a part initiated one and does not include re-negotiation between resources and orders. The general structure of the bidding between order and resource can be seen in the Figure 5.3.

As it can be seen in the Figure 5.3, order holon sends an announcement, related with the candidate operation which is to be processed next in its job route, without any information of the resources that are capable of processing that operation. Then resource holon, that can process candidate operation, prepares a proposal and sends it to the order holon. Upon receiving proposals, order holon selects the most suitable one from the proposals and sends a task offer to the corresponding resource. The resource holon selects one of the order holons from the task offers received in the same bidding session and sends an accept message to selected order holon where as a reject message is sent to all remaining order holons. A contract is made between the accepted order holon and resource holon; rejected order holons send task offer to the second most wanted resource holon if it is available, if not they re-announce the task.

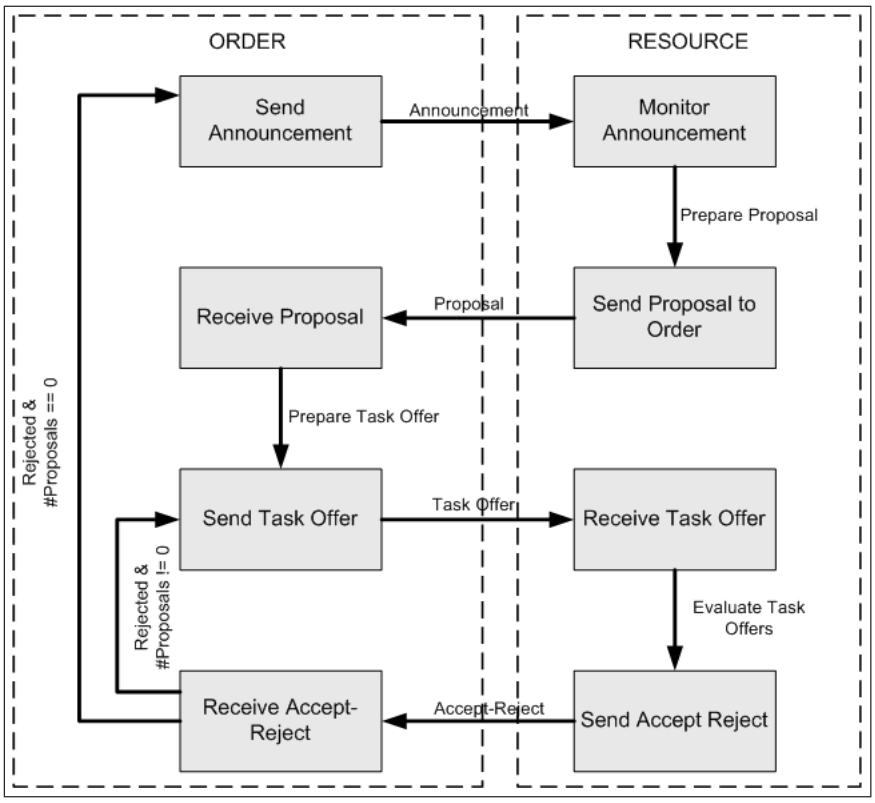

Figure 5.3. General structure of the bidding between resource and part

A bidding session of orders and resources is enabled with message exchanges between corresponding resources and orders. Messaging structure of the proposed simulation implementation will be explained in detail, in Section 5.1.5. The exchanged messages are transfered to convenient formats which helps evaluation of the messages by corresponding objects. These message formats are derived from the steps of the bidding session naturally. In Table 5.2 message types and their brief definitions are given. In Appendix B.1 class diagrams of the message types can be seen.

CommunicationManager component handles bidding for resource which has necessary functions and attributes for entering and completing a bidding session. The main procedure for bidding of the resource holon is *MessageTimerProcedure* function of CommunicationManager which enables bidding by calling necessary functions of CommunicationManager. *MessageTimerProcedure* is called by MainScreen in every timer tick. The UML diagram of *MessageTimerProcedure* is given in Appendix B.2. In Table 5.3 the pseudo codes of the MessageTimerProcedure functions are given.

Table 5.2. Message types used in a negotiation session

Message Type	Explanation
HolonicMessage	Base class for messages
TaskMessage	Task announcement message sent by order
ProposalMessage	Proposal message sent by resource in order to answer a task announcement
TaskOfferMessage	Offer message sent by order that includes basic attributes of the order and its offer to the resource
OrderMessage	Accept or reject message sent by resource to corresponding order

As it can be seen in the Table 5.3 first, received cell messages are executed to update the status of the resource in order to take account of changes in the state of the resource, then the bidding session of CommunicationManager starts. To manage bidding a *NegotiationRound* object is used which stores all received tasks, task offers, and sent proposals. The class diagram of *NegotiationRound* is given in Appendix B.3. *NegotiationRound* object has various functions and data structures that manages a bidding session of CommunicationManager.

If there is not any uncomplete *NegotiationRound* at hand, in *MonitorTaskAnnouncement* procedure, CommunicationManager receives all *TaskAnnouncement* messages and creates a *NegotiationRound* object to track bidding session. Sequence diagram of *MonitorTaskAnnouncement* is given in Appendix B.4. After this, CommunicationManager prepares proposal by using received task announcements and sends them to corresponding order holons in *SendProposalToPart* procedure. Sequence diagram of *SendProposalToPart* procedure can be seen in Appendix B.5. The bid of the resource holon for the announced task is included in the proposal whose calculation is important for the overall performance of the manufacturing system such that the bid will affect the decision of the part in order to select a resource. Bidding algorithms of the simulation implementation will be explained in detail in Section 5.1.2. After this, CommunicationManager collects task offer messages of the sent proposals with *ReceiveTaskOffer* procedure. The sequence diagram of the procedure is given in Appendix B.6. Then

in *SendAcceptRejectMessage* procedure CommunicationManager waits for receiving all task offers of sent proposals and sends an *accept message* to accepted part and *reject message* to other parts. Sequence diagram of *SendAcceptReject* procedure is given in Appendix B.7. With this, resource holon adds the accepted task to its *ScheduleList* and finishes the negotiation.

Table 5.3. Pseudo code of the MessageTimerProcedure functions

Function	Pseudo Code
InterpretCellMessage()	$foreach(message)$ Call necessary functions
MonitorTaskAnnouncement()	$if(currentNegotiationRound == null)$ Open *currentNegotiationRound* Collect task announcement messages
SendProposalToPart()	$foreach(TaskMessage)$ Prepare proposal Send proposal
ReceiveTaskOffer()	$foreach(Proposal)$ ReceiveTaskOffer of currentProposal
SendAcceptReject()	Select accepted task Send accept message to accepted task Send reject message to rejected task(s) Terminate *currentNegotiationRound*

On the other hand, bidding of an order holon is enabled with *OrderTimerProcedure* which is called in every timer tick for all order holons in the resource by MainScreen. The sequence diagram of *OrderTimerProcedure* is given in Appendix B.8. There are various functions of order holon to provide bidding with other resource holons. The pseude codes of these functions are given in Table 5.4

A bidding session for an order holon starts with end of its operation in a resource holon. With this event, order holon determines its next operation according to its process plan and sends a *task announcement* to all processing resources in the system with

Table 5.4. Pseudo code of the OrderTimerProcedure functions

Function	Pseudo Code
SendAnnouncement()	Prepare TaskAnnouncement Send TaskAnnouncement
ReceiveProposal()	*while(ProposalTimeTick < AllowedTick AND Proposal.Count > Desired)* Collect proposal messages
SendTaskOffer()	Prepare TaskOffer, award a resource Send TaskOffer
ReceiveAcceptReject()	Receive AcceptRejectMessage Make a contract with the resource

SendAnnouncement procedure. The sequence diagram of *SendAnnouncement* procedure is given in Appendix B.9. After this, in *ReceiveProposal* procedure, order holon collects a specific number of proposal. If needed number of proposals is not reached for a specific time period, order holon cancels collecting proposals. The sequence diagram for *ReceiveProposal* procedure is given in Appendix B.10. When order holon is ready to send task offer, it prepares a *TaskOffer* with awarding the resource holon with the lowest bid value including its offer to resource and sends *TaskOffer* message which identifies the awarded resource. This procedure is enabled with *SendTaskOffer* function, whose sequence diagram is given in Appendix B.11. Here offer formulation is again an important issue for a bidding algorithm. Detailed information about offer formulation will be given in the next section. Upon sending the *TaskOffer*, order waits for the *OrderMessage* which includes the answer of the resource to the offer. If order holon is accepted by the awarded resource then a contract is made between them. Otherwise, order holon re-announces the task if there is not any alternative proposals. If there are alternative proposals available, it awards the most suitable resource among the remaining proposals. The sequence diagram of *ReceiveAcceptReject* procedure is given in Appendix B.12. After the contract with an operation resource order holon enters a bidding session with a material handling resources for the transfer task which is essentially the same as the bidding session for an operation.

As seen above, an order and resource holon exchange a set of messages for a contract. For a bidding session the order does not need to know number or availability of resources in the system. There is not a higher controller which interfere to this bidding session from outside. In Table 5.5 used message types in a bidding session and their senders and receivers are given. Also the activity diagram for bidding session can be seen in Appendix B.13.

Table 5.5. Message traffic in a bidding session

Message Type	Sender	Receiver
AnnouncementMessage	Order Holon	Resource Holon
ProposalMessage	ResourceHolon	Order Holon
TaskOfferMessage	Order Holon	Resource Holon
OrderMessage	Resource Holon	Order Holon

5.1.2.2. Bidding Algorithms of the System. In a bidding session there are two important issues. The first one is the bid calculation of the resource which is included in the *proposal message* and the other one is offer calculation of the order which is included in the *task offer message*. As mentioned above, an order holon selects awarded resource according to the bid criteria. Thus, bid calculation of a resource becomes an important issue for a convenient matching of a resource and an order. Basically, the resource calculates the expected time candidate order would spent on the resource for its candidate operation considering several factors related with the resource. The factors considered in bid calculation for a processing resource are given below:

1. Total expected process times of the scheduled orders which has operations that use the same processor type with the next operation of the candidate order.
2. The expected process time of the candidate operation.
3. If the processor which would be allocated by the next operation of the candidate order is currently busy, half of the expected process time of the order.
4. The expected process times of the orders in the input buffer of the resource (if any) which has operations that use the same processor with the candidate order.

5. The expected process time of the order that is being transfered to the processor at the time of the bid calculation, which will use the same processor with the next operation of the candidate order.

These expected process times are obtained from the historical data collected for the operations performed by the resource. The resource with the smallest bid will be selected by the order.

The second important point for the bidding session for the matching of a resource and an order is the offer calculation. Offer value is included in the *task offer message* of the order to the awarded resource. As mentioned above, the resource selects the order based on the offer criteria. For this selection, various decision criteria can be used. In the proposed simulation implementation this decision is named as *evaluation algorithm* and there are four main algorithms are proposed for this. In Table 5.6 these evaluation algorithms are given with their definitions.

5.1.2.3. Dispatching Messages and Algorithms. The bidding sessions of the proposed system does not enable transfer of a part from its location resource to its destination resource with the matched material handling holon. To achieve this, a series of *call messages* are exchanged between holons. There are two types of call messages used in the system: *delivery message* and *pickup message*. After a successful bidding session, the operation resource must send a call message to material handling resource to inform it that a free space is reserved for the oder holon. Also the resource which is the current location of the mentioned order should inform the material handling resource that the order holon is in the output buffer of the resource and is ready for pickup. The former message is called a *delivery message* where as the latter one is called *pickup message*.With these two types of messages AGV holon avoids deadlocks that can be caused by trying to deliver a part which has not a place reserved in the destination input buffer or trying to pickup a part which is not placed in the output buffer of the location resource. The working of the system with this rationale is given below.

Cell holon has information about the orders which are scheduled to itself with the bidding sessions and tracks if an order in its scheduled list has made a contract with a material handling resource with the *MonitorMaterialHandling* procedure. In this procedure, the resource adds the scheduled orders to a collection if they made a contract with the material handling resource. The sequence diagram of *MonitorMaterialHandling* is given in Appendix B.14. Then when a place in the input buffer becomes available, the resource selects an order and sends a delivery type call message to the material handling resource of the order with *SendCallMessage* procedure. In *SendCallMessage* procedure, if the resource has a free space in the input buffer, it selects an order from its schedule list according to some criteria. The sequence diagrams of *SendCallMessage* procedure and its sub-procedure related with order selection, *FindOrderToCall*, are given in Appendix B.15 and Appendix B.16 respectively. On the

Table 5.6. Evaluation algortihms of the proposed system

Evaluation Algorithm	Definition
Slack	Select the order with minimum slack $Offer = dueDate - remainingProcessingTime$
Receival Time	Select the order with the minimum arrival time to the FMS $Offer = arrivalTime$
Due Date	Select the order with minimum due date $Offer = dueDate$
Alternative Cost	Discriminates orders that have alternative resources from those with no alternatives. M is an incentive for the order without alternate availability of order for the candidate operation $Offer = \begin{cases} slack - Bid1 + Bid2 & \text{proposals.Count} > 1 \\ slack - M & \text{otherwise} \end{cases}$ Bid1: Proposal with best bid Bid2: Proposal with second best bid

other hand, the destination resource sends a pickup type call message to the material handling resource when the order is placed in the output buffer of the resource. When AGV holon stores all the orders that are ready for pickup and has place for delivery. From these orders AGV holon selects a number of them equal to its slack capacity and collects and delivers them with a sequence according to some criteria using *FindOrderToDispatch* procedure.

Several rules are proposed for the selection of orders by the resource from the scheduled order list. A processing resource uses a *calling algorithm* when it has an available input buffer for the incoming order. A material handling resource uses a *dispatching algorithm* to determine the sequence of collecting candidate parts among those which are ready to pickup and have place in the destination resource from its scheduled order list when it has available space on board. Here the *Parameter* calculation for an order changes with the selected algorithm. In Table 5.7 *calling and AGV dispatching algorithms* of the proposed system are listed and explained in detail below.

Table 5.7. *Calling/Dispatching Algorithms* of the proposed system

Algorithm	Explanation
Order List	Uses order's entry time to the schedule list as Parameter
Slack	Uses slack values of the orders as Parameter
Arrival Time	Uses arrival time values of orders as Parameter
Due Date	Uses due dates of orders as Parameters
Processing Time	Uses processing time of the orders as Parameter
Parametric	Uses a set of parameters calculated from various attributes of the orders as Parameter
Apparent Tardiness Cost	Uses a modified version of Apparent Tardiness Cost (ATC) single machine scheduling rule index value as Parameter

All these algorithms are applied to the orders that are in the schedule list of the resource at that time and has fulfilled some requirements. A cell holon decides only

when there is a free input buffer and considers only those scheduled orders which are already matched with a material handling resource. In addition to these, a material handling holon only considers orders that has place in the destination resource and is ready for pickup when it has a free location on its deck.

Slack, Receival Time, Due Date and *Processing Time* calling algorithms are similar to each other which use slack, receival time, due date and processing time values as the Parameter value respectively and selects the order with the minimum Parameter. In *Order List* algorithm the first order in the schedule list is selected. These *calling algorithms* can be thought as simple priority rules that are used by cell holons for calling and AGV holon for dispatching.

Parametric rule is a modified version of the negotiation algorithm proposed by MacChialori and Riemma (2002). Basically algorithm uses a set of sub-parameters and derives the *parameter* value by multiplying these sub-parameters. Here the sub-parameters should direct the *parameter* to the same objective. The sub-parameters of the proposed algorithm is given below.

First sub-parameter is *Slack* parameter and related with the slack values of the considered orders. The formulation of *Par1* for order$_i$ is as follows:

$$\text{Par1}_i = 1 - R \times (Slack_i - minSlack) \tag{5.1}$$

where
$Slack_i$: slack value of order$_i$,
minSlack: minimum slack value of orders that are considered.

The rationale behind this formulation is setting *Slack* sub-parameter of orders that have relatively smaller slacks than the others close to 1 and those that have relatively greater slacks than the other orders close to 0. Note that the order with the minimum slack has *Slack* sub-parameter value as 1. On the other hand the formulation also prevents consideration of orders whose slack values differs from the minimum slack

more than $1/R$. If an slack value of an order differ from the minimum slack more than $1/R$ value then its *Slack* sub-parameter is set to 0. This approach gives importance to *Slack* sup-parameter among the other sub-parameters such that only the orders whose slack values lie in a range which is equal to $1/R$ ratio can be called. In the proposed algorithm R value is set to 0.001 which leads a consideration of orders in a range of 1000 time units.

Second sub-parameter for the *Parametric* algorithm is *Order Weight* sub-parameter and it is related with the priority value of the orders. The weights are used to give relative importance to the orders. The formulation is as follows for order$_i$:

$$\text{Par2}_i = Weight_i / maxWeight \tag{5.2}$$

where
$weight_i$: relative importance of order$_i$,
$maxWeight$: maximum weight value of orders that are considered.

With this formulation orders with high weight values among the scheduled orders of the resource will have *Weight* sub-parameter close to 1 and orders with lower weight values will have *Weight* sub-parameter close to 0. This formulation gives a higher chance to be called for orders with higher weight values.

The last sub-parameter of the *Parametric* calling algorithm is *Remaining Processing Time* sub-parameter and it aims to give priority to orders which has more work to do. The formulation is as follows:

$$\text{Par3}_i = \alpha + (1 - \alpha) \times \frac{RemProcTime_i}{maxRemProcTime} \tag{5.3}$$

where
$RemProcTime_i$: remaining processing time of order$_i$,
$maxRemProcTime$: maximum remaining processing time value of orders that are considered

α: adjustment parameter

This formulation sets *Remaining Processing Time* sub-parameter of the order close to 1 if it has a higher value of remaining processing time among the scheduled orders of the resource. Note that sub-parameter is close to α if order has a lower relative remaining processing time value. In the above formulation α is selected in a way to adjust the relative impact of Par3 among others.

The Parameter value of an order is calculated as follows after the sub-parameters are determined:

$$\text{Parameter}_i = 1 + \text{Par1}_i \times \text{Par2}_i \times \text{Par3}_i \tag{5.4}$$

The order with the highest Parameter value is selected to be called for the free space in the input buffer by the resource. Also AGV holon uses the same algorithm to select an order for matching.

The last rule proposed is *Apparent Tardiness Cost* (ATC) which uses the well known one machine dispatching rule with the same name. In the original ATC rule a ranking index is calculated dynamically whenever the machine becomes free (Pinedo 1995). The algorithm works similar to the original ATC rule; when a place becomes free in the input buffer of the resource, it calculates index values for all orders in its schedule list and selects the order with the highest index value. The formulation is given below;

$$\text{Parameter}_i = \frac{Weight_i}{PTime_i} \times exp\left\{-\frac{MAX(DueDate_i - RPTime_i - currentTime, 0)}{k \times MPTime}\right\} \tag{5.5}$$

where

k: scaling parameter,

$PTime_i$: processing time of order$_i$ in the resource,

$DueDate_i$: due date of order$_i$,

$RPTime_i$: remaining processing time of order$_i$,
$currentTime$: the current simulation time,
$MPTime$: the mean processing time value of all orders that are considered.

Scaling parameter k is taken as 3 which is advised by Vepsalainen and Morton (1987) for a dynamic shop floor environment. The *Parameter* value of an order is set to 0 if the difference between its slack and minimum slack is greater than 1000 time units. The rationale behind this is preventing excessive waiting times of orders with relatively smaller weight values when all the orders in the system are late. The order with the highest Parameter value is selected.

5.1.3. Virtual Modules of the System

Simulation implementation of the proposed system does not have any physical component. The physical components of the BUFAIM model factory are modeled in the proposed simulation implementation as virtual modules of the system. The virtual mimics of the physical components are shared by two different P/DS applications. Thus they will be explained here for once. In Table 5.8 virtual modules of the proposed simulation implementation and their corresponding physical components are given.

Table 5.8. Modules of the simulation implementation and corresponding physical components

Module	Corresponding Physical Device
Virtaul Agv	AGV
Virtual Robot	Robot
Virtual Processor	Processor

A virtual module have three main components which are; visual interface, MessageController and SimulationController. Visual interface gives status information data to user, related with the current executions of the virtual module. MessageController component manages communication with the cell by receiving and sending messages whereas SimulationController component emulates physical component with an event

based structure. Some of the virtual modules also collect necessary data related with their responsibilities.

A resource in the simulation implementation can have different types and number of virtual modules according to the configuration of the system. This configuration can be changed easily by the user with adjusting layout and simulation databases, for example a cell holon can have two processors and one robot. In addition to this, some virtual modules are special modules that can be used only in a specific resource. For example VirtualAGV can only be used in AGV holon and VirtualASRSRobot is only available for ASRS holon. The detailed information related with the configuration of resources with virtual modules by using simulation parameters database is given in Section 5.3.2.

Basically, all physical components are modeled as close as possible to their corresponding physical equivalents. Their messaging structure with resources and behaviors are protected in the simulation implementation. Virtual modules has all necessary attributes for emulating the physical components. Also with adjusting these attributes different physical components can be modeled in the system. Below, detailed explanation of virtual modules and their relation with LPs are given.

5.1.3.1. Virtual Robot Modules. VirtualRobot module stands for robots that serve cells in the BUFAIM model factory. There are two kinds of virtual robots in the proposed system, namely VirtualRobot and VirtualASRSRobot which only differ in the messages exchanged with their controller cells. Virtual robot simulates transfer tasks of the robot arm with an event based fashion. VirtualRobot and cell communicates via RS232 protocol as in the real-time application with the use of necessary programming tools. In addition to this, messages exchanged between LPs and VirtualRobots remains the same as the ones exchanged between physical robots and real-time cell controllers. Basically, VirtualRobot creates necessary events according to the messages received from cell and sends necessary messages generated by the execution of events. The class diagram for components of VirtualRobot is given in Appendix C.1. In Appendix D.1

and Appendix D.2 screen shots of VirtualRobot and VirtualASRSRobot can be seen respectively.

As mentioned above, like all virtual modules, VirtualRobot has three main components, namely MessageController, SimulationController and robot visual interface. When a message is sent to VirtualRobot, MessageController component receives and decodes this message where as SimulationController evaluates this decoded message and creates necessary event. Visual interface outputs necessary information about the current events and unit loads in transfer. In P/DS implementation, message strings are same with the real-time control applications, in other words, cell sends the same messages and VirtualRobot responses with the same messages after a task is completed. Like the physical robot in BUFAIM model factory, *RS232* protocol is used such that, virtual robot listens to a specific *COM* port which is the port that resource component sends messages to virtual robot. Virtual robot interprets the received messages and sends necessary messages to a specific *COM* port which is listened by resource component. Thus a messaging structure which is as close as possible to the real-time messaging structure is obtained.

The behavior of physical robots are modeled with events in the virtual robots. Events and messages have an ensuing structure that they creates new events and/or messages with their executions. The class diagrams of virtual robot events can be seen in Appendix C.2. Main definitions of the virtual robot events are given below;

1. *StartTransfer:* This event is created with a transfer command from the cell. A *StartTransfer*event sets current state of the robot as busy and schedules either an *EndTransfer* or *BreakDown* event as the next event.
2. *EndTransfer:* This event stands for a transfer task of the robot and can have different execution times with the type of the transfer which enables simulating all transfer tasks of the physical robots. With execution of this event, a completion message is sent to cell which informs that transfer task is completed.

3. *BreakDown:* This events is created if a scheduled break down is detected within the execution time of a transfer task. With this event robot becomes broken down and enters repair state. As a result, a *EndOfBreakDown* event is scheduled.
4. *EndOfBreakDown:* This event emulates the repair process for the robot after the occurrence of a *BreakDown* event. When this event is executed the robot becomes available and finishes remaining transfer tasks.

As explained above, there is a sequential relationship between messages and event such that all behaviors of the robots are simulated with a chain of events following each other. In Appendix C.3 this sequential relationship is given.

5.1.3.2. Virtual Processor. Virtual processor emulates processor of a cell in the proposed simulation application which has a similar structure with virtual robot in many ways but there are slight differences between two modules. In the real-time applications of BUFAIM model factory, processors were simulated via a video film. In simulation implementations processors are modeled as separate modules that exchange messages with their cells. Virtual processor has three main components, namely MessageController, SimulationController and processor visual interface. The relation between these components and messaging structure is same with virtual robot module. Also screen shot of the processor visual interface can be seen in Appendix D.3.

The processing of a part is emulated with an event structure and and resource component and virtual processor inform each other during their executions. Messages between resource and virtual processor are exchanged with *RS232* port. With these messages a virtual processor creates necessary events and emulates processing of a part or a break down in the system. In Table 5.9 formats of exchanged messages and their brief explanation are given.

Table 5.9. Messages exchanged between resource and virtual processor

From	To	Message String	Explanation
Resource	VirtualProcessor	*StartProcess-UnitloadName-JobType*	This message infroms virtual processor that a part is placed to processor
VirtualProcessor	Resource	*EndProcess-UnitloadName-JobType*	This message informs resource that virtual processor has finished the process and part is ready to be transfered
VirtualProcessor	Resource	*BreakDown*	This message informs resource that virtual processor is broken down and in repair
VirtualProcessor	Resource	*EndOfBreakDown*	This message informs resource that virtual processor is available again after the repair procedure

Like virtual robot, virtual processor has an event based structure which is driven by messages received from resource component. Virtual processor simulates behaviors of processor with execution of these events. The class diagram for virtual processor events is given in Appendix C.4. Also brief explanation of these events can be seen below:

1. *StartProcess:* This event is created with a start process command from the resource. It does not simulates any behavior of the processor but schedules other events like *EndProcess* or *BreakDown*. A *StartProcess* event sets current state of the processor as busy and creates next event with considering a break down occurrence.
2. *EndProcess:* This event stands for a processing of a part by the processor and can have different execution times with the type of the part and operation which enables simulating different operations in a virtual processor. With execution of this event, a completion message is sent to resource which informs that operation is completed.
3. *BreakDown:* This events is created if a scheduled break down is detected within the execution time of an operaiton. With this event processor becomes broken down and enters repair state.
4. *EndOfBreakDown:* This event emulates the repair process for the processor after the occurrence of a *BreakDown* event. When this event is executed the processor becomes available and finishes remaining operations.

With these events virtual processor module emulates all the behaviors of a physical processors which are triggered by other events or messages sent from resource. The flow of events of a virtual processor is given in Appendix C.5.

5.1.3.3. Virtual AGV Module. Virtual AGV module simulates AGV with a specific load capacity and speed. The simulated module specifically mimics the physical AGV of BUFAIM model factory[5] . To summarize, the physical AGV travels on a pre-

[5]Detailed information can be found about the physical AGV of the BUFAIM model factory in Polat (2003)

determined path in one direction. AGV and AGV controller communicates via a radio frequency system. AGV controller sends a string message that includes the code of the route which will be executed. AGV decodes this message and executes the programming component related with decoded message. The messaging between AGV and AGV controller has a hand-shaking structure such that when a message is received or an execution of a message is started an inform message is sent to the sender entity. In simulation implementation these basic structure is fully protected and the communication structure is modeled with parallel *RS232* ports. Basically virtual AGV and AGV controller exchanges messages same as the real-time implementations. But of course, in simulation implementation, virtual AGV emulates the behaviors of the the physical AGV with an event based structure. When a command is received from AGV controller, virtual AGV evaluates this command and sends necessary messages and creates necessary events.

Virtual AGV module has two main components which are SimulatonController and SimAGV. SimulatonController component is responsible with messaging with AGV controller, executing necessary events, managing the behavior of the virtual AGV and giving visual output for the user whereas SimAGV is a data structure for representing the AGV. The class diagram of SimulationController and SimAGV component of the virtual AGV is given in Appendix C.6. Also the screen shot of the virtual AGV module is given in Appendix D.4.

Like other modules of the simulation implementation, virtual AGV module has an event based structure which is triggered by other events or messages received from AGV controller. Below events related with AGV and their brief explanations are given. Also in Appendix C.7 and Appendix C.8 class diagram and flow of these events can be seen.

1. *AStartMoveEvent:* This event simulates the movement of the physical AGV which is triggered by a *start move command* from AGV controller. With execution of this *AStartMoveEvent* current location of the virtual AGV is updated according to the start move command received from AGV controller. Also a *AEnd-*

MoveEvent with the calculated travel time of the virtual AGV from the entries of received message or a *ABreakDownEvent* with respect to break down distribution is scheduled .

2. *AEndMoveEvent:* This event stands for the completion of a move command. With the execution of this event a *completion message* is sent to AGV controller.
3. *AStartSeizeNodeEvent:* When virtual AGV reaches to a destination node AGV controller decides if the virtual AGV will seize this node for a part transfer or continue its voyage. This event is created if a seize node command is received from AGV controller.
4. *AEndSeizeNodeEvent:* This event stands for the completion of a seize node event, i.e alignment to a transfer dock. With the execution of this event a *completion message* is sent to AGV controller.
5. *ABreakDownEvent:* This event emulates a break down occurrence in the virtual AGV. If a scheduled break down is detected in a *AStartMoveEvent* then this event is scheduled as the next event. With the execution of this event virtual AGV becomes unavailable and repair procedure is started.
6. *AEndOfBreakDownEvent:* This event stands for the completion of a repair process on the virtual AGV. With execution of this event a *AStartSeizeNodeEvent* or a *AStartMoveEvent* is scheduled as next event according to the state of the AGV before the *AEndOfBreakDownEvent*.

5.1.4. Logical Processors of The System

In this section LPs of the simulation implementation of distributed real-time control architecture will be explained in detail. As it is mentioned before, every resource entity in the system is thought as an individual LP and run on a different computer in the simulation application. There are three types of LPs in the system; VirtualAGVHolon, VirtualCellHolon and VirtualASRSHolon, where last two LPs are very similar to each other. In the Table 5.10 below LPs of the system and modules allocated by these LPs are given.

Table 5.10. LPs of the proposed simulation implementation

LP	Allocated Virtual Module
VirtualCellHolon	VirtualRobot VirtualProcessor
VirtualASRSHolon	VirtualASRSRobot
VirtualAgvHolon	VirtualAgv

5.1.4.1. Virtual CellHolon and Virtual ASRSHolon LPs. VirtualCellHolon and VirtualASRSHolon LPs are resource holons which are similar to each other in various ways. The slight differences come from the allocated virtual modules and messaging structure of the LPs with their corresponding virtual modules. Apart from these differences, two resource holons are similar, they share all the components of a resource holon and uses the same logic for co-operating with other holons. VirtualASRSHolon can be thought as a VirtualCellHolon which does not need virtual processor modules for its executable operations (*Receival* and *Shipment*) and allocates virtual ASRSRobot module which only has slight differences from the virtual robot module. Basically a proessing resource holon negotiates with order holons for operational decisions and homes order agents of the allocated parts. To fulfill these requirements the necessary class structures are mentioned in the previous sections.

VirtualCellHolon LPs of the system have virtual processors that can execute various operation and a virtual robot which enables part transfer within the cell and between cell and material handling resources. These LPs negotiate with parts for a commitment on an operation that they can execute and process these parts virtually. VirtualCellHolon LPs are represented by the visual interfaces of their components. In Appendix D.5, D.6 and Appendix D.7 screen shots for MainScreen, OrderScreen and CommunicationManager components can be seen respectively.

VirtualARRSHolon enables part receival and shipment in the manufacturing system. It does not have any virtual processor but allocates a virtual ASRSRobot for part transfer from cell to material handling devices and vice versa. Parts arrive to manufacturing system at VirtualASRSHolon LP and also they are shipped at this LP.

OrderScreen and CommunicationManager visual interfaces of VirtualASRSHolon LP and VirtualCellHolon LPa are same, but VirtualASRSHolon LP has a different MainScreen visual interface which is given in Appendix D.8.

5.1.4.2. Virtual AGVHolon LP. Virtual AGVHolon LP is a resource holon which controls AGV of the manufacturing system. Virtual AGVHolon LP has the general structure of a resource holon, but its cell manager component, namely AgvManager, is specialized in managing virtual AGV module of the proposed simulation application. Basically Virtual AGVHolon LP executes material handling tasks in the manufacturing environment and has responsibilities for detecting deadlock situations in the manufacturing system and applying deadlock recovering procedure. As a resource, this LP also enters negotiation with order holons for transfer tasks and co-operates with other resources to enable production in the manufacturing system. In Appendix D.9 and D.10 screen shots for MainScreen visual interface and OrderScreen visual interface of Virtual AGVHolon LP can be seen. Note that CommunicationManager visual interface is same with Virtual CellHolon LPs.

As it is mentioned above Virtual AGVHolon LP executes its manufacturing tasks as a resource holon. AGV related tasks can be stated as matching with orders and determining routing of the AGV. Order matching is handled with various algorithms that are derived from negotiation algorithms like *Apparent Tardiness Cost* or *Parametric* and traditional algorithms like *Nearest Pickup*. In addition to these responsibilities, deadlock detection and recovery in the manufacturing system is handled by this LP. During long simulation runs, it is observed that the system can sometimes enter deadlocks. To overcome this problem, a deadlock detection and recovery mechanism is added to system which are explained below. In an FMS deadlock occurs when none of the parts in output buffers can be transfered to their destination work centers because of the physical allocation of the input buffers of their destination cells by other parts. In this situation the system is called *deadlocked* such that any material transfer between work centers can not be done. There are various deadlock avoidance methods in the literature. Valckenaers and Brussel (2003) give some examples of these solu-

tions methodologies. Deadlock avoidance is a complex research area on its own right which is not within the scope of this study. In the proposed simulation implementation there is only a deadlock detection mechanism and a procedure for solving the deadlock situation. In the manufacturing system deadlock occurs when all the input buffers of destination cells of the parts in the output buffers, are physically occupied by other parts. Specific to facility layout of BUFAIM Model Factory an example of a deadlock situation that can be seen in the system is given in Figure 5.4.

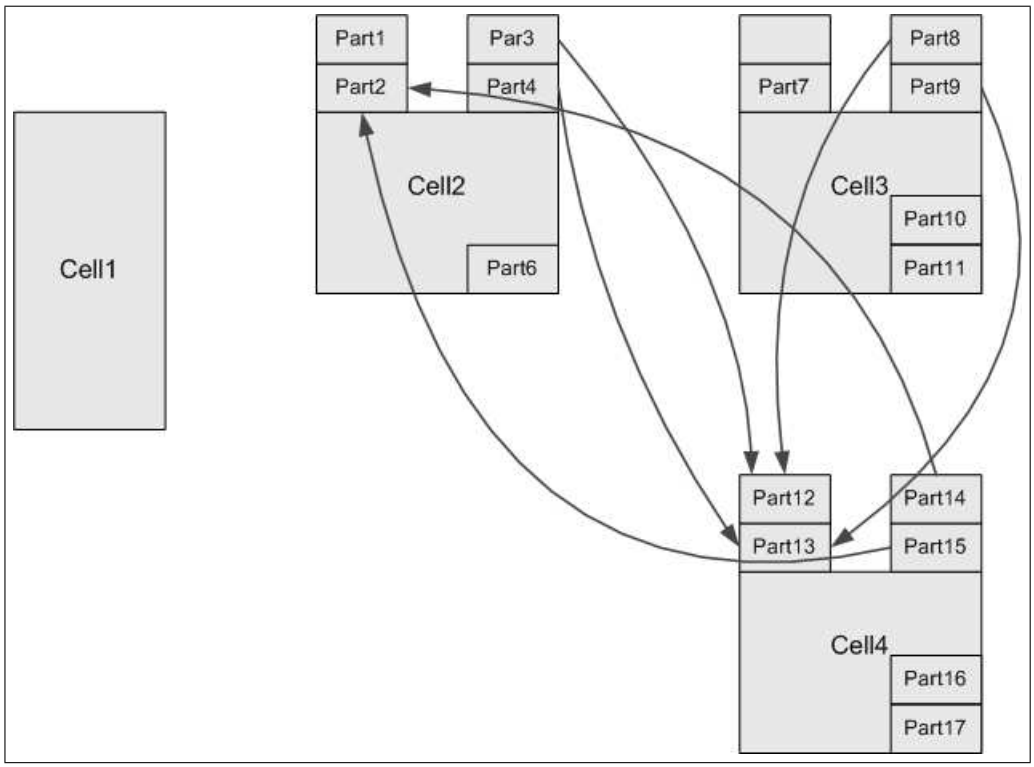

Figure 5.4. Model factory specific deadlock example

As can be seen in the Figure 5.4, all the parts in the output buffers of Cell2 and Cell3 *(Part3, Part4, Part8, Part9)* are matched with Cell4 whose input buffer is full and will not be free because, destination input buffers *(input buffers of Cell2)* of parts in its output buffer *(Part14, Part15)*, are allocated and a transfer between Cell4 and destination cells can not be done. To resolve deadlock situations such as this, Virtual ASRSHolon LP, which is the ASRS cell of the manufacturing system is used as a central buffer. Briefly, when Virtual AGVHolon LP detects a situation that for a period of time AGV could not be matched with any parts, AgvManager starts deadlock detection procedure. In a deadlock situation resource holons can not

send a *Delivery* type *Call Message* to Virtual AGVHolon LP because of allocation of their input buffers. Thus for detecting deadlock, AgvManager controls if the cause of waiting idle without any matching is the lack of active orders in the manufacturing system or a deadlock situation. Here, the allowed idle waiting time is greater than the maximum processing time among the processing times of the all operations in the job mix plus maximum transfer time of a part from processor to output buffer. The rationale behind this is giving enough time to a resource in order to process a part and transfer it to output buffer thus freeing any allocated input buffer. When idle waiting time exceeds this allowed time deadlock detection procedure is initiated. In this procedure, AgvManager simply checks if the schedule list is crowded enough for a deadlock situation. If so, AgvManager controls all order holons in the schedule list if they can be transfered. When there is not an order in the schedule list that can be transfered AgvManager starts the resolve deadlock procedure. In Appendix A.8 activity diagram for detect deadlock procedure is given. Deadlock recovery procedure is triggered when a deadlock situation deteced. In this procedure simply AGV collects as many part as its load capacity to free output buffers of the cells and unloads these parts to Virtual ASRSHolon LP. These parts are then called by their destination cells when an allocated capacity in the input buffer is freed and provided these parts are selected by the active calling algorithm. The activity diagram for resolve deadlock procedure is given in Appendix A.9.

5.1.5. Messaging Structure and Causality Error Handling Mechanisms

In this section messaging structure of the proposed P/DS application of the distributed SFCA will be explained with the causality error (CE) handling mechanisms. In the simulation implementation it is seen that *MSMQ* technology is not fast enough for a simulation study. Thus the messaging structure of the simulation implementation is changed to *TCP/IP* protocol.

5.1.5.1. Messaging Structure of Simulation Implementation. In the simulation implementation it is seen that *MSMQ* technology is not fast enough for a simulation study.

Thus the messaging structure of the simulation implementation is changed to *TCP/IP* protocol because of the speed advantage. Briefly, holonic messages of the proposed simulation implementation are transformed in to convenient format for *TCP/IP* network and are sent to destination holon(s). When a holon receives a message it decodes the message and turns it into necessary holonic object. All these messaging activities of holons are handled by *SimulationMessageController* (SMC) component of a resource holon. SMC manages all the messaging activities of resource and order holons.

Basically SMC has two main responsibilities. First one is sending messages of the resource holon and order holons residing, to their receivers in convenient format and receiving messages and decoding them into suitable holonic structure. There are five types of holonic messages in the proposed system which are used for negotiation of holons and standard communication to accomplish physical material transfer task. These message types are;

- *TaskMessage*
- *ProposalMessage*
- *TaskOfferMessage*
- *OrderMessage*
- *CellMessage*

The usage of these messages and their structures were mentioned in Section 5.1.2.1. In Appendix A.10 class diagram of SMC is given.

Every message that will be sent to an outside holon is stored in the corresponding data structures of SMC. The messages to be sent outside are stored as an holonic message object in the SMC and before sending these messages SMC encodes them into suitable format. In every timer tick SMC sends these messages to their receivers. An example message is shown in Table 5.11. Detailed structure of messages and examples is given in Appendix E.

Table 5.11. An example of message encoding-*TaskMessage*

< MessageTime > 1950 < /MessageTime >
< Name > $TaskMessage$ < /Name >
< TaskType > $Operation$ < /TaskType >
< OrderID > $Order10$ < /OrderID >
< JobType > $JobA$ < /JobType >
< OperationName > $Milling1$ < /OperationName >
< Receiver > $Cell1$ < /Receiver >

5.1.5.2. CE Handling Mechanisms. P/DS model must guarantee synchronized execution to avoid causality errors. It is worthy to remember that causality error is the affect of a future event to a past event, which is absolutely not desired. In distributed simulation, causality errors must be corrected or their occurrence must be prevented. In this study synchronization of the LPs is checked in two directions. First, a centralized barrier approach is used to prevent increase of any discrepancy among the simulation times of distributed LPs. In centralized barrier approach, every LP sends a confirmation message to a barrier controller that they have reached the barrier primitive. When all the LPs of the system reaches to barrier primitive, the barrier controller sends a release message to all LPs in the system. With this mechanism distributed LPs execute with synchronized simulation clocks in every barrier primitive. The second approach for preventing causality errors is executing messages in the time stamped order.

To implement a centralized barrier approach a synchronization controller is used which is called Synchronization Holon (SH). Synchronization holon is responsible for applying centralized barrier in the simulation environment. In Appendix D.11 and Appendix A.11 the screen shot and class diagram of SH can be seen respectively. Briefly, when an LP reaches to barrier primitive it sends a *BarrierReached* message to SH by specifying its identity and pauses its execution. SH tracks the synchronization messages and controls if all LPs in the system has reached the barrier primitive. When SH receives *BarrierReached* messages from all LPs it sends a *ResumeSim* message to all LPs to release them from the barrier primitive. An LP which received a *ResumeSim* message resumes its execution. Thus, at every barrier primitive all simulation clocks in

the system becomes synchronized. In Appendix A.12 sequence diagram of synchronization by centralized barrier with SH can be seen. In Figure 5.5 illustration of centralized barrier with SH is given.

The other method applied for synchronization is executing messages with time stamp order. To achieve this, every message is sent with a time stamp value which is controlled by SMC component of the receiver LP to decide if the message is safe to execute. When a message of a specific type is requested in simulation clock time T, SMC filters all the messages with time stamp values smaller or equal to T, and releases these safe messages. Note that every received message which has a smaller time stamp value than the current simulation time, is a causality error message. In Appendix A.13 and in Appendix A.14 sequence diagrams for adding received messages and releasing messages can be seen.

5.2. Simulation Implementation for the Layered Control Architecture

In this section simulation implementation for the layered control architecture will be explained in detail. First the general structure of the proposed simulation implementation will be given. Then logical processors of the system will be explained. Lastly, the communication structure of the system and causality error handling mechanisms will be mentioned. Note that simulation implementation of layered and distributed SFCAs share various modules. These shared modules are already described in Section 5.1.

5.2.1. General Structure of the System

Apart from some new modules and functionalities, the system structure is same as the real-time layered implementation. As explained in Section 4.3.1 the layered system is composed of controller arranged in three levels: namely a central controller at the shop level, cell controllers at cell levels and devices at the equipment level. In simulation implementation these control entities are modeled as LPs and allocates some

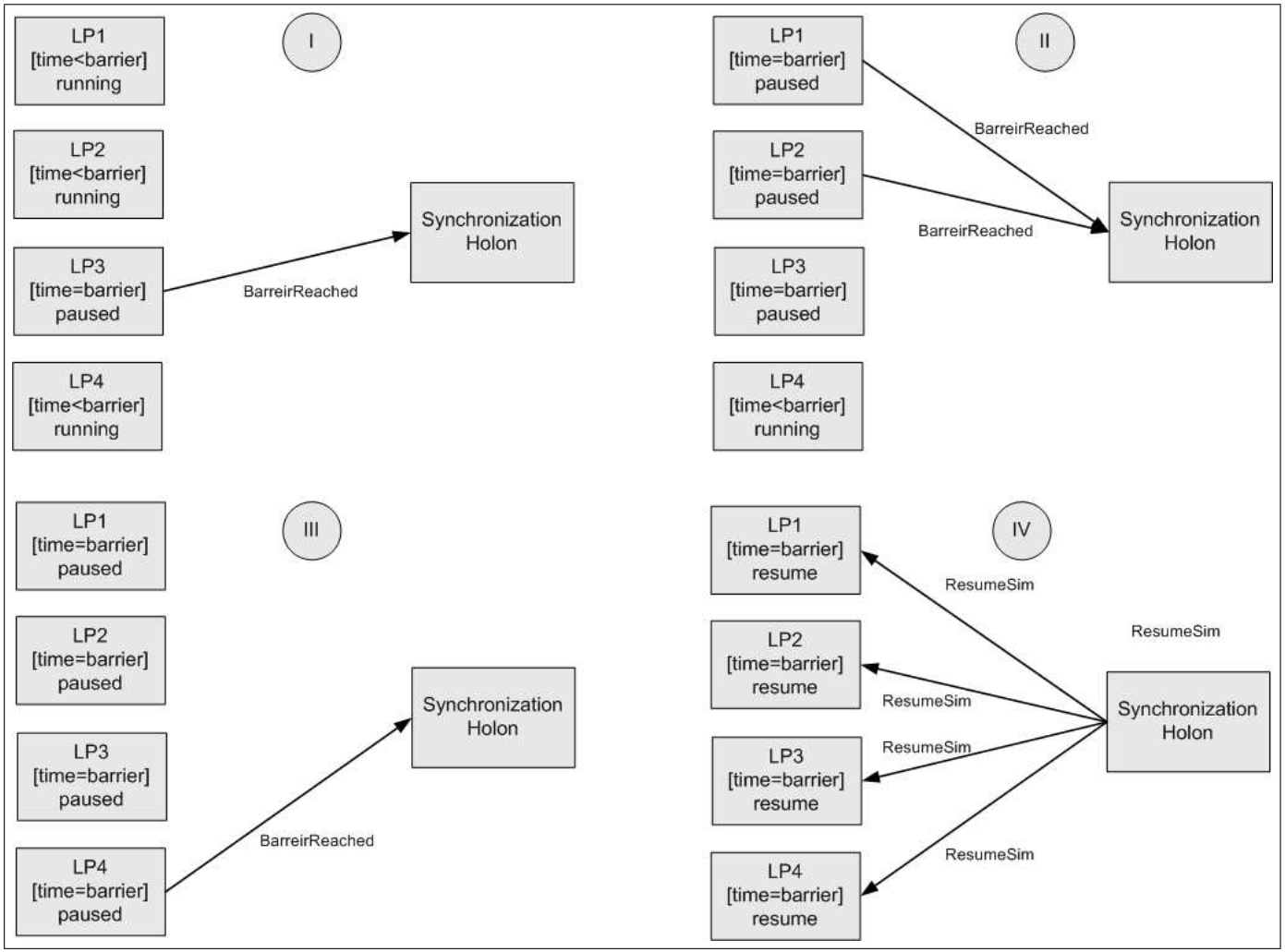

Figure 5.5. Illustration of centralized barrier with SH

of the components given above. Allocation of modules among the controllers and corresponding LPs are given in Table 5.12.

Table 5.12. Allocation of modules among the controllers

Controller	Corresponding LP	Allocated Module
Central Controller	CentralController LP	Virtual AGV Synchronization Manager
Cell Controller	CellController LP	Virtual Robot Virtual Processor
ASRS Controller	CellController LP	Virtual ASRS Robot

Time stepped distributed simulation logic is applied by using a timer programing tool in the LPs. Timer programing tool calls an event in every pre-determined equal time intervals. In the proposed implementation a timer, with a time interval equal to the time step value of the simulation, used to increase the current simulation time of the LPs by one unit, in a timer tick. In every timer tick event, resource based events and simulation based events are executed according to the execution time of the event.

5.2.2. Logical Processors of the System

In this sub-section the logical processors (LP) of the system will be explained in detail. Each controller in the real-time application is modeled as an LP in the proposed simulation implementation. Every LP in the system has some virtual devices which are mimics of the real physical devices. The virtual devices are modeled in a way that their parameters can be adjusted to enable re-configuring the layout according to the experimental needs. The detailed explanations of two LP types in the system, namely CentralController LP and CellController LPs are given below.

5.2.2.1. CentralController LP. CentralController LP is the highest level controller in the system which manages and controls the shop floor. It has two type of responsibilities which can be grouped as P/DS related and shop floor related. CentralController LP manages the production in the shop floor by sending necessary messages to Cell-

Controller LPs and VirtualAgv, in reference to the updated snapshot of the system, which is crafted by messages received from CellController LPs and VirtualAgv. All the resource allocation, dispatching decisions in the shop floor are performed by the CentralController LP via using corresponding algorithms. The class relations diagram of CentralController LP can be seen in Appendix F.1 and general structure of the CentralController LP is given below in Figure 5.6.

The shop floor related executions of the CentralController LP is same as real-time implementation, except some minor differences. During the execution of the program CentralController LP manages and controls all the system. According to the messages received from lower level controllers and AGV, CentralController LP updates the system state and creates necessary decisions for production, and sends necessary messages to CellController LPs and AGV to execute these decisions[6] . In long simulation runs it is observed that the system can enter deadlocks because of the heavy part load. The CentralController LP in simulation application, is also responsible for detecting and removing deadlock situations in the shop floor, with its knowledge about the general status of the system. CentralController LP has four main components: MainForm visual interface, SimulationManager, CentralManager and VirtualAgv. MainForm visual interface manages all the components of the CentralController LP; it creates necessary objects, gets user inputs, prepares layout, initializes other components, handles communication with other LPs and displays animation of the shop floor. The view of the MainForm visual interface is given in Appendix G.1. The execution of MainForm is similar to layered architecture implementation but there are some additional functionalities or changes in responsibilities. Initialization and finalization of simulation is handled by MainForm visual interface. The class diagram of MainForm visual interface is given in Appendix F.2. MainForm visual interface also enables a time stepped simulation with timer programming tool. In every time tick event, events related with the shop floor and P/DS are executed. P/DS related events which are barrier primitive, batch event and simulation termination is controlled by SimulationManager in every

[6]For detailed information about the layered control architecture of BUFAIM Model Factory refer to Polat (2003)

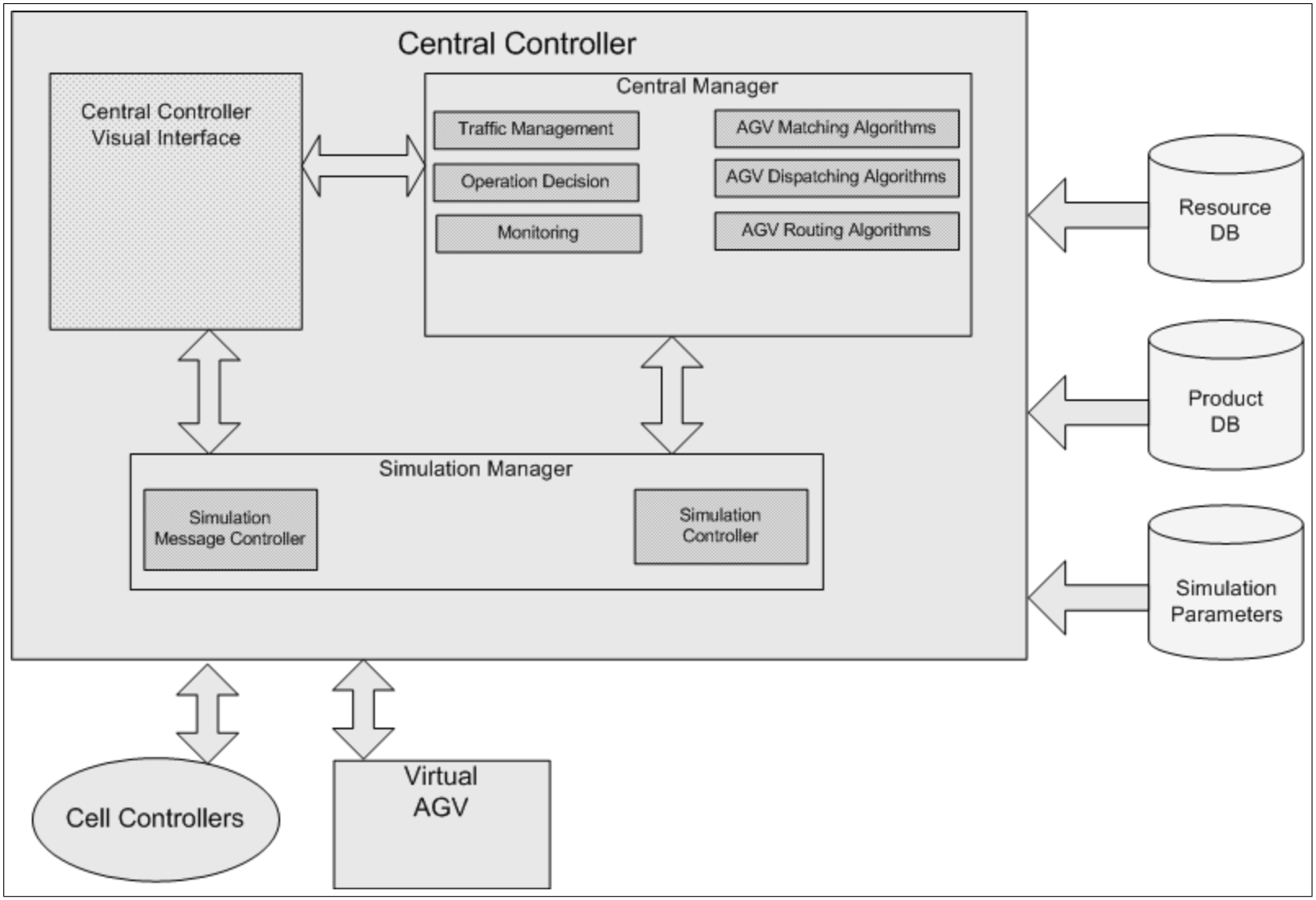

Figure 5.6. General structure of CentralController LP

timer tick with *AdvanceSimulation* procedure. Sequence diagram of timer event and *AdvanceSimulation* procedure is given in Appendix F.3 and Appendix F.4 respectively.

Another component of CentralController LP is SimulationManager which handles all executions related with P/DS. It has two sub-components SimController which is responsible for controlling P/DS events and collecting statistical data for a batch and SimulationMessageController which is responsible for controlling communication issues. The communication structure and causality error (CE) handling mechanisms of the system will be explained in detail in Section 5.2.3. Screen shot and class diagram of SimulationManager and its sub-components are given in Appendix G.2 and Appendix F.5 respectively.

The Virtual AGV represents which was controller by the AGV holon in the distributed architecture, is now controlled by the central controller in the layered approach. VirtualAgv is visually represented by a form in CentralController LP (Appendix G.3). The messaging between MainForm and VirtualAgv is enabled by RS232 connection in the simulation implementation which uses the same structure with real-time application. Necessary messages, in the convenient format, are prepared by CentralManager object, and written to RS232 port. VirtualAgv reads a specific port and when a message is received, evaluates the received command and creates and executes necessary events. With this methodology, communication structure between CentralController and AGV is modeled as close as to the real-time application. The message exchanging procedures between CentralController LP and VirtualAgv is same as the real-time application. In every message a hand shaking protocol is applied[7] . The detailed information about simulation implementation of VirtualAgv module and its procedures is given in Section 5.1.

CentralManager component has the layout and job mix information about the manufacturing system and updates the dynamic part of these data structures and takes necessary decisions by using specific algorithms. CentralManager uses an event

[7]For further information about Agv and CentralController messaging procedures refer to Polat (2003)

based execution mechanism for monitoring production[8] . There are various algorithms of CentralManager for decision processes of a FMS (i.e. process order decision algorithms, AGV matching algorithms), which were given in Section 4.2. The execution related responsibilities of CentralManager is similar to real-time layered simulation implementation. Brief explanations of the algorithms which are used in this thesis are given below:

Operation Decision Algorithms:

- *Earliest Expected Finish Time:* Considers all the expected finish time of the all alternative operation of a unit load and selects the alternative operation with the smallest expected finish time.
- *Smallest Queue Workload:* Considers work load of the candidate input buffers of all alternative operations of a unit load and selects the alternative operation with the smallest input buffer work load.
- *Smallest Queue Workload with Schedule List:* Considers work load of the candidate resource of all alternative operations of a unit load with considering the scheduled unit loads of candidate resource and selects the alternative operation with the smallest input buffer work load.

Routing Algorithms:

- *Shortest Path:* Shortest route between location and destination is selected.

Dispatching Algorithms:

- *Nearest Active Station:* Delivery stations of the unit loads on the AGV and pickup stations of the assigned unit loads are considered and the nearest station is selected as the destination.

[8]In the real-time application of the layered SFCA The structure for managing shop floor has an event based logic similar to event driven simulations . Received messages from CellControllers creates events for CentralManager and CentralManager sends various messages with the execution of events. Again refer to Polat (2003) for detailed information of event structure of layered control implementation of BUFAIM

Matching Algorithms:

- *Nearest Pickup:* The unit load on the nearest output buffer is selected.
- *Apparent Tardiness Cost:* The unit load with the maximum apparent tardiness cost index is selected.
- *Parameter:* The unit load with the maximum parameter value (which is multiply of three different sub-parameters) is selected.

Another responsibility of CentralManager is detecting and removing deadlock situations in the shop floor. In long simulation runs it is observed that the system enters deadlock which is explained in detail in Section 5.1. Briefly CentralManager controls if a deadlock situation is at hand, which is if the manufacturing system does not execute a material transfer event between cells for a specific time unit which is greater than the maximum processing time in the job mix plus the transfer time from processor to output buffer. The idea behind this is giving enough time to cells to complete their processing and free their input buffer with transferring a part from input buffer to processor. When this time limit is exceeded deadlock detection procedure is started. If a deadlock situation is detected in the system then, deadlock recovery procedure is initiated. In deadlock recovery procedure, as many parts as AGV capacity allows are loaded from output buffers of cells to AGV and transfered to ASRSController LP to free allocated output buffers. Activity diagrams of deadlock detection and recovery procedures are given in Appendix F.6 and F.7 respectively.

5.2.2.2. Cell Controller LP. CellController LPs mimic the cell level entities of the real-time layered control implementaiton of BUFAIM. CellController LPs are commanded by CentralController LP but, handle all the resource related tasks in the cell domain. CellController LPs are similar to real-time application; they receive messages from CentralController LP and according to these messages execute necessary operations within the cell. It is needless to say that the physical components of real-time layered control architecture are modeled in the simulation application as virtual modules which

are explained in detail in 5.1. The class relations diagram is given in Appendix F.8. The general structure of a CellController LP can be seen in Figure 5.7.

There are two main types of CellController LP in simulation implementation of layered control architecture. The first one is processor cell controller and the second one is ASRS cell controller. Processor cells can perform operations whereas ASRS cell enables receival and shipment of the parts. Virtual ASRSController does not have a processor but a virtual ASRS robot whereas virtual CellController LPs can allocate virtual processors and virtual robot. The structural differences come with the messaging structure of LPs and virtual modules, apart from these differences two LPs are similar. The communication structure between virtual modules and CellController LPs are modeled as close as possible to real-time application which is explained in Section 5.1. In additioto this virtual module configuration of the CellController LPs are similar to virtual holons which is explained in Section 5.1.

As it can be seen in Figure 5.7 there are two main components of CellController LP; virtual component and cell controller component. Cell controller component has three components which have different responsibilities. MainForm component coordinates all other components, gives visual output for the user and homes programming tools. Class diagram of MainForm is given in Appendix F.9. Also time stepped simulation is enabled by MainForm with a similar methodology of CentralController LP. A timer programming tool is used with a time interval equal to time step which executes a set of events related with resource and P/DS in every timer tick. In Appendix F.11 sequence diagram for timer tick event can be seen. Resource related executions of a CellController LP are managed by CellManager component. CellManager component executes necessary events for enabling processing and transfer of parts within the cell. The class diagram of CellManager component is given in Appendix F.10.

The last component is SimulationManager which handles P/DS related executions of cell controller which are barrier primitive, batch event and simulation termination. The strucutre of SimulationManager is same with the SimulationManager component of CentralManager LP. It has two sub-component; SimulationMessageController which

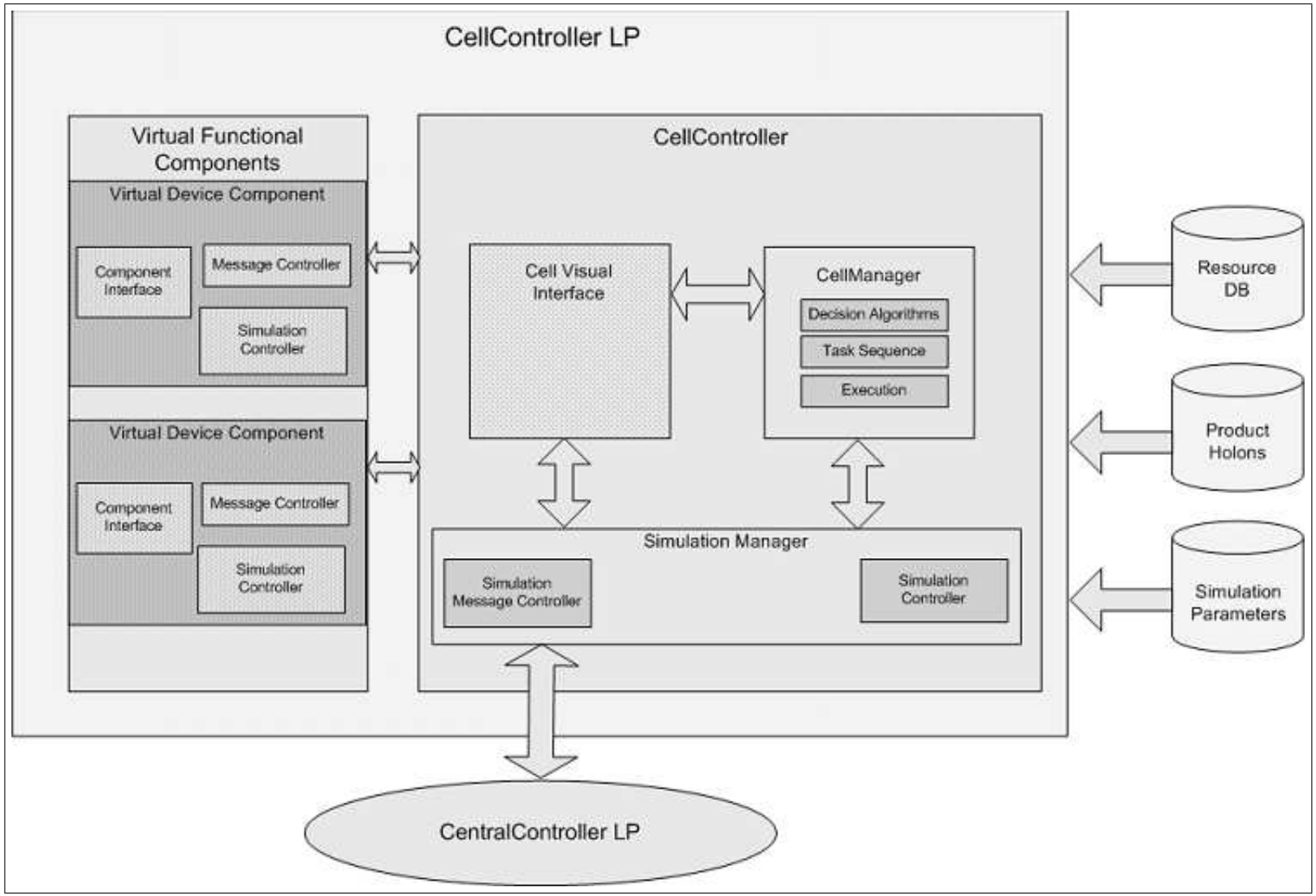

Figure 5.7. General structure of CellController LP

controls messages with considering P/DS issues and SimulationController component which enables P/DS related events such as barrier primitive, batch event and simulation termination. SimulationManager component has the same structure of the SimulationComponent of CentralController LP.

5.2.3. Messaging Structure and Causality Error Handling Mechanisms

In this section messaging structure of the proposed simulation implementation and an important concept of P/DS, causality error (CE) handling mechanisms will be explained in detail. The main communication structure of the P/DS application is similar to real-time layered control architecture. Briefly, in simulation application of layered SFCA, there are message exchanges between CentralController and CellController LPs such that CentralController LP sends command like messages to CellController LPs and CellController LPs send information messages to CentralController LP. For example, a transfer task of a part from output buffer to AGV, CentralController LP sends a command message to CellController LP for the task and CellController LP sends an inform message to CentralController LP when it completes the task. In this section additional functionalities added to the communication mechanisms for P/DS considerations will be explained[9] .

To implement centralized barrier approach in layered simulation application, CentralController LP is used as the barrier controller. Briefly in every time tick SimulationManager component of CellController LP controls if simulation clock is reached to barrier primitive. When CellController LP reaches to barrier primitive, it pauses its execution and sends a *BarrierReached* message to CentralController LP. When SimulationManager component of CentralCOntroller LP detects a *BarrierReached* message, it stores this information and tracks every *BarrierReached* messages. After SimulationManager component of CentralController LP ensures that all CellController LPs reached to barrier primitive, it sends release messages to all CellController LPs. With these release messages, all LPs starts execution with synchronized simulation clocks.

[9]For detailed information about the messaging structure of layered control architecture refer to Polat (2003)

The sequence diagram of barrier primitive procedure is given in Appendix F.12. In Figure 5.8 application of centralized barrier approach in simulaiton implementation is illustrated.

The other CE handling mechanism is related with messaging between CentralController LP and CellController LPs. As it is mentioned in previous sections CentralController LP sends command messages to CellController LPs and CellController LPs respond these messages by executing corresponding events and/or sending necessary messages. In simulation application it is not desired that an LP executes a message which has a time stamp value higher than its simulation clock. To apply this rule in this simulation application all messages are sent with a time stamp value in the system and all LPs execute messages according to the time stamp values of messages. These functionalities are handled by SimulationMessageController module both in CentralController LP CellController LPs which is a sub-component of SimulationManager. When a *TCP/IP* message is received by MainForm visual interface, proper functions of SimulationMessageController is called. Also CentralManager (or CellManager in CellController LP) gets messages for execution from SimulationMessageController, which filters messages according to their time stamps. In fact SimulationMessageController stores every received message and releases them only the simulation time reaches the message time. In other words, if a message has a higher time stamp value than simulation clock, the message is released to CentralManager when simulation clock reaches that time step value. On the other hand if the message has a time stamp value lower than the simulation clock of the LP then this message is identified as a *causality error* and stored. The sequence diagrams for receiving and execution of messages are given in Appendix F.13 and F.14 respectively. To minimize occurrence of any causality errors, the best values of barrier primitive and time step value must be determined.

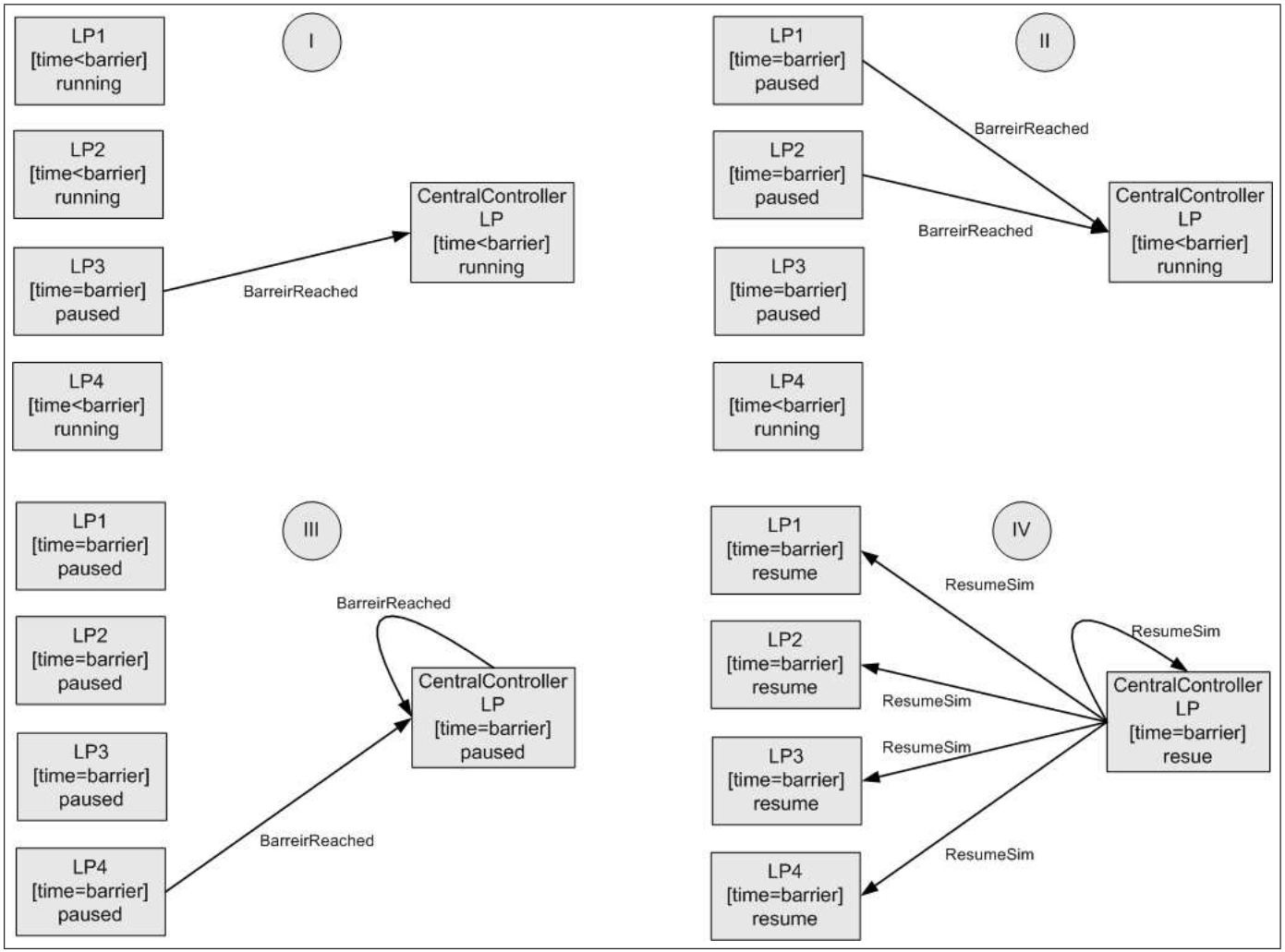

Figure 5.8. General structure of CellController LP

5.3. Shared Databases

In the simulation implementation there are two main types of data; shop floor related and simulation related data. Shop floor related data are used for configuring the layout of the shop floor and determining the product types routes and operations. Simulation related data include specific simulation parameters like time step value or barrier interval and information about virtual module configuration of cells. Simulation implementations of the layered and distributed SFCAs uses the same type of databases for storing and retrieving these.

5.3.1. Shop Floor Databases

Shop floor database is stored in two databases, layout and job mix database which are *Extensible Markup Language* (XML) type shared databases. Layout database stores information related with the facility layout like AGV routes, nodes, lanes, work centers and AGVs. Job mix database holds information about product types their definitions, process routes and operations. For more information about configuring system with these layout and job mix parameters refer to Gönen (2005). With this initialization structure the system can be configured easily by changing either or both *XML* based database. Adding a new operation or changing capacity of an output buffer can be done with small modifications in the related database.

5.3.2. Simulation Related Parameters

The other group of data are simulation related parameters which are for enabling P/DS. Simulation related parameters are stored in another *XML* based database. First group of data in the simulation parameters are related with the run of the simulation which are time step, barrier primitive and total simulation time. The other group stores information about the virtual devices of the resource holons. In the simulation implementation resource holons get P/DS data and initialize virtual components with respect to the simulation parameters database. In Table 5.13 P/DS related data, their brief explanations and example values are given.

Table 5.13. P/DS related data of the simulation parameters

Data	Explanation	Example Value (unit)
Time Step	Time scale factor of the simulation	0.5 (scalar)
Barrier Interval	Barrier primitive value	50 (time units)
Batch Interval	Run time for an individual batch	100 (time units)
Total Simulation Time	Total run time of simulation	1000 (time units)

Simulation related parameters are important for statistical and P/DS concerns. Batch interval and total simulation time parameters can affect statistical validity of the output results. Thus care must be taken for selecting the values of these parameters. In addition to this, time step and barrier interval values can affect model validity such that small time step values can overrun communication capacity with heavy message traffic in small time periods or long barrier interval values can cause synchronization loss between simulation clock values of the LPs. To prevent this the most suitable combination of barrier interval and time step values must be selected. It is good to decrease time step value in a simulation study, as much as possible to obtain shorter simulation run times. But with decreasing time step value, LPs in the system will loose synchronization in simulation clocks. To reduce the synchronization loss in simulation clocks a good choice of barrier interval value can be helpful, unfortunately the tuning of barrier interval will have a positive effect with a limited degree. To sum up, with a good combination of barrier interval one can decrease time step value to an acceptable level without causing too many CEs in the system. To obtain a combination of barrier interval and time step value, which does not cause too many CEs and have an acceptable simulation run execution time, a series of simulation runs are applied with different time step and barrier interval values. For layered simulation implementation Koşucuoğlu and Boyacı (2006) used this procedure and tried to find an acceptable combination for time step and barrier interval values. The same procedure is applied to distributed

simulation implementation. The two different simulation implementations gave same results for barrier interval and time step values (see Table 5.14).

Table 5.14. Parameter values that gives minimum number of CE

Parameter	Value (unit)
Time Step	0.15 (scalar)
Barrier Interval	8 (time units)

The other group of simulation related parameters are virtual module configuration of cells. In initialization LPs read this database and create their virtual modules according to the configuration given in the database. Simulation related parameters holds data about the types and number of virtual robot, virtual ASRS robot and virtual processors of a cell and configuration of the virtual modules. Data related with virtual AGV module is included in layout database. In Table 5.15 virtual modules and corresponding data included in the simulation parameters are given. Also in Appendix H.1, Appendix H.2 and Appendix H.3 examples of *XML* type database for virtual ASRS robot, virtual robot and virtual processor are given respectively.

5.4. Statistical Data Collection

Data collection and interpreting these data are, maybe the most important parts of a simulation study. Thus, performance measures of the system must be determined and these measures of performance must be collected during a simulation run. In this section, data collection methodologies will be explained in detail, statistical procedures for interpreting these data will be mentioned in Chapter 6. Basically there are three groups of performance measures that we are interested. First group is resource related performance measures, for example utilization of a processor or average load of an AGV. The other group is part related data which can be average flow time or average weighted lateness of parts. The last group is bidding related data like number of completed sessions per unit time or average completion time of a bidding session. The first two data groups are collected both in simulation implementations of layered and

distributed SFCAs. Bidding related data is only collected for distributed SFCA. Below these data groups and their collection strategies are explained.

Table 5.15. Virtual modules of simulation implementation and data related with them

Virtual Module	Related Data
Virtual Robot	Handling tasks and their durations Breakdown distribution Repair time distribution
Virtual ASRS Robot	Speed Breakdown distribution Repair Time distribution
Virtual Processor	Executable operations Breakdown distribution Repair time distribution
Virtual AGV	Speed Seize time Capacity

Resource related performance measures are collected with the *FMS.NET Statistics* objects. Basically every entity in the simulation study has necessary *Statistics* objects for collecting performance measures. For detailed information about statistical data collection methods of *FMS.NET* refer to Gönen (2005). In Table 5.16 these performance measures are given with corresponding resources.

Part related performance measures are collected with *OrderStatistics* object. The class diagram of *OrderStatistics* object can be seen in Appendix A.15. Basically *OrderStatistics* object holds *JobStatistics* objects which are used for collecting statistics related with a job type. In Table 5.17 the data that is collected by *JobStatistics* object and their brief explanations is given.

Table 5.16. Resources and collected performance measures

Resource	Performance Measures
Cell	Buffer length
	Waiting time in buffer
	Robot utilization
	Processor utilization
AGV	Number of loads on board
	Utilization
	Busy

The *Time*, *Lateness* and *Weighted Lateness* performance measures are a list of values of the parts of corresponding job type. Thus these performance measures must be turned into meaningful statistical data. For this, every performance measure is identified with a *StatisticsUnit* object which holds basic statistical values of related data. These values are;

- *mean:* Average of the performance measure
- *deviation:* Standard deviation of the performance measure
- *min:* Minimum value of the performance measure
- *max:* Maximum value of performance measure

Table 5.17. Performance measures for a job type

Statistics	Explanation
Completed	Number of parts completed of a job type
Submitted	Number of parts submitted of a job type
Time	Flow time of the completed parts of a job type
Lateness	Lateness values of the completed parts of a job type
Weighted Lateness	Weighted lateness values of late parts according to weight value of parts of a job type

In this manner, every performance measure of a job type is identified with expressive statistical values given above. These statistical expressions are stored and

calculated with *StatisticsUnit* objects. Each *JobStatistics* object holds *StatisticsUnit* for each performance measure which can be emphasized with statistical measures like average, standard deviation, minimum and maximum values. And *OrderStatistics* object contains a *JobStatistics* object for each job type in the job mix.

The last data group is used for bidding related performance measures which is only collected for simulation implementation of distributed SFCA. During the execution of the simulation run, resource holons collect bidding related data from order holons that are residing on the resource and when the simulation is completed all these data collected by resource holons are harvested by Synchronization holon and transfered into meaningful results. In the simulation study there are four basic performance measures related with bidding some of which was used by Veeramani and Wang (2004). These performance measures and their brief explanation is given in Table 5.18 below.

Table 5.18. Performance measures used for evaluating bidding scheme

Statistics	Explanation
Total Bidding Sessions	Number of negotiations completed by order holons through the simulation run
Bidding Time	The time that takes for completing a negotiation
Received Messages	Number of messages received by an order holon in a bidding session
Sent Messages	Number of messages sent by an order holon in a bidding session

Note that *Total Bidding Sessions* is an overall system performance measure, whereas others are order related. The last three performance measures are harvested according to the job types of the order holons and some specific statistical expressions are calculated for these performance measures which are same with the statistical expressions of part related performance measures. To transform these part based negotiation related data into meaningful statistical expressions, they are stored in *NegotiationStatisticsUnit* object which can hold basic data and statistical expressions related with

it. Every part based performance measure is expressed with this object. These *NegotiationStatisticsUnit* objects are held in *JobTypeNegotiationResult* objects so that for each job type all the necessary statistical expressions can be calculated. Also *JobMixNegotiationResults* manages *JobTypeNegotiationResult* objects and enables direct access to overall and job type based performance measures easily. The class diagram of objects related with data collection of negotiation based performance measures are given in Appendix A.16.

6. PERFORMANCE BASED COMPARISON OF TWO ARCHITECTURES

In this chapter performance based comparison of two architectures will be given. First the statistical approaches for obtaining results for these values will be explained. Then results for performance measures of the two simulation implementations will be given with different algorithm combinations.

6.1. Experimental Setting for the Simulation

There are two main issues related with simulation experiments. First problem that must be resolved is initial transient problem which is dealt by discarding the warm-up period of the simulation runs. In the simulation implementations *Welch's* procedure is applied for determining the warm-up period m for resource related performance measures. The test results show that a warm-up period with length 5000 simulation time units is appropriate.

The other problem is obtaining a sample of observations for the performance statistics. The first approach for this problem is known as replication and deletion approach. In this methodology a series simulation runs with length m is made and from each simulation run warm-up period l is discarded. This methodology brings a heavy computational time burden in our case in order to obtain a good sample size. The other approach Batch Means Method. With this method a run length of m is made and resulting observations are divided into n batches of length k. Note that $m = n \times k$. Then first p batches are deleted as warm-up period length l which is equal to $p \times k$. If k is selected large enough, it can be shown that resulting batches are uncorrelated. To obtain valid results for measures of performance batch length issue must be considered to prevent any autocorrelation between batches (Kelton and Law 1991). Fishman (1978) proposed a test procedure to determine a batch size which

yields uncorrelated observations. The simulation parameters obtained using this procedure are depicted in Table 6.1.

Table 6.1. Simulation parameters for batch means method

Parameter	Value (time units)
Run Length	79200
Batch Length	7200
Number of Batches	11

In order to discard the statistics that correspond to warm-up period first batch is not considered in the statistical calculations which has a length greater than the warm-up period.

6.2. The Test Problem

Two SCFAs are compared in a prepared shop floor environment with a specific job mix with using corresponding rules and algorithms. The BUFAIM model factory is selected as sample facility layout. The processor cells are configured with two type of processors with different capabilities and a robot. Shipment and receival of parts are enabled in ASRS cell. For material handling tasks between cells one AGV is used. In Table 6.2 below some basic information related with resources of the system are given:

Table 6.2. Information related with the resources of the system

Resource	Capacity	Input Buffer	Output Buffer
ASRS	2000	1000	1000
Cell1	2	2	2
Cell2	2	2	2
Cell3	2	2	2
AGV	4	NA	NA

Processor cells of the shop floor can execute various operations with their allocated virtual processors. In Table 6.3 allocated virtual processor and corresponding

executable operations of the work centers are given. Note that every operation can be processed in two different work centers.

Table 6.3. Processor configuration of cells

Cell	Processor	Executable Operations (Process Time)
Cell2	Milling	Milling2 (170)
	Turning	Turning2 (160)
Cell3	Milling	Milling1 (100)
		Milling2 (130)
	Turning	Turning1 (110)
Cell4	Milling	Milling1 (130)
	Turning	Turning1 (80)
		Turning2 (120)

In the test layout job mix there are four parts types with different operations. These parts are given in Table 6.4 below.

Table 6.4. Job mix and process plan of the test problem

JobA	JobB	JobC	JobD
Receival	Receival	Receival	Receival
Milling1	Turning2	Milling1	Milling2
Turning2	Milling2	Turning2	Turning1
Milling2	Turning1	Milling2	Shipment
Turning1	Shipment	Shipment	
Shipment			

When Table 6.3 and Table 6.4 is examined it is observed that every part has two alternatives for each operation. With this resource configuration and job mix a process flexibility is gained to some degree. For the arrival of parts to ASRS two different level of load weights are considered. In normal part load, parts enter the system according to an exponential distribution with mean 0.003, whereas in heavy part load parts enter the system according to an exponential distribution with mean 0.004. The part which

enters the system can be one of the part types with equal probabilities, in other words part type is determined by a uniform distribution within the interval [1;5]. There are two attributes of the a part which are randomly generated and used in the system. In Table 6.5 these attributes and their corresponding distributions are given.

Table 6.5. Distributions of the attributes of parts

Attribute	Distribution
Weight	U[1;5]
Due Date	N[1000;100]

6.3. Algorithm Sets

Using this test problem two different SFCAs are tested under different algorithm sets. As it was mentioned in the previous chapters different algorithm sets for different decisions can be used in the simulation implementations. In distributed SFCA there are three main algorithm types used in the system. First one is *offer evaluation* algorithms related with the matching of a resource and an order. The second one is *calling algorithms* related with the summoning of parts by corresponding resource. And the last one is AGV matching algorithms used for determining collection order of parts. In Table 6.6 below algorithms and rules that can be used for these decisions are given.

Table 6.6. Algorithms of the distributed SFCA

Offer Evaluation	Calling	AGV Matching
Slack	Slack	Slack
Arrival Time	Arrival Time	Arrival Time
Due Date	Due Date	Due Date
Alternative Cost	Processing Time	Processing Time
	Parametric	Parametric
	Apparent Tardiness Cost	Apparent Tardiness Cost
		Nearest Pickup
		Order List

In simulation implementation of distributed SFCA using different combinations of these algorithms creates different levels of information sharing. For example in *ATC* algorithm an order must share all its encapsulated information with resource holon via messaging. On the other hand some algorithms like *Process Time* does not require additional information sharing between resource and order holon. In the proposed system three levels of data encapsulation is identified. The first level is "high" level where there is no need for additional information sharing to implement corresponding algorithm combination. In "moderate" level where some extra information must be shared between order and processing resource holons in order to implement the corresponding algorithm combination. "Low" level encapsulation needs sharing of information between all resources and orders. In Table 6.7 levels and their corresponding algorithm combinations are given.

Table 6.7. Encapsulation level and algorithm combinations

Encapsulation Level	Algorithm Combination (Offer Evaluation-Calling-AGV Matching)
High	Slack(S)-Processing Time(PT)-Nearest Pickup(NP) Slack-Processing Time-Order List(OL) Slack-Order List-Order List Slack-Order List-Nearest Pickup
Moderate	Slack-Parametric(Par)-Order List Slack-Parametric-Nearest Pickup Slack-Apparent Tardiness Cost (ATC)-Order List Slack-ATC-Nearest Pickup
Low	Slack-ATC-ATC Slack-Parametric-Parametric

Note that in Table 6.7 the *offer evaluation* algorithm is fixed as *Slack* and the other algorithms proposed in Table 6.6 are not used for this decision. This is because in our preliminary simulation runs the case where more than one holon simultaneously competes for the same resource is not observed. In another set of preliminary runs, the duration of the bidding of session is extended to 40 seconds to force the resource

holon to wait orders that would ask for service within a closed time window. Still the number of times more than one order needs evaluation is very scarce (i.e. 1/80 percent of bidding sessions). Therefore it is obvious that offer evaluation algorithms will have no or negligible effect on performance and consequently it is set as *Slack* as a default algorithm.

The basic reason for this observation is as follows: In the current bidding scheme cancellation of previous contracts is not allowed therefore having a many to many assignment case where resource and order sets both contain more than one element is very purely coincidental and rare. Trying complex schemes such as those with negotiation is left outside the scope of the current study as further research, since the main aim of the thesis is to develop a test bed.

The decision sets for layered SFCA used in the simulation experiments are operation decision, AGV matching, routing and dispatching. In Table 6.8 below corresponding algorithms to these decision sets are given. Note that in layered SFCA implementation all information is available in the Central Controller. The six different combinations are given in Table 6.9.

Table 6.8. Algorithms of the layered SFCA

Decision Set	Algorithms
Operation Decision	Earliest Expected Finish Time (EEFT) Smallest Queue Workload (SQW) Smallest Queue Workload with Schedule List (SQWSL)
Routing	Shortest Path
Dispatching	Nearest Active Station
Matching	Nearest Pickup Apparent Tardiness Cost Parametric

In the simulation experiments these two SFCA will be compared with using the above decision algorithm combinations. Also different levels of encapsulation will be

evaluated with comparing algorithm combinations of distributed SFCA with each other and algorithm combinations of layered SFCA. With this methodology we are intent to see if there is a benefit of sharing information between distributed entities and a dominance of one SFCA to other with respect to performance measures.

Table 6.9. Algorithm combinations for layered SFCA

Operation Decision	AGV Matching
EEFT	NP
EEFT	ATC
EEFT	Parametric
SQW	NP
SQW	ATC
SQW	Parametric
SQWSL	NP
SQWSL	ATC
SQWSL	Parametric

6.4. Simulation Results

In this section simulation results for algorithm combinations of SFCA will be given for heavy and normal part load. Confidence intervals for the performance measures are calculated according to the procedure given by Fishman (1978) for average flow time, lateness and weighted lateness. In Table 6.10 and Table 6.11 simulation results of heavy part load is given for distributed and layered SFCAs. In Table 6.12 and Table 6.13 simulation results of normal part load is given for distributed and layered SFCAs. In the heavy load, total submitted parts in the system during the simulation run is 335 whereas it is 235 in normal load. Bold values in the tables shows the best results obtained in the test problems.

Table 6.10. Heavy load simulation results for distributed SFCA

Encapsulation Level	Algorithm Combination	Completed	Flow Time			Lateness			Weighted Lateness		
			Mean	LL	UL	Mean	LL	UL	Mean	LL	UL
High	S-OL-OL	162	14543	9184	19901	181011	175727	186296	537923	523933	551914
	S-OL-NP	173	16772	11465	22078	268227	262997	273456	779596	764980	794213
	S-PT-OL	266	6063	4357.3	7770	121284	119578	122990	121284	119578	122990
	S-PT-NP	316	3178	2258	4098	68301	67403	6919	218737	215698	221776
Moderate	S-Par-NP	**334**	2405	1988	2821	45505	45091	45918	142429	140987	143871
	S-ATC-NP	332	**2376**	1961	2792	**44420**	44011	44828	**136494**	135114	137873
	S-Par-OL	313	4441	3801	5081	99144	98490	99798	302381	300236	304527
	S-ATC-OL	299	6005	4892	7118	145710	144704	146717	429720	426071	433368
Low	S-Par-Par	325	3258	2647	3869	70080	69474	70686	211298	209206	213390
	S-ATC-ATC	317	3517	2993.3	4041	73803	73313	74292	223113	221484	224743

Table 6.11. Heavy load simulation results for layered SFCA

Algorithms Combination	Completed	Flow Time			Lateness			Weighted Lateness		
		Mean	LL	UL	Mean	LL	UL	Mean	LL	UL
EEFT-NP	209	14265	10023	18506	242123	237884	246362	242123	237884	246362
SQW-NP	288	6986	5729	8243	158979	157727	160231	502884.6	498526.3	507242.9
EEFT-ATC	274	5900	4767	7033	127147	126178	128115	397222	393616	400829
SQW-ATC	318	3861	3315	4407	85864	85324	86404.5	252767	250856	254678
EEFT-Par	290	8150	5993	10307	178162	176010	180314	544522	537247	551797
SQW-Par	290	4515	3675	5354	103218	102384	104052	293926	291414	296438
SQWSL-NP	**332**	**2559**	2061	3057	**47050**	46645	47454	**159598**	157857	161339
SQWSL-ATC	327	3007	2586	3427	62269	61859	62678	183089	181725	184453
SQWSL-Par	324	2981	2457	3506	60810	60290	61329	179478	177779	181178

Table 6.12. Normal load simulation results for distributed SFCA

Encapsulation Level	Algorithm Combination	Completed	Flow Time			Lateness			Weighted Lateness		
			Mean	LL	UL	Mean	LL	UL	Mean	LL	UL
High	S-PT-OL	227	1734	1397	2071	18817	18489	19146	57652	56603	58701
	S-PT-NP	**229**	1547	1392	1701	14288	14141	14435	44742	44227	45256
Moderate	S-Par-NP	225	**1471**	1308	1634	**12280**	12148	12412	**37308**	36951	37665
	S-ATC-NP	223	1523	1420	1626	12825	12730	12919	39682	39331	40033
	S-Par-OL	227	1610	1428	1791	15948	15739	16156	15948	15739	16156
	S-ATC-OL	225	1579	1407	1750	14505	14341	14669	44761	44233	45288
Low	S-Par-Par	225	1591	1413	1769	14805	14637	14973	44408	43894	44922
	S-ATC-ATC	223	1612	1420	1804	13464	13355	13574	44764	44233	45295

Table 6.13. Normal load simulation results for layered SFCA

Algorithms Combination	Completed	Flow Time			Lateness			Weighted Lateness		
		Mean	LL	UL	Mean	LL	UL	Mean	LL	UL
EEFT-NP	226	1735	1537	1934	18232	18042	18421	57379	56701	58056
SQW-NP	227	**1611**	1409	1813	**15530**	15343	15717	**47390**	46775	48005
EEFT-ATC	**229**	1841	1505	2177	21634	21307	21961	64372	63411	65334
SQW-ATC	225	1666	1447	1886	16691	16493	16889	50509	49822	51196
EEFT-Par	228	1809	1543	2075	20494	20237	20752	62464	61651	63276
SQW-Par	212	1641	1409	1872	15421	15209	15634	45349	44718	45979
S-PT-NP	316	3178	2259	4098	68301	67404	69199	218738	215699	221777
SQWSL-NP	332	2559	2062	3057	47050	46646	47455	159599	157857	161340

The results for heavy load shows that high data encapsulation level algorithm combination *Slack-Processing Time-NP* for distributed SFCA outperforms all the algorithm combinations of layered SFCA except *SQWSL-NP* for flow time, lateness and weighted lateness performance measures. In addition to this, algorithm combination also copes with low data encapsulation level algorithms for these performance measures. For completed parts performance measure, high level algorithms gives close results for best algorithm combinations of low level algorithms of distributed SFCA and layered SFCA. Moderate level algorithm combinations of distributed SFCA outperformed all other algorithm combinations for all performance measures. Also note that *SQWSL-NP* algorithm combination of layered SFCA gave the best results among the algorithm combinations of layered SFCA.

The results for normal load does not give very different results from heavy load case. Because of the relaxation of work load almost every algorithm combinations of distributed and layered SFCA give close results for completed parts. For flow time, lateness and weighted lateness performance measures, moderate level algorithm combinations outperformed all others. In addition to this high level data encapsulation algorithm *Slack-Processing Time-NP* for distributed SFCA outperformed algorithm combinations of layered SFCA and gave close results to low level algorithm combinations of distributed SFCA. In addition, *SQW-Parametric* algorithm combination of layered SFCA gave the best results among the algorithm combinations of layered SFCA.

In Table 6.14 the algorithm combinations of the two different SFCAs which have close intelligence level and algorithm combinations which gave the best results are given together. As it can be seen in Table 6.14 although layered SFCA gives better results in the algorithm combinations which have close intelligence level distributed SFCA algorithms are competes in all performance measures. In addition to this when we examine the best algorithm combinations of the two SFCAs, distributed SFCA gives better results with moderate encapsulation level algorithm combinations and a high encapsulation level algorithm combination copes with other algorithm combinations.

Table 6.14. Comparison of algorithm combinations with similar intelligence level of different SFCAs.

Algorithms Combination	Completed	Flow Time			Lateness			Weighted Lateness		
		Mean	LL	UL	Mean	LL	UL	Mean	LL	UL
S-ATC-ATC	317	3517	2993	4041	73803	73314	74293	223114	221484	224743
SQWSL-ATC	327	3007	2587	3428	62269	61859	62679	183089	181726	184453
S-PAR-PAR	325	3258	2647	3870	70081	69475	70687	211299	209206	213391
SQWSL-PAR	324	2982	2457	3506	60810	60291	61329	179479	177779	181178
S-ATC-NP	332	2377	1961	2792	44420	44011	44829	136494	135114	137874
S-PAR-NP	334	2405	1988	2822	45505	45092	45919	142430	140988	143872
S-PT-NP	316	3178	2259	4098	68301	67404	69199	218738	215699	221777
SQWSL-NP	332	2559	2062	3057	47050	46646	47455	159599	157857	161340

Another important data group is bidding related results. Performance measures for bidding related results are total bidding sessions, bidding time, number of received messages and number of sent messages. Here results for heavy and normal load will be given for a specific algorithm combination. These performance measures can be used to compare two different negotiation structure which is not in the scope of this thesis. In Table 6.15 results of *Slack-Parametric-Nearest Pickup* for heavy and normal load is given.

Table 6.15. Bidding results for normal and heavy load

	Heavy Load	Normal Load
Total ♯ Bidding Sessions	1962	1411
Mean Bidding Time (Confidence Interval)	2.35 (2.22;2.47)	2.43 (2.27;2.58)
Mean Received Messages (Confidence Interval)	2.34(2.32;2.36)	2.36 (2.32;2.40)
Mean Sent Messages (Confidence Interval)	1.99 (1.97;2.01)	2 (1.99;2.01)

As it is expected total number of completed bidding sessions in heavy load is greater than the normal load. Other performance measures are very close to each other and they do not differ from each other significantly which is expected.

7. CONCLUSIONS AND FURTHER STUDIES

In this thesis layered and distributed SFCA of BUFAIM model factory are compared via a P/DS approach. All the physical components of the model factory are modeled as virtual modules and communication structure of the two SFCA are totally protected. The simulation software is constructed above the present implementations of SFCAs in a modular fashion using an object-oriented approach. Thus the improvements gained in this thesis are readily applied to real-time shop floor control implementations. Since P/DS implementations of layered and distributed SFCAs are developed and compared, we will try to examine two different SFCAs in two ways: implementation related results and experimental results.

The real-time applications of SFCAs were built using an object oriented approach. Thus developing P/DS implementations of these applications was not very difficult. Virtual modules and new classes for statistical data collection are developed for once and used for both implementations. For components that have a communication link with other components like centralized barrier controller, implementation for the simulation implementation of SFCAs differed. In layered control architecture, centralized barrier controller can not be created as a separate module because it does not belong to any of the layers. Thus the centralized barrier controller is included into Central Controller LP (which brought additional code burden for this implementation). But in distributed control architecture the module for centralized barrier approach is added as another holon which communicates with oher holons for its centralized barrier responsibilities. Note that additional messaging burden for the two applications are same.

Various algorithms are developed for for decision processes of the SFCAs. These algorithms require different levels of information sharing. In simulation implementation of layered SFCA this does not violate any architectural concerns since all the shop floor is controlled by the central controller and all the information is accessed by this controller. But in distributed SFCA, sharing additional data brings questions about

the level of the distribution of the architecture. Thus to investigate the effects of information sharing, different levels of data encapsulations are determined in terms of algorithm combinations used for decision processes. The results showed that algorithm combinations with moderate data encapsulation level give the best results compared with all algorithm combinations of layered and distributed SFCAs. In addition to this, an algorithm combination, which enables a high level data encapsulation, composed of *Slack*, *Processing Time* and *Nearest Pickup* algorithms outperformed all the algorithm combinations of the layered SFCA and coped with algorithm combinations that enables low level data encapsulation of distributed SFCA. To sum up sharing information improved performance measures of the distributed SFCA to a degree.

The developed simulation implementations can be used as test beds for various experimental purposes. Any procedures for detecting and recovering various kinds of disturbances can be developed and tested in the simulation implementations. In the thesis, already a deadlock detection and recovery procedure is developed and tested. Other procedures for break down detection and recovery can be build and tested using these simulation implementations. All these new procedures can be easily implemented to real-time software with re-using the simulation code. In addition, for both SFCA, new algorithms for various decision processes can be developed and tested using the simulation implementation. In this study new resource allocation and dispatching algorithms for distributed SFCA and AGV matching algorithms for layered SFCA are developed and tested. Also these new algorithms can be directly used in real-time control applications.

Another further research area can be developing new bidding schemes with new structures for order and resource. The bidding structure developed for the distributed SFCA can be updated to a negotiation structure which enables a multi-stage negotiation and re-negotiation between resources and orders. All these structural developments can be done by changing related modules like *CommunicationManager* and *Order Holon* without any other changes in the structure. Also all these developments can be used in real-time application of distributed SFCA after they are tested in the simulation implementation by only inserting related modules. Different negotiation

structures can be compared according to the performance measures that are used in bidding related results efficiently.

APPENDIX A: Unified Modeling Language (UML) Diagrams for Simulation Implementation of Distributed SFCA

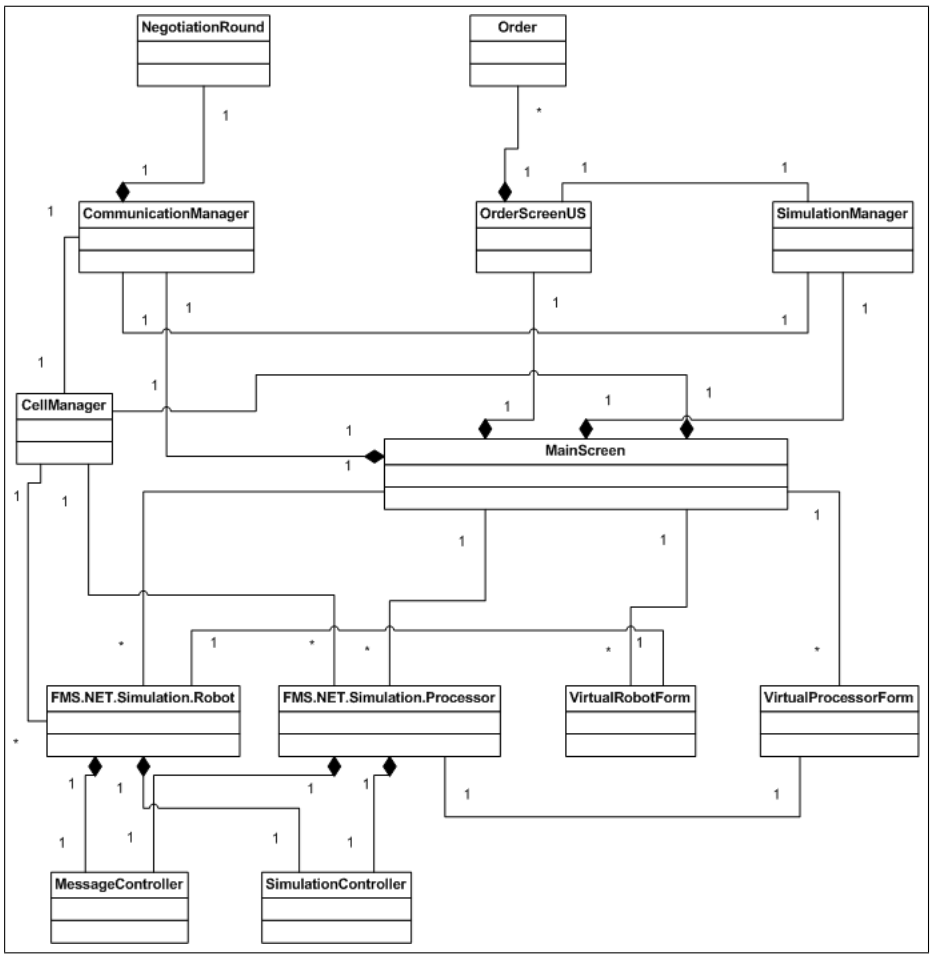

Figure A.1. Class relation diagram of a resource holon

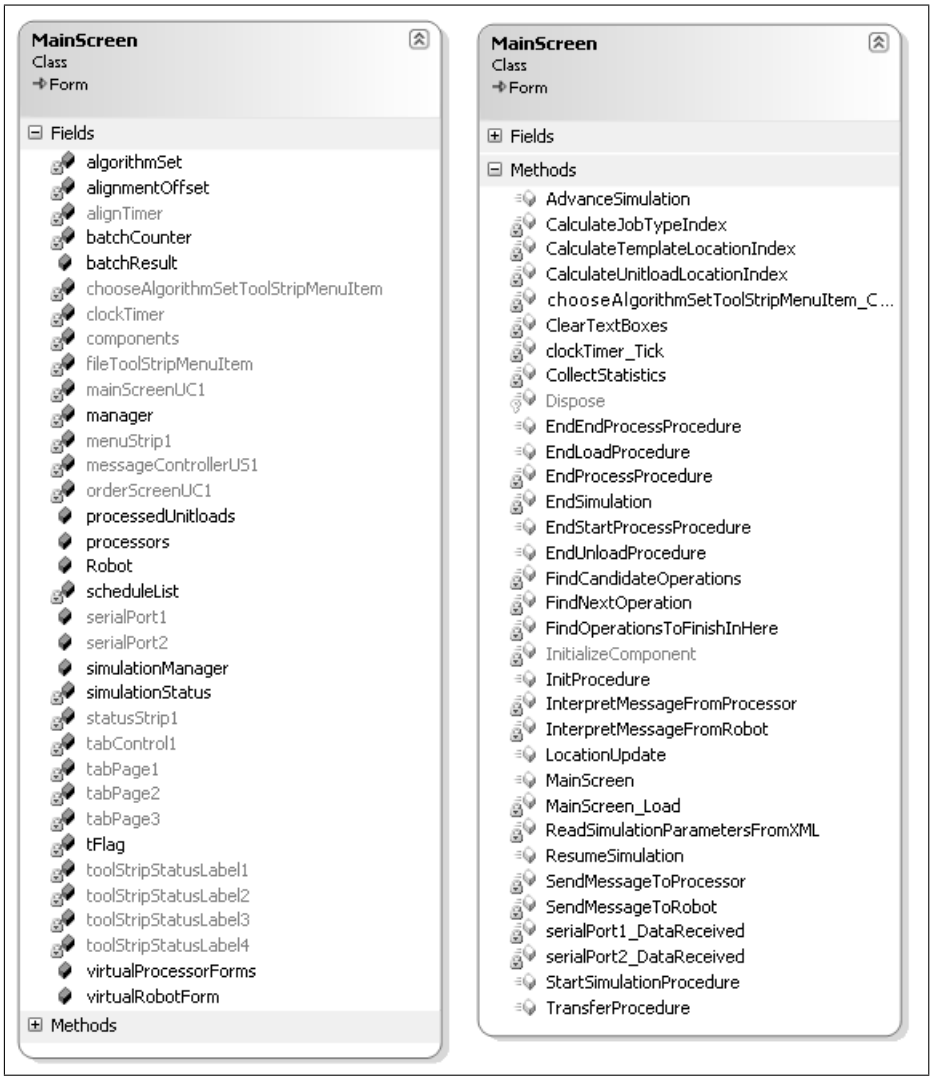

Figure A.2. Class diagram of MainScreen object

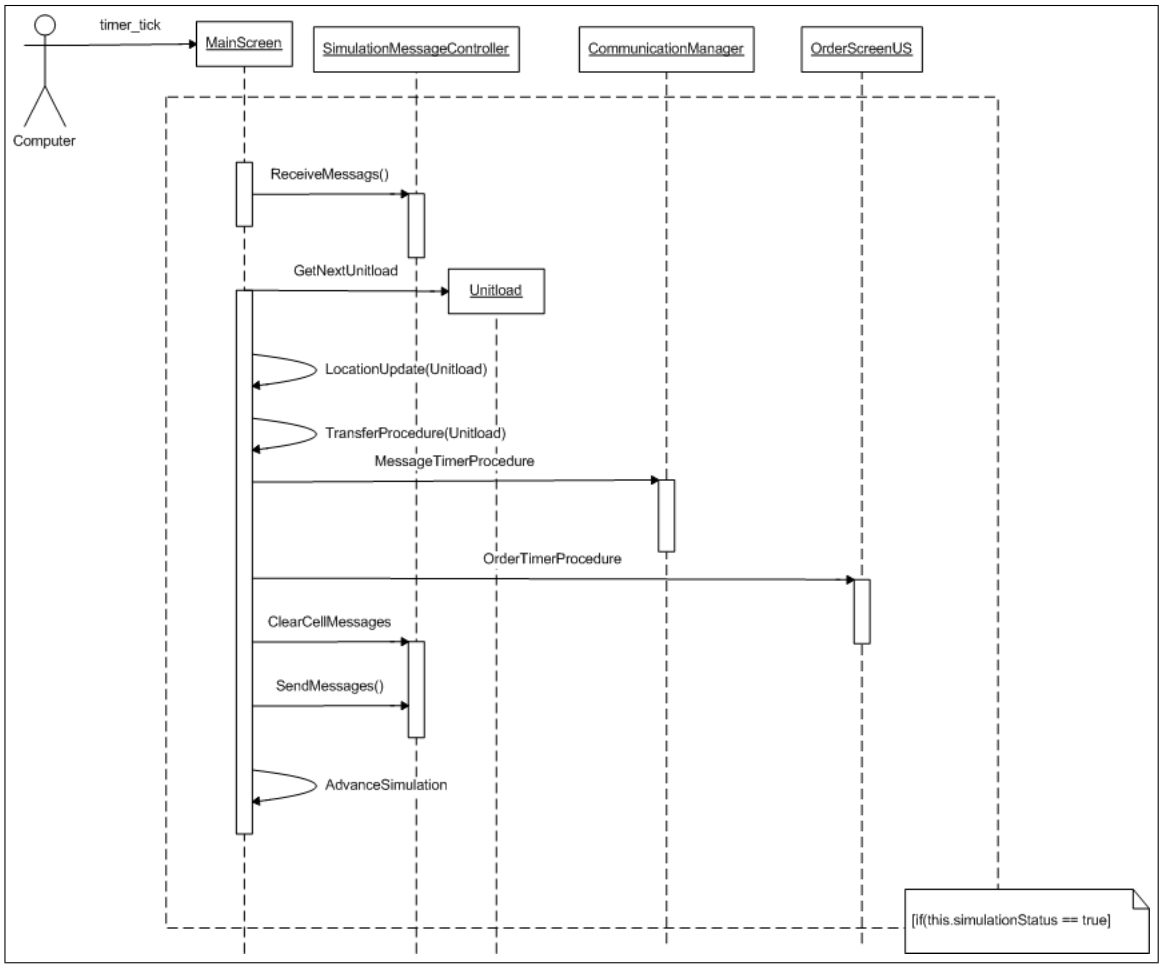

Figure A.3. Sequence diagram for tick event of timer

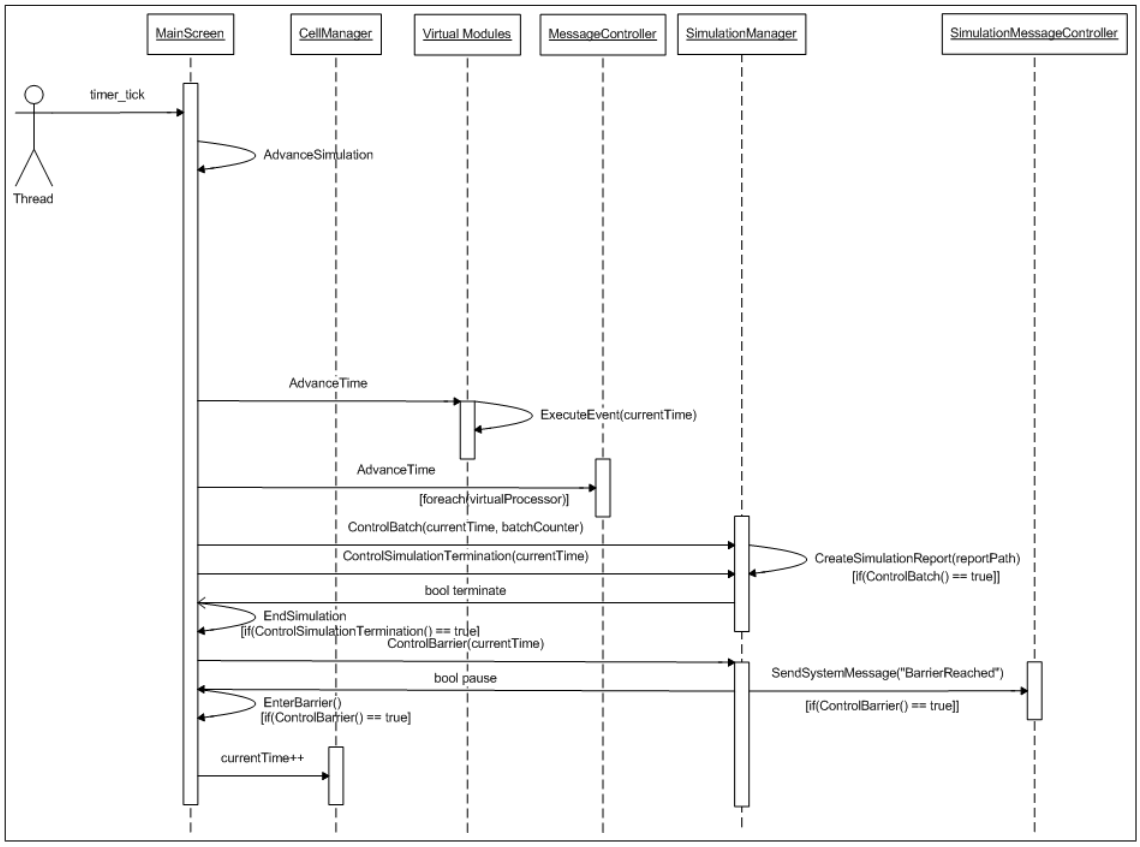

Figure A.4. Sequence diagram for AdvanceSimulation function

Figure A.5. ResourceManager and CommunicationManager class diagrams

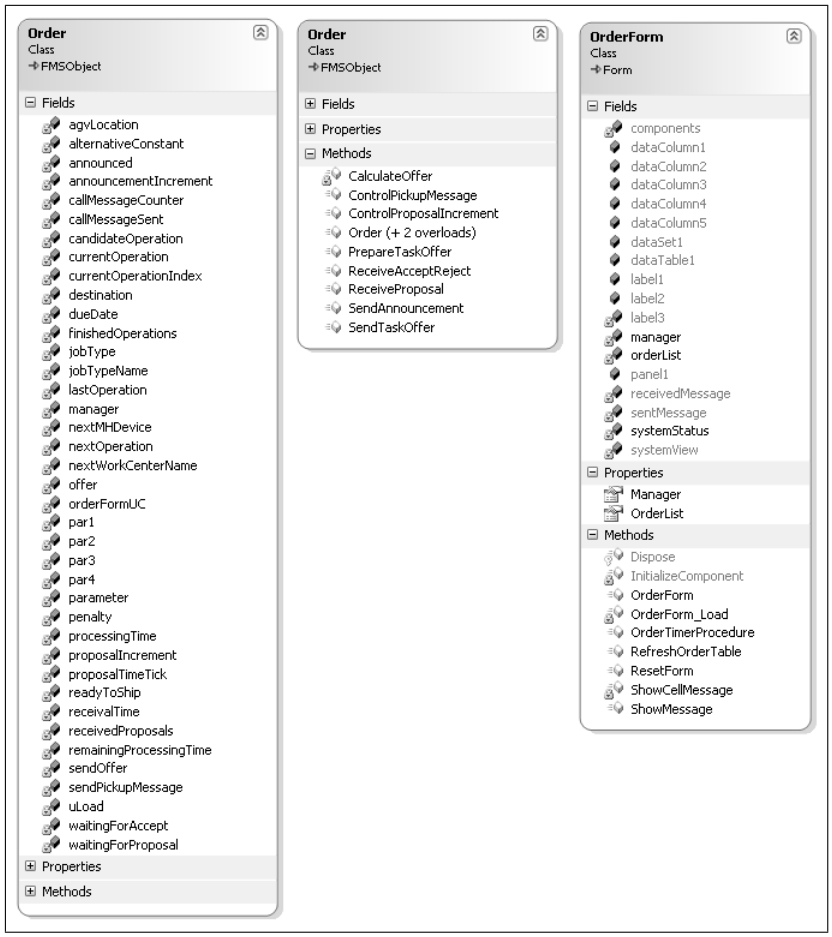

Figure A.6. OrderScreen and order holon class diagrams

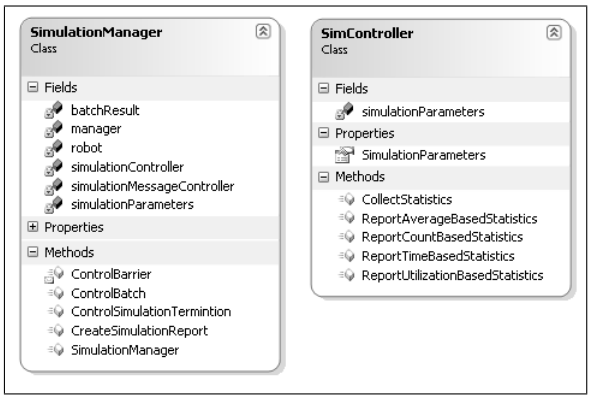

Figure A.7. SimulationManager and SimController class diagrams

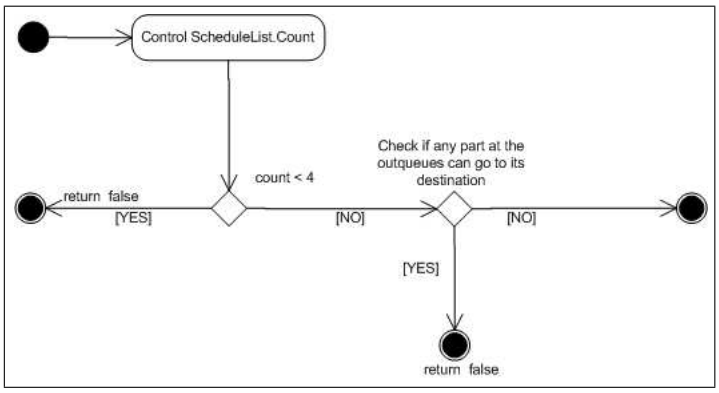

Figure A.8. DetectDeadlock procedure activity diagram

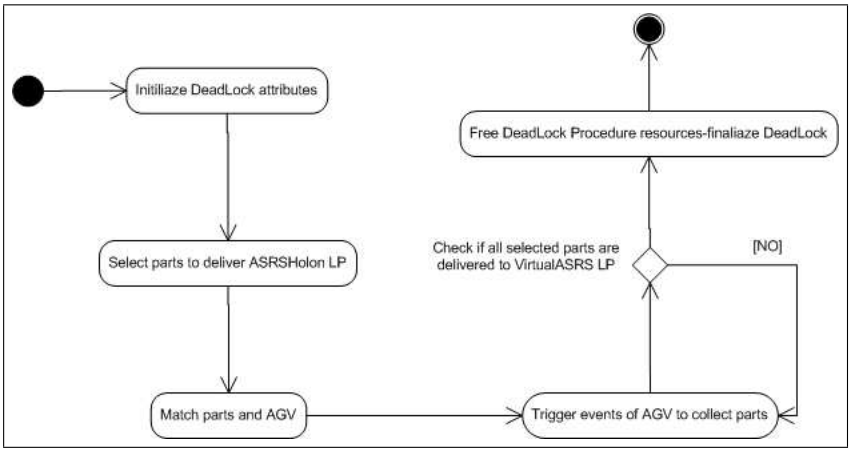

Figure A.9. Deadlock procedure activity diagram

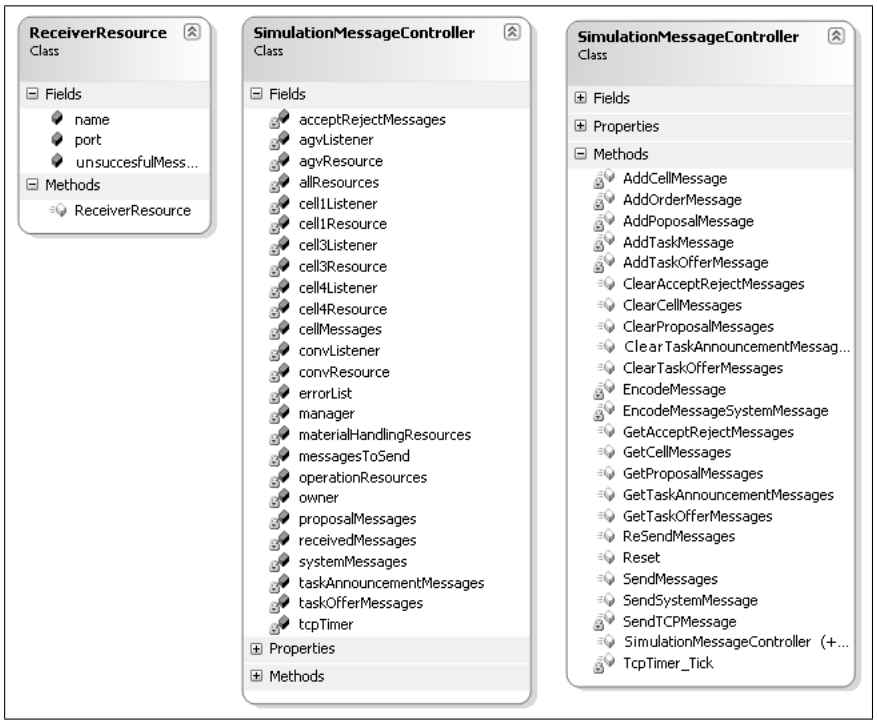

Figure A.10. SimulationMessageController class diagram

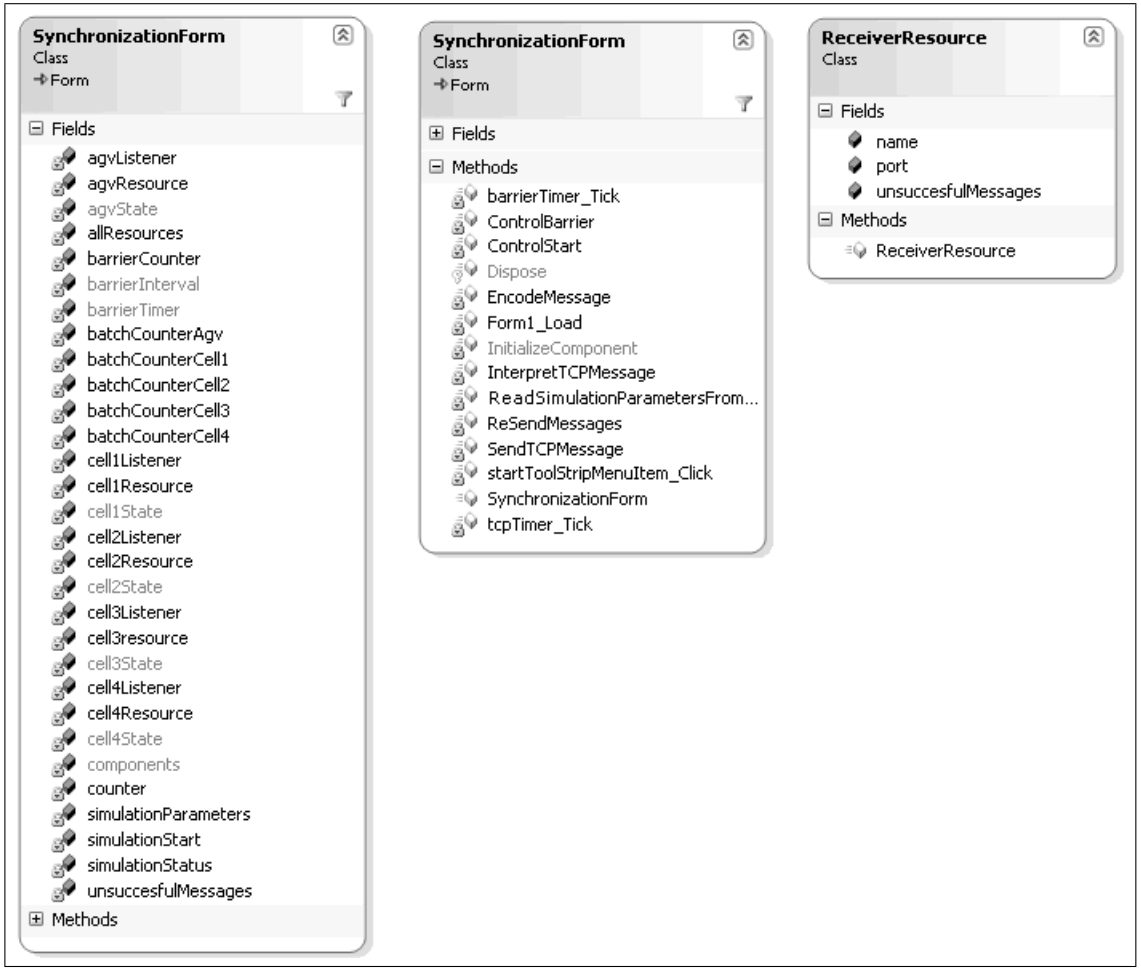

Figure A.11. Class diagram for Synchronization Holon

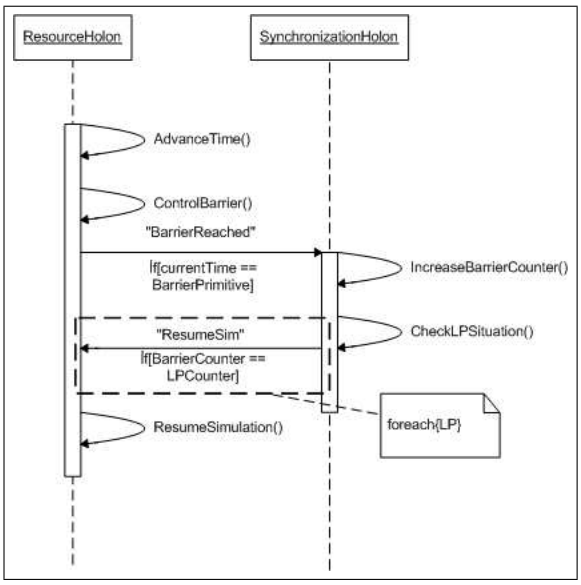

Figure A.12. Sequence diagram of barrier primitive application

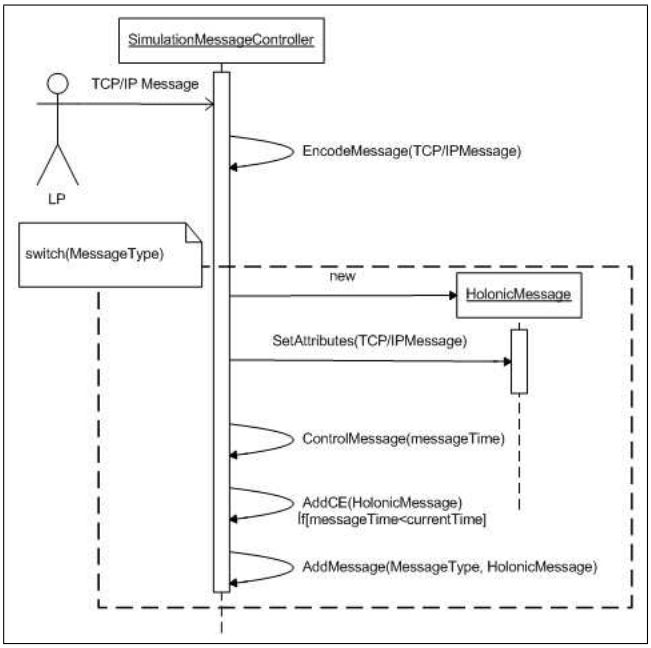

Figure A.13. Sequence diagram of storing received messages

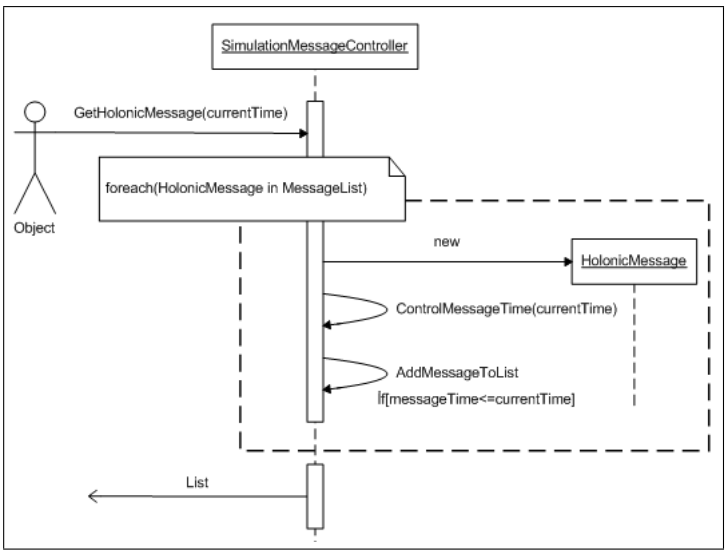

Figure A.14. Sequence diagram of releasing received messages

Figure A.15. Class diagrams for OrderStatistics and related objects

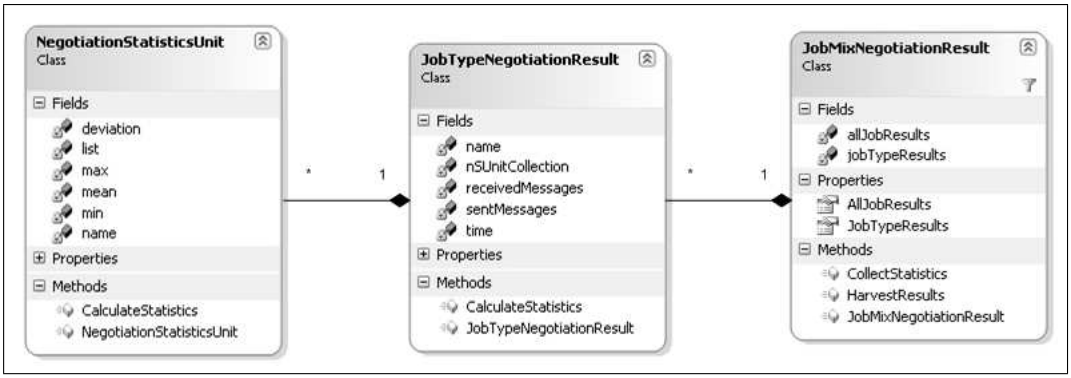

Figure A.16. Class diagram for JobMixNegotiationResult and related objects

APPENDIX B: UML Diagrams for Bidding Procedures and Bidding Related Classes

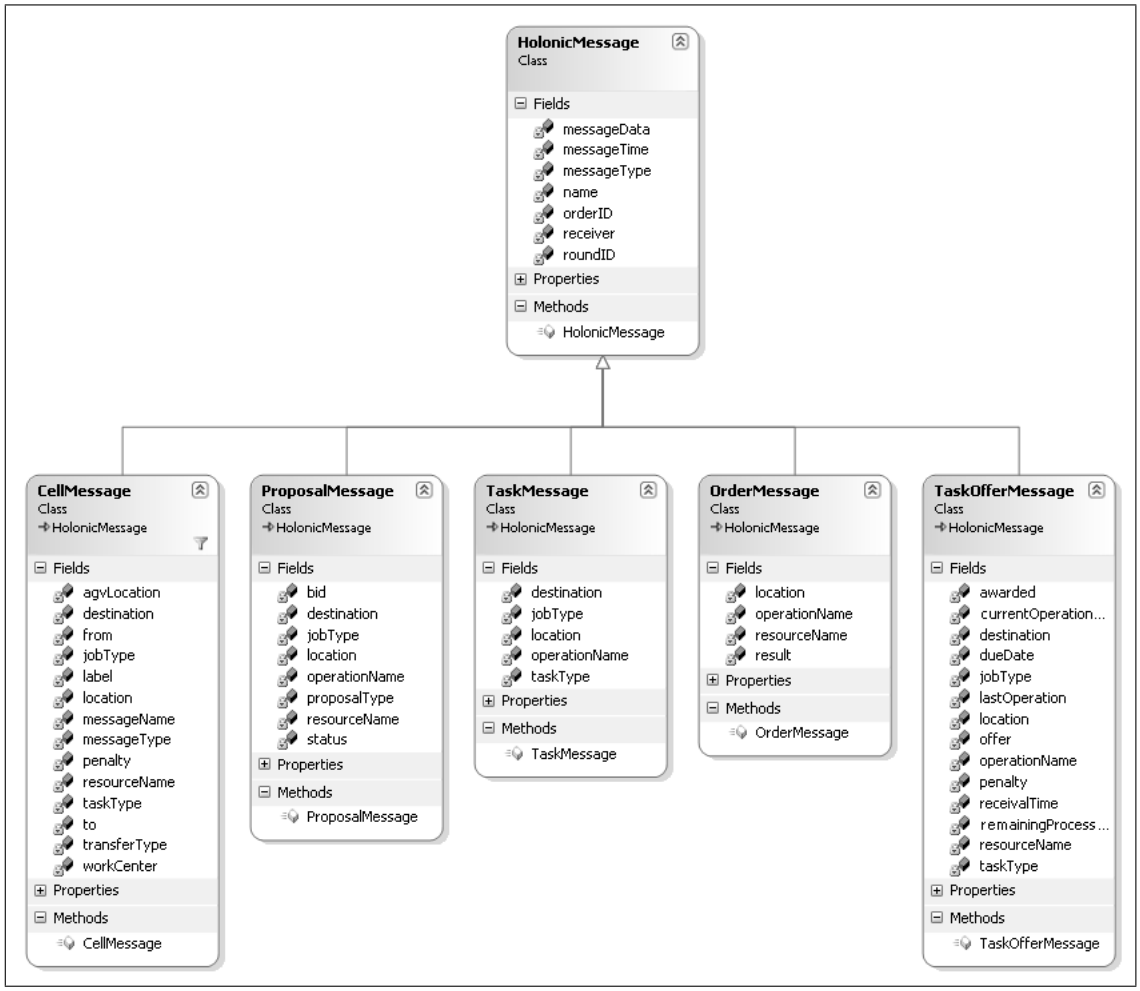

Figure B.1. Class diagrams for HolonicMessage and its derived classes

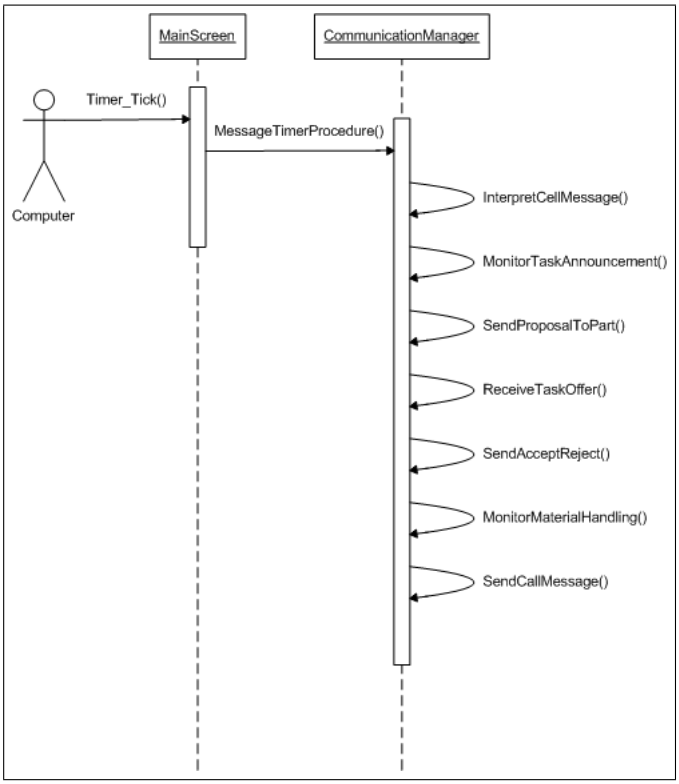

Figure B.2. MessageTimerProcedure sequence diagram

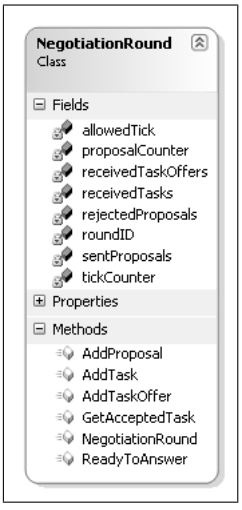

Figure B.3. NegotiationRound class diagram

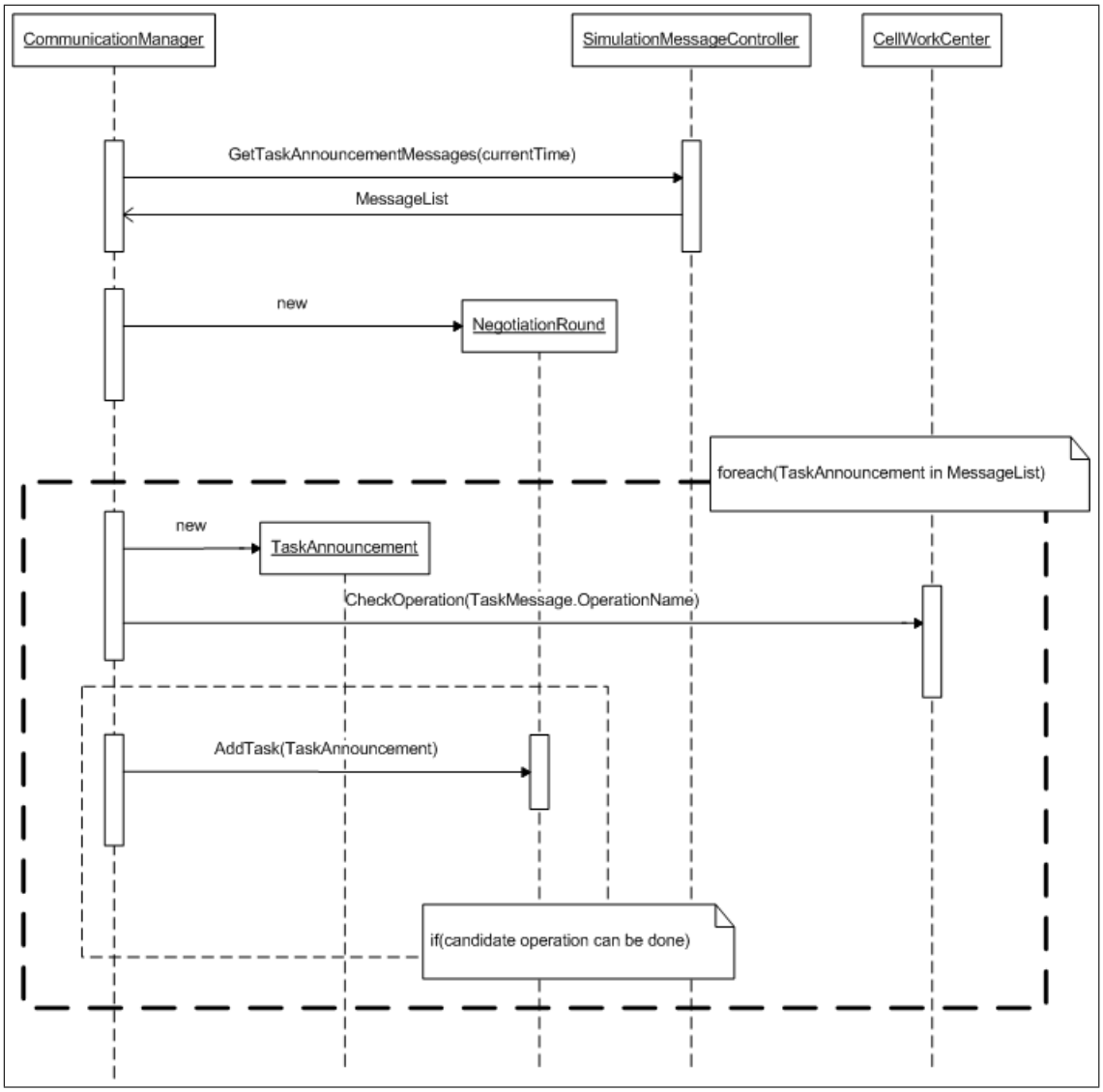

Figure B.4. MonitorTaskAnnouncement procedure sequence diagram

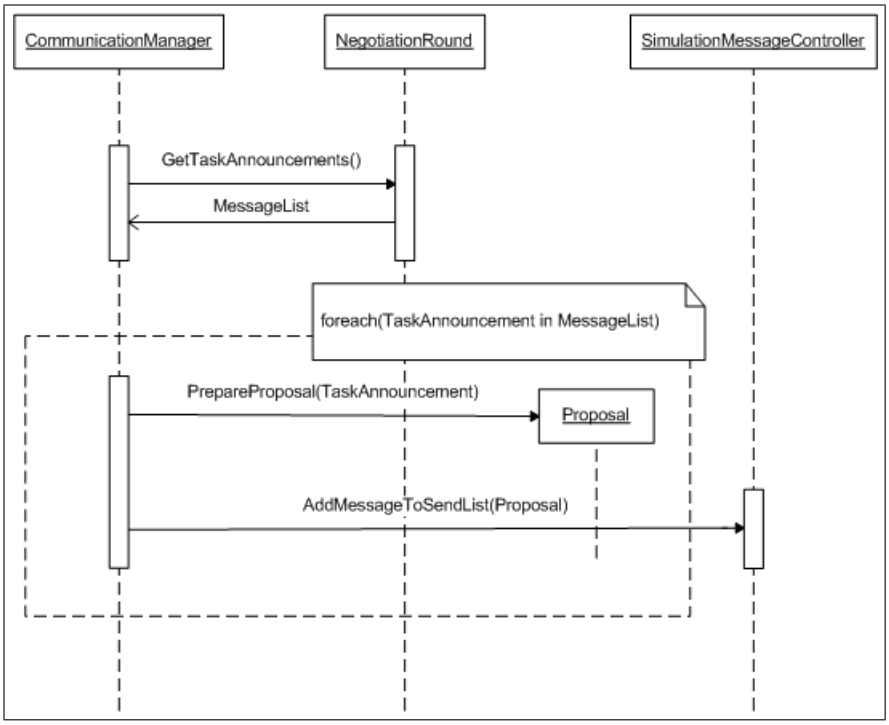

Figure B.5. SendProposalToPart procedure sequence diagram

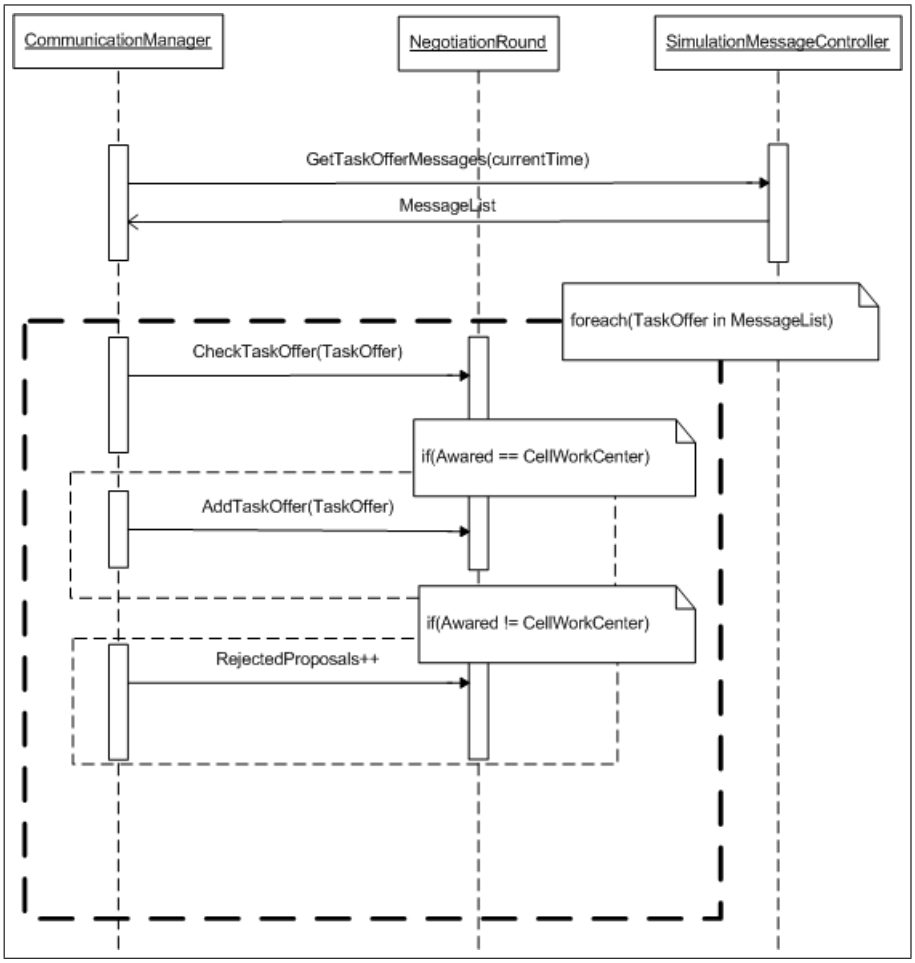

Figure B.6. ReceiveTaskOffer procedure sequence diagram

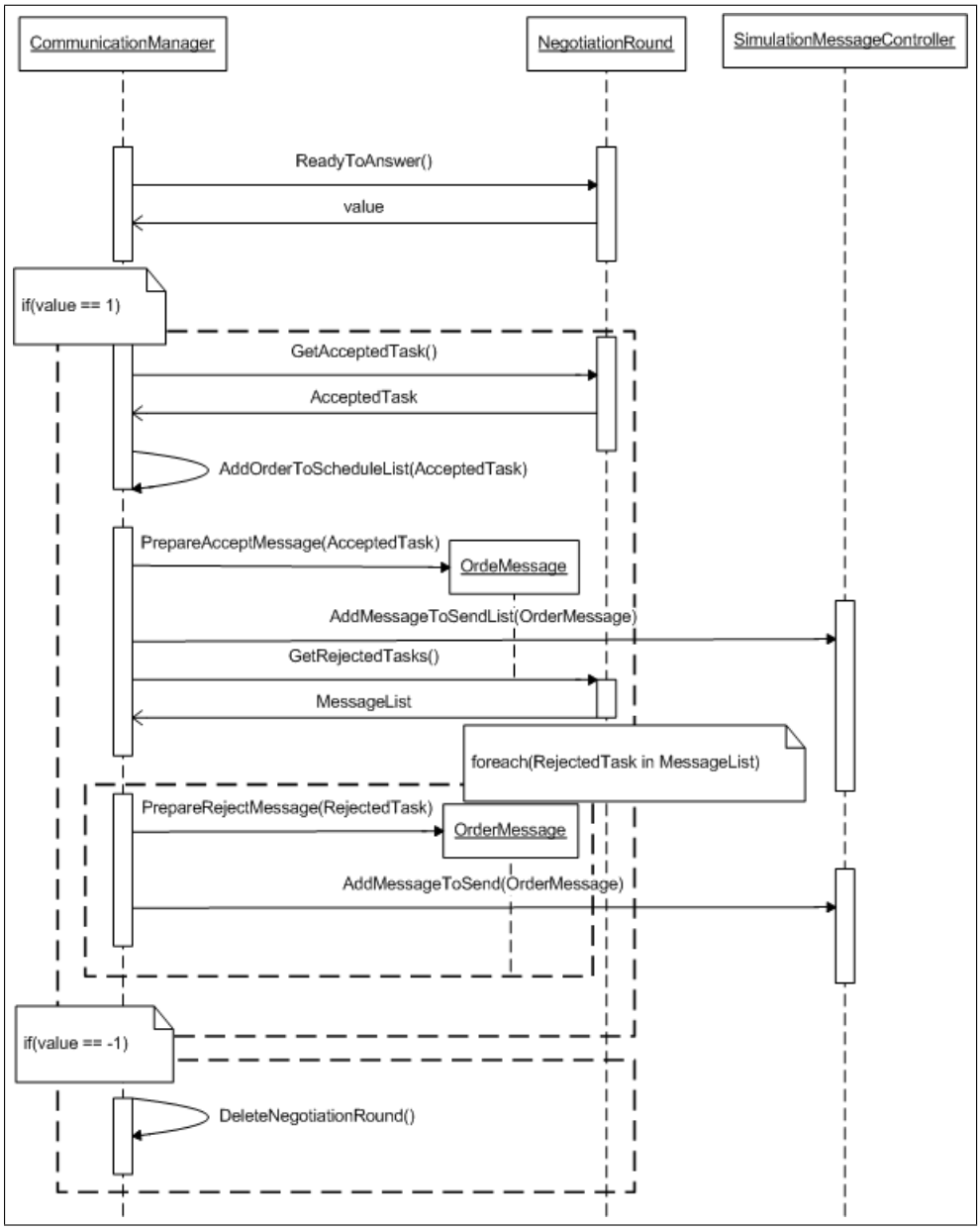

Figure B.7. SendAcceptReject procedure sequence diagram

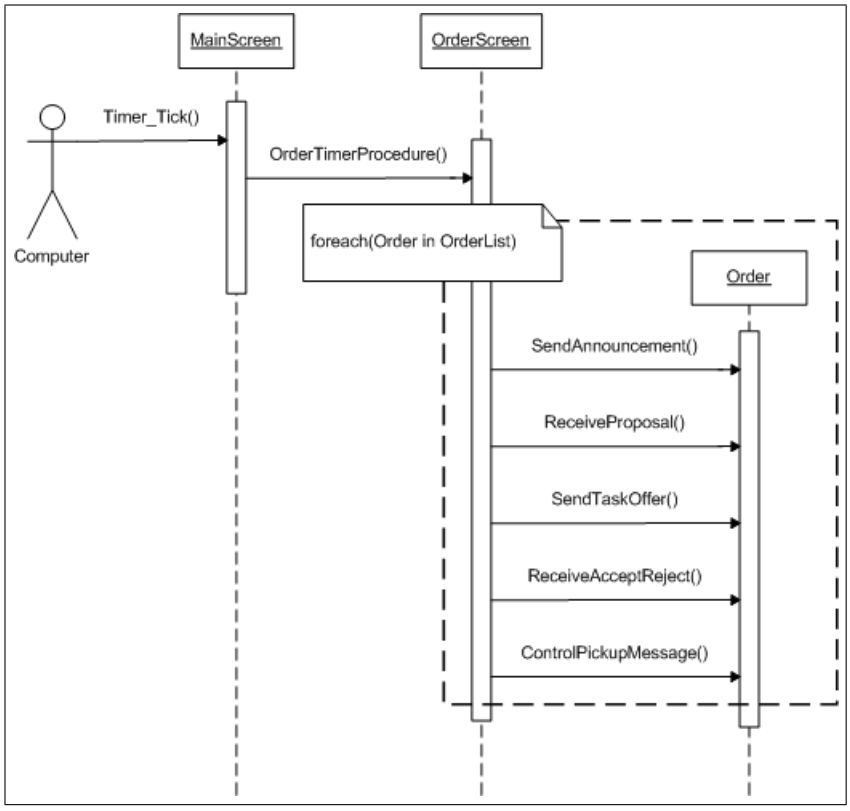

Figure B.8. OrderTimerProcedure sequence diagram

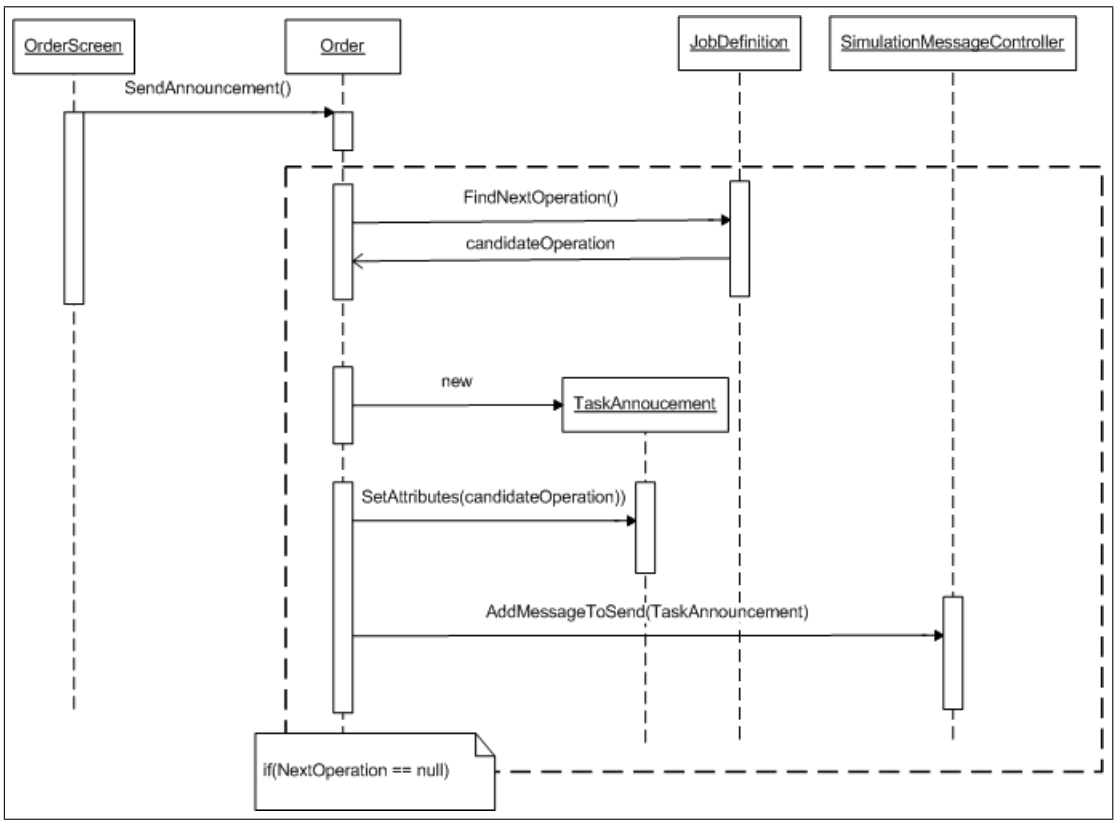

Figure B.9. SendAnnouncement procedure sequence diagram

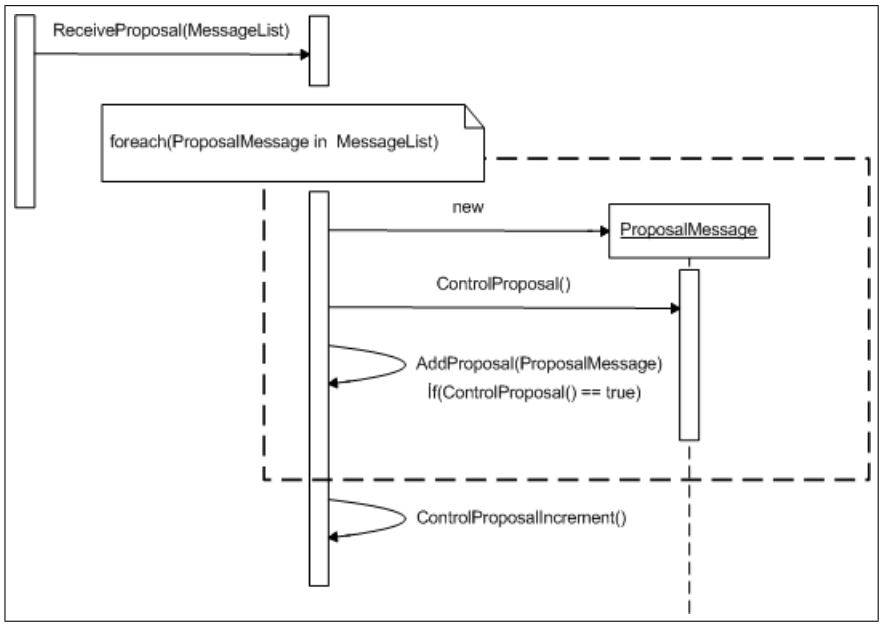

Figure B.10. ReceiveProposal procedure sequence diagram

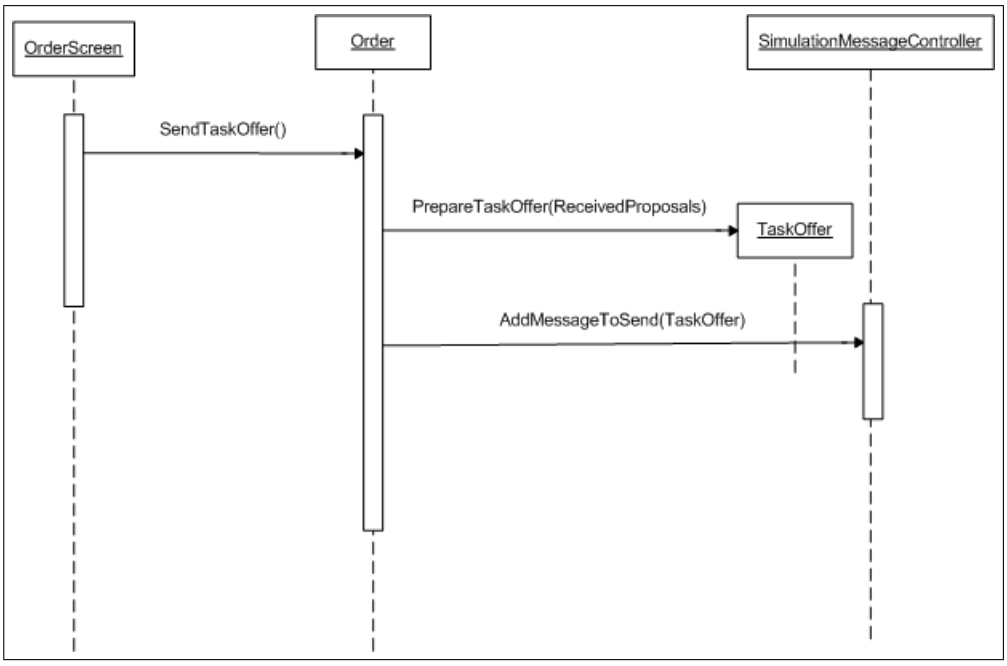

Figure B.11. SendTaskOffer procedure sequence diagram

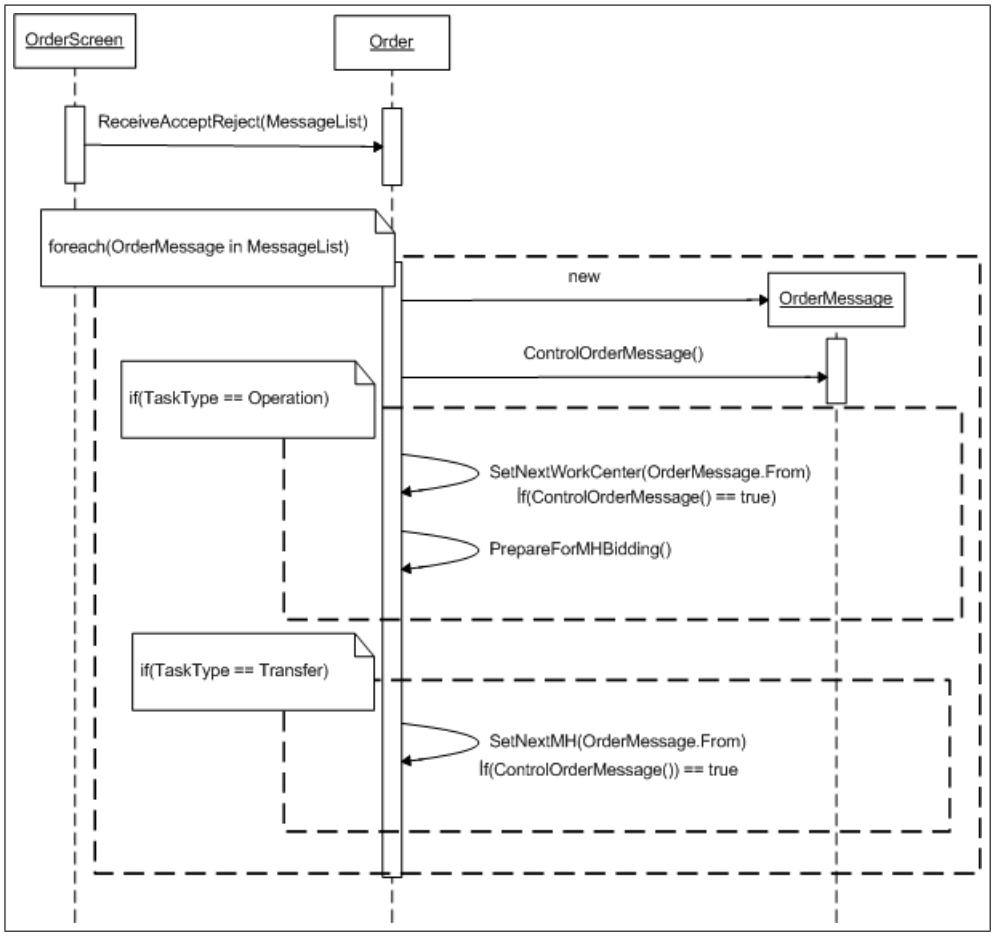

Figure B.12. ReceiveAcceptReject procedure sequence diagram

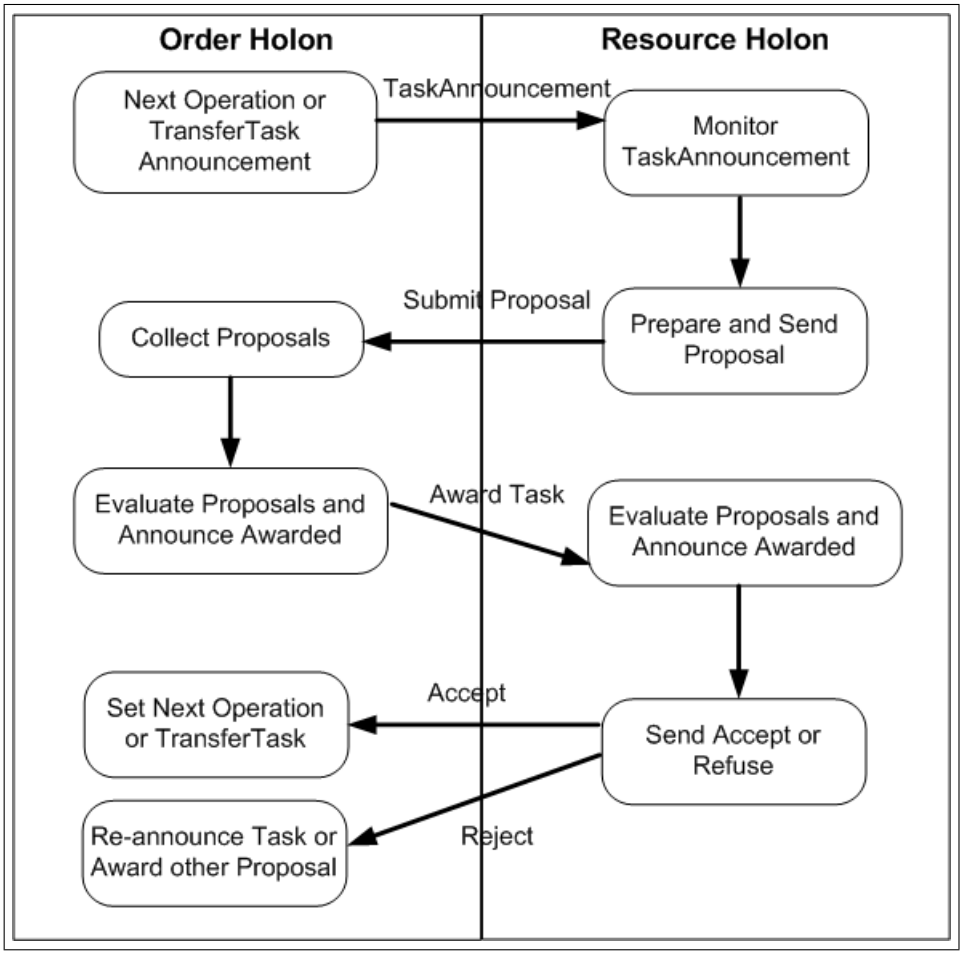

Figure B.13. Activity diagram for bidding session

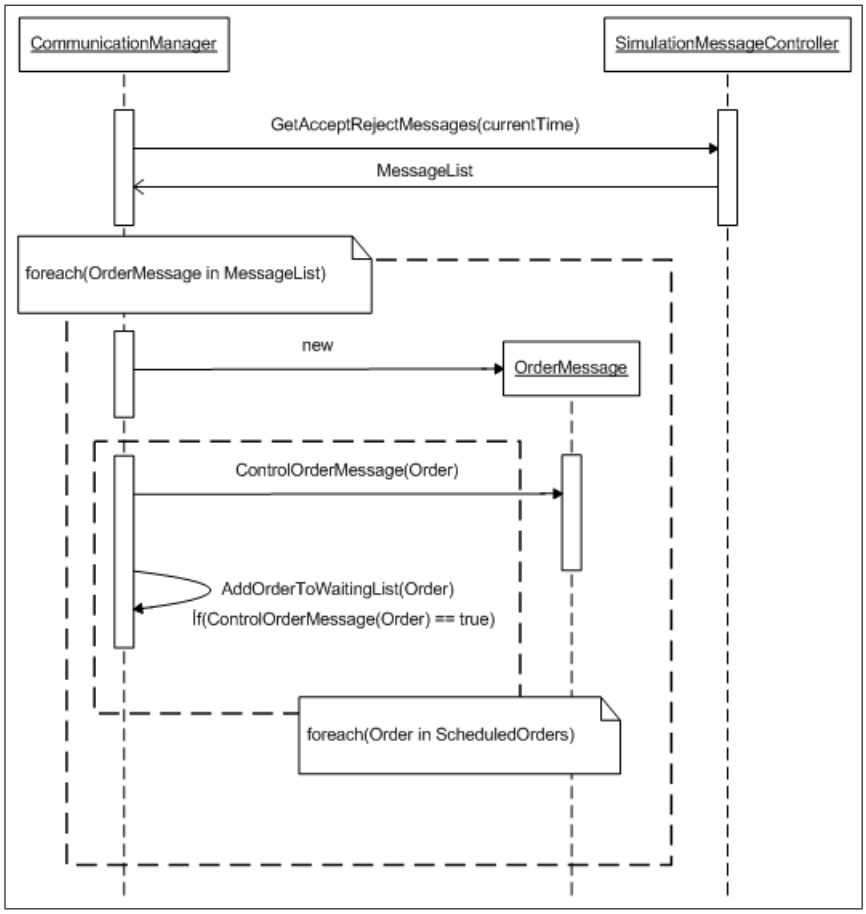

Figure B.14. Sequence diagram for MontorMaterialHandling procedure

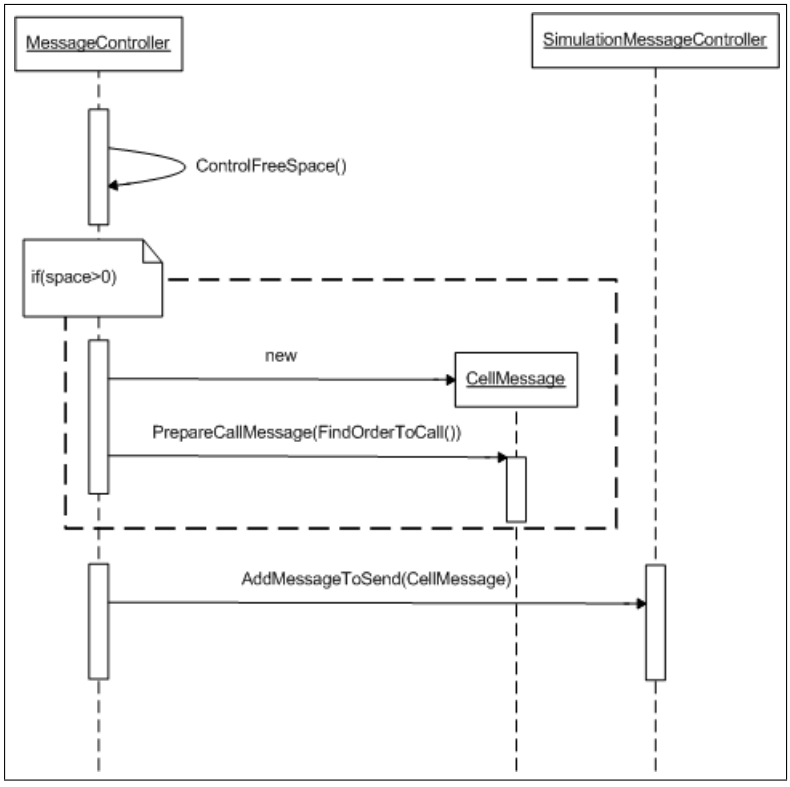

Figure B.15. Sequence diagram for SendCallMessage procedure

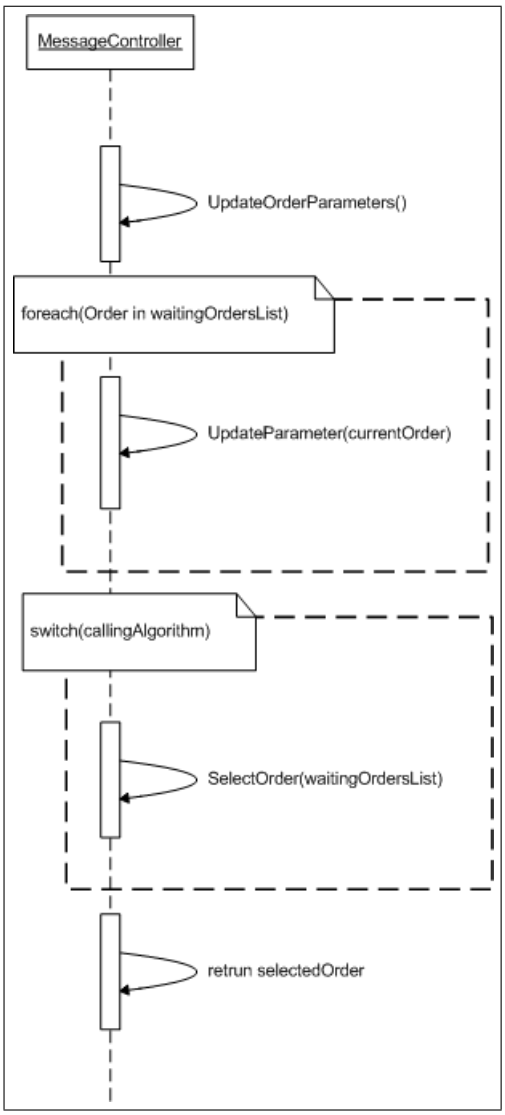

Figure B.16. Sequence diagram for FindOrderToCall procedure

APPENDIX C: Diagrams for Virtual Modules of the Simulation Implementations

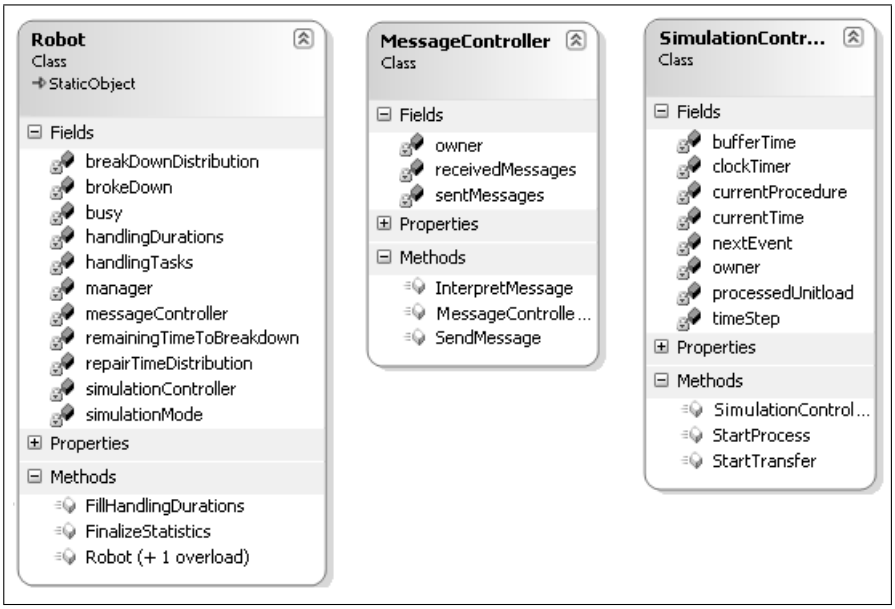

Figure C.1. Class diagrams for VirtualRobot components

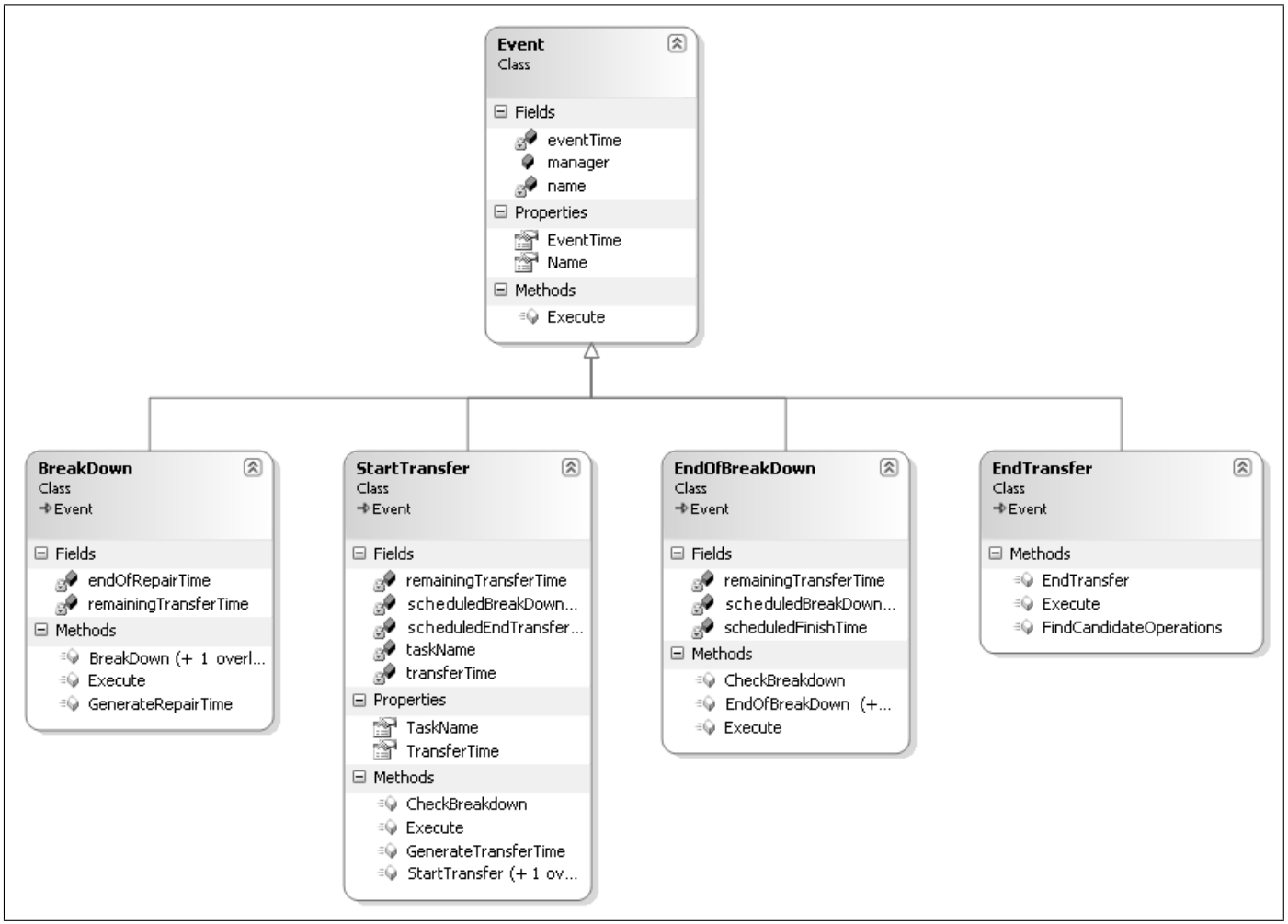

Figure C.2. Class diagrams for virtual robot events

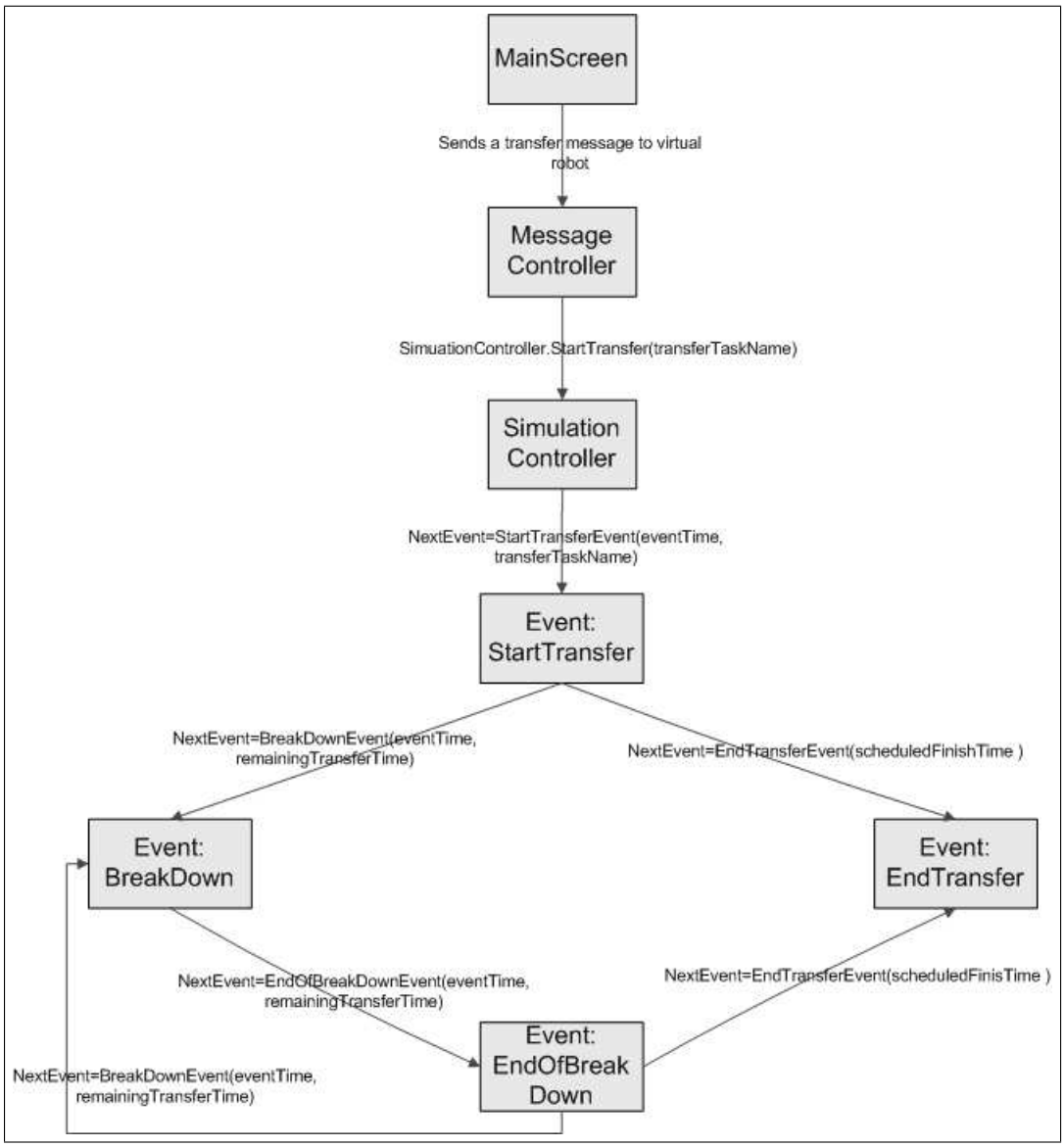

Figure C.3. Event flow structure of the virtual robot events

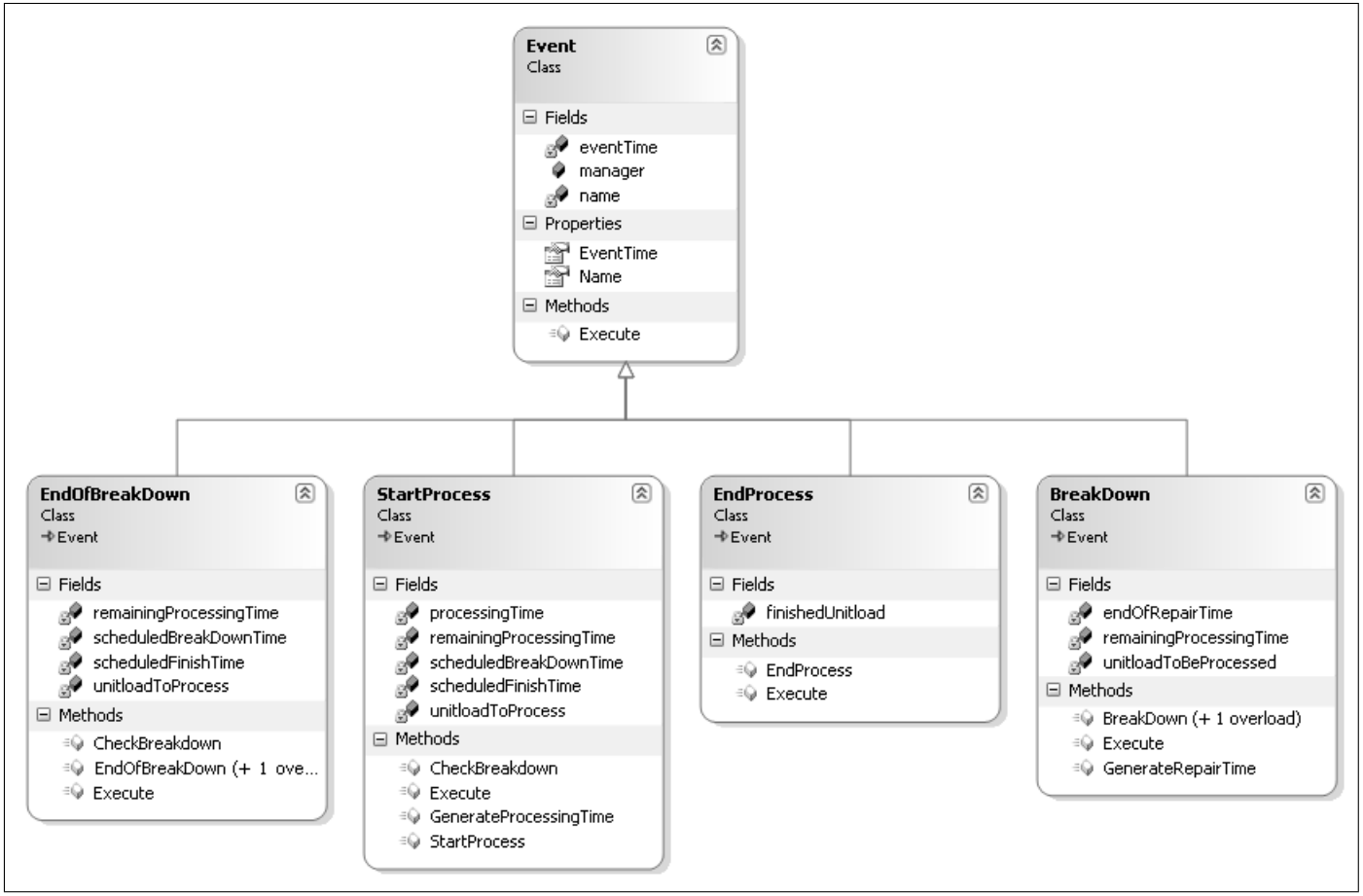

Figure C.4. Class diagrams for processor events

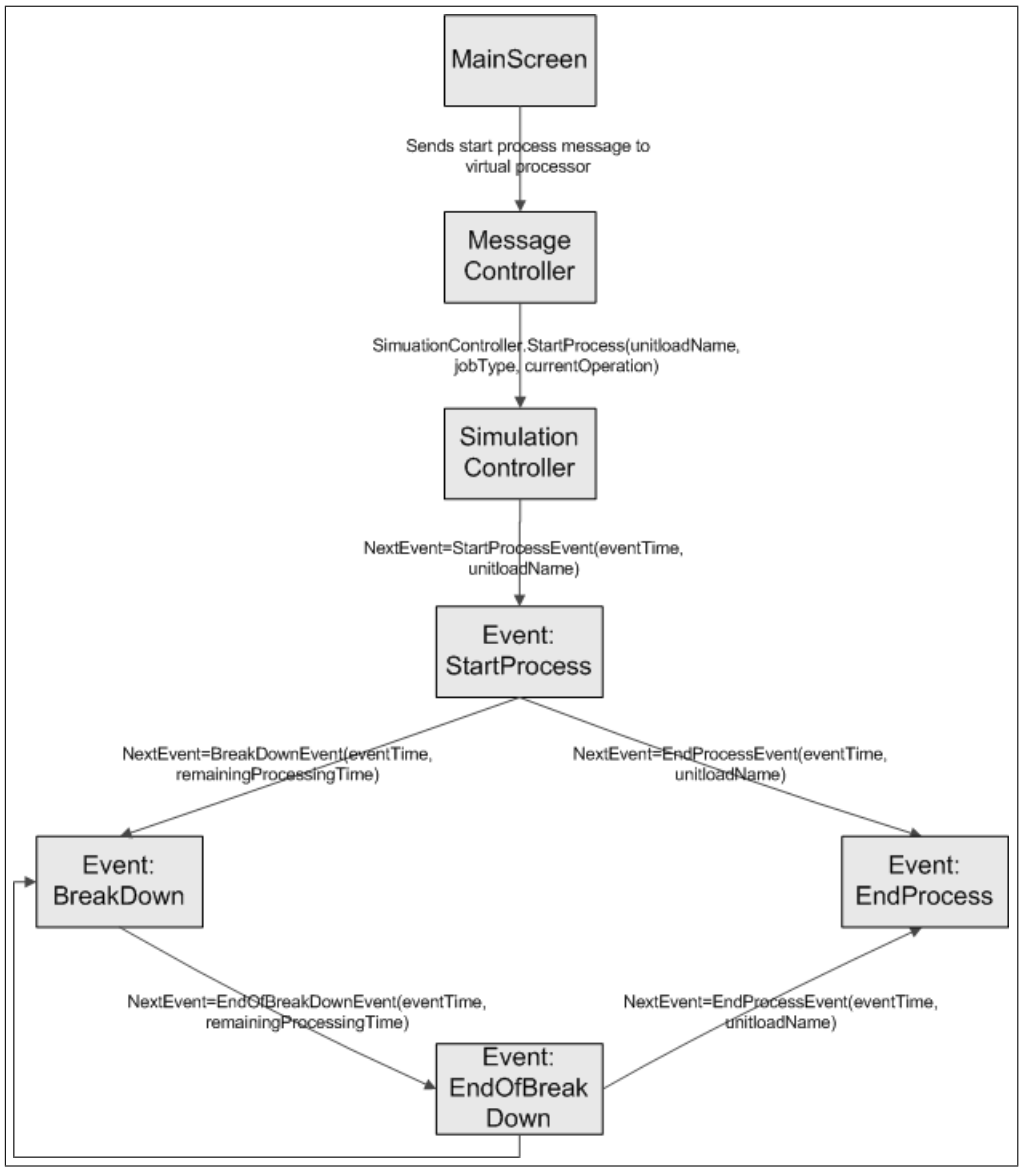

Figure C.5. Event flow structure of the virtual processor events

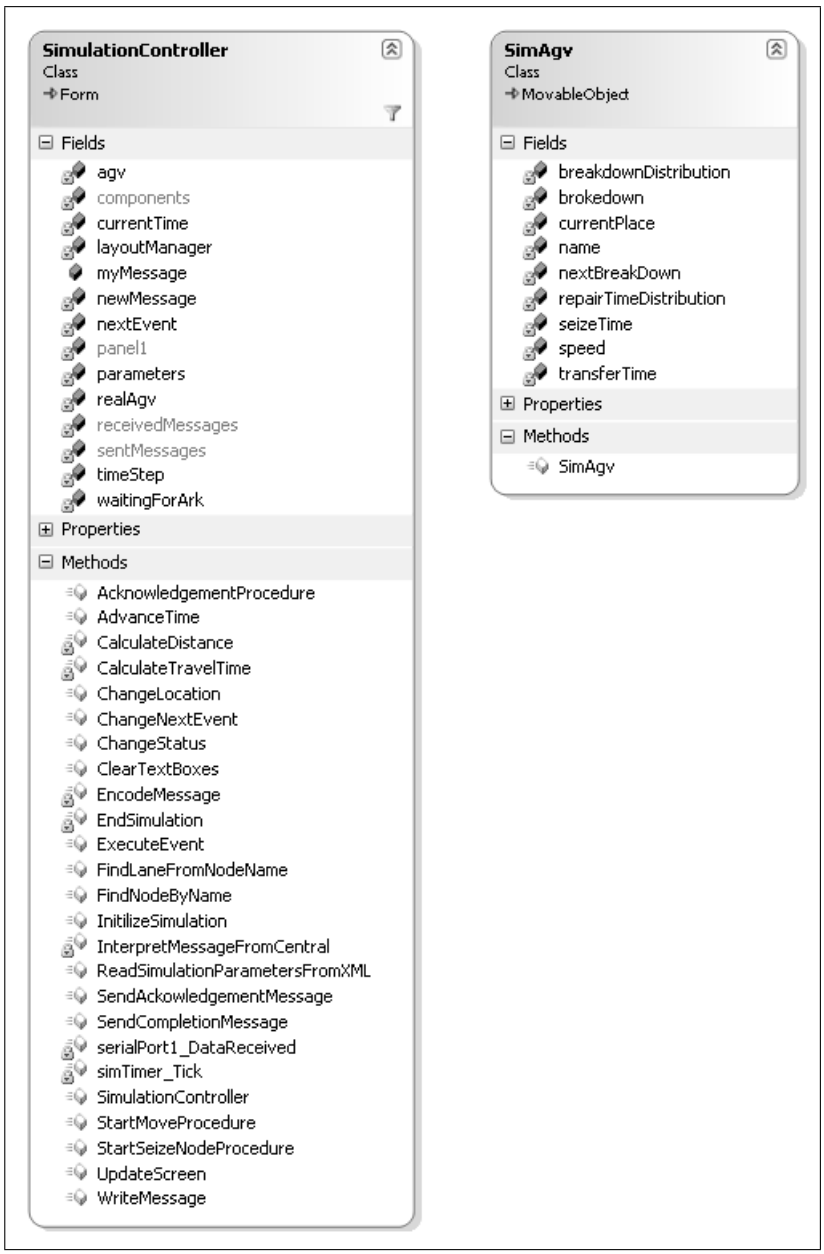

Figure C.6. Class diagrams for virtual AGV

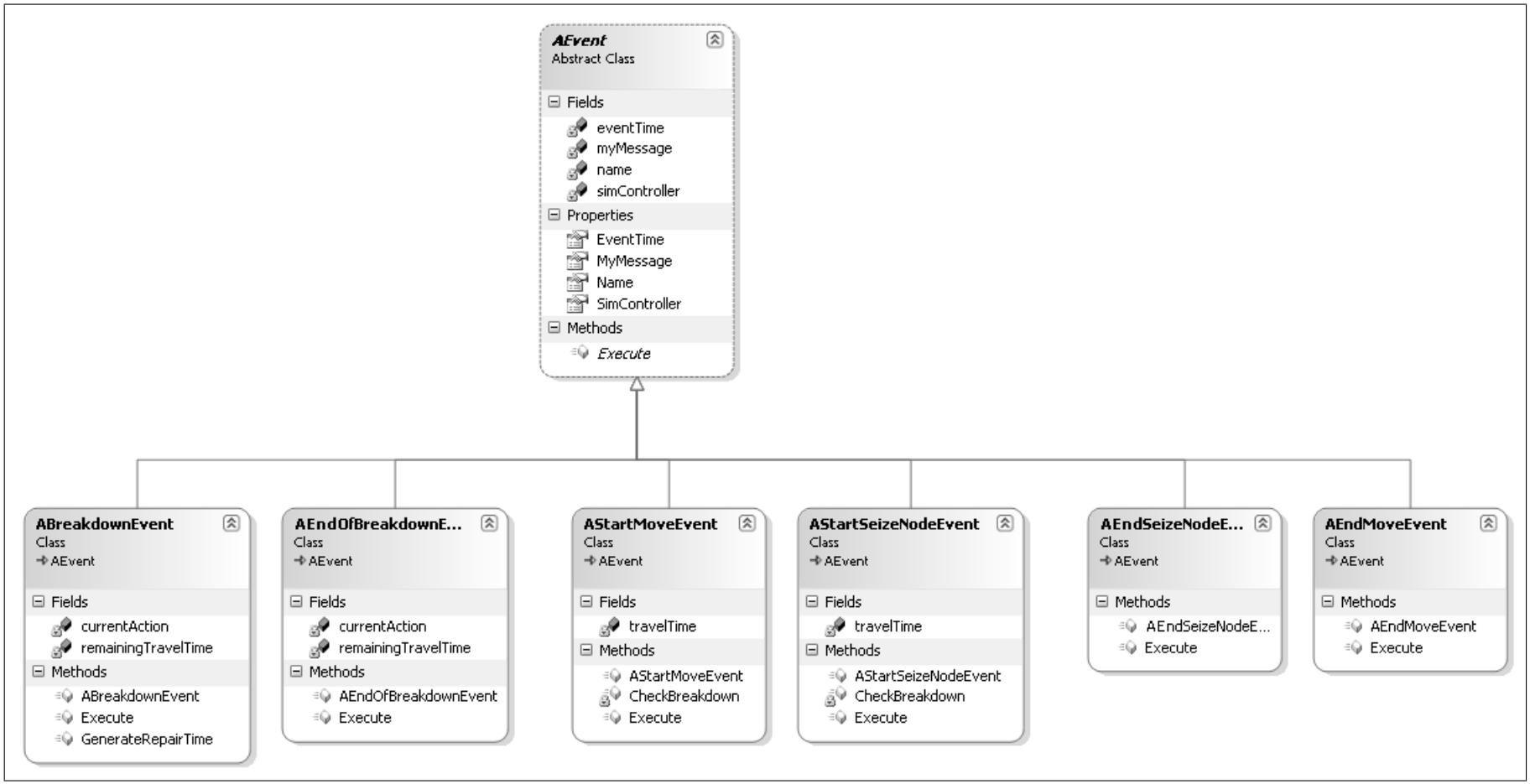

Figure C.7. Class diagrams for AGV events

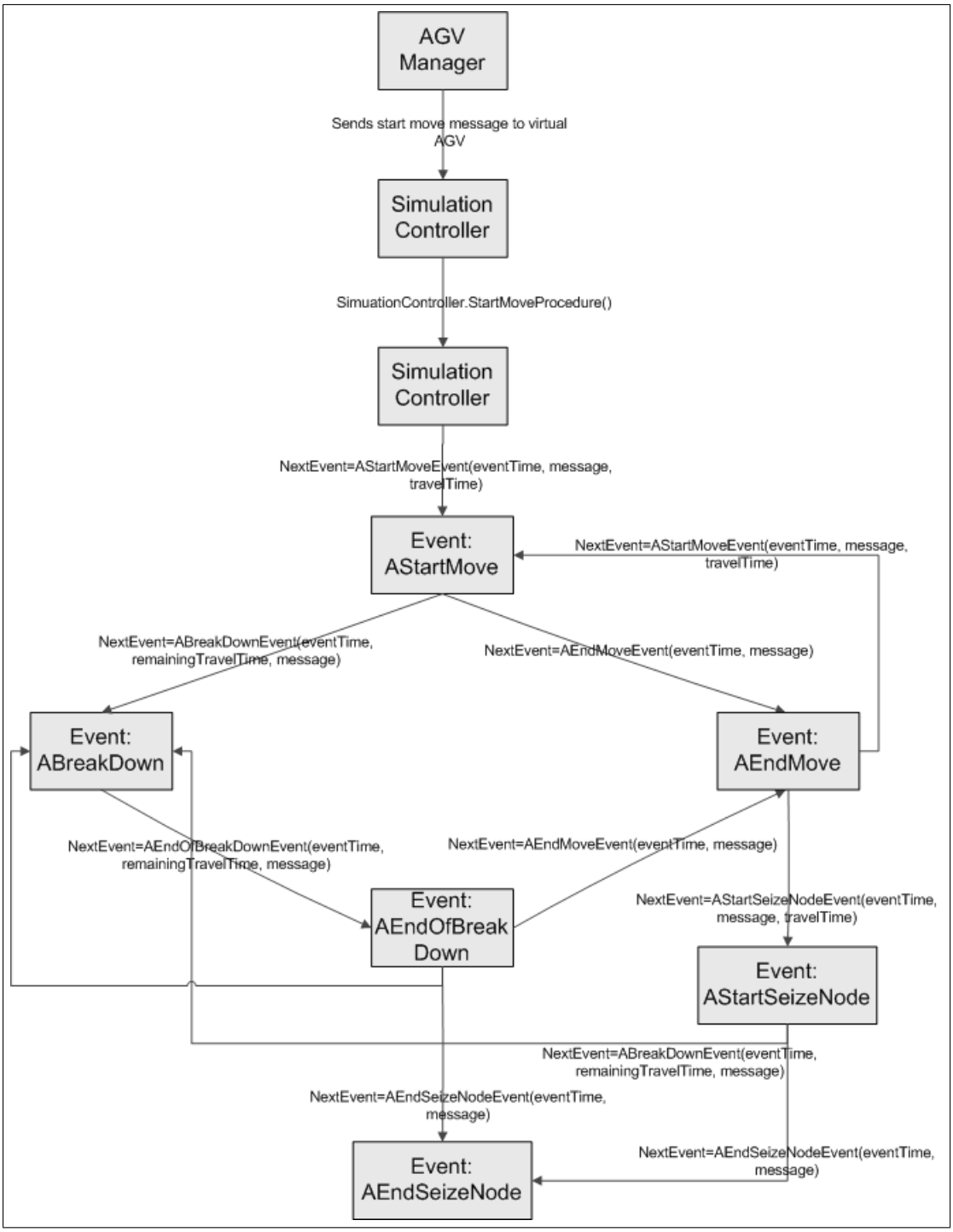

Figure C.8. Event flow structure of the virtual AGV events

APPENDIX D: Screen Shots for Simulation Implementation of Distributed SFCA

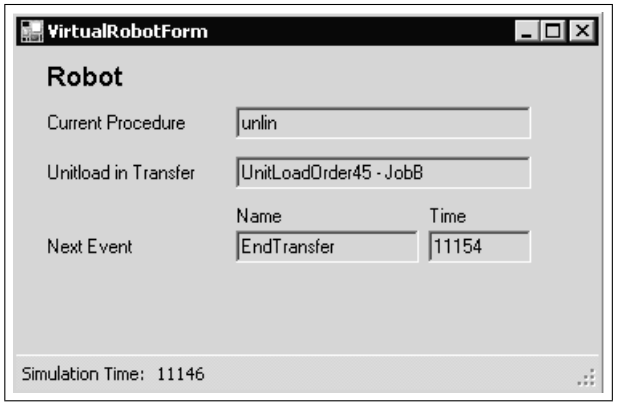

Figure D.1. Screen shot of virtual robot

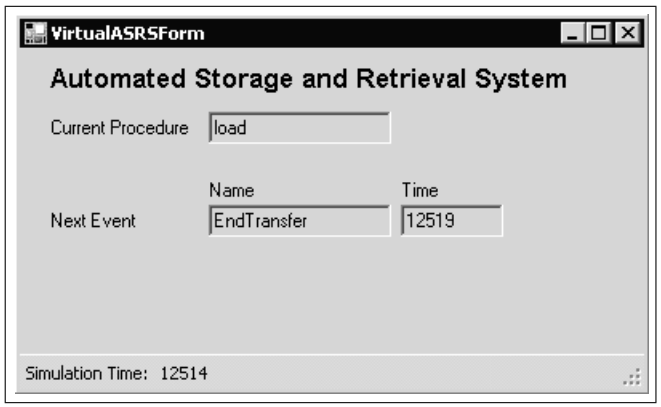

Figure D.2. Screen shot of VirtualASRSRobot

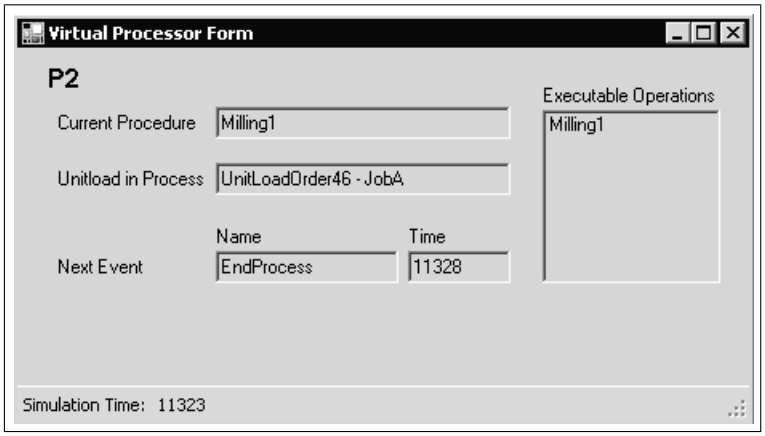

Figure D.3. Screen shot of virtual processor

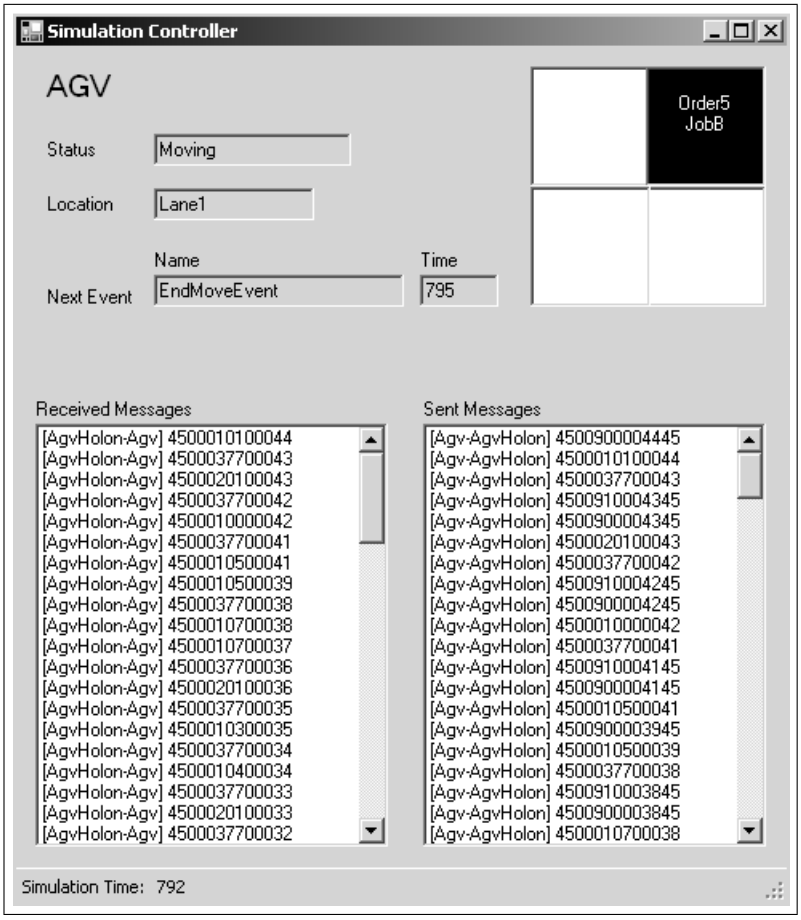

Figure D.4. Screen shot of virtual AGV

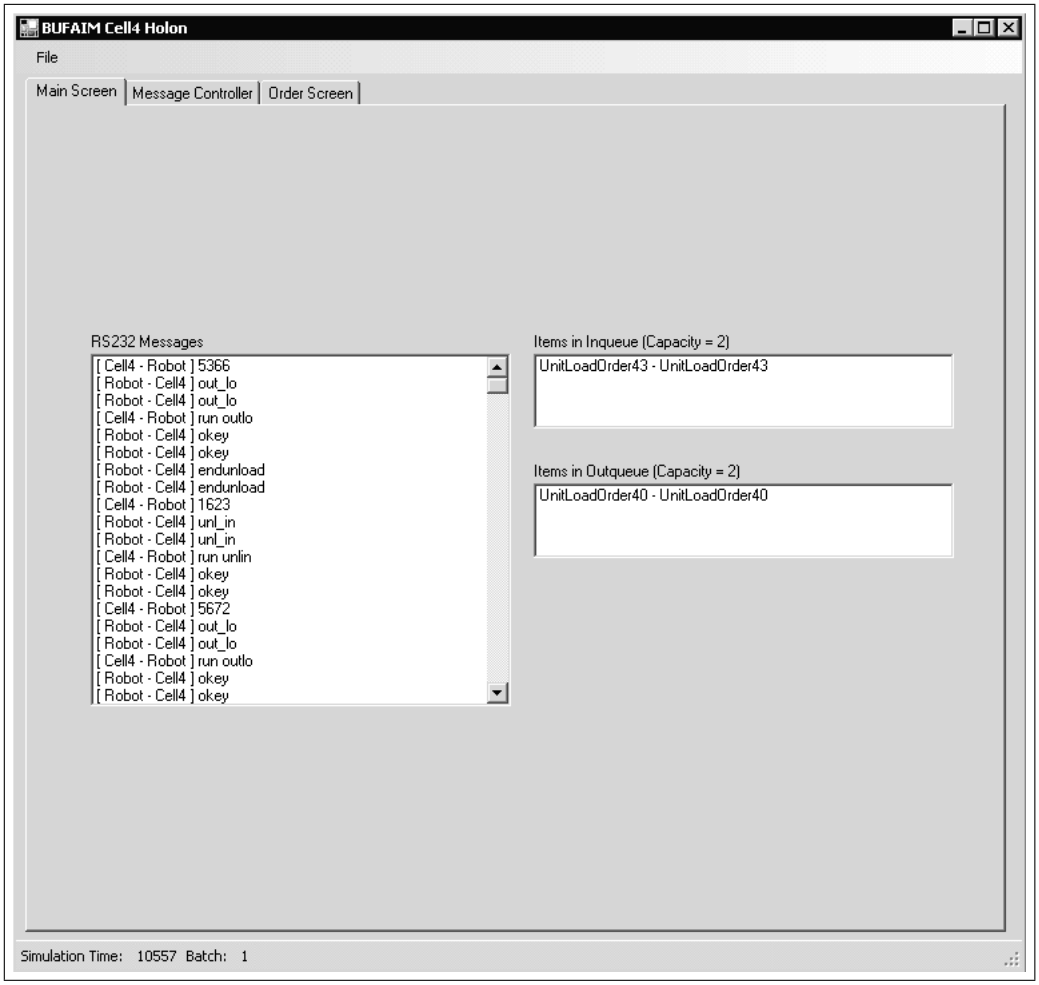

Figure D.5. Screen shot of MainScreen visual interface of VirtualCellHolon Logical Processor

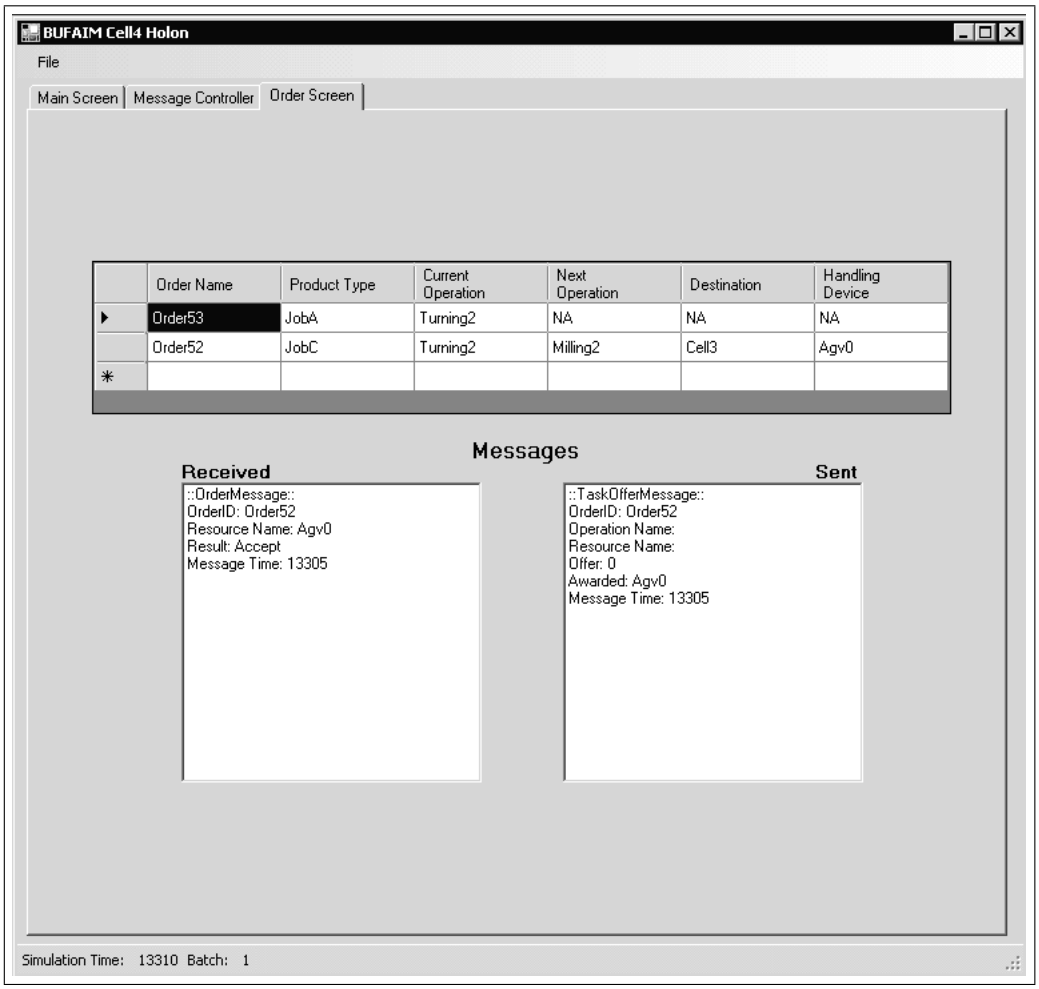

Figure D.6. Screen shot of OrderScreen visual interface

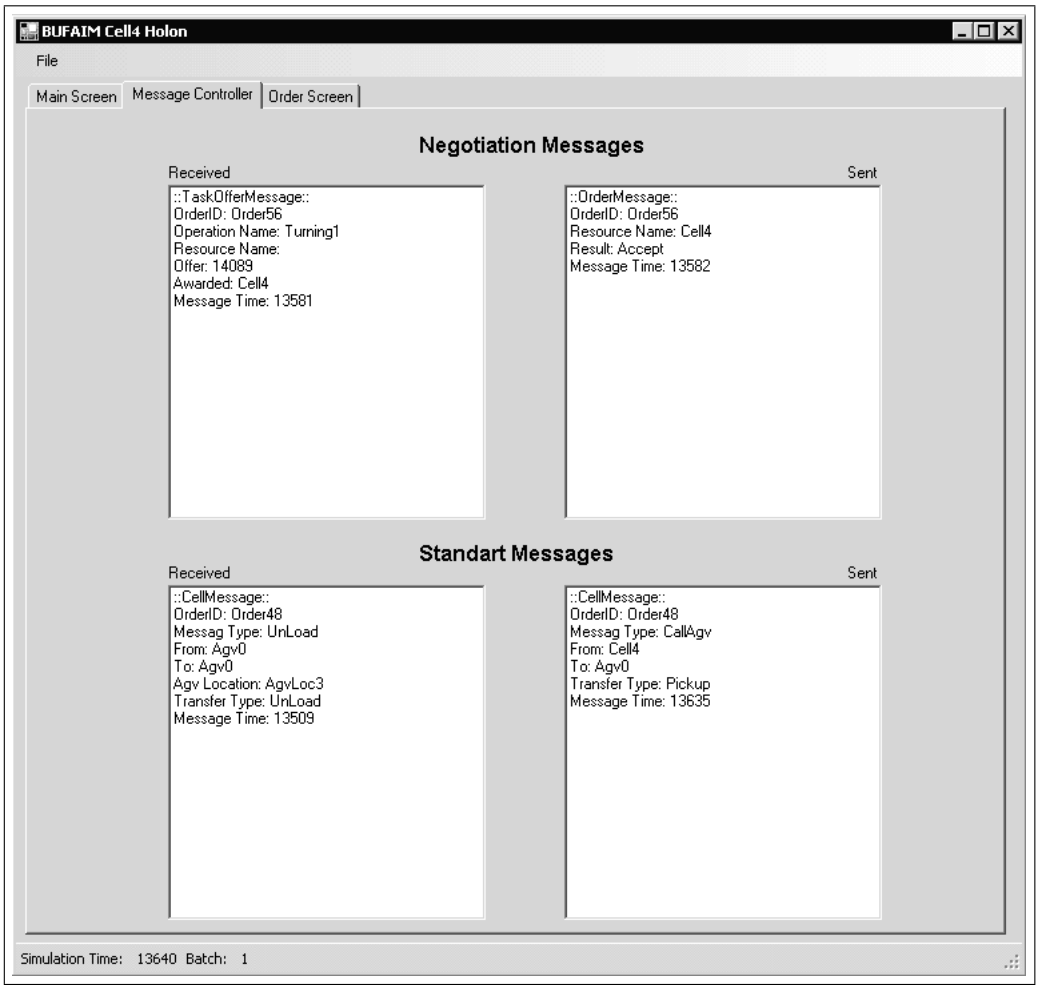

Figure D.7. Screen shot of MessageController visual interface

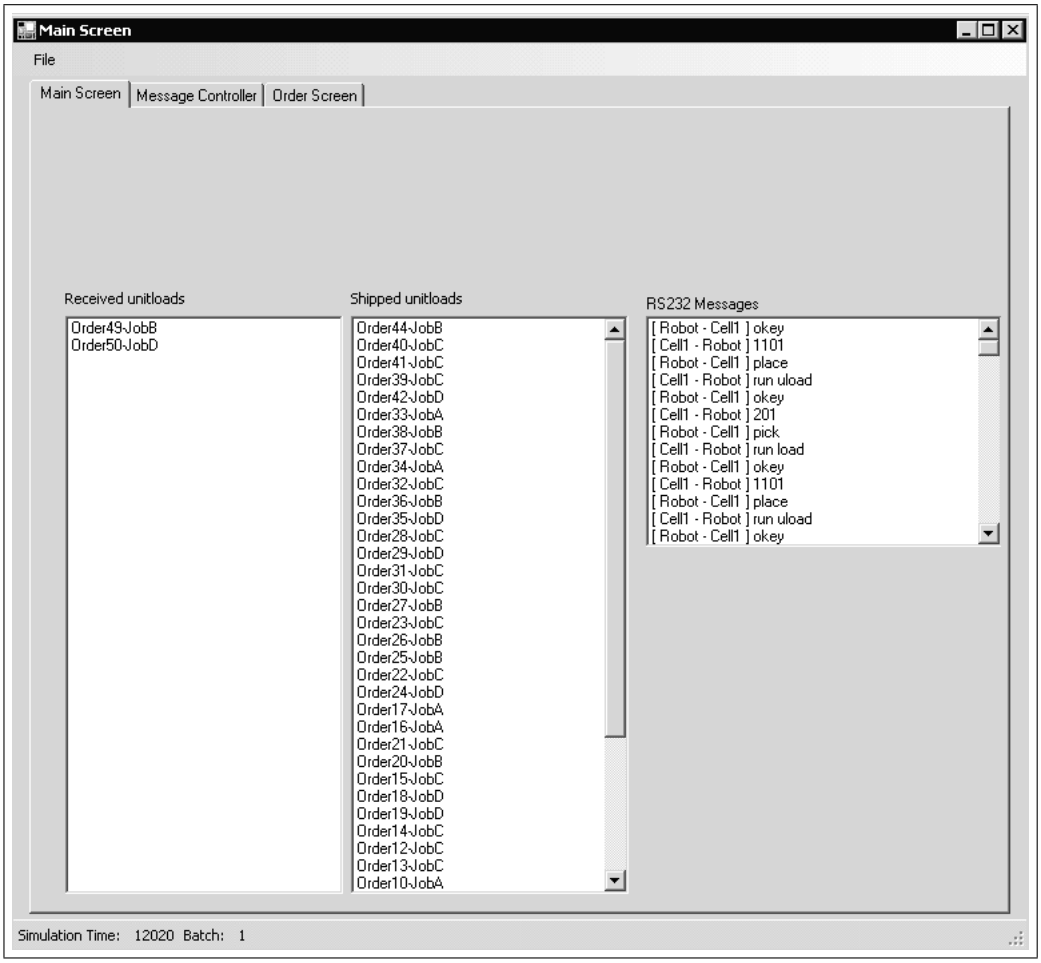

Figure D.8. Screen shot of MainScreen visual interface of VirtualASRSHolon Logical Processor

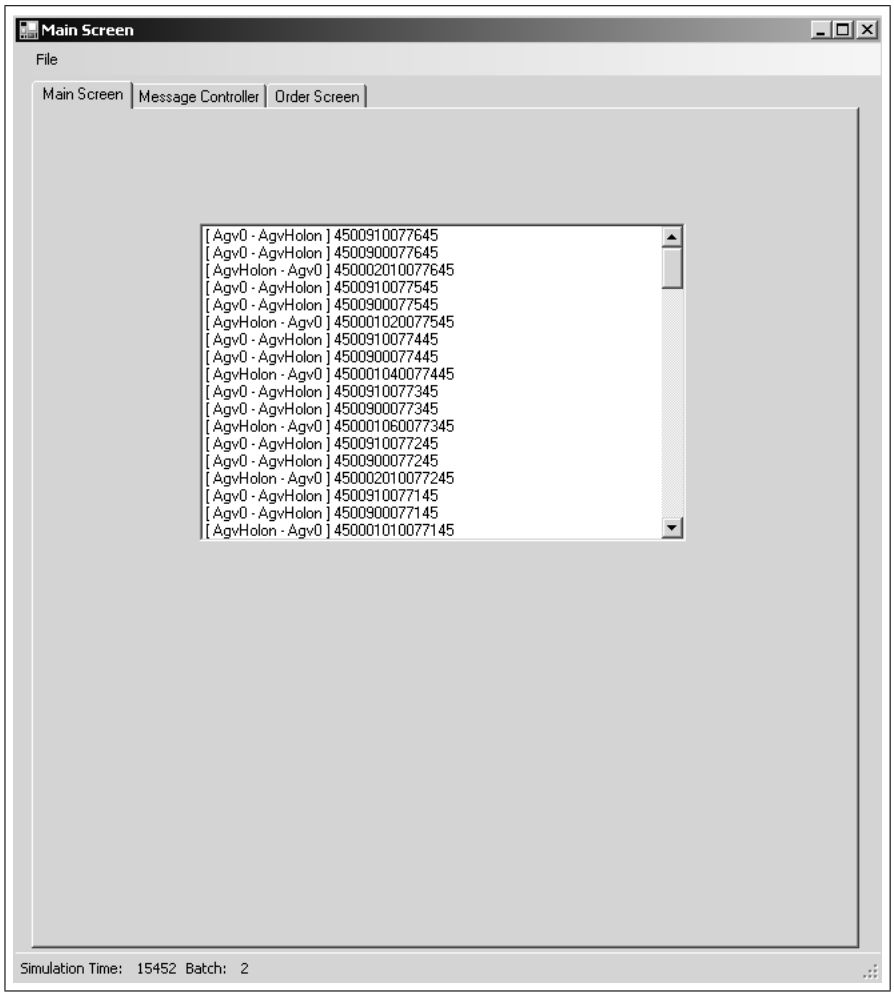

Figure D.9. Screen shot of MainScreen visual interface of VirtualAGVHolon Logical Processor

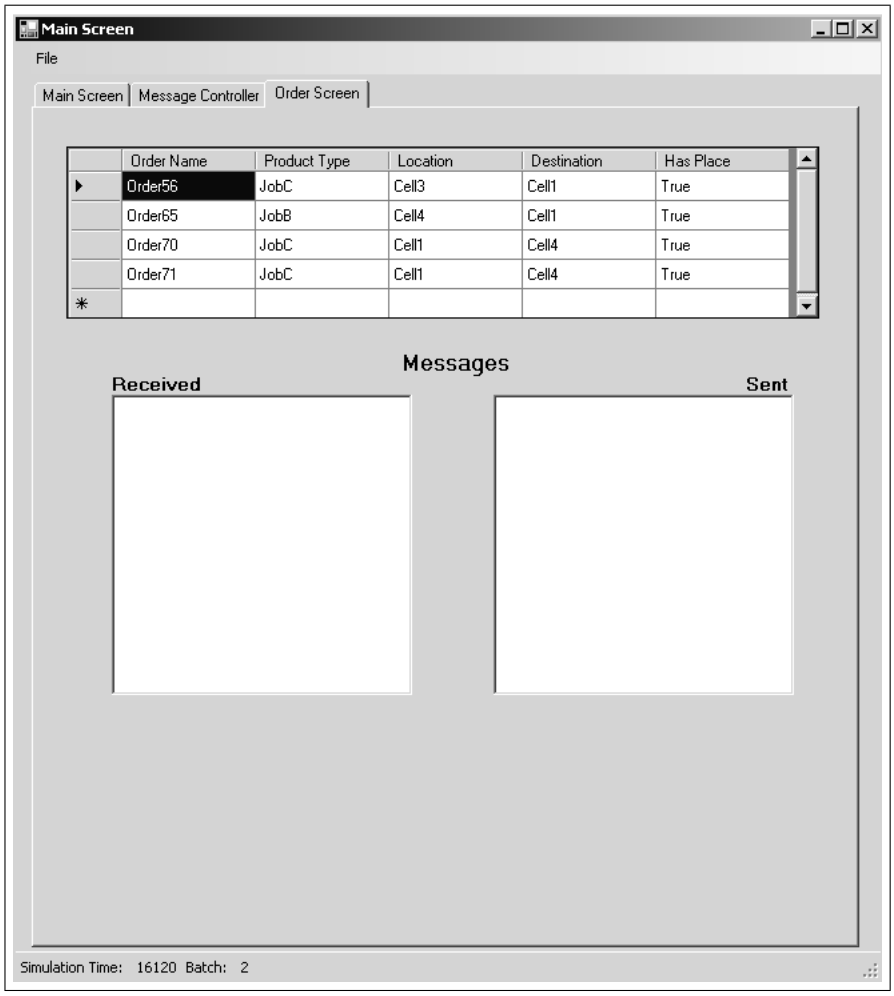

Figure D.10. Screen shot of OrderScreen visual interface of VirtualAGVHolon Logical Processor

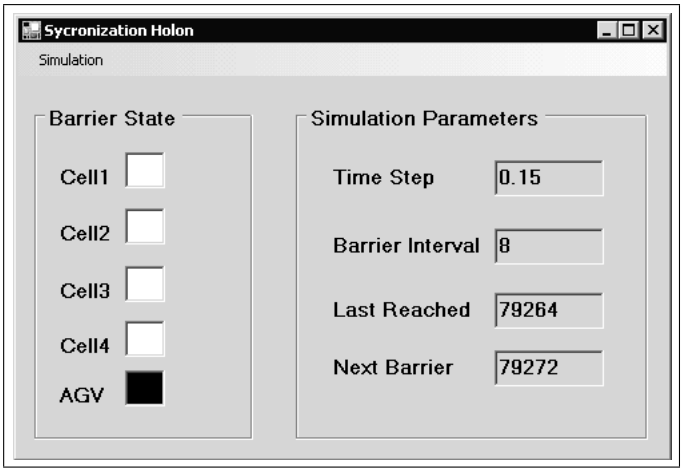

Figure D.11. Screen shot of Synchronization Holon

APPENDIX E: Message Structures and Example Messages for Simulation Implementation of Distributed SFCA

Table E.1. TaskMessage-transfer task structure

< MessageTime > 1950 < /MessageTime >
< Name > $TaskMessage$ < /Name >
< TaskType > $TransferTask$ < /TaskType >
< OrderID > $Order10$ < /OrderID >
< JobType > $JobA$ < /JobType >
< Location > $Cell1$ < /Location >
< Destination > $Cell3$ < /Destination >

Table E.2. ProposalMessage structure

< MessageTime > 1950 < /MessageTime >
< Name > $ProposalMessage$ < /Name >
< ProposalType > $Operation$ < /ProposalType >
< OrderID > $Order10$ < /OrderID >
< OperationName > $Milling1$ < /OperationName >
< Bid > 450 < /Bid >
< RoundID > 1948 < /RoundID >

Table E.3. TaskOfferMessage-operation structure

< MessageTime > 1950 < /MessageTime >
< Name > $TaskOfferMessage$ < /Name >
< TaskType > $Operation$ < /TaskType >
< OrderID > $Order10$ < /OrderID >
< OperationName > $Milling1$ < /OperationName >
< Offer > 2577 < /Offer >
< JobType > $JobA$ < /JobType >
< Awarded > $Cell3$ < /Awarded >
< RoundID > 1947 < /RoundID >
< DueDate > 2950 < /DueDate >
< RemainingProcessingTime > 430 < /RemainingProcessingTime >
< Penalty > 4 < /Penalty >
< ReceivalTime > 1948 < /ReceivalTime >
< LastOperation > $Receival$ < /LastOperation >
< CurrentOperationIndex > 1 < /CurrentOperationIndex >
< Receiver > $Cell1$ < /Receiver >

Table E.4. TaskOfferMessage-transfer task structure

< MessageTime > 1950 < /MessageTime >
< Name > $TaskOfferMessage$ < /Name >
< TaskType > $Operation$ < /TaskType >
< OrderID > $Order10$ < /OrderID >
< Awarded > $Agv0$ < /Awarded >
< JobType > $JobA$ < /JobType >
< Location > $Cell1$ < /Location >
< Destination > $Cell3$ < /Destination >
< RoundID > 1947 < /RoundID >
< Receiver > $Cell1$ < /Receiver >
< DueDate > 2950 < /DueDate >
< RemainingProcessingTime > 430 < /RemainingProcessingTime >
< Penalty > 4 < /Penalty >
< ReceivalTime > 1948 < /ReceivalTime >
< ProcessingTime > 130 < /ProcessingTime >

Table E.5. OrderMessage structure

< MessageTime > 1950 < /MessageTime >
< Name > $OrderMessage$ < /Name >
< OrderID > $Order10$ < /OrderID >
< ResourceName > $Cell3$ < /ResourceName >
< Result > $Accept$ < /Result >

APPENDIX F: UML Diagrams for Layered Simulation Implementation

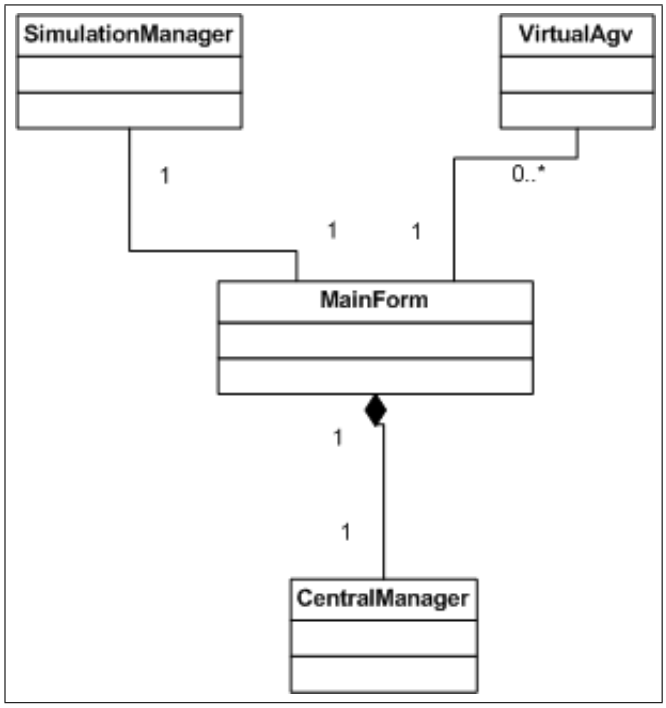

Figure F.1. CentralController LP class relations diagram

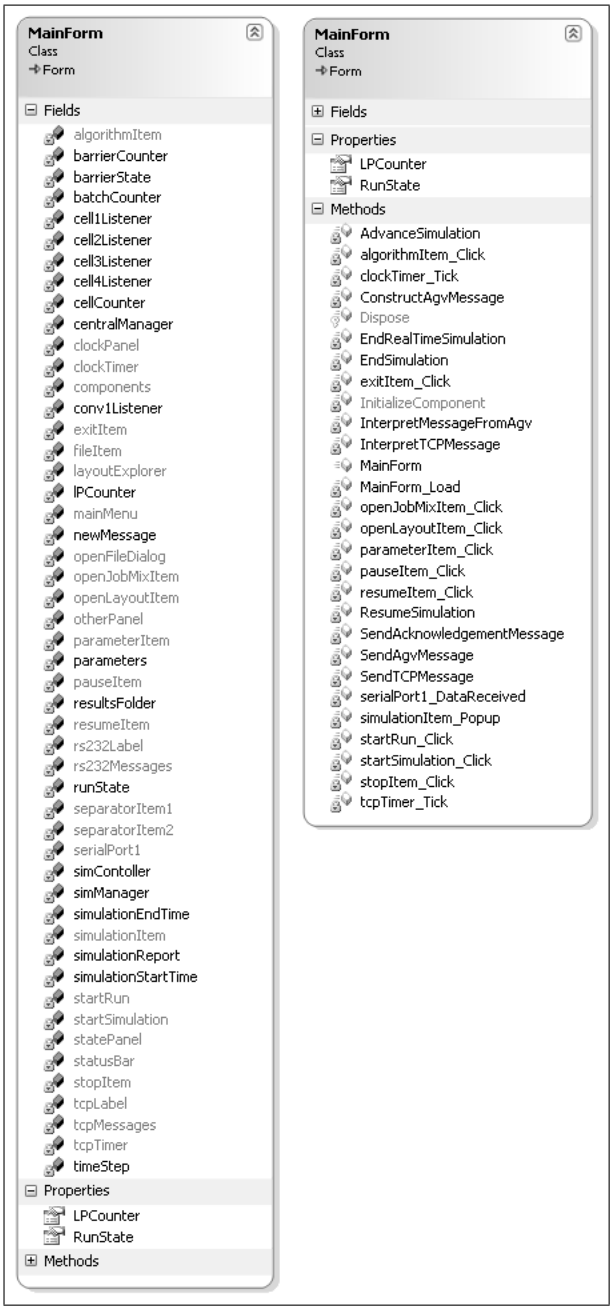

Figure F.2. MainForm class diagram

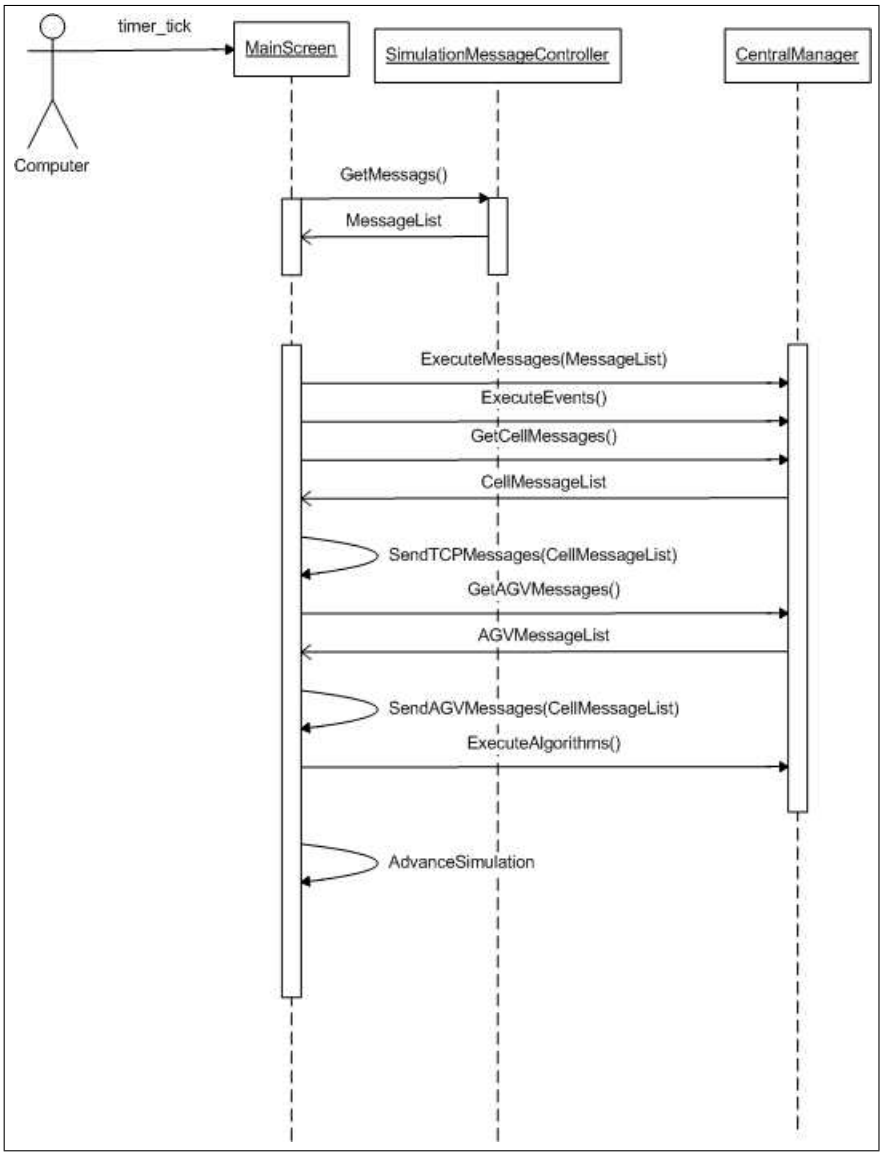

Figure F.3. CentralController LP timer event sequence diagram

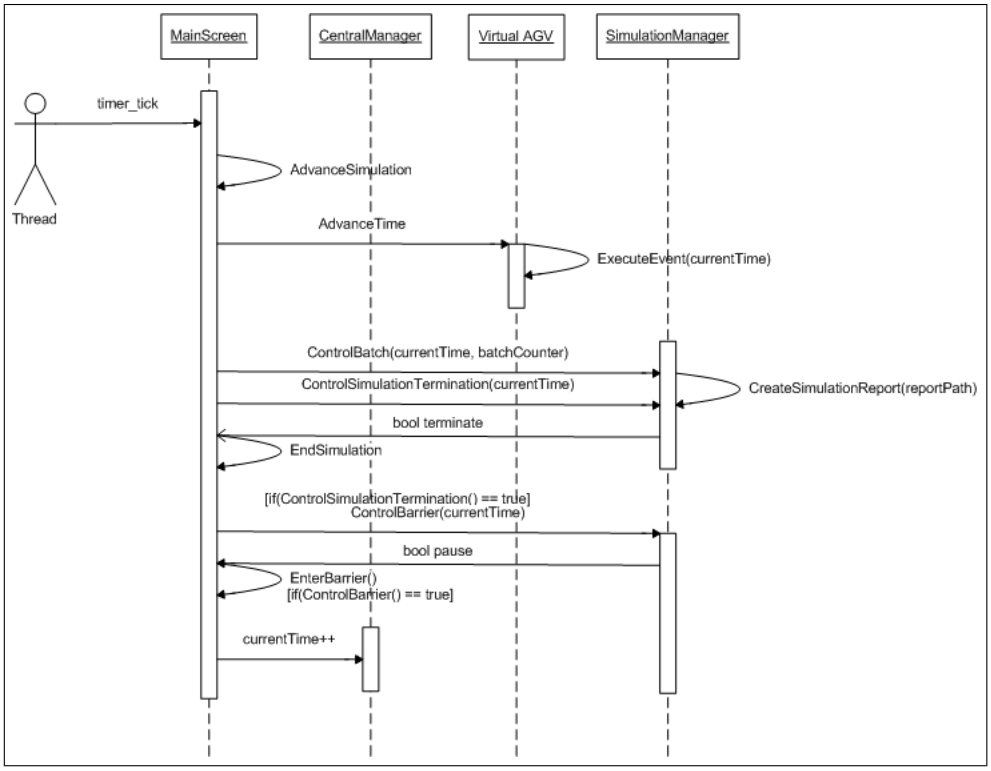

Figure F.4. CentralController LP AdvanceSimulation procedure sequence diagram

Figure F.5. SimulationManager and its sub-components class diagrams

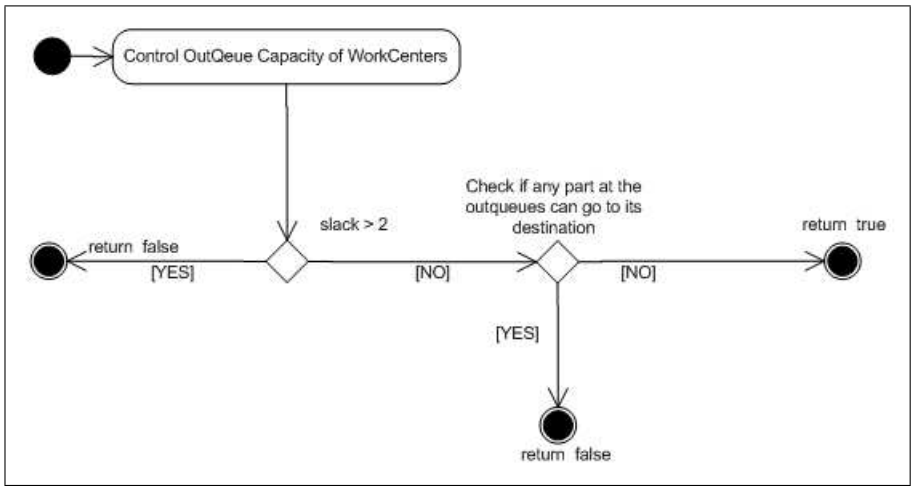

Figure F.6. Activity diagram for DetectDeadLock procedure

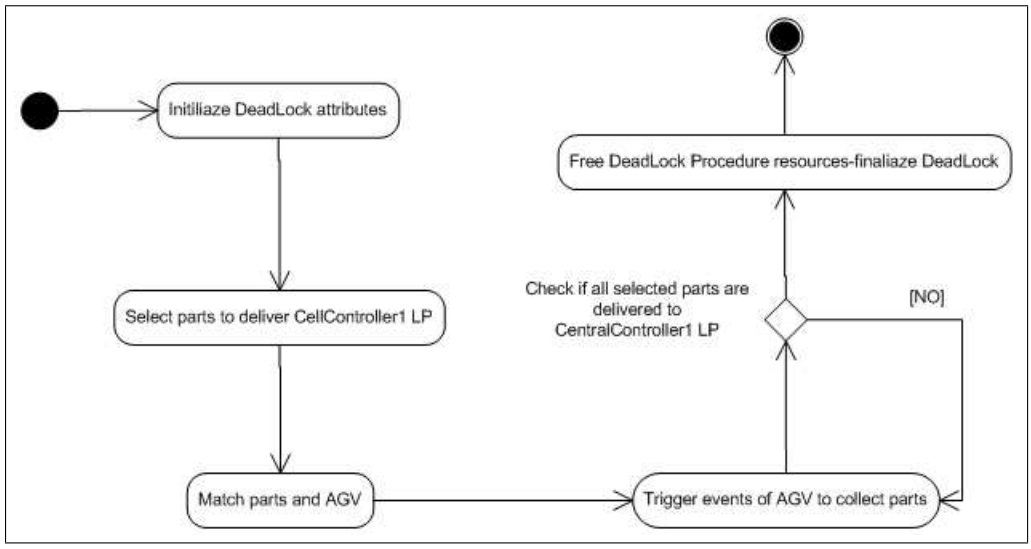

Figure F.7. Activity diagram for ResolveDeadLock procedure

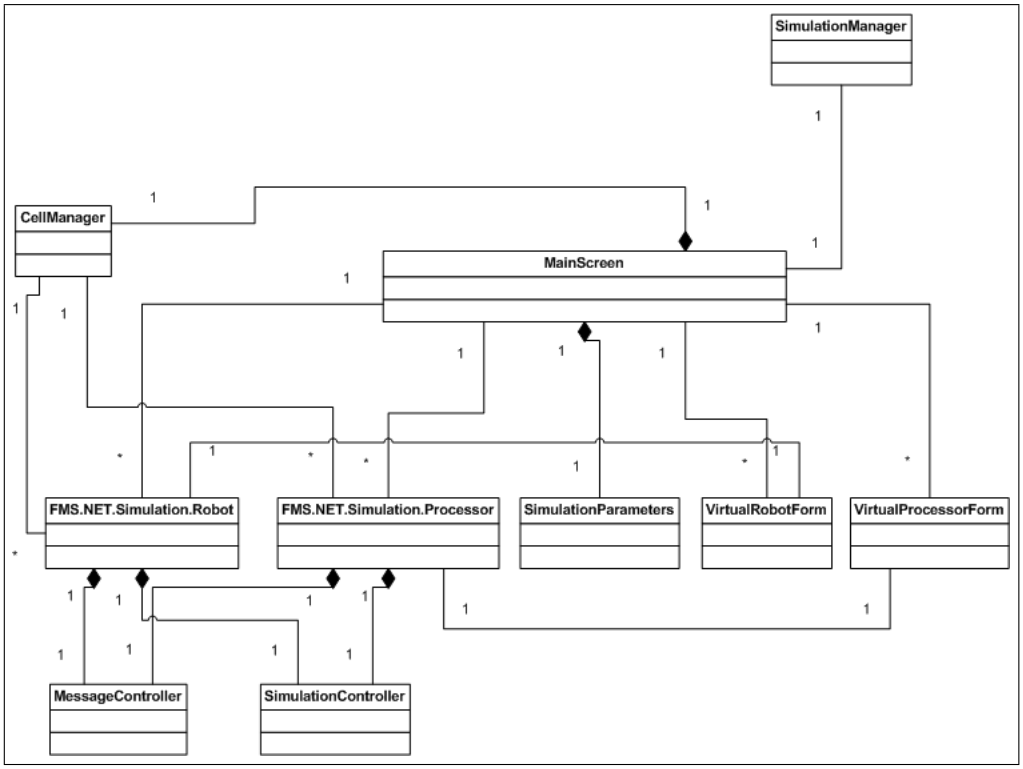

Figure F.8. CellController LP class realeation diagrams

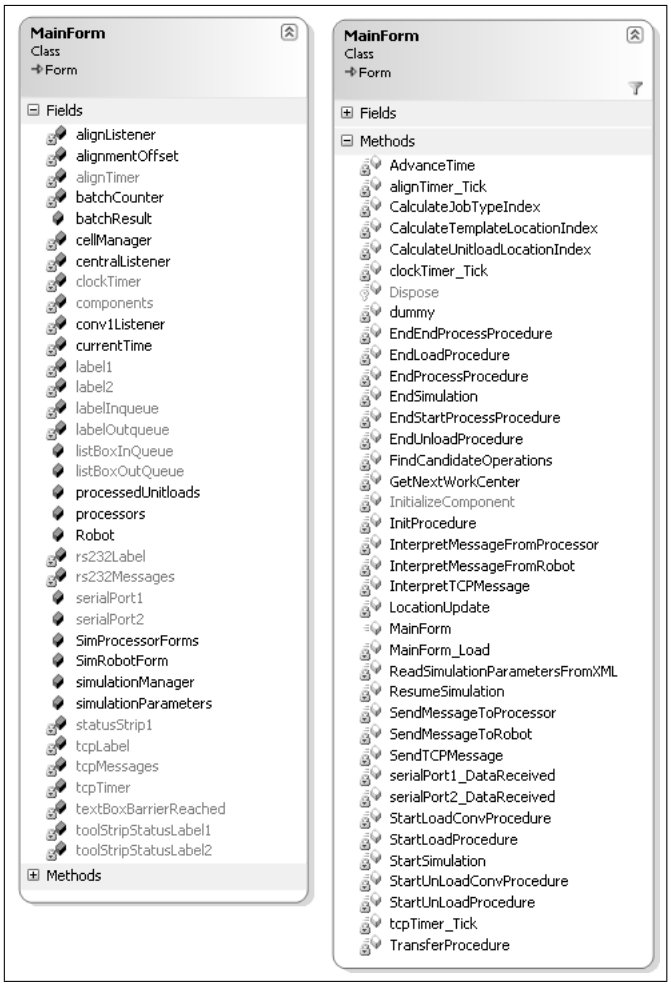

Figure F.9. MainForm class diagram

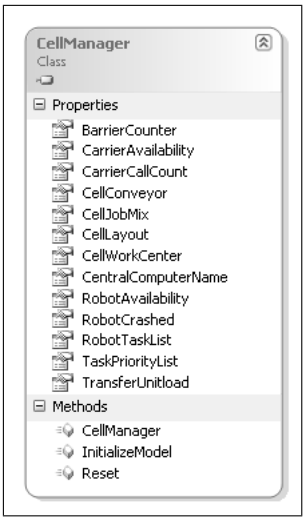

Figure F.10. CellManager class diagram

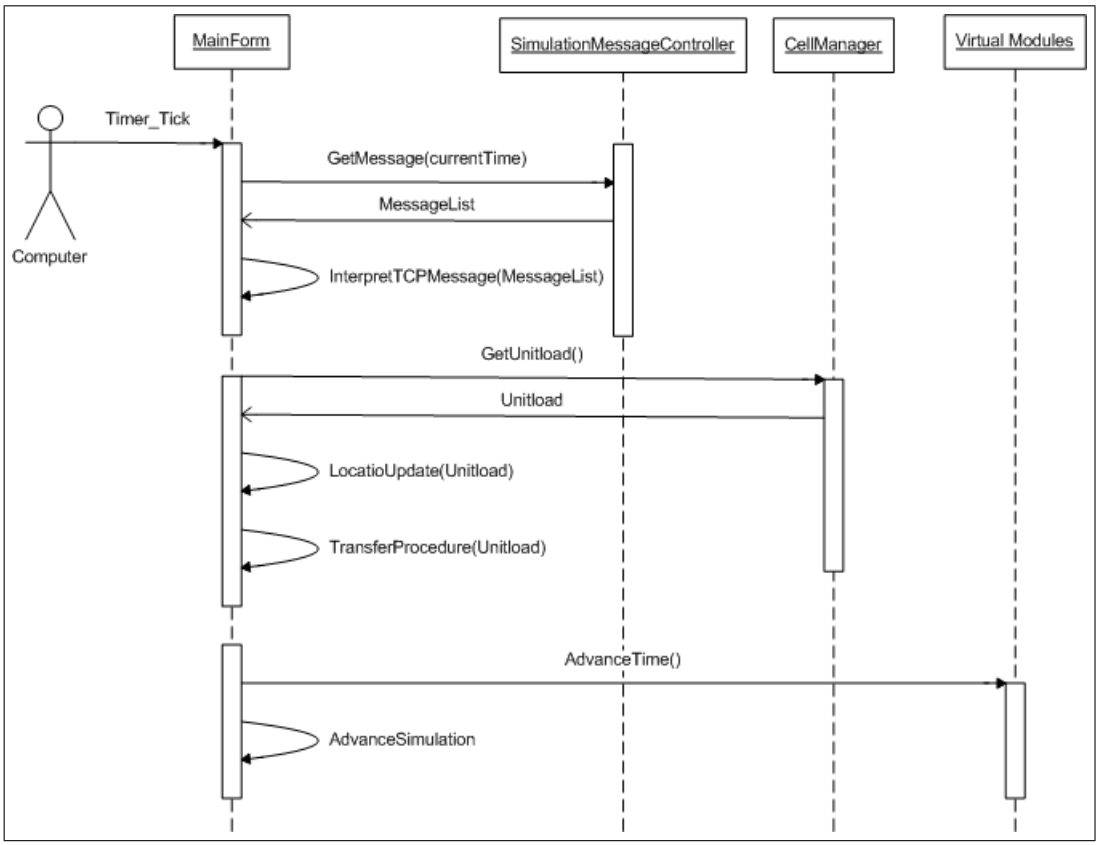

Figure F.11. Sequence diagram of timer tick event of CellController LP

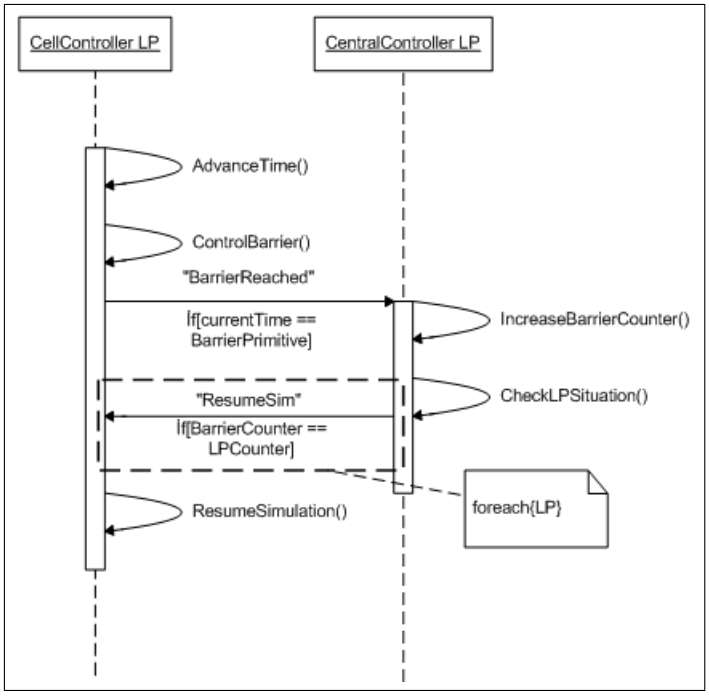

Figure F.12. Sequence diagram for centralized barrier approach

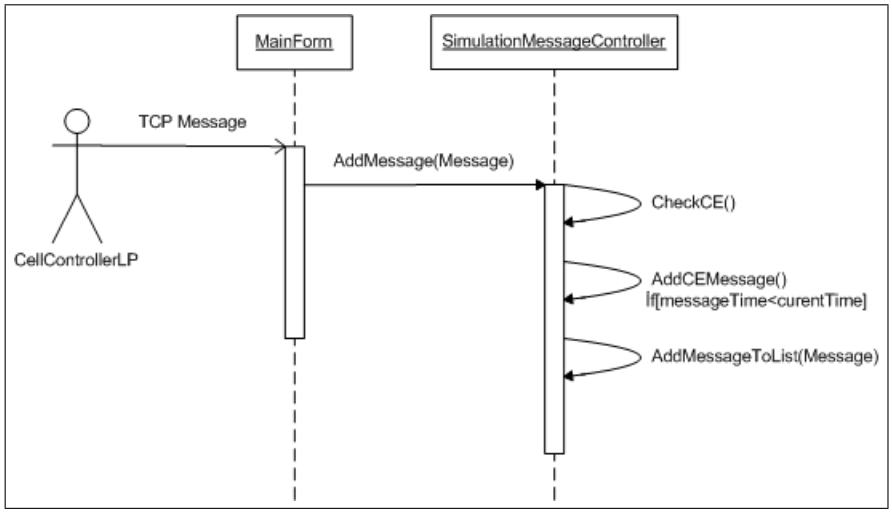

Figure F.13. Sequence diagram for accepting a received message

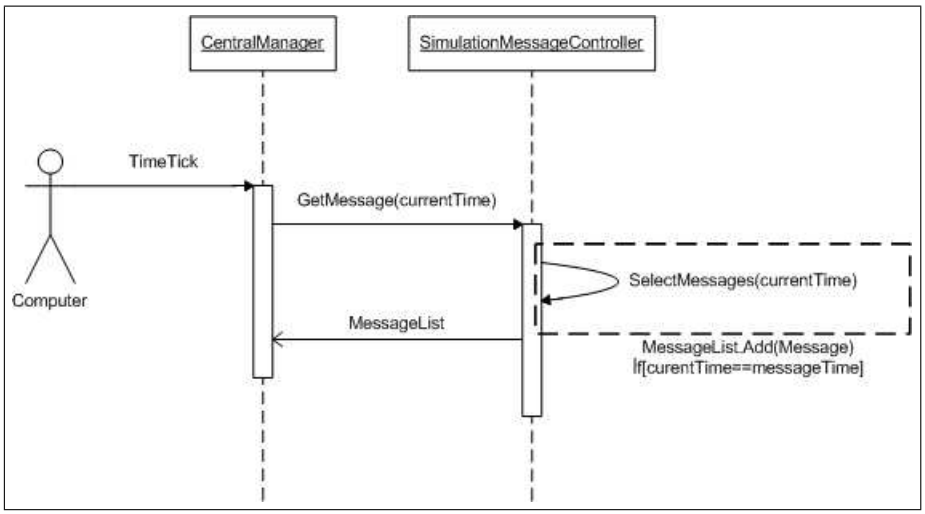

Figure F.14. Sequence diagram for selecting a received message according to time stamp value

APPENDIX G: Screen Shots of Simulation Implementation of Layered SFCA

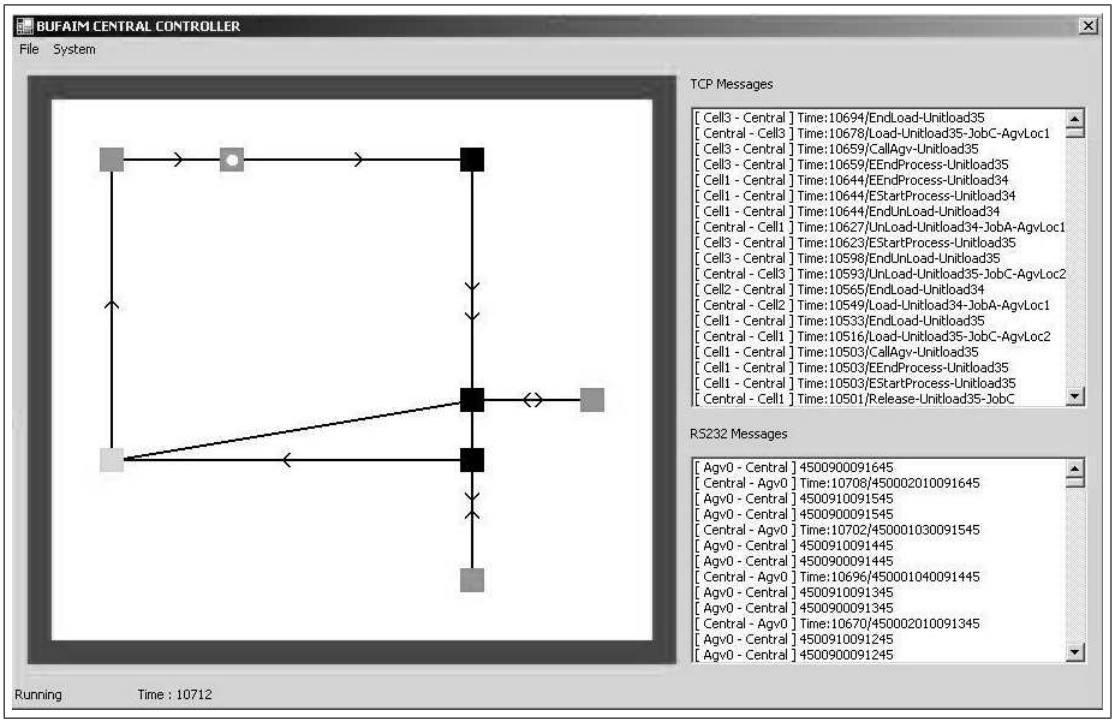

Figure G.1. MainForm visual interface screen shot

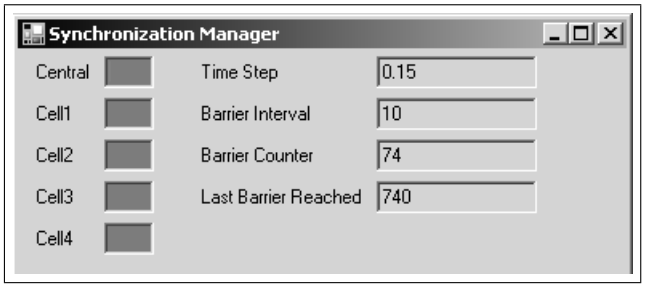

Figure G.2. SimulationManagerForm visual interface screen shot

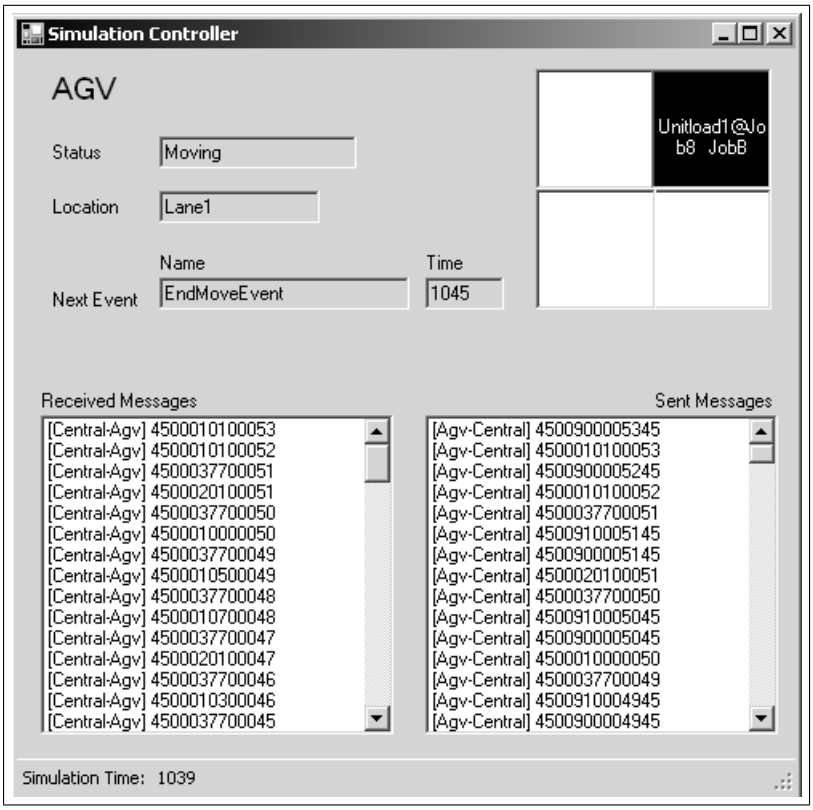

Figure G.3. Screen Shot of VirtualAgv visual interface

APPENDIX H: Examples for XML Type Database Structure for Configuring Virtual Modules

Table H.1. XML based database example of virtual ASRS robot

```
<ASRS>
    <Name>ASRS< /Name>
    <Capacity>10000< /Capacity>
    <SimulationMode>true< /SimulationMode>
    <BreakDownDistribution xsi:type="ExplonentialRVGenerator">
        <Value>1000< /Value>
    < /BreakDownDistribution>
    <RepairTimeDistribution xsi:type="FixedRVGenerator">
        <Value>100< /Value>
    </RepairTimeDistribution>
    <XSpeed>0.1< /XSpeed>
    <YSpeed>0.1< /YSpeed>
    <ZSpeed>0.1< /ZSpeed>
< /ASRS>
```

Table H.2. XML based datebase example of virtual robot

```
<Robot>
    <Name>RobotOfCell2< /Name>
    <Capacity>1< /Capacity>
    <SimulationMode>true< /SimulationMode>
    <BreakDownDistribution xsi:type="FixedRVGenerator">
        <Value>INF< /Value>
    < /BreakDownDistribution>
    <RepairTimeDistribution xsi:type="FixedRVGenerator">
        <Value>INF< /Value>
    < /RepairTimeDistribution>
    <HandlingTasks>
        <HandlingTask>
          <Name>unlin< /Name>
          <DurationDistribution xsi:type="FixedRVGenerator">
            <Value>10< /Value>
          < /DurationDistribution>
        < /HandlingTask>
        <HandlingTask>
          <Name>outlo< /Name>
          <DurationDistribution xsi:type="FixedRVGenerator">
            <Value>10< /Value>
          < /DurationDistribution>
        < /HandlingTask>
        <HandlingTask>
          <Name>inpr< /Name>
          <DurationDistribution xsi:type="FixedRVGenerator">
            <Value>10< /Value>
          < /DurationDistribution>
    < /HandlingTasks>
< /Robot>
```

Table H.3. XML based datebase example of virtual processor

```
<FMS.NET.Simulation.Processor>
        <Name>P1< /Name>
        <Capacity>1< /Capacity>
        <SimulationMode>true< /SimulationMode>
        <ExecutableOperations>
            <Operation>Turning2< /Operation>
        < /ExecutableOperations>
        <BreakDownDistribution xsi:type="FixedRVGenerator">
            <Value>INF< /Value>
        < /BreakDownDistribution>
        <RepairTimeDistribution xsi:type="FixedRVGenerator">
            <Value>INF< /Value>
        < /RepairTimeDistribution>
< /FMS.NET.Simulation.Processor>
```

REFERENCES

Anussornnitisarn P., S. Y. Nof and O. Etzion, 2005, "Decentralized Control of Cooperative and Autonomous Agents for Solving the Distributed Resource Allocation Problem", *International Journal of Production Economics*, Vol. 98, pp. 114-128.

Balasubramanian S. R., W. Brennan and D. H. Norrie, 2001, "An Architecture for Metamorphic Control of Holonic Manufacturing Systems", *Computers In Industry*, Vol. 43, pp. 13-31.

Babiceanu R. F. and F. F. Chen "Development and Applications of Holonic Manufacturing Systems: a Survey", *Journal of Intelligent Manufacturing*, Vol. 17, pp. 111-131.

Bilge Ü, A. Polat, Y. Tunç and M. Gönen, 2002, *Real-time Shop Floor Control Implementations in BUFAIM Model Factory*, Boğaziçi University Techinal Report

Brussel H. V., J. Wyns, P. Valckenaers, L. Bongaerts and P. Peeters, 1998, "Reference Architecture for Holonic Manufacturing Systems: PROSA", *Computers in Industry*, Vol. 37, pp. 55-74.

Bongaerts L., L. Monostori, D. Mcfarlane and B. Kadar, 2000, "Hierarchy in Distributed Shop Floor Control", *Computers in Industry*, Vol. 43, pp. 123-137.

Buzacott J. A. and D.D. Yao, 1986, "Flexible Manufacturing Systems: A Review of Analytical Models", *Management Science*, Vol. 32, No. 7, pp. 790.

Cho S., 2005, "A Distributed Time-Driven Simulation Method For Enabling Real-Time Manufacturing Shop-Floor Control", *Computers and Industrial Engineering*, Vol. 49, pp. 572-590.

Dewan P. and S. Joshi, 2002, "Auction Based Distributed Scheduling In A Dynamic Job Shop Environment", *International Journal of Production Research* , Vol. 47, pp. 1173-1191.

Fishman S. G., 1978, "Grouping Observations in Digital Simulation", *Management Science*, Vol. 24, No. 5, pp.510.

Gou L., P. B. Luh and Y. Kyoya, 1998, "Holonic Manufacturing Scheduling: Architecture Cooperation Mechanisms and Implementation", *Computers in Industry*, Vol. 37, pp. 213-231.

Gönen M., 2005, *BUILD.NET: A Graphical Application Generator for Object Oriented Software and Sapmle Applications*, M. S. Thesis, Boğaziçi University.

Fujimato M. R., 2000, *Parallel and Distributed Simulation Systems*, John Wiley and Sons, New York, USA.

Kelton W. D. and A. M. Law, 1991, *Simulation Modeling and Analysis* McGraw-Hill Inc., New York, USA.

Koşucuoğlu D. and B. Boyacı, 2006, *Distributed Simulation for Real-Time Control of Automated Manufacturing Systems*, IE 492 Project Report, Boğaziçi University.

Kotak D., S. Wu, M. Fleetwod and H. Tamoto, 2003, "Agent Based Holonic Design and Operations Environment for Distributed Manufacturing", *Computers in Industry*, Vol. 52, pp. 95-108.

MacChialori R. and S. Riemma, 2002, "A Negotiation Scheme for Autonomous Agents in Job Shop Scheduling", *International Journal of Computer Integrated Manufacturing*, Vol. 15 pp. 222-232.

McDonnel P., G. Smith, S. Joshi and S. R. T. Kumara, 1999, "A Cascading Auction Protocol As a Framework For Integrating Process Planning And Heterarchical

Shop Floor Control", *The International Journal of Flexible Manufacturing Systems*, Vol. 11 pp. 37-62.

Pinedo M. R., 1995, *Scheduling Theory, Algorithms and Systems*, Prentice Hall, New Jersey, USA.

Polat A., 2003, *Simulation Based Shop Floor Control Implementation in BUFAIM*, M. S. Thesis, Boğaziçi University.

Singh N., 1996, *Systems Approach to Computer Integrated Design and Manufacturing*, John Wiley and Sons Inc, New York, USA.

Siwamogsarham T. and C. Saygın, 2004, "Auction Based Distributed Scheduling and Control Scheme for Flexible Manufacturing Systems", *International Journal of Production Research*, Vol. 40, No. 3, pp. 547-572.

Usher M. J., "Negotiation Based Routing in Job Shop via Collaborative Agents", *Journal of Intelligent Manufacturing*, Vol. 14, pp. 485-489.

Tunç Y., 2005, *Real-Time Holonic Shop Foor Control Implementation in BUFAIM*, M. S. Thesis, Boğaziçi University.

Valckenaers P. and H. V. Brussel, 2003, "Deadlock Avoidance in Flexible Flow Shops with Loops" *Journal of Intelligent Manufacturing*, Vol. 14, No. 1, pp. 137-144.

Veeramani D., K. J. Wang and J. Rojas, 1998, "Modeling and Simulation of Auction Based Shop Floor Control using Parallel Computing", *IIE Transactions*, Vol. 38, pp. 773-783.Vol. 14, pp. 199-217.

Veeramani D. and K. J. Wang, 2004, "Performance Analysis of Auction-Based Distributed Shop-Floor Control Schemes from the Perspective of the Communication System", *International Journal of Flexible Manufacturing Systems*, Vol. 9, No. 2, pp. 121-143.

Vepsalainen A. P. J. and T. E. Morton, 1987, "Priority Rules for Job Shops with Weighted Tardiness Cost", *Management Science*, Vol. 33, No. 8, pp. 1035.

Yang C. O. and J. S. Lin, 1998, "The Development of a Hybrid Hierarchical/Heterarchical Shop Floor Control System Applying Bidding Method in Job Dispatching", *Robotics and Computer Integrated Manufacturing*,

Yuh G., J. Lin and J. J. Solberg, 1992, "Integrating Shop Floor Control Using Autonomous Agents", *IIE Transactions*, Vol. 24, No. 3, pp. 57-71.