

University of Warwick institutional repository: <http://go.warwick.ac.uk/wrap>

A Thesis Submitted for the Degree of PhD at the University of Warwick

<http://go.warwick.ac.uk/wrap/36393>

This thesis is made available online and is protected by original copyright.

Please scroll down to view the document itself.

Please refer to the repository record for this item for information to help you to cite it. Our policy information is available from the repository home page.

An Agent Based Compositional Framework for Supply Chain Simulation

Raghu Arunachalam

Submitted for the degree of Doctor of Philosophy



**University of Warwick
Department of Engineering**

October 2000

CONTENTS

LIST OF FIGURES.....	i
LIST OF TABLES.....	ii
ACKNOWLEDGMENTS.....	iii
DECLARATION.....	iv
ABSTRACT.....	v
ABBREVIATIONS.....	vi
1. INTRODUCTION.....	1
1.1 Objectives.....	6
1.2 Methodology.....	6
1.3 Chapter plan.....	7
2. A REVIEW OF DISCRETE EVENT SIMULATION AND SUPPLY	
CHAIN MODELLING.....	9
2.1 The need for models.....	9
2.2 World views.....	12
2.2.1 Event based approaches.....	12
2.2.2 Activity based approaches.....	14
2.2.3 Three phase approach.....	16
2.2.4 Process based approaches.....	17

2.3 World views and problem description.....	20
2.4 Overview of developments in discrete event simulation.....	21
2.4.1 Programming approaches.....	21
2.4.2 Non programming approaches.....	23
2.5 Supply chain modelling.....	25
2.6 Supply chain dynamics.....	32
2.7 The role of DES in supply chain modelling.....	35
2.8 Requirements for DES of supply chains.....	39
2.9 Web based and distributed modelling.....	43
2.10 Conclusion.....	46
 3. LARGE SCALE MODELLING - A REVIEW.....	 48
 3.1 Modularity and the world views.....	 50
3.2 Object oriented simulation.....	60
3.3 DEVS and hierarchical modelling.....	62
3.3.1 Large scale modelling decomposition and synthesis.....	65
3.3.2 Variable structure modelling and DEVS.....	67
3.4 Coupling schemes.....	71
3.5 High level architecture.....	73
3.6 Conclusion.....	76

4. PARALLEL DISCRETE EVENT SIMULATION - A REVIEW.....	78
4.1 The problem.....	78
4.1.1 The structure of asynchronous PDES.....	79
4.1.2 Causality errors.....	80
4.2 PDES algorithms.....	82
4.2.1 Conservative approach.....	82
4.2.1.1 The Chandi and Misra approach.....	82
4.2.1.2 Synchronous approach and conservative time windows.....	86
4.2.2 Optimistic approach.....	88
4.2.2.1 Rollback control and antimessages.....	89
4.2.2.2 Global virtual time.....	91
4.2.2.3 Variations of time warp.....	93
4.3 Conservative versus optimistic techniques – a critique.....	94
4.4 PDES based simulation languages.....	96
4.5 PDES of manufacturing systems - a review.....	97
4.6 Conclusion.....	101
5. A FRAMEWORK FOR COMPOSITE MODELLING.....	103
5.1 Design rationale.....	103
5.1.1 Heterogeneous composite modelling.....	104

5.1.2 Composite model synthesis.....	106
5.2 Functional view of the architecture.....	110
5.3 Composite modelling framework - Process view.....	114
5.3.1 Model Taxonomy.....	114
5.3.2 Model selection.....	118
5.3.3 Agents and model synthesis.....	120
5.3.4 Composite model building.....	124
5.3.5 Simulation of the composite model.....	132
5.3.5.1 Model encapsulation and <i>model_agents</i>	133
5.3.5.2 Distributed computer infrastructure.....	135
5.3.5.3 Simulation super executive.....	136
5.3.5.4 Local simulation executive.....	138
5.3.5.5 PDES controller.....	139
5.4 Conclusion.....	140
 6. PDES CONTROLLER FOR HERMIS.....	 141
 6.1 The conservative approach.....	 143
6.2 Optimistic approach.....	147
6.3 PDES paradigm.....	150
6.3.1 Modified LP architecture.....	152
6.3.2 PDES algorithm.....	153
6.4 Proof of correctness.....	158

6.5 Discussion.....	166
7. CONCLUSION.....	170
7.1 Summary of work.....	170
7.2 Conclusions.....	173
7.3 Contribution.....	175
7.4 Suggestions for future work.....	176
REFERENCES.....	178
APPENDICES.....	191
A - JAVA SOURCE CODE FOR PDES CONTROLLER.....	191

LIST OF FIGURES

Figure 2.1	Forms of supply chain network [Scwarz 81].....	26
Figure 2.2	Demand amplification in a supply chain [Hulian 87].....	32
Figure 3.1	Relationship between CFG, IG and HIG.....	54
Figure 3.2	Coupling two models using the Entity-Connection view... ..	56
Figure 3.3	An example of a simplified PaInt model [Davis 96].....	57
Figure 3.4	The class Hierarchy of simulation objects in SmartSim.....	62
Figure 3.5	Separation of description of behaviour and structure in DEVS models	68
Figure 3.6	Simulation of federates in HLA.....	76
Figure 4.1	An example of an LP framework.....	80
Figure 4.2	Causal relationships.....	81
Figure 4.3	An example of deadlock.....	84
Figure 5.1	Levels of composition (modified from [Davis 96]).....	104
Figure 5.2	An example of a supply chain of a personal computer manufacturer.....	108
Figure 5.3	A functional view of the composite modelling framework.....	110
Figure 5.4	Class hierarchies: a) PC manufacturer, b) Computer monitor Manufacturer.....	116
Figure 5.5	<i>Transfer-Entity</i> heirarchy of a customer order placed with a PC Manufacturer.....	118
Figure 5.6	<i>synthesis_agent</i> architecture.....	122
Figure 5.7	Blackboard architecture of composite modeller.....	125
Figure 5.8	Supply chain of a computer manufacturer.....	129

Figure 5.9	MI hierarchy of a.Computer assembler, b. Motherboard manufacturer, c. CPU manufacturer.....	131
Figure 5.10	Architecture of simulation mechanism.....	135
Figure 6.1	Cyclic relationship in supply chain models.....	145
Figure 6.2	Modified LP architecture.....	153

LIST OF TABLES

Table 4.1	Example of event scheduling in a synchronous approach.....	87
Table 4.2	Comparison of conservative versus optimistic schemes.....	96
Table 4.3	Summary of PDES in manufacturing.....	101

ACKNOWLEDGEMENTS

During the course of my study I have been fortunate to have had the support of many.

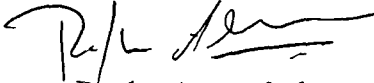
Foremost I owe my sincerest gratitude to my supervisor Mr. Rajat Roy for having been a constant source of guidance and inspiration. I would like to thank the members of the simulation team for always being helpful and a joy to be amongst.

I am grateful to the Warwick Manufacturing Group and the Department of Engineering for making my study possible by supporting me financially.

Finally, I wish to thank my parents for being a tremendous source of encouragement.

DECLARATION

I declare that the work presented in this thesis, unless otherwise acknowledged, is my own and has not been previously submitted for any academic degree.



Raghu Arunachalam

ABSTRACT

To survive in an ever increasing global and competitive marketplace, organisations are forging strategic alliances to gain a competitive advantage over their rivals. Consequently, it is now recognised that it is not sufficient to look at organisations in isolation, but view them in the wider context of the supply chain. In order to design and manage supply chains it is necessary to understand and predict the behaviour of such systems.

The ability to perform detailed studies of dynamic behaviour has made discrete event simulation (DES) an invaluable tool in the design and analysis of manufacturing systems. DES has been used to model individual stages of a supply chain, but rarely has it been applied comprehensively across the entire chain. The multi-faceted nature of supply chains makes the creation of a single model that represents all aspects of the chain difficult. A compositional framework, termed HerMIS (Heterogeneous Model Integration and Simulation), is proposed that allows pieces of a supply chain to not only be studied in isolation, but in the context of the other parts as well.

Three requirements are identified for the development of HerMIS. These are: (1) to support a compositional approach so as to allow multi-faceted modelling, (2) to function in a distributed environment where models and information about them are distributed at different locations amongst various organisations, and (3) to provide an execution mechanism that allows the composite model to be simulated efficiently.

A class based taxonomy of component models and their interaction is conceived that forms the basis of a representation scheme for composite modelling. An agent based paradigm that employs a collection of *synthesis_agents* and *model_agents* is devised to support the distributed operation of the framework. The *synthesis_agents* function as sources of knowledge for synthesising composite models and are used in conjunction with an interactive blackboard based system to guide the user in creating composite models. Each of the *model_agents* incorporate a discrete event model of a supply chain component, and supports the distributed simulation of the composite model.

Finally, a parallel discrete event simulation algorithm is proposed that enables the composite model to be simulated on a network of computer workstations. The algorithm is based on the optimistic PDES approach and takes into consideration some of the operating characteristics of a composite supply chain model.

ABBREVIATIONS

AS	Activity Scanning approach (Discussed in Chapter 2 page 14)
CFG	Control Flow Graph (Discussed in Chapter 3 Page 52)
CCP	Component Configuration Packet (Discussed in Chapter 5 Page 137)
DEDM	Discrete Event Dynamic Modelling (Discussed in Chapter 2 Page 12)
DES	Discrete Event Simulation (Discussed in Chapter 2)
DEVS	Discrete Event simulation Specification (Discussed in Chapter 3 Page 62)
ES	Event scheduling approach (Discussed in Chapter2 Page 12)
GISS	Generalised Interactive based Simulation System (Discussed in Chapter 3 Page 59)
GPPL	General Purpose Programming Language (Discussed in Chapter 2 Page 21)
GUI	Graphical User Interface
GVT	Global Virtual Time (Discussed in Chapter 4 Page 91)
HerMIS	Heterogeneous Model integration and Simulation (Discussed in Chapter 5)
HCFG	Hierarchical Control Flow Graph (Discussed in Chapter 3 Page 52)
HIG	Hierarchical Interconnection Graph (Discussed in Chapter 3 Page 52)
HLA	High Level Architecture (Discussed in Chapter 3 Page 73)
IG	Interconnection Graph (Discussed in Chapter 3 Page 52)
LP	Logical Process (Discussed in Chapter 4 Page 79)
MI	Model Interaction (Discussed in Chapter 5 Page 114)
OO	Object Oriented
OOS	Object Oriented Simulation (Discussed in Chapter 3 Page 60)
PAInt	Process Activity Interaction (Discussed in Chapter 3 Page 57)
PDES	Parallel Discrete Event Simulation (Discussed in Chapter 4)

PI	Process Interaction (Discussed in Chapter 2 Page 17)
PP	Physical Process (Discussed in Chapter 4 Page 79)
SES	System Entity Structure (Discussed in Chapter 3 Page 65)
SPL	Simulation Programming Language (Discussed in Chapter 2 Page 22)

CHAPTER 1

Introduction

In an increasingly global and competitive market place, the notion of supply chain and its management has been the focus of many researchers and practitioners alike. In a fast changing environment organisations are focusing on their core competencies and forging strategic alliances that share knowledge and resources to gain a competitive advantage and retain flexibility over their competitors.

A number of terms such as extended enterprise, virtual enterprises [Bleecker 94] and symbiotic networks [Alter 93] have been used to describe these integrated enterprises. In order to get the benefit of such integrated enterprises it is clearly not enough to optimise one part of the supply chain but to look at it as a whole. In addition to understanding the components of the system, it is important to understand the relationship between the various components and the behaviour that consequently emerges.

The developments in network technologies such as the internet and the use of EDI has enabled the creation of inter-organisational systems that support these emerging close relationships between organisations. However, the primary focus has been on implementing transaction systems, such as order, invoice, and payment systems [Turner 93]. In addition to integrating transaction processes across the supply chain, there is also

a need for integrated decision support systems that help to plan and predict the performance of the system.

Discrete event simulation (DES) has been used as a tool to analyse the performance of existing and proposed manufacturing systems for a number of years now. It has been used to support decision making at various levels from strategic analysis of a proposed plant to aiding day to day operations, for instance, in a scheduling system. The ability to create detailed models that closely conform in structure, dynamic and stochastic behaviour, to the manufacturing system being modelled, has made DES an attractive technique for what – if analysis. However, DES has largely been restricted to modelling individual pieces, such as a single production facility of a supply chain and is seldom employed to model the *entire* supply chain [Geller 95]. Even in cases where an attempt has been made to model the supply chain more comprehensively, the modelling effort can often be described as ‘modelling in the small’, a term coined by Zeigler [Zeigler 84].

Often a model is created with the purpose of aiding a given decision objective. For instance, the objective may be to study the potential capacity of a proposed plant. Once this objective is met the model is viewed as having served its purpose and is often discarded. Even in the case of a DES model developed in the context of a scheduling system, where a model is used over a long period of time, it is still generally confined to this application. If for instance the scheduling problem is to be studied at a supply chain level instead of at a local level, then typically, a new model is created from scratch rather than integrating several local models. Ulgen and Gunal [Ulgen 98] estimate that seventy

to eighty percent of simulation models developed in the automobile industry fall into this category of ‘modelling in the small’.

The nature of supply chain management is such that a multiplicity of objectives needs to be addressed. DeKok and Bertrand [DeKok 95] use the notion of a hypothetical supply chain manager and production manager to illustrate the division in responsibility and consequently the objectives in controlling the supply chain. The supply chain manager is concerned with objectives at a global (supply chain wide) level. This may include performance measures, such as lead-times of individual production facilities in the supply chain or inventory levels between production facilities etc. His aim is to improve the overall performance of the supply chain. On the other hand, the production manager is concerned with objectives at a more local level. His objectives are to take into consideration local constraints such as limited capacity and/or limited inventory storage etc. and still meet his obligation to his customers. In addition, within a particular entity of the supply chain, decision-making objectives may vary based on the management level of the decision-maker. For instance at a corporate level the objectives may deal with capacity, production rate etc, while at a shop floor level one may be concerned with sequencing and scheduling on an hour by hour basis. This existence of a multiplicity of objectives in the management and control of a system needs to be addressed in the development of a modelling methodology that supports decision making. Zeigler [Zeigler 84] uses the term multifaceted modelling to describe such a methodology.

Two general approaches are used in the design of manufacturing systems: aggregate refinement and decomposition [Heim 94]. Aggregate refinement attempts to model a system using a single model that incorporates details of all aspects of the systems. Such models allow a more holistic view of the system in consideration. However, when dealing with large systems one is restricted to building models that are fairly coarse (not detailed, rough cut), since attempting to increase the detail of representation of the system can result in large and unwieldy models that are hard to maintain and verify. The decomposition approach, on the other hand, is based on the 'divide and conquer' strategy. Instead of modelling all aspects of the system in a single model, the system is partitioned into a number of independent components each of which are modelled separately. The decomposition approach simplifies building models of complex systems. However, treating the system as being composed of independent units prevents a more 'holistic' analysis. Creating models by decomposition allows one to perform detailed analysis of parts of a system in a manageable fashion, but ignores potentially vital interactions between the components of a system.

Both the approaches viz. aggregation refinement and modelling by decomposition are useful but not sufficient. In the context of supply chain modelling, an aggregation refinement approach may be used to create a coarse model of the supply chain, whereas the decomposition approach may be used to model individual manufacturing facilities at a more detailed level. However, quite often there is a need for a model that is detailed but still takes into account the broader context in which it operates. In addition, creating new models from scratch for every objective is wasteful. Instead, a compositional approach that reuses models and allows models representing the various facets of the system to be

simulated either in isolation or together, may be more appropriate. In this way knowledge embedded in previously constructed models can be reused and new models tailored from existing ones.

A multifaceted modelling methodology needs to support modelling at two levels, viz. model building and composite model building. At the model building level the methodology needs to be based on sound modelling and software principles. This would allow for good design, enable easy model development, documentation, reuse etc. In addition, models need to be based on a modular design that incorporates a well defined interface so as to allow the inclusion of the model in a model composite and thus support multifaceted modelling. At the composite model building level the methodology needs to support an integrated framework that allows the management of the models that represent the various facets of the system. The framework needs to provide a means of storing, classifying and retrieving models, and a mechanism for synthesising composite models from them. Once created, the composite model needs to be simulated to satisfy the objectives of the modelling exercise. Thus, the framework needs to include a means of simulating the integrated model.

A number of modelling formalisms, worldviews, frameworks, and tools currently exist. However, no single formalism/tool satisfies all users [Goble 91] [Hooper 86]. Due to a combination of conceptual ease, cost, skill etc., modellers select appropriate formalisms/tools to build their models. This thesis deals with developing a framework that supports multifaceted modelling at the composite model building level by coupling together models built using existing model building formalisms and tools.

1.1 Objectives

The primary objective of this thesis is to develop a multifaceted discrete event modelling framework, based on a compositional approach, for the modelling of supply chains. In order to achieve this the following objectives also need to be met.

1. Analyse and identify the fundamental concepts and components required for the framework.
2. Review literature to evaluate how the required concepts and components have been applied.
3. Develop concepts, structures, and relationships of a framework that can be used to aid multifaceted modelling of supply chains.
4. Develop a mechanism to simulate the composite model created by the framework.

1.2 Methodology

We employ a ‘Phenomenological, descriptive-Interpretive’ approach [Galliers 87] in the design of the modelling framework.

The field of study is reviewed in breadth to identify the various techniques applied to solve the design problem. The modelling framework is then developed by integrating and

synthesising the various approaches to arrive at a novel approach that satisfies the objective of the research.

1.3 Chapter plan

The thesis is composed of seven chapters. Chapter 2 begins by reviewing the area of discrete event simulation. This is followed by a review of the type of supply chain models found in the literature. Finally, a set of broad requirements for the modelling framework is identified.

Chapter 3 and Chapter 4 review concepts found in the literature that are relevant to the requirements identified in the previous chapter. Chapter 3 reports how a number of researchers have tackled issues relating to composite modelling. Modularity, coupling schemes and the DEVS methodology [Zeigler 84] are some of the items described in this chapter. Chapter 4 reviews the area of parallel discrete event simulation (PDES). A number of algorithms are critically reviewed in this chapter.

Chapter 5 and Chapter 6 describe a compositional modelling framework for the modelling of supply chains. Chapter 5 presents aspects relating to the building of a composite model. In Chapter 6 the development of a mechanism for simulating the composite model is described. A novel PDES algorithm is presented in this regard and a mathematical proof of validity is included.

Chapter 7 provides a summary of the work reported in the thesis and conclusions are drawn. Finally, a few suggestions for future work are expressed.

CHAPTER 2

A review of discrete event simulation and supply chain modelling

This chapter begins by presenting a review of discrete event simulation in general and its application in modelling manufacturing systems in particular. The objective here is to introduce basic concepts and terminology and provide an overview of simulation techniques currently used. This is followed by a review of various attempts in the literature to model supply chains.

2.1 The need for models

The unceasing quest for knowledge and the desire to control one's destiny has been a fundamental characteristic of mankind throughout their history. The ability to create models of the world around us and use these has been an important part of this endeavour. Rosenbluth and Wiener [Rosenbluth 45] substantiate the importance of model building by saying:

“No substantial part of the universe is so simple that it can be grasped and controlled without abstraction. Abstraction consists in replacing the part of the universe under

consideration by a model of similar but simpler structure. Models are thus a central necessity of scientific procedure."

A number of definitions of models exist in the literature. Minsky [Minsky 86] defines models quite simply as: "*any structure that a person can use to simulate or anticipate the behaviour of something else*". He considers models, in particular 'mental models', as embodiment of knowledge. For the discussion here a definition of models provided by Pidd [Pidd 94] for use in the context of management systems is used.

"A model is an external and explicit representation of part of reality as seen by the people who wish to use that model to understand, to change, to manage and to control that part of reality."

Models can take a multitude of forms. For instance, models could be mental models (as described by Minsky earlier) or physical models (scale models) as in the case of aircraft wind tunnel models or mathematical models such as a differential equation that models the behaviour of a chemical reaction or logical models where the behaviour of a system is described by a set of logical rules. Typically, these rules are encoded on a computer and simulated. Naylor et al. define simulation as: "a numerical technique for conducting experiments on a digital computer, which involves certain types of mathematical and logical models".

Computer simulation models can be classified in several ways [Rubinstein 98]:

1. Static versus Dynamic Models: Static models are those that do not evolve over time. In contrast, dynamic models represent the behaviour of systems over time.
2. Deterministic versus Stochastic Models: Models that incorporate at least a single random variable in the representation of the model are termed stochastic models, while models that incorporate non random (deterministic) variables exclusively in their representation are termed deterministic models.
3. Continuous versus Discrete Models: Models can also be classified in the way the notion of time is handled. In continuous models the state of the model changes continuously with respect to time. Continuous models generally employ a system of differential or difference equations to express a model of a particular system. Examples of continuous models include models of air flow on aircraft wings, models of chemical reactions and system dynamics models etc. Simulation of these models are performed by solving the differential equations either on an analogue computer or digital computer. On the other hand, discrete models update their state instantaneously at a finite number of discrete points in time. The manner in which the state is transformed is expressed using a logical state transition function as opposed to mathematical equations. Simulation is performed by employing discrete event simulation software on a digital computer. Discrete event models are usually used in the modelling of queuing network systems such as manufacturing systems etc.

This thesis deals only with discrete, dynamic and stochastic simulation models. Such models have collectively been termed as *discrete event dynamic models* (DEDM) or *discrete event simulation* (DES) in the literature. Unless otherwise mentioned, subsequent references to models mean DEDM.

2.2 World views

Early work in the area of DES concentrated on ways of conceptualising the problem in a manner so as to enable it to be encoded as a set of computer routines. The static structure was described as a set of entities with its associated attributes, which provided a data processing environment in which the dynamic behaviour was simulated [Kiviat 69]. The dynamic structure involved structuring the behaviour such that one can identify when the next event will occur and what routine to execute at that point [Tocher 65]. Three basic ways of structuring the problem into computer routines evolved from the early work in modelling - namely, event based, activity scanning and process interaction [Kiviat 69]. These were termed world views as they allowed the structuring of a model based on different perspectives of the system.

2.2.1 Event based approaches

Event based approaches [Markowitz 63] [Fishman 73] were the most commonly used approach in the early days of simulation. This was partly due to the popularity of SIMSCRIPT, a simulation language based on the work by Markowitz [Markowitz 63] at

the RAND co-operation. However, later versions of SIMSCRIPT have emphasised the process interaction view, and this has been one of the reasons for the decline in recent years of event based approaches[Pidd 88]. Other reasons for the lack of popularity of the event based approach are discussed later in this section.

An event based approach consists of a set of event routines that capture the changes of state that occur at different points in time in the flow of the simulation. The routines describe the consequences of the occurrence of an event. The consequences are represented as event notices, which identify the time at which these events occur, and are scheduled on to the event list. An event based control program provides a list, termed the event list, into which event routines schedule event notices. This has prompted the use of the term event scheduling (ES) for describing this approach. The simulation control program maintains the list in increasing order of time of event notices. Simulation occurs by selecting the next event in the event list and executing the relevant event routine.

A major drawback cited by a number of researchers [Pidd 88] [Laski 65] is the inherent difficulty in developing event routines. In the case of simple models where limited interaction occurs between entities this may not be a problem, but developing event routines for more complicated models can be difficult and error prone. This is because the effect of all the consequences of executing an event need to be taken into consideration in developing the event routines. Further, this also inhibits modification and incremental building of models. A small change in the model may involve the reworking of a number of event routines.

The flip side of increased complexity of model development is that event based approaches, in general, are more efficient than the other world view approaches in terms of runtime.

2.2.2 Activity based approaches

One approach to simplifying the model development process in relation to the event based approach, is to take the complexity out of the model building and transfer it to the simulation executive. Thus, rather than have the modeller determining all the consequences of the occurrence of an event, the modeller simply describes the conditions upon which events occur and the simulation executive can be used to determine when these conditions are satisfied and schedule events. Activity based approaches [Buxton 62] [Tocher 65] are based on this philosophy.

An activity routine consists of a test head, that describes a set of conditions that need to be true so as to schedule the start of the activity, and a number of action statements. The action statement expresses the change of state accompanied by the start of the activity and the time at which the activity will complete, i.e. end event. The advantage of including a set of conditions in the activity is that unlike in the event based approaches the consequences of the execution of an event need not be explicitly mentioned. Instead, a set of conditions is used to evaluate the scheduling of the events.

The conditions in the test head of an activity are based on the status of entities or the simulation time or both. If all the necessary entities required for the activity to commence are available, the action statements in the activity are executed and the entities are made unavailable to other activities. Each of the entities are associated with a time cell which informs when in the future the entity will again become available.

The simulation executive scans the time cells of the entities and determines when the next entity is going to become free, indicating the next event time. This entity is then made available. The test heads of the activities are then scanned to see if this change in status of entities triggers any of the activities. Activities that have been triggered then execute their action statements. This process is then repeated. The need to scan the entities and activities repeatedly, in order to progress the simulation, has led to the name activity scanning (AS) by some researchers.

A key feature of the activity scanning approach is that the activities are represented as compact self-contained units. Conditions and actions that are only relevant to the activity are included in its description, thus enabling greater clarity and model extensibility. The downside is that as the consequences of executing events are not determined beforehand, as is the case in the ES approach, repeated scanning of entities and activities are required to progress the simulation time, resulting in significant overhead in runtime of the simulation

In practice, due to the poor run time performance of AS models this approach is rarely used in its original form. The three phase approach, a hybrid, that combines some of the runtime benefits of the ES approach while improving on the representational ease of AS has to a large degree superseded the AS approach.

2.2.3 Three phase approach

The three phase was first described by Tocher [Tocher 63] in the context of modelling a steel mill. Tocher argues that in most real world systems two types of events exist, viz. conditional events and bound events. Conditional events are triggered when a certain condition is true. On the other hand, bound events are those that follow the completion of another event. Distinguishing between the two types of events allows for creating a hybrid approach. Bound events can be scheduled directly as in the ES approach, while Conditional events can utilise a scanning procedure, akin to the AS view.

The separation of bound and conditional events also allow for a more modular representation of activities. In the AS approach (original) the description of the activities were fragmented as the start and end events are described in different activity routines. The three phase approach allows for a more complete definition of an activity, where an activity describes a period of time during which one or more entities are engaged in some aspect of behaviour of the real world system. An activity is described by a conditional event and one or more bound events. The conditional event describes the conditions for the start of the activity and the bound events schedule the termination.

Simulation occurs in three phases- 'A' phase, which advances the time to the next scheduled event, 'B' phase that executes all scheduled events at that time, 'C' phase scans the conditional events and executes them. Thus, this approach has been termed the three phase approach or the ABC approach.

2.2.4 Process based approaches

The various world views conceptualise the modelling problem in different ways. The ES approach considers all the events that occur in a system and models them as individual event routines, which describe all the consequences of executing them. On the other hand, the AS approach breaks down the behaviour of the system into a number of unique activities and describes their behaviour. In a sense, both these approaches describe the consequences of the occurrence of an event on the entities in the simulation. In the event based approach, all the consequences of executing an event are located in that event routine, while in the case of the AS approach only the consequences on the entities that are involved in the activity are included.

Process based approaches, as the name suggests, conceptualises a system as consisting of a number of processes each describing the behaviour of an entity or class of entities throughout its life cycle.

The system can be modelled by either representing the life cycle of temporary entities (parts, WIP), as in GPSS, or representing the behaviour of permanent entities (machines),

as is possible in SIMULA. The former is termed as a transaction based process view and the latter a server based process view.

As a process has to include the passage of time in its description, there needs to be a method of synchronising the various process routines. This is accomplished by deactivating and reactivating the execution of the process. A process may be deactivated for a fixed period of time or until some condition is met. The former is referred to as an unconditional delay and the latter as a conditional delay [Pidd 88]. In the case of an unconditional delay the reactivation of the process can be scheduled in an event list. Conditional delays, however, require that after the execution of every event the process be checked for reactivation.

Interaction between processes are commonly described using some intermediate object [Franta 79] [Birtwistle 79], Mutual exclusion and producer – consumer synchronisation are two such approaches. In mutual exclusion, synchronisation is achieved by acquiring and releasing a common resource. This is used to model a resource in a transaction based process view. Producer – Consumer synchronisation is used in a server based view where temporary entities are exchanged via a common buffer.

The simulation executive of a process based approach is responsible, at each point in the simulation, in moving the entity as far as possible through its process cycle. A popular approach in accomplishing this is to use two lists – a future event list and a current event list. The future event list contains entities (processes) that are unconditionally delayed

and are scheduled to activate at some fixed time in the future. This list is used to determine the time of next event. The current event list contains two types of entities. Entities that are waiting to be reactivated based on some condition, i.e. entities that are conditionally delayed, and unconditionally delayed entities that are to reactivate at the current simulation time. The simulation progresses in three stages as follows:

1. Future event scan: This involves finding the unconditional entities that are to be reactivated next. The simulation clock is then updated to this value.
2. Move entities: The entities in the future event list that are to be reactivated at the current simulation time are moved to the current event list.
3. Current event scan: The entities in the current event list are scanned repeatedly with a view to move the entities through their processes. An entity that is permitted to reactivate will move through its process until the termination of the process or a further delay. The entities are then rescheduled in the future event list (unconditional delay) or the current event list (conditional delay). The scan is performed repeatedly until none of the entities in the current event list can move further through their process. The executive then goes back to step 1 and the simulation continues.

The PI view has the benefit of being the more natural way of representation compared to the other world views [Pidd 88]. This has to do with the similarity between the way a modeller, in particular a novice, conceives a model and the manner in which the routines are structured in the PI view. It is common for a modeller to conceive the working of a

system by breaking it down into a number of entities and then understanding the behaviour of each of them in turn.

2.3 World views and problem description

Behaviour of production systems are commonly expressed in one of two ways: from the perspective of the machines or from the perspective of the material that flow through the system. Tocher [Tocher 65] categorised a number of simulation packages available at that time based on the perspective of their model description, i.e. machine based or material based. He suggests that both perspectives could be used for modelling manufacturing systems; however, one or the other would be more suitable depending on the nature of the manufacturing system.

In the case of a mass production system, where there is low variability in the type and characteristics of the materials that flow through the system and considerable complexity in the structure and behaviour of the machines, a machine based view may be more appropriate. Conversely, he states that in systems with high material variability in conjunction with a simple machine behaviour and layout, modelling using a material based focus may be more suitable.

The three world views described in the previous section may be used to conceptualise models in either perspective. However, in practice one perspective may be more natural than the other for a particular world view. Activity and event based approaches are most

commonly associated with machine based representation while process based systems are more natural with a material based focus.

2.4 Overview of developments in discrete event simulation

Discrete event simulation software in one form or another has existed over a period of forty years and to varying degrees has paralleled the development of computer technology. Simulation software today in general belongs to one of the two categories: programmable approaches and non-programmable approaches. Although, a few simulation packages exhibit characteristics of either category; an example of this is SIMPLE ++ [Aesop 94]. SIMPLE ++ is basically a simulation programming environment but it also includes templates of common domain objects that can be customised and integrated with a GUI to create models.

2.4.1 Programming approaches

Early attempts at building simulation models were restricted exclusively to being built using general purpose programming languages (GPPL). As simulation models are a type of software, using a GPPL provides the most direct route to model development. However, the process can be quite arduous, particularly for large models. Programs that represent simulation models can be thought of as consisting of three levels of routines – level 1 which provides routines for the executive that manages the scheduling of the events and progressing the simulation clock, level 2 contains routines that specify the model, and level 3 which includes auxiliary routines that aid the simulation process such

as statistical distributions, random number generators etc. As only the level two routines vary from model to model, reusing the level one and level three routines can reduce the development time of models.

Some of the early simulation software was based around this idea. Routines that reappear in every simulation such as the level one and level three routines were provided as libraries in a GPPL. Models were coded in a GPPL and appropriate routines from the libraries were then incorporated to create the simulation model. An early example of a simulation library was SIMON [Hills 65], the initial version of which was based on ALGOL and later adapted to FORTRAN. More recently, SEEWHEY [Fiddy 81] offered a library of FORTRAN routines that supported an event scheduling approach and included visual interactive support.

A disadvantage of the flexibility offered by a GPPL is that it provides a more general syntax, the use of which may not be natural to the modelling task. Simulation programming languages (SPL) provide a more problem specific syntax. The syntax is designed to bear a closer resemblance to the way in which the modelling problem is conceptualised, thus simplifying the process of translating the model into computer code. In addition, SPLs typically include a hidden executive that handles the sequencing and scheduling of events. The modeller, thus, focuses his attention on describing the behaviour of the model in terms of the syntax provided.

The combination of a hidden executive and modelling view implicit in the syntax may result in a simpler, in relation to GPPL, albeit less flexible tool for model building. A number of SPLs have been developed over the years. Some of the popular languages are SIMULA [Birtwistle 79], the SIMSCRIPT family [Markowitz 63], and more recently MODSIM [CACI 93].

2.4.2 Non programming approaches

Developing models in either a GPPL or a SPL requires one to be skilled in programming. Some of the early non programming approaches to developing simulations were based on the idea of what has been termed ‘Block Structured Languages’ by Pidd [Pidd 95]. These were developed with an aim to allow non-programmers, typically experts in the domain system, to be able to create models. Block structured languages provide a diagrammatic representation scheme akin to flowcharts. A set of symbols or blocks is provided from which a flow diagram that represents the system is constructed on paper. Each of the blocks in the diagram is associated with a set of attributes which customises how the block behaves in the model. The modeller then translates the diagram into a sequence of commands and attributes, each representing a block. GPSS (General Purpose Simulation System) [Greenberg 72] and HOCUS [Hills 71] were two of the earliest modelling systems and SIMAN [Pegden 90] is a more recent addition to this category of modelling software. GPSS and SIMAN use block diagrams to represent a transaction oriented process view, while HOCUS employs activity cycle diagrams to describe an activity based approach to modelling.

The provision of blocks and their associated attributes avoids the need for programming and therefore simplifies model building. The benefit however come with the cost of decreased flexibility. For instance, the origins of GPSS lie in the modelling of telecommunication networks. Consequently, modelling systems that are not similar in nature will be harder to represent. GPSS, due to its transaction based process view, has some inherent limitations. For example, systems that involve limited interaction between transactions are easier to model, however if complicated interactions occur between transactions, then modelling using GPSS may not be appropriate [Pidd 94] [Gordon 79].

Block structured systems were designed in the era of non-interactive and text based computing. With the advent of interactive GUI based computing, a number of simulation packages have appeared that use these features to aid the description of models. Law and Kelton [Law 91] use the term simulators while Pidd [Pidd 95] terms them as ‘visual interactive modelling systems’ (VIMS). A simulator provides a set of pre-built objects that correspond closely to the objects in a real life system of a particular domain. Icons are used to represent these objects and a GUI is used to customise and integrate these objects to create a simulation model. The presence of pre-defined objects makes it simpler for modellers to conceptualise and create models, but affects the modelling flexibility. As long as the system to be modelled bears a close resemblance to the structure of the simulator, model building can be greatly simplified. Examples of simulators are WITNESS [ISTEL 96] and Arena [SMC 93] which is based on SIMAN. Indeed, SIMAN, a block structured system, now comes as part of ARENA, a VIMS system, thus ensuing the benefits of such systems.

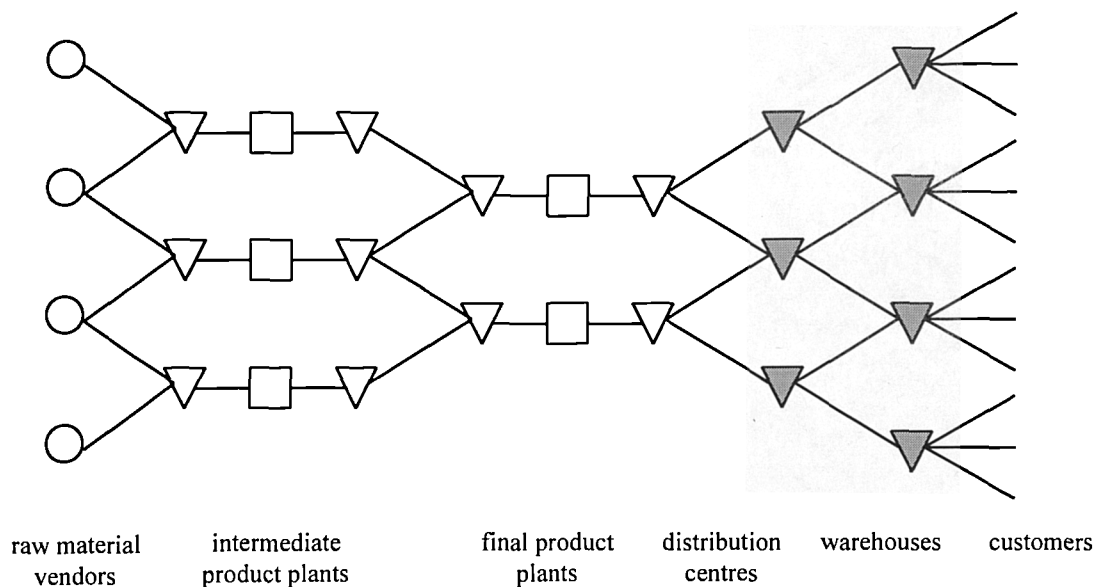
2.5 Supply chain modelling

In this section modelling techniques found in the literature to model supply chains are reviewed. The review is not exhaustive or detailed but the objective here is to provide a flavour of the various attempts at modelling supply chains and bring out some of the issues involved in using the various techniques.

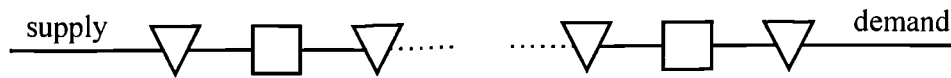
A supply chain is the network of organisations that are involved, through upstream and downstream linkages, in the different processes and activities that produce value in the form of products and services in the hands of the ultimate customer [Christopher 92]. Typically, supply chains consist of three stages: procurement, production and distribution [Thomas 96]. Each of these stages may in turn involve a number of facilities that may cross organisational and geographical borders. Models of supply chains (Figure [2.1]) have used a network structure of nodes and links to describe the various activities involved in the supply chain, where the nodes represent production or stocking facilities and the links between nodes describe the flow of information and material between the various facilities of the supply chain. The procurement stage of the supply chain consists of a network of stocking facilities that control and store the ordering of raw materials or intermediate products required by the production facilities. The production facilities consume the material provided by the procurement stage and transform it into the final product. The transformation process involves the flow of material through a sequence of processing facilities linked by intermediate stocking points. Finally, the distribution stage consists of a network of stocking nodes that direct the product from the production

facilities to the final customer. The stocking nodes consist of various levels of warehouses and the final retail outlet where the customer demand is satisfied.

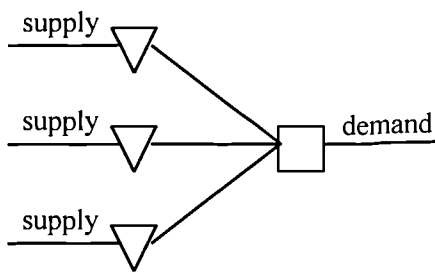
Although the various stages of a supply chain may be organisationally or geographically separated, decisions made at each stage have an effect on other stages. For example, lot sizes at the production stage through their effect on material procurement patterns affect the stocking policy at the procurement stage. Similarly, lead-times of products affect the stocking policy at the distribution stage of the supply chain. The aim of supply chain modelling is to study these interactions between the various stages of the supply chain in order to better control and manage the performance of the supply chain in its entirety.



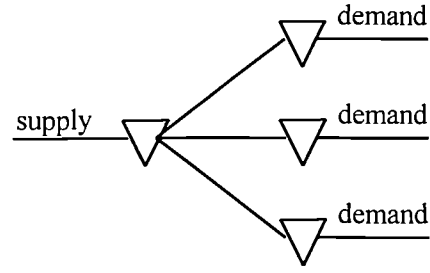
(a) A general supply chain network



(b) Serial structure



(c) Assembly structure



(d) Arborescent structure



inventory



production facility

Figure 2.1 Forms of supply chain network [Schwarz 81]

The supply chain network illustrated in Figure [2.1 a] is a general representation and in practice is quite complicated to use in developing models. A number of attempts in the literature have instead focussed on simplifying the supply chain network. Schwarz [Schwarz 81] has identified three such networks:

1. **Serial structure:** The network (Figure [2.1b]) is made of nodes such that every node, apart from the first and last one, connects to only one other predecessor and successor node. The simplicity of such networks attracted their use in a number of early models. Supply chains in practice are rarely structured in such a manner and consequently their value is limited.
2. **Assembly structure:** Here the network (Figure [2.1c]) is composed such that each node has exactly one successor node, but can potentially have a number of predecessor nodes. These networks typically represent assembly operations where a number of parts/subassemblies feed into a production node where they are assembled into a single subassembly/product.
3. **Arborescent structure:** These networks (Figure [2.1d]) are usually useful in structuring the distribution stage of the supply chain. Each node is restricted to exactly one predecessor node, but can be connected to a number of successor nodes.

v

Supply chain models have been classified in the literature by many authors [Thomas 96] [Vidal 97] [Ballou 92] based on the level at which they aid the decision making process. Operational models have been developed to aid short-term decision making, typically involving a time horizon in days or even hours. Operational supply chain models deal with issues such as selection of production batch sizes, choice of transportation mode etc. Strategic models, on the other hand, have a longer time horizon, typically in months or years. They deal with issues such as opening and closing of facilities (both plant and distribution centres), selection of location for the production of a new product, optimising on taxes and duties incurred and so on.

Much of the past effort in modelling supply chains has not considered the supply chain in its entirety, thus avoiding the ensuing complexity. Instead, researchers have tended to focus only on parts of the supply chain. Thomas and Griffin [Thomas 96] classify operational models that take into account only part of the supply chain into three categories: buyer-vendor, production-distribution, and inventory-distribution models.

Buyer-vendor models deal with the buying and selling of raw material and/or intermediate components (subassemblies) between the various stages of the supply chains. Traditionally, inventory models have ignored vendor issues and focussed instead on determining the optimal order quantities for the purchaser. Thomas and Griffin [Thomas 96] point out that focussing solely on the purchaser ignores two potential opportunities. Firstly, it may be possible to reduce costs without changing the ordering policy by investing in newer material handling systems or data handling technology, such as electronic data interchange (EDI). Secondly, the firms can jointly determine an order quantity that is optimal to the buyer and the vendor. Determining an optimal quantity for the buyer and vendor is the aim of many of the models presented in the literature. Monahan [Monahan 84] considers the case of a single product single buyer - single vendor structure. Assuming that vendors purchase in a lot for lot fashion, Monahan develops an expression that determines the factor 'K' by which the optimal order quantity should be increased by.

Lee and Rosenblatt [Lee 86] extend the work of Monahan by allowing the vendor to purchase in any quantity rather than lot for lot and requiring a minimum profit margin. They present an algorithm that finds the optimal order quantity increase factor 'K' and the optimal order quantity for the vendor. They show that the optimal order quantity for the vendor is an integral multiple of the optimal order quantity of the buyer.

Other authors have looked at alternatives to the single buyer – single vendor structure. Kohli and Park [Kohli 94] consider a single vendor – group of buyers scenario. They investigate the case of a homogeneous group of buyers that buy from a single vendor. They exploit the reduction of transaction costs attained by joint ordering. They derive expressions for optimal joint ordering quantities. Lau and Lau [Lau 94] investigate single buyer – multiple vendor structures where the buyer has a choice between a low cost, high lead-time supplier and a high cost, low-lead time supplier. They obtain expressions assuming deterministic demand and stochastic lead-time of a known distribution, for a total cost function that is dependent on the optimal order quantity, the optimal reorder point and the ratio of the orders placed with each of the suppliers.

Production-distribution models link aspects of production with those of distribution planning. They look at the relationship between production batch sizes and distribution batch sizes, impact of selecting various transportation options etc. Although one can find a number of examples of production planning models or distribution planning models in the literature, models that consider the two jointly are rare. Part of the difficulty in creating such models has to do with the fact that either of these aspects (production

planning or distribution planning) of the supply chain are quite difficult to model individually [Thomas 96]. Consequently, the models are complicated to formulate or a number of simplifications need to be made to make the model more tractable. Examples of the latter are Production-distribution models developed by Benjamin [Benjamin 90] and Haq et al. [Haq 91]. Both these models assume linear transport costs and thus have limited applicability in practice.

Much of the early effort in supply chain co-ordination was in the area of modelling inventory-distribution systems. Clark and Scarf [Clark 60] pioneered some of the earliest work in this area. They consider a serial inventory system with stochastic customer demand. Penalty costs are used to enable co-ordination between the inventories. The penalty cost charged on an inventory is the total cost of the succeeding inventory, which is incurred due to the failure of the inventory under consideration to deliver the ordered quantity.

Finally, Cohen and Lee [Cohen 88] present an ambitious attempt to include all parts of the supply chain in a single model. They describe a model that consists of four analytical sub-models, viz. (1) Material control, (2) production control, (3) finished goods stockpile, and (4) distribution network control. Each of these are tractable stochastic models and can be solved to optimise relevant operating policies for each of the sub-models. However, optimising performance measures for the entire chain requires a heuristic procedure due to the intractable nature of the computations involved. Although Cohen and Lee make a number of assumptions in deriving the model that restrict its

application in practice, the fact that the various aspects of a supply chain were integrated in a single model makes this a good starting point for future models [Slats 95]

2.6 Supply chain dynamics

A frequently cited example that epitomises supply chain dynamics is given by Houlihan [Houlihan 87]. He describes a supply chain comprising five stages that are connected to one another serially as shown in Figure [2.2].

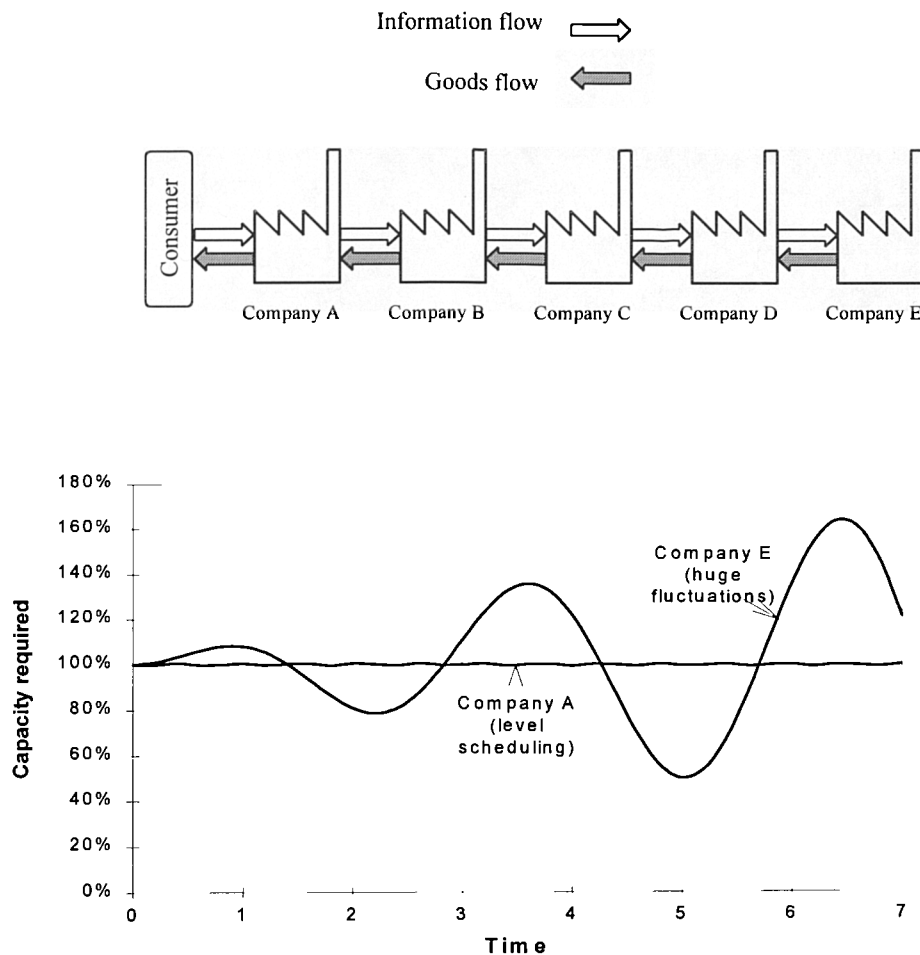


Figure 2.2 Demand amplification in a supply chain [Houlihan 87]

If for instance, a small variation in customer demand occurs at stage A then this variation in demand is propagated to the subsequent stages. However, due to the inevitable time delays in transmission and the various control policies at each of the stages, the responses tend to exaggerate at each subsequent stage. Thus, what was a small variation in demand of a few percent at stage A results in a much larger variation at stage E as illustrated in Figure [2.2]. This phenomenon of demand amplification has been coined by Burbidge [Burbidge 84] as the law of industrial dynamics. He states, “ If demand for products is transmitted along a series of inventories using stock control ordering, then the demand variation will increase with each transfer”.

This phenomenon was first studied by Forrester [Forrester 61] and presented in his book **Industrial dynamics**. Forrester applied the concepts and techniques of control theory to study the dynamic behaviour of industrial systems. He presents the case of a production-distribution system that consists of four stages namely, a factory, a warehouse, a distributor inventory and a retailer inventory. The flow of information and material between the various stages of the supply chain are modelled. Unlike a DES model, the behaviour of individual entities and interactions between them are not expressed in a descriptive fashion. Instead, the rates of flow of information and goods between the various stages and the manner in which they vary with time are expressed using first order difference equations. The model is described as a feedback system where decisions to change system parameters such as production rate are based on current state and the previous state. An important issue in control theory is the study of the stability of feedback systems. Stability deals with determining the response of a feedback system to

changes in input. Forrester uses these techniques to study the dynamic behaviour of the above mentioned production distribution system. He uses a system of 72 first order difference equations that describe the relationship between the stock levels, production rates, rate of orders placed by customers etc. The production-distribution system is simulated by solving the system of difference equations at each point in time to determine the values of the various system variables.

Forrester uses the model of the production-distribution system to demonstrate the amplification of demand as it propagates through the stages of the supply chain, due to an initial small change in customer demand at the retailer. He studies the effects of various production, distribution, and storage policies on demand amplification. He considers three ways of reducing demand amplification:

1. By reducing delays in the system so as to allow a quicker response to any changes in demand. For example, by decreasing production time and delays in order processing etc.
2. By limiting the number of stages in the supply chain. Forrester reports improvement in system stability by eliminating the distribution stage in his production-distribution model.
3. By changing inventory policy. He reported an improvement by reducing the re-order quantity and increasing re-ordering frequency.

More recently Towill and his colleagues at the University of Wales at Cardiff have contributed to the area of supply chain dynamics. Towill [Towill 93] discusses how the systems dynamics methodology can be extended by incorporating ideas of servo theory. Towill [Towill 91] illustrates that significant reduction in demand amplification can be achieved by integrating various decision making mechanisms in the supply chain. This allows the better use of information available rather than acting only on distorted orders/information received from the previous echelon in the supply chain.

The phenomenon of demand amplification and fluctuation in stock levels has also been exhibited by using other modelling techniques. Southhall et al [Southall 88] describe the use of a discrete event model to study this phenomenon. In addition to studying stepwise responses as in the case of the Forrester model, they incorporate the existence of uncertainty such as random fluctuation of customer demand and also in production lead-times and study their effects on supply chain dynamics. Lee et al [Lee 97] study the effects of systems dynamics within the framework of classical inventory theory. They prove the existence of demand distortion and its amplification as it propagates upstream in a supply chain. They use the term ‘bull whip effect’ to describe this.

2.7 The role of DES in supply chain modelling

A large proportion of the supply chain modelling effort, as is evident from the review in the previous section, has focussed on developing analytical models of supply chains. Models have been based on mathematical formulations of queuing theory, multiple integer programming (MIP), difference equations and so on. However, such models are

limited in the level of detail they incorporate. For example, questions such as what are the effects of a machine breakdown are often important in the design and management of the supply chain. This requires not only studying the steady state behaviour of the system but also investigating the dynamic behaviour of the system at a very detailed level. Most analytical models, systems dynamics models being an exception, evaluate behaviour under steady state conditions and ignore transient affects. In addition, researchers as opposed to practitioners are often tempted by the potential elegance and simplicity offered by closed form solutions. However, in practice problems in supply chains tend to be complicated and finding simple closed form solutions are far from trivial. A number of authors have questioned the applicability of analytical models in practice [Slats 95] [Askin 93]. Slats et al [Slats 95] cite that a large proportion of analytical models fall prey to the need to make a number of unrealistic assumptions in order to reduce the complexity and make the computation tractable. In addition, discrete event simulation provides a valuable tool for constructing models that include random behaviour of a large number and wide variety of components [Askin 93]. Consequently a few researchers have looked at applying discrete event simulation to the modelling of supply chains.

Geller et al [Geller 95] describe the use of DES in the modelling of the IRIDIUM supply chain. The IRIDIUM project co-ordinated by Motorola consists of a number of partners in the telecommunications and space industry. The objective of the project is to provide a digital, satellite-based, cellular, personal communications network. The project involves the development and launch of a number of satellites in a short time frame. Geller et al describe the need to predict resource requirements and optimise schedules so as to

minimise costs and meet the aggressive launch targets. Discrete event models of the different stages of the supply chain were built by the individual partners using a common tool. WITNESS, a visual interactive simulation tool, was chosen for this purpose. Depending upon the requirements of the study, models of varying levels of detail were developed, from detailed representations to black box representations that use a simple time delay to characterise a stage. The models of the various stages of the supply chain were implemented as WITNESS sub-models. These were then integrated to create a model of the supply chain.

The supply chain models then were used in two studies to predict shared resource requirements and the alignment of production and delivery schedules. Although a traditional simulation tool and a simple mechanism for model integration was used, the authors attribute a substantial saving in time and cost to the supply chain modelling study.

VanDuin et al [VanDuin 92] describe the development of TASTE, an acronym for The Advanced Simulation Tool for logistic Engineering, a simulation tool for the modelling of supply chains. It consists of a library of pre-defined logistic components that can be configured to represent a supply chain using a graphical user interface. The components of TASTE have been designed so as to enable them to be used in a wide variety of supply chains. Configurable components such as distribution centres, production units, transport mechanisms etc are provided as part of the library. Slats et al [Slats 95] report the successful use of TASTE in a number of enterprises in the Netherlands. TASTE

provides a tool for modelling a supply chain at a high level (low detail). Individual supply chain components are not modelled in great detail. For instance, production units are not linked to detailed models that represent the manufacturing facility.

Slats et al [Slats 95] incorporate TASTE as a part of DPSS, a distributed planning support system. DPSS is based on the idea of logistic laboratories. The creation of logistic laboratories is motivated by the need to support analysis of the performance of entire logistic chains from several perspectives and levels of decision making viz. strategic, tactical, and operational. Slats et al [Slats 95] advocate the use of an experimental environment, termed logistic laboratory that consists of a set of logistic models and sub-models based on optimisation, heuristics, and simulation, from which models reflecting various perspectives of the supply chain can be created. DPSS consists of four types of models, three of which are based on optimisation and the fourth is a DES model for which TASTE is used.

Roy and Meikle [Roy 95] describe the role of DES in finite capacity scheduling. They state that the role of DES based scheduling is limited, in particular in the case of large and complex enterprises, to the shop/cell level, due to a lack of an integrating framework that incorporates models of the various stages. The authors envision a DES tool that includes knowledge of the structure of the enterprise so as to extend the DES scheduling framework to include the supply chain and thus provide more accurate schedules. A model of an enterprise can then be created that includes links to the various cell/shop models. Models of various levels of detail can be included in the framework. Typically, a

coarse model may be sufficient at the enterprise level, while a more detailed model can be used at a local level.

2.8 Requirements for DES of supply chains

One approach to modelling the supply chain could be to use a traditional ‘flat’ simulation framework. Although such a framework is suitable and has been used successfully for modelling single manufacturing cells or single flow lines in detail, extending it to a supply chain level or even a factory level would not be practical. Popplewell and Yu [Popplewell 94] mention that the human effort in building such a model would be enormous and it would result in the use of excessive computer memory and slow run times to be of any use. In addition, the model would fail to reflect the structure of the supply chain, thus not aiding the understanding of its behaviour. An alternative is to couple a number of models that represent the various components of the supply chain to create a more comprehensive model of the supply chain. The coupling of models could occur in a top down fashion where the supply chain is decomposed into its constituent parts and coupled together, or it can take place in a bottom-up manner where existing models are coupled to synthesise a composite model that represents the supply chain.

The facilities that constitute the supply chain are often dispersed geographically. Consequently, the models that represent the various stages of the supply chain may be created and maintained by different people and stored in various locations. The supply chain modelling framework, thus, must operate in a distributed environment where

composite models are created by integrating such distributed component models. The advent of the World-Wide-Web and developments in networking technologies has enabled organisations to integrate a number of operations such as order processing etc. Adopting these concepts and technologies to the area of modelling methodology is an important requirement for modelling supply chains. Web based simulation is a new and fast growing area that looks at how the internet and web based technology can influence modelling methodology. In the context of developing a supply chain modelling framework the support for distributed modelling provided by web based simulation is of interest.

Once a composite model is created, the modelling framework requires a mechanism to execute (simulate) the model. In the case of individual models, a simulation executive manages the execution of events in a time ordered fashion. Events are scheduled on to an event list and the simulation is progressed by executing the event with the earliest scheduled time. The simulation of composite models is more complicated as events in one component model may affect events in another component. Thus, there needs to be a means of sequencing and scheduling the events generated by the various components in a composite model. One approach is to extend the event list mechanism to include not only local events but events generated across the entire composite model. All the models in a composite could employ a common global event list on to which events would be scheduled. A super-executive would determine the next set of events to be executed and inform the associated component models to execute them. Such an approach has its attraction in its simplicity, however, using a global event list is not a very efficient

mechanism, both in terms of scalability and runtime performance. As the number of models in a composite increase so do the number of possible events, and consequently the size of the event list grows making its management difficult. Further, run times of large models may be high as the execution of the event list is sequential and only events that are scheduled for the next-event time are executed.

An option to improve runtimes is to locate the various component models in several processing nodes rather than a single one. For instance, the component models could be executed in a network of workstations with each component model assigned to a workstation. This approach has the benefit of allowing local events that occur in each of the component models to be executed in parallel rather than sequentially, thus improving the speed of execution. In addition to assigning the component models to run on individual processors, a mechanism to co-ordinate the local execution of events in each of the processors needs to be present. Employing a global event list allows for exploiting the potential parallelism in event execution to a limited extent as only events that are scheduled for the next-event time can be processed simultaneously. A number of alternate algorithms that address this issue have been presented under the banner of parallel discrete event simulation (PDES) in the literature. Incorporating PDES in the supply chain modelling framework will enable the efficient execution of large composite models. Run time performance of a simulation is an important attribute and indeed has a bearing on the successful satisfaction of the modelling objectives. Prohibitively large run times can restrict the amount of experimentation possible which can be detrimental to the simulation study.

The researchers belonging to the PDES community and the modelling methodology community have by and large worked in isolation and consequently the application of PDES techniques in mainstream modelling efforts have been limited and not very successful [Fujimoto 92]. A major criticism of the PDES effort is that the primary focus has been on improving runtime performance and this has been quite often at the cost of other modelling requirements. In addition, the objective of research in PDES has not just been motivated by the need to provide an efficient runtime platform for discrete event simulation. A number of researchers [Chandy 93] have looked at PDES also as a case study or vehicle to explore concepts and techniques in the wider context of distributed computing. Thus, as Page [Page 93] notes, a sizeable proportion of the research does not address the PDES effort in the larger context of the modelling endeavour.

Similarly, developments in web based simulation need to be conducted in the context of other developments in modelling methodology. In summary, the modelling framework needs to support the following three requirements:

1. A compositional approach.
2. Functioning in a distributed environment.
3. PDES techniques.

In order to gain an understanding of the intricacies involved in developing the modelling framework, the relevant literature pertaining to each of the three requirements is first

reviewed. The knowledge thus gained is then used to develop a novel supply chain modelling framework. This chapter concludes by providing an overview of web based modelling and its support for distributed modelling. The review of compositional approaches and PDES techniques found in the literature are presented in chapters three and four respectively.

2.9 Web based and distributed modelling

Almost every area of human activity has been transformed by the impact of the world-wide-web (WWW) and computer simulation is no exception. Web based simulation is a new field that investigates the application of web based technologies to simulation. One of the earliest researchers to foresee the potential impact of the web on simulation was Fishwick [Fishwick 97]. He described a number of potential impacts of this new technology on the discipline of simulation, including: (1) education and training, (2) publication, and (3) simulation programs. It is the last category, viz. web based simulation programs, that has received the most interest and it is this that is of interest here in the context of supporting distributed modelling.

A fundamental characteristic of the WWW is its distributed nature. Thus, integrating it with traditional simulation techniques is a natural avenue to support distributed modelling. Page et al. [Page 97] identify two basic forms of web based simulation. The first form involves a remote server on which a simulation runs. A client uses a form-based CGI script to set up an experiment for the simulation. The server performs the

simulation and returns the results, which is displayed by the CGI script. Such an approach has the benefit that existing simulation software can be used. The second form involves using Java applets. Simulation models are encoded in a Java applet which is downloaded using a web browser by a client who wishes to run it. The simulation is then run locally and the user interacts with the applet in the course of performing the simulation. An obvious disadvantage is that existing simulation tools cannot be incorporated in this approach.

Lorenz et al. [Lorenz 97] describe a third form of web based simulation that combines the two approaches described above. A disadvantage of using a remote server based simulation is that interaction with users is limited to CGI forms and thus visual interactive simulation is not possible. Lorenz et al. propose an architecture where the simulation runs on a remote server. However, instead of using a CGI interface a Java applet acts as a client and provides local support for animation and visual interaction.

A number of simulation languages, such as *silk*, SIMJAVA, and NCOS, have been developed, based on Java, with a view of exploiting the web oriented characteristics of this language. Healy and Kilgore [Healy 97] describe *silk*, a general purpose simulation language based on Java. Java's built in support for multi-threading is used by *silk* as a basis for describing the behaviour of entities in a process oriented simulation. Distributed simulation is then achieved by mapping threads onto multiple processors. In addition, the use of *JavaBeans* to create reusable component models supports distributed modelling.

McNab and Howell describe the development of SIMJAVA [McNab 96]; a simulation package that is a Java version of Sim++, a discrete event simulation library for c++. SIMJAVA is based on the process-interaction world view. A SIMJAVA simulation consists of a set of objects each running on their own execution thread. Objects communicate with one another by sending event messages over ports. Page et al. [Page 97] describe how the Remote Method Invocation (RMI) feature of Java can be incorporated into SIMJAVA to enhance its support for distributed modelling. RMI employs the distributed object model of Java. In this model, objects, termed remote objects, make available their functionality by allowing other objects to access their methods. Thus, components of a model can be distributed over the internet and interaction between them can occur by invoking appropriate methods in remote objects.

Colvin and Beaumariage [Colvin 98] present a simulation environment termed 'The Network-Centric Simulation Object System (NSCOS). NSCOS is a prototype simulation environment, based on Java, primarily for use in manufacturing systems simulation. Models in NSCOS are described by aggregating a collection of simulation objects. Allowing the simulation objects to be located either locally or on a network enables distributed modelling. They describe the application of NSCOS to the simulation of a semiconductor fabrication facility. Models of the semiconductor facility are created by coupling together a range of *workstation_objects*. Some of these *workstation_objects* may be located on a remote computer connected to the Internet. The *class_loader* feature of Java, which allows classes to be loaded from the web, is used to provide the mechanism for supporting remote *workstation_objects*.

Sarjoughian and Zeigler [Sarjoughian 1998] describe an implementation of the DEVS formalism in Java. DEVSJAVA enables the amalgamation of a proven set theoretic modelling formalism with the web-centric capabilities of Java. DEVSJAVA supports the creation of standalone and applet based atomic and composite models. A web browser based graphical SES (GSES) is used to synthesise hierarchical models from atomic and coupled models. The platform independent nature of Java and the support for distributed object computing allows DEVSJAVA to support a collaborative modelling environment.

2.10 Conclusion

A broad review of DES techniques is presented in this chapter. The various types of simulation tools are categorised, and their potential benefits and pitfalls discussed. A key conclusion is that it is impossible to find a single formalism/tool, that can be used across the board, to satisfy the requirements of all users. A judicious choice of the modelling framework needs to be made based on the nature of the modelling problem.

This is followed by a review of approaches to modelling supply chains. A number of approaches, both analytical and simulation exist in the literature. Based on the knowledge gained by the review, a set of requirements for the DES of supply chains are identified. The literature pertaining to one of the requirements, viz. web based and distributed modelling, is reviewed here. In Chapters 3 and 4 a review of the other two

In Chapters 3 and 4 a review of the other two requirements, namely support for compositional modelling and the incorporation of PDES, is presented.

CHAPTER 3

Large scale modelling - a review

A number of researchers [Zeigler 84] [Luna 92] [Ulgen90] have identified the need for coupling together models to represent a wider system. Zeigler [Zeigler 84] describes the notion of modelling in the large, where individual models are developed with the aim of not just studying one aspect or facet of a wider system, but to be part of a larger model of models, representing the multi-faceted characteristics of the system. In the previous chapter the ability to couple models to create composite models was specified as a requirement in aiding the modelling of supply chains. Providing an environment in which such multi-faceted modelling can take place requires a revision of the way in which aspects of behaviour and description of interaction are represented. [Davis 96].

This chapter provides a review of the research addressing the problems of representing and structuring models to aid large scale modelling. Increasing the modularity of models has been an underlying theme in the literature. Modularity and the support for it provided by the world views is first looked at.

The need to develop larger and more complex computer programs has motivated software engineers to develop mechanisms to structure problems to aid the programming

process. A number of these ideas have been adopted by the discrete event simulation community in their endeavour to model increasingly complex systems. Modularity, an important software engineering concept, has been suggested as a natural choice for simulation, particularly so in large scale modelling [Oren 79].

Cota and Sargent [Cota 92] describe modularity as a combination of locality and encapsulation. Locality has to do with locating all relevant parts of a system in one place. Encapsulation, on the other hand, is the strict protection of this information by a well defined interface in such a way as to allow it to be changed without affecting other parts of the system. The identification of what constitutes a relevant piece of information and, thus, requires to be localised is clearly subjective and depends on the objective of the modelling exercise. For example, if the modelling exercise is to study the flow of entities and various routing schemes in say a discrete part manufacturing environment, a material based view, where all aspects of behaviour of a part are localised in one place, may be more suitable. The localisation of all routing information in the flow entities would make for easier changes to routings.

The requirement of coupling together models to describe a larger system is of interest in the context of supply chain modelling. Modularity in this case has to do with allowing model components to be reused in interchangeable coupling configurations. This requires the modularization of behaviour and coupling information, thus allowing behaviour and coupling specification to be modified independently. Modularity can exist at the modelling object level, the model level or both. At the modelling object level individual

objects can be coupled to create a model. At the model level, models can be coupled to create a composite model.

3.1 Modularity and the world views

The traditional world views have been the basis for simulation in a number of commercial simulation packages. The implications for modularity of modelling using world views is investigated here. The process interaction view is considered first. The process world view provides locality of object [Overstreet 86], i.e. the behaviour of an object (entity) is contained within a process routine. Information about interaction among objects however is distributed, and mutually supportive code in the interacting entities is required to co-ordinate interaction [Blunden 67]. Traditional process based languages do not provide locality of the conditions for reactivation of a process. Languages such as SIMAN and GPSS allow a process to deactivate and subsequently reactivate another process. Deactivation of a process is supported by the ‘pre-empt’ block in GPSS and SIMAN, and ‘cancel’ and ‘reactivate’ instructions in SIMULA.

With the objective of improving the modularity of the process based view, Cota and Sargent [Cota 92] proposed a modified process view based on the active server approach [Henrikson 81]. They argue that design decisions are often associated with servers or resources and the active transaction approach employed in SIMAN and GPSS result in the distribution of conditional information regarding the acquisition of resources among the transaction entities. They modify the notion of active state of a process to include not

only the time left to complete the current activity, but also the conditions under which the activity can be pre-empted and others scheduled. The reactivation of a process is described by the following structure:

Wait t

or when condition 1

or when condition 2

....

....

or when condition n

Where t is the time for the completion of the current event and condition 1 to condition n are the conditions under which the activity will be pre-empted.

By employing a combination of an active server approach and modifying the active state of a process, the modified process view localises conditional information within the process. As mentioned earlier encapsulation of process behaviour requires the strict protection of state by a clearly defined interface. In the example given in [Cota 92], Cota and Sargent describe a repair shop process modelled using the modified process approach where interaction between processes is accomplished via a passive entity (buffer). The repair shop accepts parts with varying priorities and performs a repair and makes it available to other processes. Parts with higher priority pre-empt repairs of parts with lower priority, which resume repair once the higher priority part has been repaired.

Interaction between processes is achieved through two buffers - an input bin and an output bin. Parts that need to be repaired are placed in the input bin by a supplier process and after the completion of repair the part is placed in the output bin by the repair process to be accessed by other processes.

The use of message passing as a means for interaction has been suggested by a number of researchers, and implemented in simulation languages based on the modified process view such as May [Bagrodia 87], Sim++ [Lomow 90], and HCFG [Fritz 95]. Message passing as the sole interaction mechanism between processes prevent them from directly modifying the state of other processes and thus encapsulates processes behaviour. In addition, the use of message based simulators provides a more natural environment for modelling distributed systems [Bagrodia 87] (Note: Supply chain models could be thought of as distributed models).

Cota and Sargent [Cota 94] describe a graph based representation of the modified process view employing message passing for inter-process communication. Control flow graphs (CFG) are self contained units that describe the behaviour of a process and can be combined to create a model. Inter-process communication occurs via message passing along channels. A process is described by a directed graph where nodes represent control states (reactivation points) and edges describe possible control state transitions. Each edge is associated with three attributes – a condition, an event, and a priority. An edge is traversed if the condition is satisfied and the event associated with the edge, which represents the actions performed in the state transition, is executed. Priorities are used to

break a tie if the conditions of more than one arc emanating from a node are true.

Conditions are based on:

1. Messages waiting to be received by the CFG.
2. A boolean function of the local variables.
3. A time delay, which holds the process until some fixed simulation time has elapsed.

By enforcing a fixed interface and employing message passing for inter-process communication, CFG provides a modular representation of processes in a modified process interaction view. Modularity, as in our earlier definition of allowing model components to be reused in mutable coupling configurations, is achieved by making a clear separation between the description of the behaviour of the process, which is described by a CFG, and the description of the interaction which is described by an interconnection graph (IG). Fritz and Sargent [Fritz 95] describe an extension of the CFG/IG framework to support hierarchical modelling. The objective here is to extend the idea of process modularity to modularity at the sub-model/model level, allowing for the creation of models which are composed of a network of other models. Hierarchical control flow graph based models make use of hierarchical interconnection graphs (HIG) to describe the interconnection between sub-models. The nodes in a HIG are not restricted to representing a single process described by a CFG, but can denote models (described by an IG). The relationship between the different structures viz. CFG, IG, HIG are illustrated in Figure [3.1].

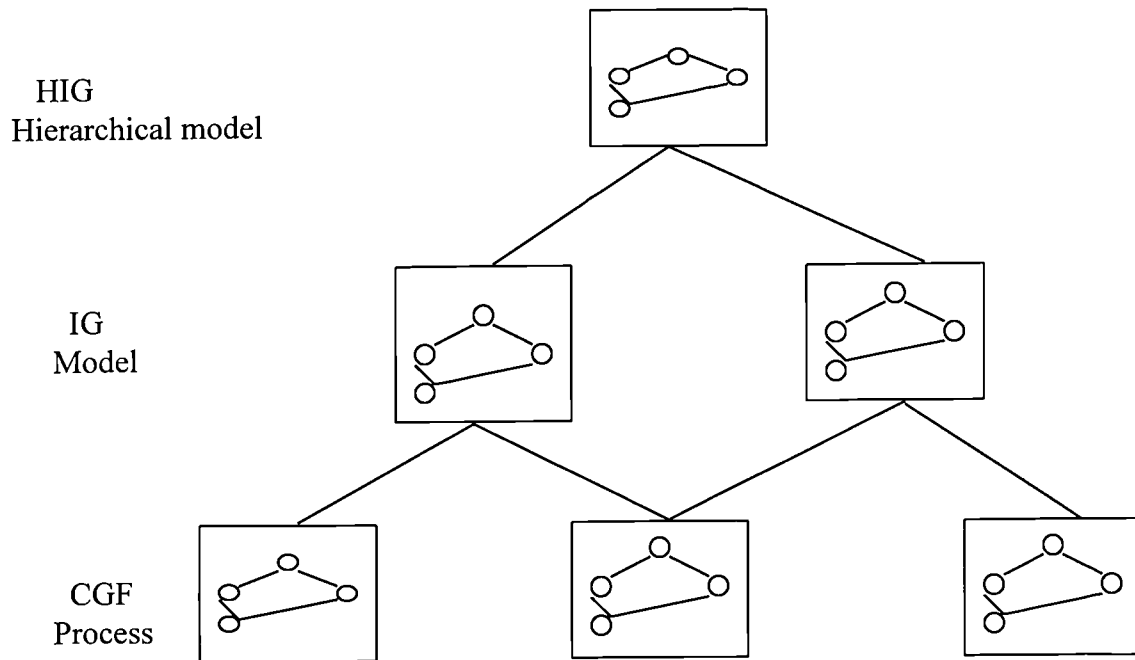


Figure 3.1 Relationship between Control Flow Graph, Interconnection Graph and Hierarchical Interconnection Graph

Approaches based on the modified process view provide modularity by employing an active server approach and using constructs to localise the conditions for interaction with the flow entities. Bagrodia et al. [Bagrodia 87] describe another message based approach to discrete event simulation that is based around the modified process view. This approach, however, does not enforce the active server model as used by Cota and Sargent [Cota 92], or for that matter the active transaction view. A general entity construct is used to model all entities, active or passive. The interaction between entities is akin to mutual interaction as described by Davis [Davis 96]. He defines mutual interaction as the

interaction between entities of equal status, in which none dominates and which allows each to proceed with internal behaviour during an activity. Modularity in the context of entities being coupled in mutable coupling configurations is, however, limited as conditional information for the interaction is distributed amongst the entities.

Pflug and Prohaska [Pflug 90] propose the Entity-Connection approach in the context of providing modularity with respect to mutual interaction. They draw an analogy between actors in a play and processes in a model. The parts of an actor contain monologues and dialogues (or polylogue). The monologues correspond to autonomous behaviour in a process and dialogues (polylogue) correspond to interaction between processes. This distinction between monologue and dialogue is the basis for the entity-connection model. An entity module encapsulates the activities of the entity that do not require co-operation, while a connection module encapsulates activities that require co-operation. During execution entities perform activities concurrently. When interaction is required the interacting entities invoke the relevant connection module that describes the interacting activity. During the execution of the connection module the entities which invoked the connection remain attached to it. The connection module has a set of conditions associated with it, such as type and number of entities. When these are satisfied, the body of the connection module is executed. After the termination of the connection module the attached entities are released and they resume concurrent execution.

Pflug and Prohaska [Pflug 90] describe a two stage modelling process. First entities and connection modules are defined. Models are then created by using the modules as building blocks. A graphical user interface is used to select and combine the required modules to create a model. The ability to subsequently use these models as building blocks in a composite model is not described by the author. However, it may be possible to extend the entity-connection framework to support hierarchical modelling by creating a 'model' object. Models can be described as an aggregation of entities and connections. A coupled model could then be described as an aggregation of models and connection objects that are shared between the models.

Model = < Entity, Connection >

Coupled model < Model , Connection >

Figure [3.2] illustrates this.

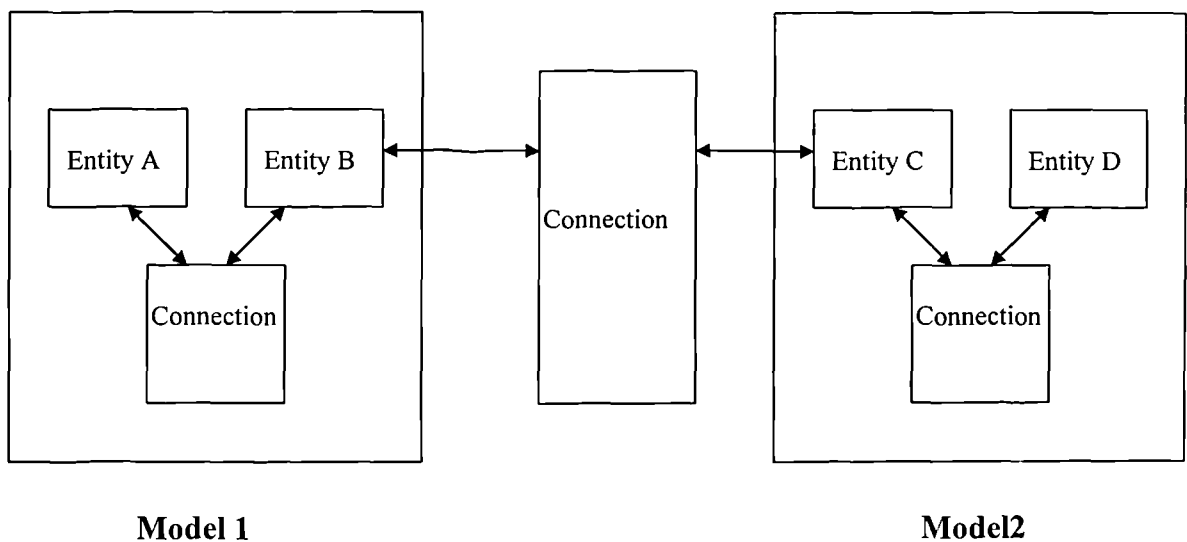


Figure 3.2 Coupling two models using the Entity-Connection view

The idea of separating the behaviour of entity from interaction has also been incorporated in PAInt [Davis 96], albeit from the perspective of the activity and process world views. Davis proposes a modelling view that combines the traditional process and activity world views, processes being similar to entities in the entity connection view and activities to connection modules. Processes are modified to improve process independence by extracting conditions for mutual interaction and placing them within activities. Waiting states are removed from the process and incorporated into the activities that describe the interaction. Processes act as tokens that are accepted and released by a series of activities, where process interaction occurs. Keys, based on the idea of guards [Bagrodia 90], are employed to check the validity of arriving entities for commencement of the activity. By separating the specification of entity behaviour (process), entity interaction (activity), and the conditions for interaction (keys), PAInt supports modularity. Figure [3.3] shows an example of a PAInt model. Two processes P1 and P2 are used that perform activities A1 and A2. Both activities A1 and A2 involve co-operative interaction between the two entities of the processes P1 and P2.

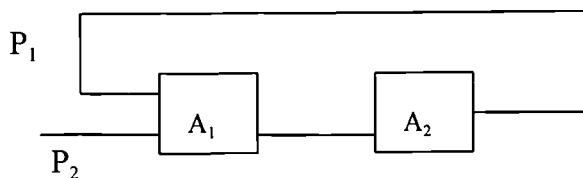


Figure 3.3 An example of a simplified PAInt model [Davis 96]

Davis describes how the mechanism of employing keys, processes and activities to develop modelling objects can be extended to create models (aggregation of modelling objects) and further to create composite models (aggregation of models) and so on, thus supporting hierarchical model construction.

Modelling objects are defined by $M = \langle k, A, P \rangle$

where k = key describing the conditions for the activity to commence

A = Set of activities

P = Set of processes

and composite models can be described by $CM = \langle k, M, P \rangle$

where k = Input interface

M = Set of models

P = Set of processes that connect the models

In the case of modelling objects, keys describe the conditions for an activity to occur, i.e. the input interface for tokens (processes) to be accepted by an activity. At higher levels of aggregation (models and composite models), keys determine the input interface to the model/composite model. Similarly, processes can be used to determine the output interface of a model. Keys and processes have an independent existence from the models, thus resulting in the separation of behaviour and coupling specification, and consequently allowing the aggregation of models in interchangeable composites. In addition, the notion of keys allows the implicit description of coupling information.

Modularity with respect to creating a composite of heterogeneous models (models employing different modelling formalisms) at varying levels of abstraction, also termed multi-modelling by Fishwick [Fishwick 93], has been investigated by De Meter and Deisenroth in the development of GIBSS (Generalised Interactive Based Simulation Specification) [De Meter 91]. Entity objects are used to represent a subsystem at some level of abstraction. These specify the attributes and logic required to simulate the entity. Interactions between entities are explicitly stated and accomplished by an interaction object. De Meter and Deisenroth suggest that message passing may be ill-suited to conceptually represent physical interactions, and describe a mechanism of direct observation /manipulation to accomplish interaction between the entity objects and interaction objects. This is similar to the idea of observation of state values suggested by Blunden [Blunden 67] to solve the ‘self containment problem’ (modularity). A three phase approach adapted to handle continuous simulation is used in the execution of the model. Hierarchical modelling as such is not supported, the model is an aggregation of entities. However, the entities in themselves may represent subsystems at different levels of abstraction.

Activity based approaches provide locality of interaction. Pidd [Pidd 88] has suggested that the activity view facilitates modular code development and modification. However, locality in terms of entity behaviour is not provided by activity based approaches, as the various activities performed by an entity are described in different routines. Unlike traditional process based approaches, conditions for interaction in activity based

approaches are localised in the test head of an activity. However, the decoupling of the test conditions from the activity is necessary in order for activities to be more modular. The idea of guards [Bagrodia 87] [Davis 96] can be used for this purpose.

3.2 Object oriented simulation

Object orientation is a popular technique for developing and implementing complex (large) computer programs. The ideas behind object orientation ironically were first demonstrated by the general purpose simulation language SIMULA [Birtwistle 79]. A combination of the difficulty involved in understanding a new modelling paradigm, and the lack of facilities such as integrated statistics collection, contributed to the lack of acceptance in the simulation community [Henrikson 81]. Since then the concepts of object orientation have matured and languages such as Smalltalk [Goldberg 89] and C++ [Stroustrup 88] have contributed to its popularity in the general programming community.

An important aspect of object orientation is the idea of objects. Objects are based on the idea of encapsulating data and methods. The data in the object can only be accessed/manipulated by the associated methods in the object. The methods thus act as an interface to the data held in the object. The parallel between object data and state, and methods and behaviour suggests object orientation as being a natural choice for simulation.

The early attempts at object orientation was in aiding the implementation of a particular world view. For example both SLAM and SIMAN are implemented in C++, although such efforts cannot strictly be classified under the term object oriented simulation. At the modelling level they do not allow the defining, modification (polymorphism) and reuse (inheritance) of modelling primitives. This view is similar to what is termed object based [Joines 97], or data driven simulators. A predefined set of objects are provided that are used to create new models by combining them in a suitable manner.

The natural correspondence between objects in the physical world and their representation as model objects has been suggested by a number of authors [Ulgen 90] [Zeigler 90]. The application of object orientation has been extended to the modelling view in a number of simulation environments. Used in combination with a graphical user interface they provide an easy and extensible modelling environment.

Simple++ [Aesop 94] employs a graphical interface which allows the coupling of different modelling objects. Modelling objects can be selected from a predefined set of objects or new objects can be created/modified from existing objects. Full access is provided to the modelling hierarchy of the objects, thus allowing inheritance and reuse of modelling objects.

Ulgen and Thomasma [Ulgen 90] describe SmartSim, an object oriented simulation environment implemented in Smalltalk for the modelling of discrete part manufacturing systems. A set of generic modelling objects are provided that are combined using a

graphical user interface to create models. The objects are part of a class hierarchy Figure [3.4] and are fully extensible. They give the example of a workstation object being extended to represent a Gauge object. Objects can be combined to create aggregate objects using the subsystem class. These aggregate objects in turn can be part of other aggregate objects and so on, thus providing a mechanism for hierarchical modelling.

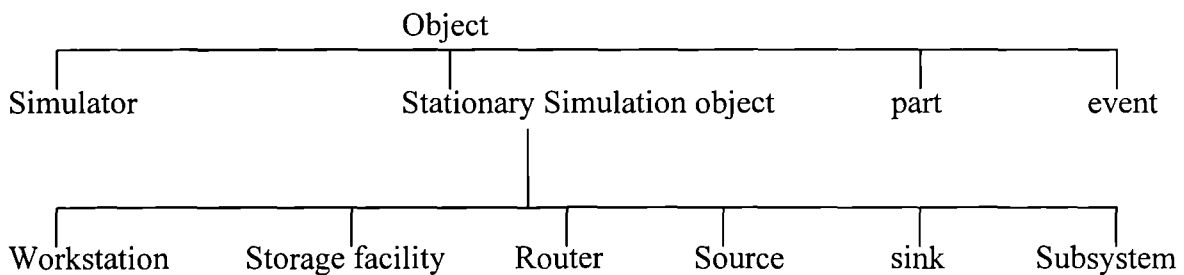


Figure 3.4 The class hierarchy of simulation objects in SmartSim

3.3 DEVS and hierarchical modelling

The motivation behind the development of DEVS [Zeigler 84] lies in providing a sound mathematical basis for the representation of discrete event models. It performs a similar role to differential equations in the representation of continuous models. Based on systems theory concepts, DEVS provides an implementation independent abstraction for the description of discrete event models. This can be used as a basis for the development of simulation languages; DEVS Scheme being one such example. Systems theory distinguishes between the behaviour of a system and its structure. It introduces the notion

of decomposition (how a system can be broken down) , and synthesis (how previously decomposed system can be reconstituted.), thus allowing hierarchical construction. By adopting decomposition and synthesis, DEVS allows the representation of discrete event models in a hierarchical modular manner.

The DEVS formalism allows one to specify basic models from which larger ones are built, and the manner in which these models are connected together in a hierarchical fashion. The basic models are termed atomic models and used to describe behaviour of individual components. Coupled models are used to describe the structure of models created from individual components. Thus, atomic models represent the behavioural aspects of a model, while coupled models the structural aspects.

A basic model contains the following information.

- A set of input ports through which external events are received.
- A set of output ports through which external events are sent.
- A set of state variables that define the state of the model.
- A time advance function that schedules change of state based on an internal transition function.
- An internal transition function that describes the state change of a model when directed by the time advance function.
- An external transition function that describes the change of state when an input is received on an input port
- An output function that describes the output generated by a state change.

An atomic model is described mathematically by the following structure

$$\mathbf{M} = \langle \mathbf{X}, \mathbf{S}, \mathbf{Y}, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, \mathbf{ta} \rangle$$

where \mathbf{X} is the set of external events that can be received by \mathbf{M}

\mathbf{S} is a set that defines the state

\mathbf{Y} is the set of events that can be output by \mathbf{M}

δ_{int} is the internal transition function which specifies the next state the system
will change to after time \mathbf{ta}

δ_{ext} is the external transition function which specifies the transition of state when
an input is received

λ is the output function which generates an external output just before an internal
state transition

\mathbf{ta} is the time advance function

A coupled model is defined by the following mathematical structure

$$\mathbf{DN} = \langle \mathbf{D}, \{\mathbf{M}_i\}, \{\mathbf{I}_i\}, \{\mathbf{Z}_{i,j}\}, \mathbf{select} \rangle$$

where, \mathbf{D} is a set of model component names,

for each $\mathbf{i} \in \mathbf{D}$

\mathbf{M}_i specifies a model described by \mathbf{i}

\mathbf{I}_i is a set containing the models influenced by \mathbf{M}_i

$Z_{i,j}$ is a set of port couplings that describe the port to port connection from model M_i to model M_j , where $j \in D$

select is a function that resolves ties when simultaneous events occur across more than one input port.

3.3.1 Large scale modelling decomposition and synthesis

Zeigler [Zeigler 84] characterises model building in terms of small scale modelling and large scale modelling. He compares small scale modelling to more traditional approaches to modelling. A model is created as a ‘one off’ with a specific objective to analyse a particular aspect of a system. Once the modelling objective is met the model may be discarded and not used again. In contrast, models in large scale modelling are developed as part of a larger model (hierarchical), and models are reused in different configurations to study different aspects of behaviour.

Zeigler proposes a structure, the system entity structure (SES), to support large scale modelling. The SES combines the dynamic formalism of DEVS with AI symbolic formalisms. It provides a structure that embodies knowledge about the system [Zeigler 84]. The SES is described using a labelled tree structure. Nodes represent entities, specialisation, or aspects. Entities representing individual real world objects or components of a decomposition of another real world object. An aspect node represents one decomposition out of a possible many of an entity (represented by the parent node). The children nodes of an aspect represent components in the decomposition of the parent.

Specialisation nodes represent alternative choices for a component. The children of a specialisation node represent variants of its parent.

Modelling occurs in two stages. In the first stage, that occurs once initially, the entire system is decomposed to include all the objects, their variants (termed specialisation) and alternate configurations (termed aspects), to create a SES. In the second stage, the SES is pruned such that a unique model is derived that satisfies the modelling objectives. Pruning begins at the root of the SES and progresses in a top-down fashion. At each stage the SES is pruned to leave a unique object under each specialisation and one aspect under each object. The coupling specifications at each aspect are then used to provide the coupling specification of the model.

Zeigler provides examples [Zeigler 90] of the usefulness of the concept of SES in supporting large scale modelling in the domain of computer architecture modelling. However Davis suggests [Davis 96] that the degree of isomorphism (similar structure and consequently interface) between entities may be limited in manufacturing systems.

The need to make explicit all possible couplings between entities in a SES can be time-consuming. Rozenblit et al. [Rosenblit 89] have suggested the use of a knowledge based system for identifying additional taxonomical information regarding constraints and relationships between objects to guide synthesis. Thomas [Thomas 94] has identified the role that model interfaces can play in providing this information. He employs a scheme where the type of model interface, as opposed to the implementation, is used as a

classification criterion to classify DEVS models. He modifies the input and output sets to include typed messages. In addition to the port name and content, the type of message is also included in messages. A class based hierarchy is developed where models belong to a class if they share the same interface. He employs this classification in the syntactic checking of possible coupling configurations in a SES.

3.3.2 Variable structure modelling and DEVS

Oren [Oren 75] introduced variable structure systems as a new system class. Variable structure models, as part of their behaviour, include in their description the possibility to change their own structure, both in terms of constituent components as well as the relationship that exists between them. Such systems exist in a number of application domains including manufacturing systems (flexible manufacturing systems, supply chains), biological/ecological systems etc. For example, depending on the type of orders and quantity a manufacturer may vary his suppliers thus reconfiguring the supply chain.

Models in DEVS are represented by atomic and coupled models. Atomic models describe the behaviour of individual components, while coupled models describe the structure of the model. This apparent separation of behaviour from structure (see Figure [3.5]) makes it difficult for DEVS based models to describe variable structure systems.

Zeigler and Praehofer [Zeigler 89] describe a two-level control hierarchy for the modelling of variable structure systems. The first level is made up of models that represent the behaviour of the system. The second level consists of controllers that

observe the dynamic behaviour of the models in level one, and accordingly change the structure of the couplings between them. The control models, however, cannot be viewed as atomic models as they violate the modularity principle. Atomic models are only allowed to change their own state. The controller models can be viewed as belonging to another class of models besides atomic and coupled models.

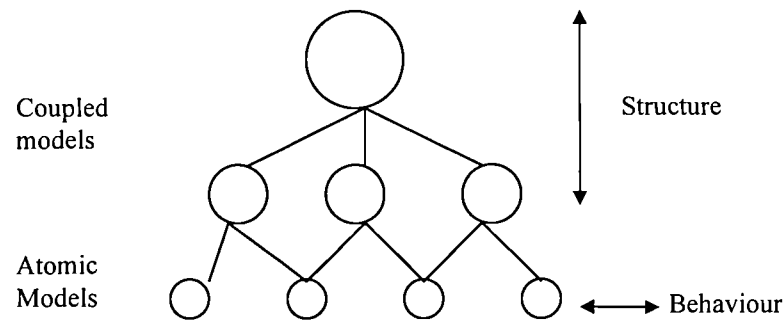


Figure 3.5 Separation of description of behaviour and structure in DEVS models

Barros et al. [Barros 94] describe an approach that maintains the modularity of atomic models. They modify the DEVS coupled models formalism with a view to support structural change. Atomic models are defined as before, while coupled models are defined as follows:

$$V_CM = \langle X, Y, \vartheta, select \rangle$$

Where,

V_CM = Coupled model incorporating variable structure behaviour.

\mathbf{X} = set of internal ports;

\mathbf{Y} = Set of external ports;

\mathbf{M} = Set of components;

\mathbf{C} =connections between components;

and \mathfrak{S} is defined by

$$\mathfrak{S} = \langle \mathbf{X}, \mathbf{Y}, \chi, \delta_{\text{int}}, \delta_{\text{ext}}, \text{select} \rangle$$

where $\chi = \langle \mathbf{M}, \mathbf{C} \rangle$

and $\mathbf{X}, \mathbf{Y}, \delta_{\text{int}}$, and δ_{ext} are as defined above in the case of DEVS.

\mathfrak{S} acts as an atomic model that handles structural change. The state of the model is defined by the set of components and the respective interconnections of the coupled model. Changes in structure are initiated by internal and external transition functions of the atomic model. They give an example of the application of the formalism in a model of a variable multi-processor network. Based on the load of incoming jobs, the processor network invokes new processors or releases processors back to a pool.

Pawletta et. al. [Pawletta 96] describe an approach based on the pruning of the SES in runtime. Coupled models are modified to include structural events that can create, delete and exchange components or connections in a coupled model. The SES uses the structural events to re-prune the SES and generate a structural variant of the model. Pawletta et al. also address the problem of dependence between behavioural state and

structural state. For example, if component ‘a’ is exchanged with component ‘b’ in a coupled model and if the two components share some common state variables, then a mechanism needs to be devised to copy the common state variables from ‘a’ to ‘b’. The problem is solved by introducing a class hierarchy based on the state variables of component models. Component models inherit state variables from parent classes. Exchanges then are permitted only between components with state classes derived from a common parent class.

Thomas [Thomas 94] mentions that the interface-oriented classification of models, described earlier in the context of syntactic checking of couplings in a SES, can also facilitate variable structure modelling. The interface classes are used to create a coupled structure of classes. Where, the classes act as placeholders for component models. The structure of the model is varied (to a limited extent) by replacing component models in the coupled structure by other models belonging to the same class.

The problem of modelling variable structure systems can be viewed from the perspective of the level of modularity of the component modules. The more modular the component models are, the less they are contextualised to a particular model configuration and consequently, can change their structure more easily.

3.4 Coupling schemes

The function of a coupling scheme is to provide a translation of the input/output alphabet used in interaction between component models. The coupling scheme together with modularity allows the creation of composite models. Davis [Davis96] classifies coupling schemes into four categories.

1. *Tightly coupled- a complete and explicit coupling is provided.*
2. *Loosely coupled - part of the coupling is made explicit and the rest is garnered from context of the interaction.*
3. *Fixed coupling - A common alphabet is used by all the component models.*
Couplings are identified as a sequence of component names/identifiers.
4. *Undefined - No explicit coupling is mentioned.*

Modelling frameworks that describe coupled models in a tightly coupled fashion typically use graph based technique to make explicit coupling information. Control flow graphs and hierarchical control flow graphs (CFG/HCFG) [Cota 94] [Fritz 95], described earlier (Section 3.1), are examples of modelling frameworks that employ tightly coupled coupling schemes. Component models are created using CFG that communicate strictly via channels. Coupling between component models is described by an interconnection graph or hierarchical interconnection graph (IG/HIG). IG/HIG explicitly describe the set of input and output channels (ports) for the coupled model to communicate with its

environment, and an explicit description of the internal coupling of the constituent component channels.

Coupled models in DEVS [Zeigler 84] are also described using a tightly coupled scheme. An explicit specification of the couplings of the various ports of each of the models in a coupled model is required to describe it. Coupled models described as above belong to a class of coupled models termed Digraph models. In addition to Digraph models, Zeigler describes another class of coupled models termed Kernel models. Kernel models provide a more loose coupling description. They are used to describe coupled models that can consist of a variable number of component models. All of these however share a fixed topology. Thus only the models that make up the coupled models need to be described. The context, i.e. topology is then used by the coupled model to work out the exact port to port couplings.

Other modelling frameworks/environments that use loose coupling descriptions include SmartSim [Ulgen 90], Cadence [Login93], and Simple ++ [Aesop 94]. All the above schemes use a static routing, which is described explicitly by a graph, but routes during model execution can be varied depending on the state of the models. SmartSim and Cadence use information in routing objects to determine exact coupling details during runtime, while Simple ++ allows the entire state of the coupled model to be involved in determining exact coupling details.

Examples of models with no coupling scheme can also be found in object oriented simulation languages such as MODSIM II [CACI 93] and APPOSTLE [Wonnacott 96]. Interaction between objects in OO languages is performed by executing methods in other objects. Care must be taken to ensure that the invoked messages actually exist in the other models. It must be noted that strictly speaking OO simulation languages do not provide an input coupling scheme, as an object does not need to know the identity of all the objects that access its methods. Output coupling, on the other hand, is made explicit in the calling object.

Bhuskute et.al. [Bhuskute 92] describe a modelling environment that employs a fixed input/output alphabet. All entities (component models) are aware of all other entities and their interfaces. Coupling can be performed by using a uniform naming scheme that describes all the constituent components and ports of a coupled model. In the scheme described by Bhuskute, all interaction occurs via buffers with a fixed naming convention.

3.5 High level architecture

High level architecture (HLA) is a specification developed by the US Department of Defence (DoD) with the aim to support the reuse and interoperation of simulation models. Although primarily developed for the requirements of the defence industry, the DoD is actively involved in promoting HLA in the civilian domain [Dahmann 98]. A special issue of the journal SIMULATION recently was part of this endeavour. The HLA builds and generalises upon past activities such as the aggregated level simulation

protocol (ALSP) [Wilson 94] and distributed interactive simulation (DIS) [DIS steering committee 94].

HLA specifies how multiple simulations termed federates inter operate with each other resulting in a larger simulation termed the federation. HLA is composed of three major components:

1. **HLA Rules:** A set of ten rules that specify the principles and conventions that need to be adhered to so as to achieve proper interaction between the federates during the simulation of a federation. In addition, it specifies the responsibilities of federate and federation designers.
2. **Object Model Template (OMT):** OMT describes the entities to be simulated and the interactions between entities in a federation. Two information models are employed for this purpose. The Simulation Object Model or SOM, one of which exists for each federate in a federation, describes the data a federate requires (consumes) and the data it produces for use by other federates in the federation. The other information model, one of which exists for every federation, is the Federation Object Model (FOM). This model defines what part of the information in the SOM is to be used by the federation. In other words the FOM describes what subset of functionality of each SOM will be used by the federation.
3. **Interface Specification:** These provide a description of the functional interface between the federates and the Run Time Infrastructure (RTI). The RTI is a software that provides a set of services to co-ordinate and allow the exchange of data between federates. The RTI itself can take a number of forms and as such is not part of the

HLA specification; the federates only need to be aware of the functional interface provided by the RTI.

Figure [3.6] illustrates the simulation of federates in a federation. The federates do not all have to be simulation models; they can represent other entities such as data collectors or even real-time human participants as in the case of war game scenarios. Figure [3.6] depicts three federates - a data collector, simulation-1 and simulation-2. Each of these federates has a federate ambassador that enables it to communicate with the RTI. The RTI in turn employs a RTI ambassador that enables it to communicate with the federate ambassador. This separation of integration facilities allows the RTI to be reused in different federations irrespective of the type of federates involved.

The object models in conjunction with the HLA rules can be viewed as providing an explicit coupling scheme. However, HLA does not contain a knowledge representation scheme, akin to a SES in DEVS, that aids a modeller in synthesising models. Recently DEVS has been extended to incorporate HLA in DEVS/HLA [Zeigler 99], where DEVS provides a mechanism for synthesising models from a family of component models while HLA provides a standard that ensures that the component models of the synthesised model can interoperate and provides a mechanism for simulation.

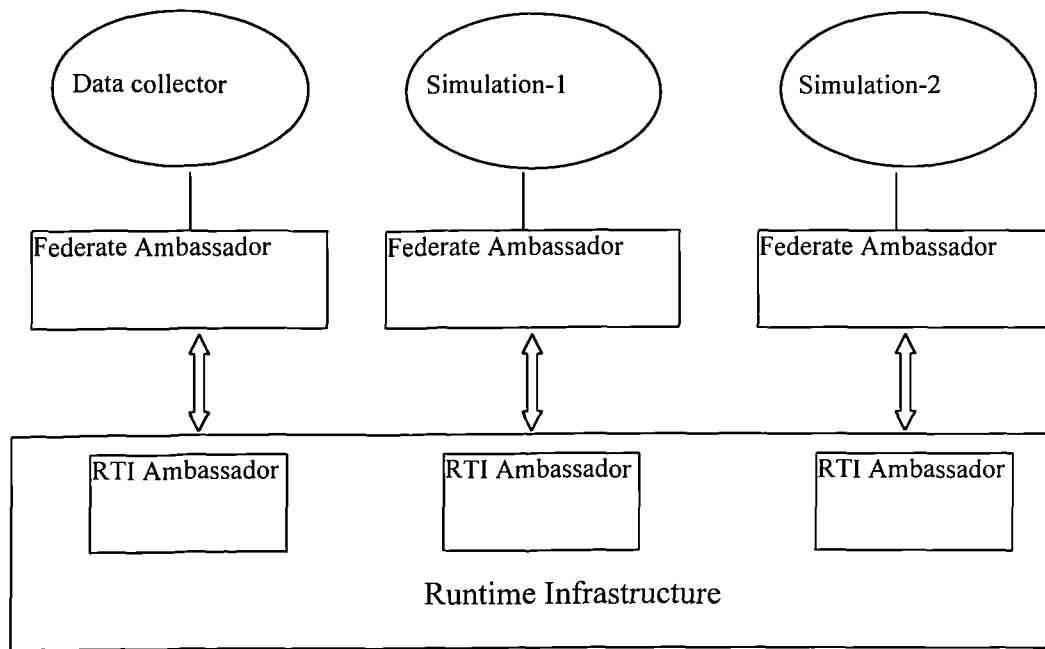


Figure 3.6 Simulation of federates in HLA

3.6 Conclusion

In this chapter approaches in the literature for structuring model objects (both entities and models) with a view to aid the building of composite models were discussed. Three issues need to be addressed in developing a large scale modelling framework. Firstly, the models need to be encapsulated to create component models. Secondly, a coupling scheme needs to be devised that specifies the interaction between the components in a composite model. Finally, a knowledge representation structure needs to be created that incorporates compositional knowledge about the domain to be modelled and guides the synthesis of models.

Various approaches to modifying the traditional world views, with the objective of creating a more modular representation, were reviewed. The modified process view of Cota and Sargent [Cota 92], the PAInt [Davis 96] approach, proposed by Davis, to integrating the process and activity modelling so as to support hierarchical modelling, and the Entity-Connection framework were some of the approaches reviewed in this context.

The DEVS methodology, reviewed in this chapter, provides a comprehensive framework that supports large scale modelling. It employs an explicit coupling scheme and supports the creation of hierarchical models. A structure termed system entity structure (SES) was proposed that described the various facets of a system and aided synthesis of composite models. The SES is pruned in a top-down fashion to create a particular instance of a composite model. The SES includes taxonomically, decompositional and coupling knowledge to help synthesise composite models.

In the next chapter a framework that supports large scale modelling by allowing the synthesis of composite models in a bottom-up fashion is described. In addition, rather than employing a single global structure to incorporate the knowledge required for synthesis, a distributed structure is devised. Further, the ability to create composite models is supported in all the approaches in the literature by extending the same mechanism that is used to create individual models from modelling entities. In Chapter 5, the need to move away from this ‘single formalism’ description is discussed and an approach that makes this distinction is described.

CHAPTER 4

Parallel discrete event simulation - a review

This chapter reviews parallel discrete event simulation (PDES), also sometimes referred to as distributed simulation. The chapter begins by defining the problems involved with simulating a model on multiple processors. PDES techniques from the literature are then presented. This is followed by a review of PDES, and its application in the area of manufacturing systems simulation. Finally, a comparison of the various techniques is presented.

4.1 The problem

PDES is concerned with the simulation of a discrete event simulation model on multiple processors. The motivation behind applying PDES techniques in the supply chain modelling paradigm developed in this thesis is two fold. Firstly, the nature of the supply chain modelling paradigm is such that a number of autonomous models that constitute the supply chain are integrated together to create a single model. For the simulation of this model, a synchronising mechanism is required that ensures that the constituent models run in step. The second issue that PDES addresses is the potentially large model sizes that can result in the modelling of the supply chain. The PDES paradigm, by

allowing concurrent execution of the models, results in improved execution time and consequently makes the simulation of supply chains more feasible in practice.

One approach to the synchronisation problem is to use a global clock to synchronise all the models in the supply chain, allowing execution to proceed in a lock step fashion. When few events occur simultaneously at a single point in simulation time, this approach, termed synchronous PDES in the literature, performs badly in terms of execution time since the number of events that can be simulated concurrently at each clock cycle is small [Fujimoto 90]. The primary focus of PDES literature has been in the simulation of asynchronous systems, where events are not synchronised by a global clock but occur at irregular time intervals. As models of manufacturing systems tend to belong to this class, i.e. asynchronous models, the rest of this chapter is concerned with asynchronous PDES techniques.

4.1.1 The structure of asynchronous PDES

Figure [4.1] depicts the structure of a typical PDES. This paradigm, first proposed by Chandy and Misra [Chandy 79], has been used as the basis in the literature to describe PDES models. The physical system is viewed as being composed of a number of physical processes (PP), that interact with each other during the course of its operation.

For example, if the physical system being modelled is a machine shop, machines, buffers, conveyors could be viewed as the PPs. The physical system is modelled by

constructing a simulator consisting of logical processes (LP), one for every PP in the physical system. Each LP includes state variables, an event list, and a local clock, that together represent the state space and event list of the equivalent serial simulator. Interactions between PPs are modelled by sending and receiving time stamped messages amongst corresponding LPs. Simulation proceeds by every LP processing the events in its input queue in time stamp order. Processing of event messages may involve the modification of state variables, the scheduling of one or more events, the sending of messages, and the updating of the clock.

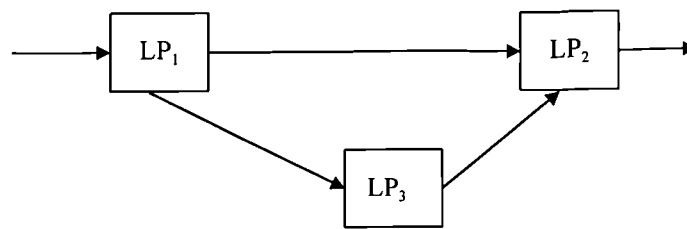


Figure 4.1 An example of an LP framework

4.1.2 Causality errors

The objective of PDES algorithms is to maintain the causality relationships between the events in the simulation and thus avoid causality errors. The following example (Figure [4.2]) illustrates this.

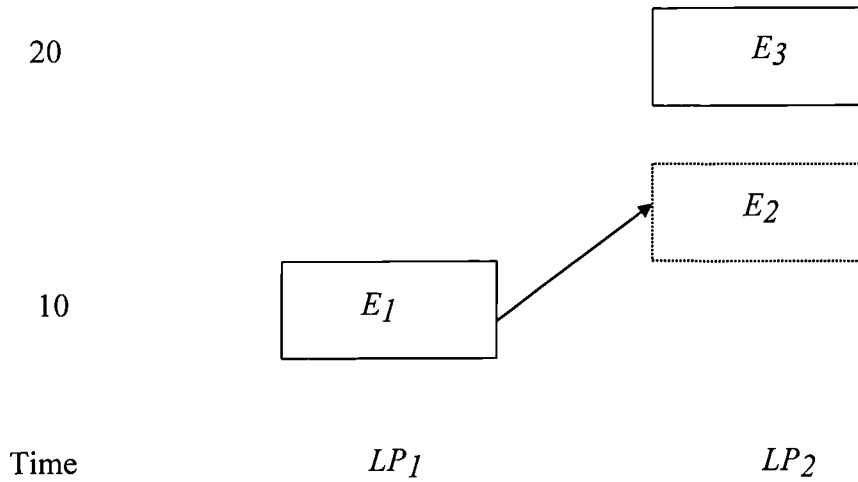


Figure 4.2 Causal relationships

Consider two events E_1 at LP_1 scheduled at time 10, and E_3 at LP_2 scheduled at time 20. If E_1 schedules an event E_2 on LP_2 at say time 15, then it is possible that the event E_2 could affect the execution of event E_3 , thus requiring the sequential execution of events E_1, E_2, E_3 . In other words, from the perspective of the physical system the cause must always precede the effect. However, it is also conceivable that the events E_1, E_2, E_3 are all independent and can be executed concurrently. The objective of the PDES algorithm is to guarantee that causality errors are avoided, but on the other hand, any inherent parallelism in the simulation is exploited to the utmost.

4.2 PDES algorithms

PDES algorithms can be broadly classified as belonging to two categories: conservative approaches or optimistic approaches. The conservative approach executes events only when it can guarantee that it is safe to do so. Causality errors are strictly avoided by this mechanism. Optimistic techniques take a different approach. Events are processed without guaranteeing causality errors. When a causality error is detected, a rollback mechanism is invoked to recover from the error.

More recently hybrid approaches (conservative/optimistic) have also been proposed. They make use of a combination of the two approaches to avoid causality errors. The following sub-sections describe these approaches in more detail.

4.2.1 Conservative approach

4.2.1.1 The Chandy and Misra approach

Early attempts at PDES were based on the conservative approach. Chandy and Misra [Chandy 79], and Bryant [Bryant 77], independently developed the first PDES algorithms. The algorithms are based on the asynchronous paradigm presented in section 4.1.1. The links between the different LPs are statically determined, and as mentioned earlier, each LP sends messages in increasing order of time. The last message received on a link can thus be used as a lower bound of the time stamp of any subsequent messages that will be sent on that link. Messages arriving on each link are stored in a FIFO queue, which because of the fact that messages are sent in an increasing order of time, is also in

increasing order of timestamp. A clock is associated with every incoming link of an LP. The clock is set to the timestamp of the message at the top of the queue or if the queue is empty, it is set to the time stamp of the last received message.

Each LP repeatedly selects the link with the smallest clock time. If the queue associated with this link contains a message, it is processed or else the LP blocks (waits). Chandy and Misra [Chandy 79] proved that this mechanism guaranteed that no casual errors occur. However, a cycle of empty queues can occur under some circumstances and this could result in a deadlock, i.e. each LP in the cycle is waiting for a message from the other, and consequently, the simulation grinds to a halt. Figure 4.3 illustrates this. LP_1 is waiting for a message from LP_2 , LP_2 is waiting for a message from LP_3 , and finally LP_3 is waiting for a message from LP_1 . All three LPs are blocked even though each of the LPs has messages in other queues that are waiting to be processed.

Null messages were proposed [Chandy 79] as a mechanism to avoid deadlock. Unlike other messages null messages have no counterpart in the physical system. They are used only for synchronisation purposes. A null message sent from LP_1 to LP_2 , with timestamp t_{null} , is a commitment by LP_1 to LP_2 that it will not send a message with timestamp smaller than t_{null} . Null messages are sent by an LP on all its output links after the processing of every event. The clock value of every incoming link is used to obtain a lower bound of the next event to be processed. This value can be coupled with knowledge specific to the simulation to arrive at a lower bound of timestamp at each link. For instance, a minimum service time may be available for any message passing

through an LP. This could be taken into account while determining the time stamp for the null message. The receiver of the null messages can then compute new bounds on its outgoing links and so on.

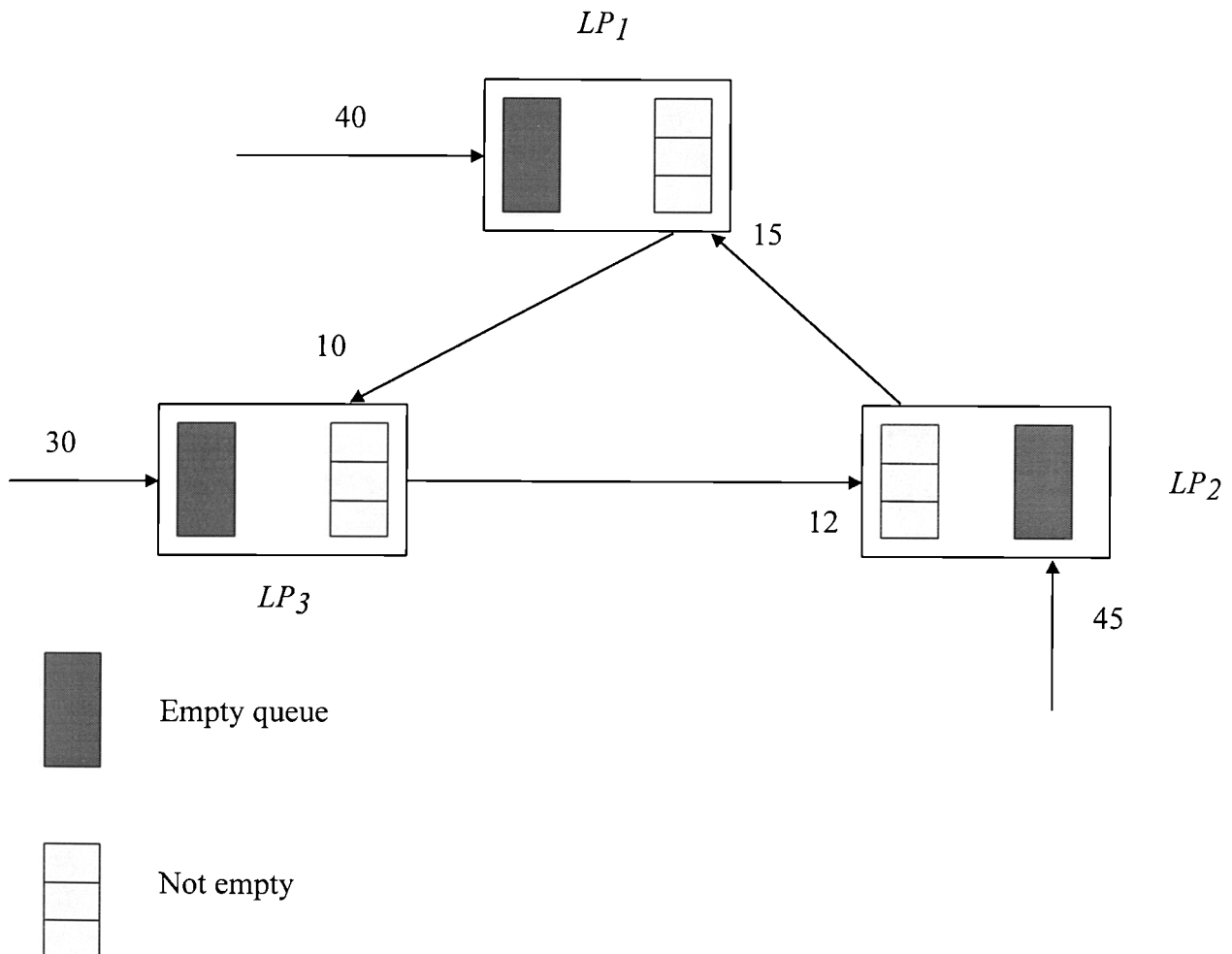


Figure 4.3 An example of deadlock

Empirical evidence [Seetalakshmi 78] has suggested that in practice a large proportion of messages tend to be null messages, and this can adversely affect performance of the simulation. A variation [Nicol 96] of the null messages protocol is to send null

messages only when requested by another LP. This technique significantly reduces the amount of null messages traffic. However, null messages take a longer time as each message requires two messages to be sent- first, a request for a null message and then the actual null message. Another variation in resolving the null message congestion problem is to entirely do away with null messages. A deadlock detection and recovery algorithm has been proposed in the literature [Chandy 81]. This algorithm works in a similar fashion to the deadlock prevention mechanism, except null messages are not employed. *The simulation progresses until it deadlocks. The deadlock is then detected* and a deadlock recovery scheme is used to resolve it.

In this scheme a simulation is in one of the two phases viz.:

1. Parallel phase: Simulation proceeds as in the null message algorithm until it deadlocks. An algorithm is used to detect the occurrence of deadlock. Misra [Misra 86] presents one such algorithm.
2. Phase interface: Initiate computations that advance the LP clocks and resolve the deadlock.

A centralised controller is used to synchronise the above actions. The controller is responsible for detecting deadlocks and ordering the LPs to move into the phase interface stage, and once the deadlock is resolved to move back into the parallel phase. The phase interface scheme works by exploiting the fact that it is always safe to process events with

the smallest time stamp. This is similar to sequential simulation. Thus, until a deadlock is detected, deadlock detection and recovery based algorithms work in an identical fashion to a deadlock avoidance scheme without null messages. Once a deadlock occurs, sequential processing of events is resorted to advance simulation time and resolve the deadlock.

4.2.1.2 Synchronous approach and conservative time windows

A number of PDES approaches belonging to this class exist in the literature [Ayani 89] [Lubachevsky 89]. These techniques are based on the following principle. A global synchronisation function is used to choose, among all the LPs, a set of events that are safe to be processed. These events are then processed. This procedure of determining safe events and executing them is continued iteratively. The different approaches vary in the way that safe events are selected. The basis for many of these algorithms is the notion of distance between LPs. The distance metric between two LPs provides a lower bound of the time that it takes for an event to propagate from one LP to the other, and possibly influence the execution of events in that LP.

For example, consider three LPs - LP_1 , LP_2 , and LP_3 . If the minimum time for an event to be propagated from LP_1 to LP_2 is 10 and from LP_2 to LP_3 is 15, then

$$dist_{ab} = 10; \text{ and } dist_{bc} = 15;$$

Table [4.1] describes the events scheduled in the three LPs:

LP_1	LP_2	LP_3
E_{11} at 5	E_{21} at 6	E_{31} at 18
E_{12} at 6	E_{22} at 17	E_{32} at 19
E_{13} at 30	E_{23} at 20	E_{33} at 100

Table 4.1 Example of event scheduling in a synchronous approach

At the start of the simulation the safe events are E_{11} , E_{21} , and E_{31} , as the distance between them prevents them from interfering with each other. At the next step, E_{12} and E_{32} are safe to process, whereas E_{22} may be influenced by E_{12} , as the distance between them is only 10.

In practice a large proportion of time may be wasted in searching for safe events. Lubachevsky [Lubachevsky 89] proposed the use of a time window to limit the search space of events. An important issue in such a scheme is the size of the window. Too small a size will limit the number of events for concurrent execution and too large a size will mean that the simulation will behave in a similar fashion to a simulation without a time window, making the window managing overhead unjustifiable. In practice, the size of the window needs to be determined by application specific information provided by the modeller.

4.2.2 Optimistic approach

As mentioned earlier, optimistic techniques do not strictly adhere to causal relationships. Events are processed as and when they arrive. An error detection mechanism is used to monitor the occurrence of causal errors. When an error is detected the system rolls back and undoes the erroneous processing of events that caused it. Optimistic techniques can be viewed as operating by constantly betting that the events processed will not result in a causality error. The assumption here is that more often than not causality errors will not occur, thus limiting the overhead due to rollbacks. Jefferson [Jefferson 85] refers to this as the temporal locality principle and conjectures that in practice many applications belong to this category. The time warp paradigm [Jefferson 85] is one of the most popular implementations of optimistic PDES.

Time warp is similar to the conservative LP paradigm in so much as that each LP has its own local clock and no global clock exists. However, LPs do not block when they can not guarantee that an event is safe to process. Each LP has a single input queue into which all the messages are stored in increasing order of time. The only constraint is that LPs follow the local causality principle i.e. events must be processed by each LP in a non-decreasing timestamp order. Execution on LPs continue until the local causality principle is violated, i.e. a message (straggler message) arrives with a timestamp smaller than the local clock, or more simply a message that should have arrived in the past arrives later. As the local causality principle cannot be violated, the LP needs to rollback to a time earlier than the time stamp of the straggler message. This is accomplished by returning the state of the LP to the past, so that events can be processed in increasing

order of time. The mechanism is termed time warp as LPs have clocks that may not agree with one another, and these clocks may go backwards and forwards in time.

4.2.2.1 Rollback control and antimessages

As mentioned earlier when a straggler event is detected by a LP the rollback mechanism is invoked. During the course of the simulation the state variables, and input and output queues are constantly saved, so that they can be used to undo erroneous computation and facilitate a rollback. However, an LP may have sent a number of messages to other LPs, and these LPs in turn, based on these event messages, may have sent more messages to other LPs. A mechanism is required to undo the effects of these messages so as to cause a complete rollback. Jefferson [Jefferson 85] developed a technique based on the notion of antimessages.

Every time a message is sent by a LP, a corresponding antimessage is stored in its output queue. The antimessage is similar to the actual message except for a single type field. Antimessages have a negative sign in that field whereas normal messages have a positive sign. Messages and antimessages are treated by the LP in the same fashion, except, antimessages do not cause any processing by the LP and they have a different queuing discipline. Whenever a message and its corresponding antimessage find themselves in the same input queue, they cancel each other. This property of antimessages makes it useful in unsending a previously sent message because of a rollback.

Consider the following example. Let a message with timestamp 100 arrive at an LP which is at virtual time 150. Clearly, this is a violation of the local causality principle, hence a rollback is ordered. The first step in the roll back process is to restore the state of the LP to a time before 100. The next step is to undo the effect of all messages sent to other LPs between the virtual time 100 and 150. This is accomplished by sending all messages in the antimessage queue that have a timestamp between 100 and 150. The antimessages in effect, now chase their corresponding actual messages and annihilate them. The manner in which the antimessage catches up with its corresponding message is described by the three possible cases that may occur.

1. The original message has been received by the LP but is still in the input queue waiting to be processed. Clearly, in this case the virtual clock of the LP is behind the timestamp of the message, as otherwise a rollback would have occurred. The antimessage gets queued in the input queue and as both message and antimessage cannot exist in the same queue, they annihilate each other, leaving the LP with no record of that message.
2. The second case is if the original message has been processed or being processed. In this case the virtual time of the LP will be greater than the timestamp on the antimessage, as the original message has been processed and virtual time has moved on. The antimessage now has a timestamp less than the virtual time of the LP. This causes the LP to rollback, bringing back the original message to the input queue, and maybe sending other antimessages to other LPs where again similar actions are taken if the antimessage has a lower timestamp than the virtual time of the LP. The rollback brings the input

queue to the same state as case 1, and consequently the message and antimessage annihilate each other.

3. The final case is if the antimessage arrives earlier than the actual message. This is possible because the underlying message passing mechanism does not enforce message ordering, i.e. a message may arrive in an order different from when they were sent. In this case the antimessage waits in the queue until its corresponding message arrives. In the meanwhile the LP may process the antimessage; however the antimessage results in no computation by the LP, and the only effect is that the virtual clock of the LP is updated. When the actual message arrives it causes a roll back which brings the antimessage back into the input queue and the message and antimessage annihilate each other.

4.2.2.2 Global virtual time

An important issue not addressed so far is the fact that the time warp mechanism seems to require infinite memory space or in other words the memory requirements for a simulation run is not bounded. State vectors and input output queues of the model need to be intermediately saved at different points in simulation time to guarantee successful rollback. Secondly, if individual LPs are constantly moving in forward and backward directions (with respect to time), how does one determine the extent to which the simulation has progressed? Time warp exploits the property of global virtual time (GVT) to overcome this.

GVT at real time r is defined as follows:

GVT = Minimum of { (The virtual clocks of all LPs at time r) and (The virtual timestamps of all messages in transit, i.e. messages that have been sent but not received.)}

GVT thus provides a lower bound of the furthest a LP can roll back. It can be viewed as a floor beyond which LPs cannot roll back. All states saved before GVT can be reclaimed as it is impossible for an LP to rollback beyond GVT.

As mentioned earlier, the virtual times of the LPs may not agree with each other and in addition, are not restricted to moving in one direction, i.e. virtual time may go forward or backward. However, GVT always moves forward although not necessarily at a constant rate, making it a virtual clock for the entire system, and thus being a measure of how far the simulation has progressed.

A number of algorithms [Lin 89] [Samadi 85] exist in the literature for calculating GVT. The basis of most of them is to temporarily stop the simulation and order all LPs to send information regarding their virtual clocks to a central controller. The central controller then uses this information to calculate the GVT. The frequency of calculating GVT is a trade-off between space and time. Calculating GVT frequently makes efficient use of memory. However, because the simulation needs to be paused, the progress of the

simulation suffers. On the other hand, infrequent calculation of GVT benefits execution time at the expense of increased memory requirements.

4.2.2.3 Variations of time warp

A variation to the rollback mechanism is to employ lazy message cancellation [Gafni 88]. In time warp when rollback occurs messages are cancelled aggressively, i.e. antimessages are immediately sent to cancel messages sent earlier. In lazy cancellations antimessages are not immediately dispatched. The LP waits to see if the re-execution of rollbacked computations regenerates the same message. If the same message is generated there obviously is no need to send the antimessage.

For example, consider a LP rolls back to time T , and there is an antimessage with timestamp $T+t$. If lazy cancellation is employed the antimessage is not sent until the virtual clock passes time $T+t$. At this time if the rollbacked computation resulted in an identical message, the antimessage is not sent, else, the antimessage is sent.

Another variation is to only employ local rollback. In this technique messages are not sent immediately. They are only sent to other LPs after the timestamp on the message is greater than GVT, in other words only when the sending LP can guarantee that in a rollback the message will not be required to be cancelled. Thus, antimessages are not employed in this scheme.

4.3 Conservative versus optimistic techniques – a critique

An important factor in the performance of a PDES algorithm is the extent to which inherent parallelism in the execution of events by LPs is exploited. At one end of the spectrum is the classic Chandy and Misra [Chandy 79] conservative approach where if, for example, it is possible that an event E_A on LP A may affect E_B , an event on LP B , then they must be executed sequentially with E_A first executed followed by E_B . If, for instance, in the simulation E_A seldom affects E_B then performance can be degraded severely. In practice, conservative approaches depend on look-ahead information to improve performance. A number of studies [Fugimoto 90] have shown the reliance of conservative approaches on the availability of look-ahead information. In addition, as mentioned earlier, look-ahead information is also required (not in the case of deadlock detection and recovery schemes.) to prevent the simulation from deadlocking. A consequence of the dependence of conservative approaches on look-ahead information is that the simulation modeller needs to concern himself with the details of the synchronisation mechanism. In addition, modification of the simulation model is made more complicated as the modeller needs to verify that any changes to the model will not affect the look-ahead properties of the model. Automatic extraction of look-ahead information from models has been investigated by a few researchers [Cota 90] [Bagrodia 90]. However, the application of such techniques are still in their infancy and are restricted to simple cases.

In contrast, optimistic approaches try and exploit as much parallelism present in the simulation as possible. They do this by executing events concurrently irrespective of

their causal dependence on one another. Any causal errors are then detected and the effects rolled back. Thus, optimistic approaches have no need for look-ahead information. An important consequence of this is that the modeller is free from concerning himself with issues of synchronisation and can concentrate on the modelling effort instead. This is much like in the case of development of traditional sequential simulations, where the modeller is oblivious to the details of the implementation of the event list. Optimistic schemes also avoid the potential for deadlock as LPs do not block in anticipation of messages. The flip side of this is that optimistic approaches incur additional overheads compared to conservative approaches as they are required to save states periodically and perform rollbacks to undo incorrect computations. This can be a problem, particularly if the simulation exhibits ‘thrashing’ behaviour where most of its time is spent in executing incorrect events and undoing their effect by rolling back. Further, the optimistic mechanism (executive) is more complicated to implement than its conservative counterpart. However, this is not a significant disadvantage, as the cost of developing an optimistic executive needs only to be paid initially during its development. Once a robust optimistic mechanism is implemented, the benefits of simpler model development can be reaped.

The choice of PDES scheme for a particular application depends on a number of factors. If for example, the application exhibits good look-ahead characteristics and a simple simulation executive is desired, then good performance can be derived from an executive based on the conservative approach. On the other hand, if look-ahead information is limited or user transparency of simulation protocol is paramount then optimistic

approaches provide an alternative choice of PDES scheme. Table [4.2] summarises issues relating to conservative and optimistic approaches.

	Conservative	Optimistic
Parallelism	Limited by worst case scenario.	Not limited
Performance	Depends on the quality of look-ahead present in the simulation.	Can exhibit thrashing behaviour
Overheads	Blocking till the avoidance of causal errors can be guaranteed, and the overhead of deadlock avoidance and recovery.	Overhead involved in state saving and recovery.
Development of simulation executive	Simple	More complex, and harder to verify robustness.
Development of simulation models	Complicated, requires the modeller to be aware of synchronisation issues. Harder to modify models.	Higher degree of transparency of synchronisation mechanism and more robust to model change.

Table 4.2 Comparison of conservative versus optimistic schemes

4.4 PDES based simulation languages

Although research in the area of PDES has been conducted for many years during which a number of PDES algorithms have been proposed and implemented, the use of PDES techniques has not been adopted by the practising simulation community at large [Fujimoto 93]. Page [Page 93] has attributed this mainly to the PDES community ignoring the methodologies adopted by the mainstream simulation community in the pursuit of performance gains. Thus in many cases the simulation techniques developed

by PDES researchers have tended to be contrived and bode little relevance to the requirements of simulation modellers. In order to remedy this a number of researchers [Bagrodia 90] [Waldorf 94] [Wonnacott 96] [West 88] have investigated the development of PDES based simulation languages that incorporate many of the features of traditional sequential simulation languages.

4.5 PDES of manufacturing systems - a review

Chen and Peng [Peng 96] argue that although PDES has shown promise in a number of areas such as telecommunications, computer networks, battlefield simulations etc, its implementation in manufacturing simulation has not been very successful to-date. They attribute the failure to lack of suitable decomposition techniques, oversimplification of the model so as to make it suitable for PDES, and the use of exotic parallel computing platforms.

.

Some of the earliest work in application of PDES techniques to manufacturing was conducted by Shires [Shires 84]. The objective of the project was to develop a multi-processing framework for the simulation of a flexible manufacturing system (FMS). The entities in the simulation were limited to buffers and machines. The scheme used was different from other PDES attempts. Their ideas were influenced by traditional parallel processing concepts. The entire simulation was divided into five modules, viz. one module to simulate the machine centres, another to simulate all the buffers, a third was used to model operation decisions, a fourth module was responsible for the collection of

statistics and animation, and the fifth module managed the event list and super-executive. The modules were programmed in Pascal and executed on an Intel SYS 310 computer controlling a set of slave processors via an Intel multibus.

A drawback with this approach is that it is not scaleable. A single event list and global clock is used, resulting in a bottleneck. In addition, in practice the communications overhead may be significant as the entities in the model are tightly coupled and a large proportion of time may be spent in message passing.

A similar approach was used by Hon and Ismail [Hon 91]. Each entity in the simulation was represented by a process that was concurrently executed on a transputer. A central “monitor” process was connected to all the entity processes and was responsible for monitoring event and state changes of the entities. This system was used to simulate a cell containing an AGV, a robot, three identical machine centres, and four buffers. The system was implemented on OCCAM based transputers.

Both of the above mentioned approaches resorted to very fine grained decomposition. Entities at the cell level were mapped on to different processors. The interactions between entities tend to be very strong in such an approach, and consequently, in practice, communication overheads negate the benefits of concurrent execution.

A few researchers have tried to exploit the structure of manufacturing systems to arrive at more efficient PDES algorithms. Nevison [Nevison 1990] proposed a simulator that

exploits the closed loop structure of some flexible manufacturing systems. The model consists of a system of conveyors that feed a number of assembly stations. Parts and WIP are routed by transfer points, under computer control, to appropriate assembly stations. The system was defined to be composed of eight building blocks. Nevison argues that one of the primary reasons that the conservative (Chandy and Misra) algorithm performs poorly is due to the flooding of null messages. He developed a set of rules, taking the structure of the system into account, that determine when a deadlock can occur. These rules are used as a basis for dictating when null messages were sent. A network of transputers was used as the hardware platform for the simulation.

Bhuskute and Mize [Bhuskute 93] exploit the deterministic routings of some manufacturing systems. A model is described by three parameters: the number of machines M , the number of parts N , and a graph that determines the routing of each part. Models are described by varying these three parameters. Each processor executes a sub-model with its own event list. Message passing is used to simulate dependencies between sub-models. The system was implemented on an Intel ips/2 concurrent computer with 32 nodes.

The Nevison and Bhuskute/Mize approaches both sacrifice generality and oversimplify the model in pursuit of improved execution efficiencies. For instance, Bhuskute and Mize ignore transportation mechanisms and parts are made to follow deterministic routings. In practice manufacturing systems are more complex and such assumptions reduce the fidelity of the simulation.

In addition, the majority of approaches base their simulation on “exotic” parallel computing hardware. Manufacturing organisations seldom have the expertise or infrastructure to support these hardware platforms.

Fujii [Fujii 94] describes applying PDES techniques to a CIM system. The factory is decomposed into areas. Each area is assigned to a processor and simulation is conducted in a sequential manner. Transportation systems connecting all the areas are modelled on a separate processor. Since the areas do not depend on each other strongly, simulation for the most part can continue independently. A time bucket algorithm, a variation of time warp, was used for synchronisation. The simulator was implemented on a network of six SPARC workstations.

This approach, to an extent, addresses the problems of decomposition and hardware platforms. A CIM factory is decomposed into loosely coupled areas, thus reducing their interdependence and consequently reducing message traffic. In addition, a network of workstations, as opposed to dedicated parallel computers or transputers, is employed to implement the simulation.

Developed by	Synchronisation scheme	Hardware platform
Robert and Shires [Shires 84]	N/A	Intel SYS 310 computer with multiprocessing slave computers connected through Intel multibus
Nevison [Nevison 90]	Conservative	Transputer network
Hon [Hon 91]	Conservative	Transputer network
Bhuskute [Bhuskute 93]	N/A	Intel ips/2 concurrent computer with 32 nodes
Fujii [Fujii 94]	Optimistic	Six Sun SPARC workstations connected by ethernet

Table 4.3 Summary of PDES in manufacturing

4.6 Conclusion

In this chapter a literature review of PDES was presented. The motivation behind using PDES techniques is two fold. Firstly to provide a mechanism to synchronise the models that constitute the supply chain and secondly execution speed. The large scale nature of supply chain models necessitates an efficient simulation engine, so as to allow the simulation to be performed in a reasonable time and thus be practical. PDES techniques can be broadly classified into conservative and optimistic techniques. Conservative techniques process events only when it can be guaranteed that causality errors will not occur. On the other hand, optimistic techniques execute events without avoiding

causality errors; if a causality error is detected, the simulation is rolled back to undo the causality error. Peng and Chen [Peng 96] believe that PDES has not been successful in practice in the simulation of manufacturing systems for two reasons. One that a generalised decomposition scheme does not exist, and second that expensive parallel processing platforms have been employed in the past to implement the simulator. In chapter 6 a PDES scheme is presented that takes a more coarsely grained approach, by concurrently simulating individual models in a supply chain rather than individual entities, than the reviewed attempts at applying PDES to manufacturing systems described earlier.

CHAPTER 5

A framework for composite modelling

The objective of this thesis, as previously mentioned, is to develop a framework for the modelling and simulation of manufacturing supply chains. Three broad requirements were identified in Chapter 2 for such a framework. The framework must:

- Address the distributed nature of component models
- Provide a mechanism for integrating component models to create model composites
- Provide a mechanism for executing (simulating) the model composites

In this chapter and the next (Chapter 6) a modelling framework HerMIS (acronym for Heterogeneous Model Integration and Simulation) that satisfies the above requirements is presented.

5.1 Design rationale

In this section some of the ideas, concepts and features that influence the design of HerMIS are introduced.

5.1.1 Heterogeneous composite modelling

An important aspect in supporting composite modelling is the mechanism employed to describe the interaction between component models in a composite. Figure [5.1] depicts the various levels of interaction in a modelling environment. In general, approaches in the literature that support composite modelling, reviewed earlier in Chapter 4, employed a uniform mechanism to describe interaction among the various levels of modelling. For instance, in the case of DEVS [Zeigler 84] and CFG [Cota 94] interactions between modelling elements were described using an explicit coupling scheme that mapped input/output ports from one modelling element on to another. Similarly, models and composite models were also created by coupling modelling objects through the same explicit mapping of the input/output ports. Again in the case of PAInt [Davis 96] the process-activity interaction model is used to express interaction at all levels of composition.

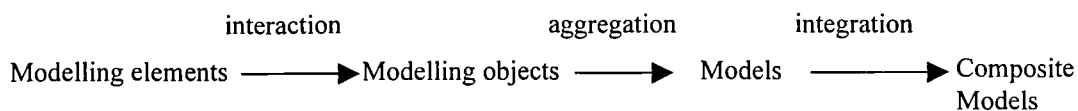


Figure 5.1 Levels of composition (modified from [Davis 96])

A uniform interaction mechanism provides a seamless modelling environment in which models are composed in a hierarchical manner. However, as the nature of interactions occurring at each level of composition are different, building models using a uniform compositional approach may complicate the modelling process. For instance, the process-activity formalism used in PAInt leads to a conceptually simple mechanism for

modelling interactions between modelling elements. This is because at this level it is natural to think in terms of activities and processes. However, at the composite modelling level, representing interaction between models is harder to conceptualise using the process-activity interaction mechanism. For example, consider a frequently occurring inter-model interaction: transferring temporary entities from one model to another. Here, a port based explicit coupling scheme, similar to an interconnection graph (IG), which is employed in the control flow graph formalism [Cota 94], may provide a conceptually simpler representation.

One advantage of making a distinction between the manner in which models and composite models are built is the support such a framework provides for heterogeneous modelling. By heterogeneous modelling it is meant that models that compose a composite model are free to be expressed in a variety of modelling formalisms. In contrast, in the case of approaches reviewed that support composite modelling (DEVS, CFG, PAInt), composite models were homogenous. For instance, in the case of DEVS all the models that compose a coupled model have to be DEVS (coupled or atomic) models.

Heterogeneous modelling allows a modeller to choose a formalism for model building that is suitable for the modelling task. A formalism may be chosen because of its naturalness for the task. If, for instance, a discrete part manufacturing system is being modelled, a formalism supporting a material based view may be more appropriate than say a machine based approach. The knowledge and expertise in using a particular formalism can also play a part in the selection of a modelling formalism. In addition, factors such as availability, popularity, cost etc. can influence the choice of a modelling

tool and, consequently, the modelling formalism employed.

5.1.2 Composite model synthesis

Creating composite models involves two activities:

1. Selecting appropriate models that constitute a composite.
2. Specifying how the constituent components in a composite interact.

As mentioned earlier, a number of composite modelling approaches required the explicit specification of the port couplings involved. Such a scheme has the advantage of being based on a sound set theoretical foundation, as demonstrated by the DEVS methodology. However, creation of composites can become quite user intensive. This is particularly so when a large number of models are involved in the composition.

An obvious solution to this problem is to use a knowledge based mechanism to guide composition. One such approach reviewed earlier employed a system entity structure (SES) to contain information of models and their possible couplings. The SES is pruned based on a given criterion to arrive at a suitable composition. Although using a SES simplifies composition, a SES needs to be created in the first instance. This can be quite a complicated process and involves the complete exposition of all possible models in a system and their various couplings [Davis 96]. Consequently, the resulting SES could be very large and unwieldy. This is particularly so in the case of variable structure composites, as every possible coupling of a variable structured system needs to be explicitly represented.

In addition, the following other reasons make a compositional approach based on a SES unsuitable for structuring models that compose a supply chain:

Hierarchical focus: The SES is fundamentally a hierarchical structure. It represents a variety of decompositions of a particular system, thus allowing for a top down synthesis of models. Consequently, if one looks at a SES, the manner in which the various entities interact with one another is not readily apparent. One has to look at the composition tree in conjunction with the SES for this information. In the composition of models of supply chains one is primarily concerned with identifying the component models and modelling the manner in which they couple and interact with one another. Hence, it may seem that a network structure that focuses on the connectivity of the various entities as opposed to a tree structure (SES) that focuses on decomposition would be more appropriate in guiding composition.

Figure [5.2] illustrates this point by considering the supply chain of a hypothetical personal computer manufacturer. Figure [5.2 a] depicts a hierarchical representation while figure [5.2 b] depicts a network representation of the supply chain. Looking at the network representation the structure of the supply chain and the interaction of the various components is readily apparent.

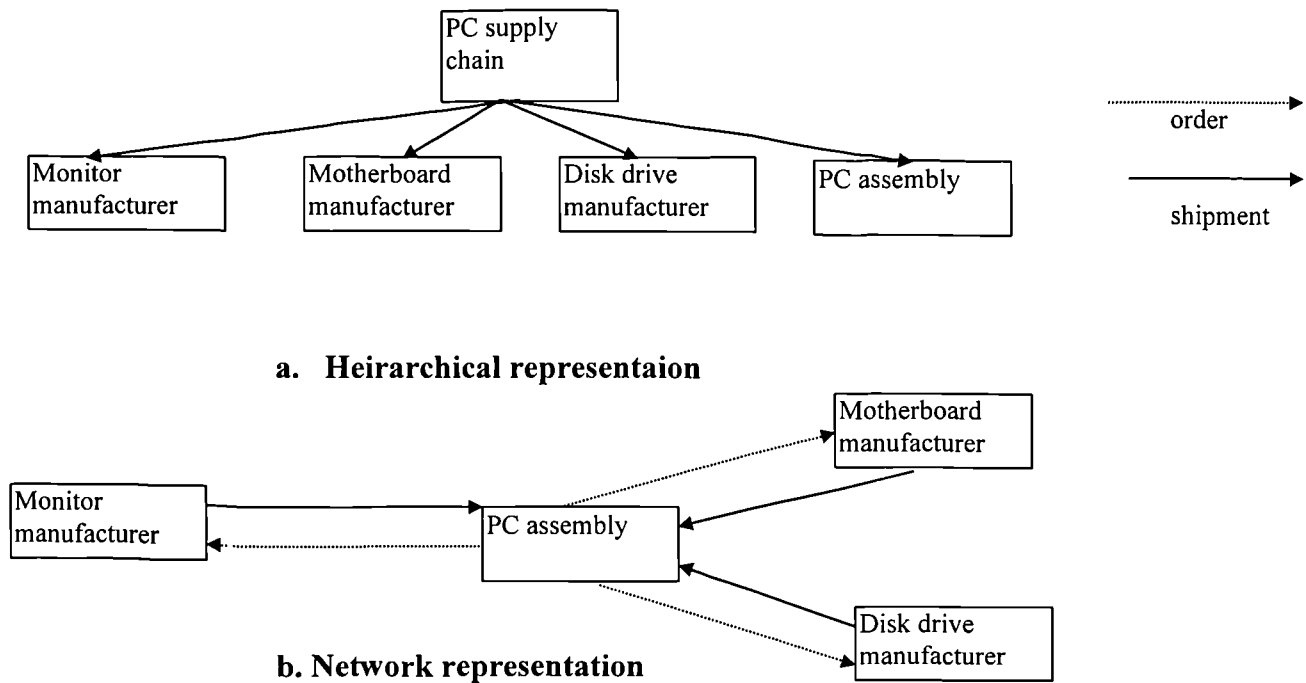


Figure 5.2 An example of a supply chain of a personal computer manufacturer

Isomorphic constraint: The manner in which a system is decomposed in a SES can affect the subsequent ability to synthesise models from it [Davis 96]. Let us again consider the example of the personal computer manufacturer illustrated in Figure [5.2]. The interactions between the entities, viz. Monitor manufacturer, Motherboard manufacturer, Disk drive manufacturer, and PC assembly, decomposed from the PC supply chain node is described by a composition graph. Care must be taken in developing this composition graph as subsequent specialisation or decomposition of any of these entities will be restricted to this coupling scheme. For instance, say, if the personal computer manufacturer deals with two types of monitor manufacturers, namely

a flat screen LCD monitor manufacturer and a CRT monitor manufacturer. The two monitor manufacturers will be represented in the SES as specialisations of the Monitor manufacturer. The interaction between the two monitor manufacturers and the PC assembly entity may be quite different. Order specification, for example, can be quite different in the two cases. Orders for CRT monitors may include information about tube size, pitch size etc. In contrast, the LCD display may include type of display (TFT, active matrix), pixel resolution etc. Thus, the ability to decompose or specialise entities in the SES is restricted by the type of interaction between the entities. If the nature of the system modelled is such that the interactions between entities is fixed for all possible subsequent specialisation and decomposition, then creating a SES is not a problem. However, if this is not the case, the creation of a SES and the subsequent synthesis of models from it can be quite difficult.

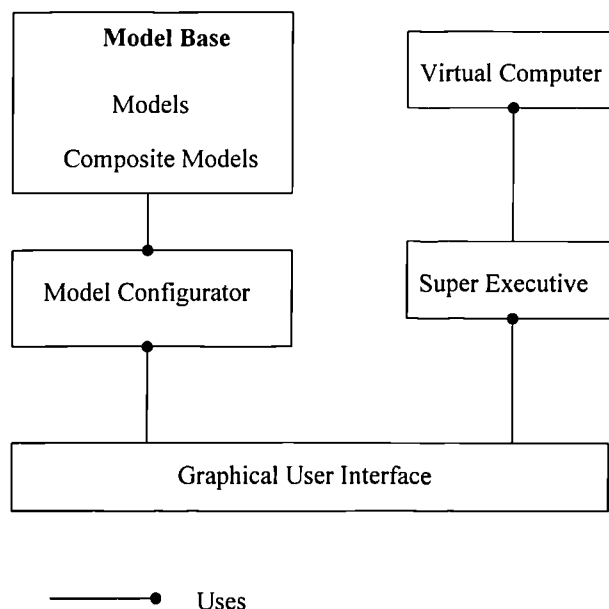
The need for decomposed or specialised entities in a SES to be isomorphic is in part due to the way in which the composition graph is structured. The composition graph incorporates information about what other entities a particular entity interacts with and the way in which this interaction occurs, i.e. the coupling scheme. If these two aspects can be described separately rather than be combined in a single structure, then it may be possible to postpone the description of the exact nature of the interaction to a more detailed level of decomposition or specialisation, thus relaxing the need for decomposed or specialised components to be isomorphic.

Global nature: Knowledge about model composition of an entire system is encoded within the SES. If models belong to more than one SES, as in the case of models in a

supply chain, each SES needs to be created individually, resulting in repetition. An alternative is, rather than employing a global structure to represent information required for composition, this information can be distributed among the various models. This allows the model and knowledge of how it interacts with other models to be created and maintained by the owners of it, thus allowing for collaborative model development.

5.2 Functional view of the architecture

In this section and the next the composite modelling framework HerMIS is described. A functional view to better illustrate the requirements of the framework is first presented. This is followed by a process view that describes all the elements of HerMIS and how they support the composite modelling process.



5.3 A functional view of the composite modelling framework

Figure [5.3] illustrates the different function modules and their interrelationship. The modules can be classified as belonging to two categories; those that support model composition, and those that support model execution.

Model composition

The modules that belong to this category are involved in providing support for the creation of composite models. Activities supported include model creation, access, composition (integration), and reuse.

Model base

The function of the model base is to act as a repository of models (atomic as well as composite). The model base provides functions for model searching, based on user-defined criteria, model access, and model storage. A model builder can search for component models for a composite model or save models for future reuse.

In addition, information regarding the function and the requirements for interfacing with other prospective models need to be stored in the model base. A mechanism to describe the types of temporary entities that the model consumes and produces in a composite is required.

Aspects of security and control need to be addressed by the model base. Models may be viewed as proprietary information. Functions that allow the imposition of restrictions on

model access and use need to be provided. Users may be granted privileges such as access only, permission to modify, permission to integrate and create composites etc.

Model configurator

The function of the model configurator is to facilitate the building of composite models.

The model configurator uses the information in the model base to create composite models. Composite model builders (they may be independent of model builders) use the model configurator to describe the different components that constitute the composite and their interrelationship. As a minimum, a mechanism to specify the inter-model coupling for the transfer of information between the input/output ports of the constituent models needs to be provided. Due to the heterogeneous nature of the modelling framework, simple mapping between input and output ports may not be sufficient. Models in a composite may employ different representation schemes for the exchange of information. For example, a model representing a memory chip manufacturing facility may represent a memory chip entity by a product code whereas another model may describe the same memory chip by a different set of attributes, say name, type, capacity, speed etc. Functions that translate between the different representations are required if meaningful interaction is to occur between the models.

In addition, issues regarding validity (syntactic) of coupling needs to be addressed by this module. Interface information of models can be used to guide or check the validity of inter-model coupling. Finally, functions to save composite models for future reuse in the model base need to be provided.

Model execution

The modules in this category deal with the simulation of composite models in the model base. The activities involved include creating a distributed computer network for the concurrent execution of the models, mapping the constituent models of a composite model onto appropriate processors, creating and managing the physical interface for communication between the models and co-ordinating activities between the models during simulation.

Super executive

This module is responsible for the synchronisation of activities between models. The super executive is analogous to the simulation executive in a traditional modelling environment. It is responsible for scheduling the inter-model activities, and the exchange of temporary entities between models need to be co-ordinated so they are sent and arrive at appropriate times.

The super executive could employ a simple mechanism that monitors the time-of-next-event of all the models and schedules the activation of an event in the model with the least time-of-next-event. Alternatively a PDES scheme may be used to exploit the inherent parallelism in the models, allowing for concurrent execution of models.

Virtual distributed computer

This module encapsulates the physical processors that compose the distributed computer. Functions are provided for creating a network computer on which the individual simulations can be distributed and will be executed. The simulations can be mapped on to the processors and be simulated using distributed algorithms. A PDES algorithms is presented in the next chapter that performs these functions.

5.3 Composite modelling framework - Process view

5.3.1 Model Taxonomy

As mentioned earlier one of the requirements for a composite modelling framework is a knowledge based representation that aids the model building process. Three types of taxonomic information are required for this purpose.

1. Classification of models based on what it models.
2. Classification of models based on the type of models it interacts with.
3. Information about how models interact with one another, i.e. coupling specification.

Three taxonomies viz. *Model-Type*, *Model-Interaction* hierarchies and *Transfer-Entity* hierarchies are used for this purpose

Model-Type categorises models based on what they represent. Models that are classed as belonging to a particular *Model-Type* all model the same system or class of system. For

example, models that represent a computer motherboard assembly line could all be classed as belonging to say, *Model-Type motherboard_mfg*. Similarly, all models of a CPU manufacturing facility could belong to *Model-Type mem_mfg*.

To support composite modelling, in addition to knowing what a given model models, it is also useful to know with what other models it may interact. *Model-Interaction* hierarchies provide this information. Continuing with the example cited earlier of computer motherboard manufacturers, a model of *Model-Type motherboard_mfg* could interact with a model of a CPU production facility to create a composite model that models a part of the motherboard supply chain.

Each *Model-Type* is associated with a *Model-Interaction* hierarchy. *Model-Interaction* classes are structured within a *Model-Interaction* hierarchy based on the *Model-Types* each class interacts with. If two models interact with identical *Model-Types* then they belong to the same *Model-Interaction* class. Inheritance relationships are applied to classes within a *Model-Interaction* hierarchy such that *Model-Interaction* classes interact with at least all the *Model-Types* that a parent class interacts with.

Model interaction is described by using ports. A *Model-Interaction* class has a port for every model it interacts with. The port specifies the *Model-Type* that it can be coupled with and the *Transfer-Entity* hierarchy to be used for the interaction. Figure [5.4 a] illustrates a hypothetical *Model-Interaction* hierarchy used to model a personal computer manufacturer. A variety of *Model-Interaction* classes can be defined depending on what

aspect of a supply chain is to be modelled. Four model classes, viz. *pc*, *pc_mem*, *pc_mon*, and *pc_mem_mon*, are depicted in Figure [5.4a]. Models belonging to class *pc* have a port that can be coupled with a customer model. Similarly, models belonging to class *pc_mem* can be coupled to a model that models a memory production facility. In addition it can also be coupled to a customer model as it inherits a customer port from its parent class *pc*. Depending on the objectives of a simulation study, models can be selected from appropriate classes and coupled to create a composite model.

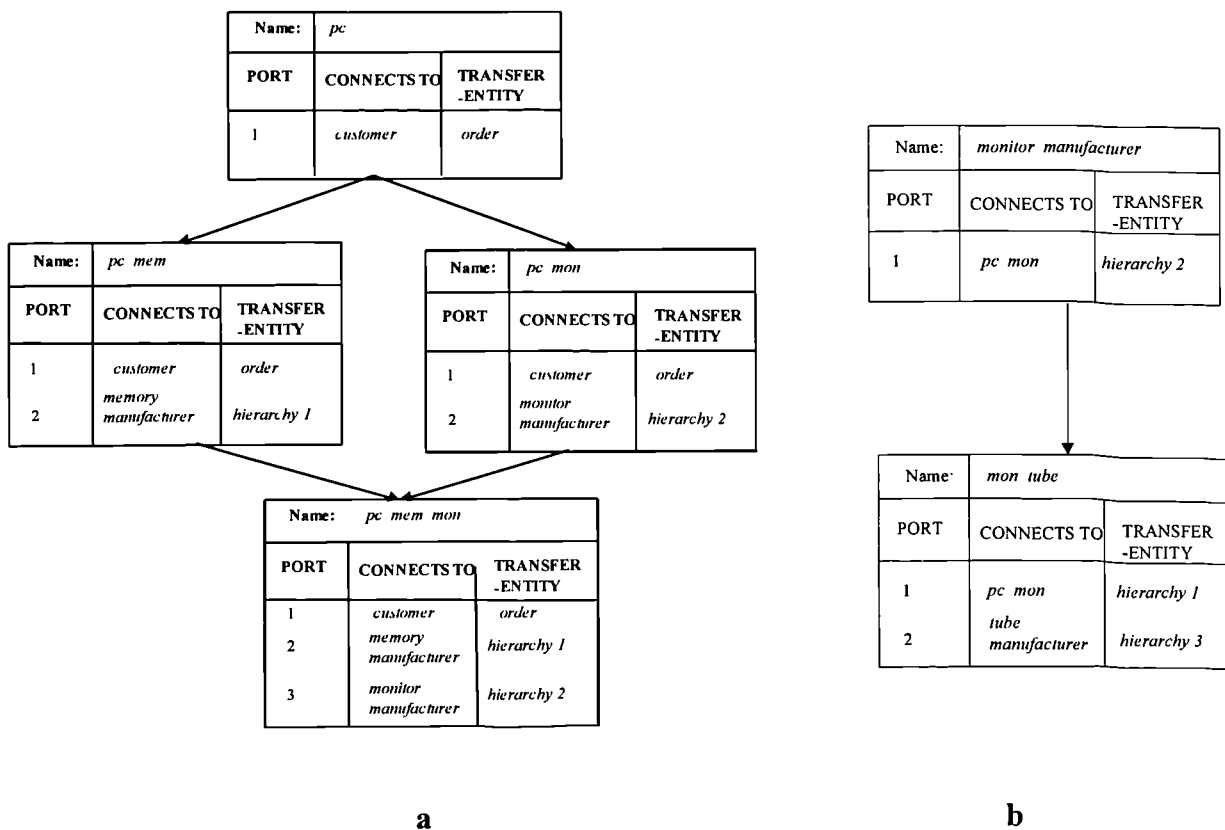


Figure 5.4 Class hierarchies: a) PC manufacturer, b) Computer monitor manufacturer

For instance if the relationship between the monitor manufacturer and the personal computer manufacturer is to be investigated, a model belonging to class *pc_mon* can be coupled with a model belonging to class *monitor_manufacturer*. Further, if a more detailed study is deemed to be required, say the influence of the monitor tube supplier is to be included, an appropriate model belonging to class *mon_tube* can replace the model belonging to the *monitor_manufacturer* class. A model of class *tube_manufacturer* can then be coupled to the *mon_tube* model to complete the network.

The *Model-Interaction* based classification guides composition by identifying the models that can be coupled to produce meaningful composites. However, it does not say anything about the manner in which the constituent models may communicate. This information is derived from a *Transfer-Entity* specification. The *Transfer-Entity* specification expresses the structure of the information that is exchanged between the interacting models via the model ports. It is structured in a hierarchy of classes to maximise reuse by inheritance. Figure [5.5] illustrates a part of the *Transfer-Entity* hierarchy of a personal computer manufacturer. Here the structure of a *Transfer-Entity* that represents an external order placed by a customer to the manufacturer is depicted. For instance, the *Transfer-Entity order_basic* may be suitable for a 'rough cut' model which does not take into account the various types of personal computers manufactured, whereas *order_advanced* may be appropriate as a *Transfer-Entity* in the case of a more sophisticated model that takes into account the exact specification of the ordered computer.

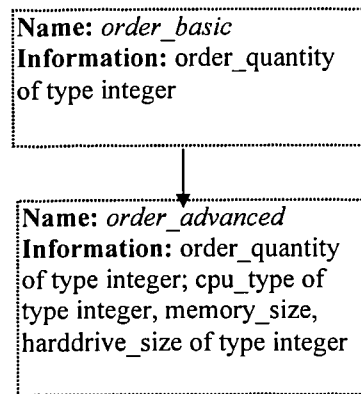


Figure 5.5 *Transfer-Entity* heirarchy of a customer order placed with a PC manufacturer

The *Model-Interaction* hierarchy in conjunction with the *Transfer-Entity* specification provide a knowledge representation scheme that aids the synthesis of composite models. The *Model-Type* and *Model-Interaction* taxonomy can be viewed as providing semantic information for the synthesis of composite models, while the *Transfer-Entity* taxonomy ensures syntactic compatibility.

5.3.2 Model selection

Model-Type and *Model-Interaction* taxonomies incorporate knowledge of models at a structural level. The *Model-Type* taxonomy in conjunction with *Model-Interaction* hierarchies of component models in a composite incorporates knowledge about the structure of the composite model. However, in addition, knowledge of behavioural aspects of models is required to aid the creation of composite models. As mentioned earlier, *Model-Types* indicate the real world system a particular model represents and the *Model-Interaction* hierarchy tells us of the structure of interaction between the models.

However, it does not tell us about the aspects of the real world system that are incorporated in the model, or at what level of detail the model is constructed. For example, various models of a motherboard assembly line can be created. At the simplest level the system can be modelled by a queue that stores orders and a server with a fixed delay that approximates the time taken to assemble a motherboard. More complex models can be created that take into consideration the type of motherboard being created, detailed sequencing and constraints involved in the assembly etc. Thus, in addition to the information provided by the *Model-Type* and *Model-Interaction* hierarchy, an additional taxonomy is required that classifies models based on the manner in which they abstract the real world system in consideration and help the user (composite modeller) select component model instances from *Model-Type* classes.

This additional taxonomy is harder to provide as a real world system can be modelled in a multitude of ways to reflect various modelling objectives. Developing a comprehensive methodology by which models that represent a common system can be classified and related to one another, based on the degree of abstraction, is beyond the scope of this thesis. However, an example of a classification scheme that may help as a starting point in tackling model selection is given below.

At the most basic level, model selection can be totally delegated to the user (composite modeller). The user can be presented with a list of models and then be requested to make a selection. In addition, the models could be tagged with a document that provides descriptive information about them. Further, models can also be classified based on the performance measures they provide. For example, all models of a system that include

levels of inventory as a performance measure can be classified together. Within a performance measure class, further sub-classification is possible. For example models that include inventory levels as a performance measure can be connected by a set of directed edges such that an edge connects a model of lower resolution with a model of higher resolution.

5.3.3 Agents and model synthesis

Software agents is a fast evolving and intensively researched area. The term agent has become so pervasive that an all encompassing definition is beyond the scope here. Instead the term agent is adopted as described by Nwana [Nwana 96]. Agents can be broadly defined as a component of software and/or hardware which is capable of acting exactly to accomplish tasks on the behalf of the user [Nwana 96].

Agents provide an elegant paradigm for developing mechanisms for model synthesis and execution in a distributed environment. Two types of agents are used in the modelling framework: *synthesis_agents* and *model_agents*. The two agent types can be viewed as supporting different aspects of the distributed nature of HerMIS. The *synthesis_agents* enable distributed (collaborative) model development and *model_agents* support the distributed execution of models. *Synthesis_agents* are described here and *model_agents* are discussed in a subsequent section (section 5.3.5.1). *Synthesis_agents* can be viewed as belonging to the class of agents termed ‘interface agents’ [Nwana 96]. Interface agents are agents that co-operate with a user in achieving a particular task. As the name suggests, *synthesis_agents* aid the modeller with the task of synthesising composite models. It incorporates knowledge about the different model

types it supports and their various interfaces. *Synthesis_agents* interact with the modeller to help select and couple appropriate models to create composite models.

Figure [5.6] illustrates the architecture of a *synthesis_agent*. The agent consists of two parts - a static component termed the agent delivery facilitator that interacts with a client and facilitates the transport of the agent, and a mobile component functional unit that is transported to the client and performs the actual function of the agent. The functional unit consists of the following parts:

1. **Agent control and interface:** This provides an interface through which the modeller interacts with the agent. In addition it is responsible for controlling the operation of the agent.
2. **Model repository:** It holds the information required for synthesising composite models. The model interaction hierarchies, entity classes, and model instances are stored here.
3. **Model-Interaction type selection method:** This method guides the user in selecting the appropriate *Model-Interaction* type for the component model. The information in the *Model_Interaction* hierarchy is then used to determine the other *Model_Interaction* types that can be coupled with it.
4. **Model instance selection method:** For every *Model-Interaction* class supported by the agent the model instance selection method provides a mechanism to guide the user in selecting appropriate model instances for the modelling task.

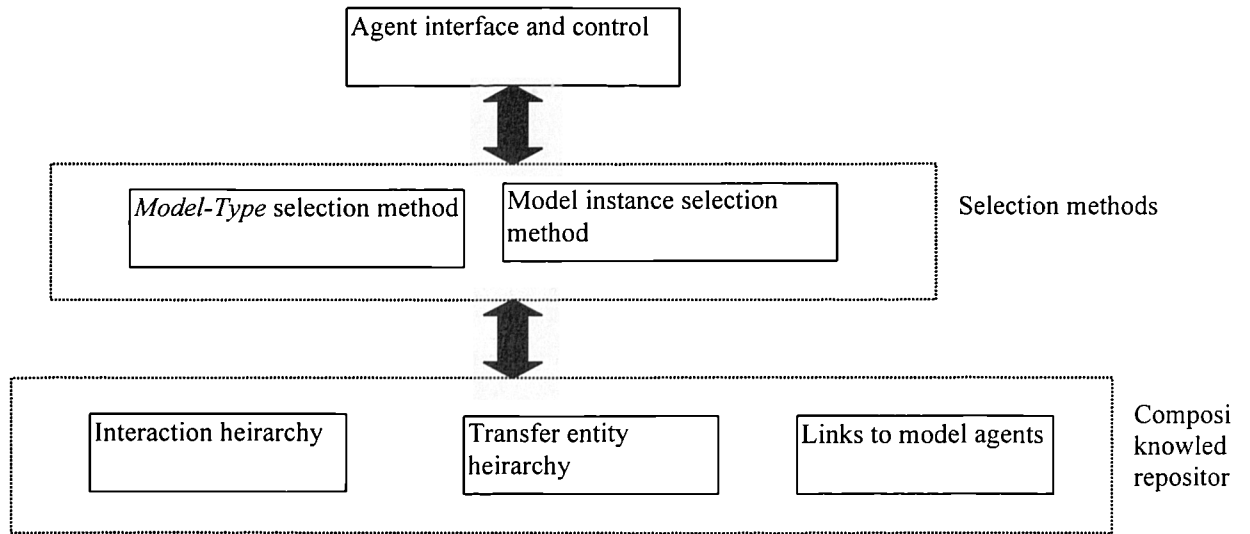


Figure 5.6 *synthesis_agent* architecture

Selection/creation of a *synthesis_agent*

The objective of model building in an environment that supports ‘large scale modelling’ is to enable the model not just to be simulated in isolation but also to be able to interact with other models, and thus support composite modelling. Consequently, a newly created model needs to be associated with an appropriate *synthesis_agent* that incorporates the knowledge required for guiding and enabling composition. It may be that a *synthesis_agent* already exists that supports models of a particular system. For example, the newly created model could be one of a family of models supported by a pre-existing *synthesis_agent*. In such a case the *synthesis_agent* is updated to reflect the existence of the new model. This involves the following steps:

1. Associating the newly created model with a *Model-Interaction* class. This may be a pre-existing class, or the *Model-Interaction* hierarchy may need to be extended to

support the interactions of the new model.

2. Associating the various ports with appropriate transfer entities from the *Transfer-Entity* hierarchy. This may involve extending the *Transfer-Entity* hierarchy.
3. Updating the selection methods of the agents to allow the selection of the newly created model.
4. Updating the model base of the *synthesis_agent* to include the newly created *model_agent*.

In case a *synthesis_agent* that supports a newly created model does not exist before hand, a *synthesis_agent* then needs to be created. Rather than creating a new *synthesis_agent* from scratch, (creating new *Model-Type*, *Model-Interaction* hierarchy, *Transfer-Entity* hierarchy etc) it is useful to reuse compositional knowledge (taxonomic, topological, and coupling) existing in other pre-existing *synthesis_agents* that support similar model types. Thus, a search needs to be performed for pre-existing *synthesis_agents* that support the *Model-Type* that the model belongs to, and the knowledge embodied in them incorporated to aid composition.

For example, consider an automobile engine manufacturer who wishes to evaluate the potential of supplying engines to an automobile manufacturer. The supply chain is modelled by coupling a model of the automobile manufacturer and its suppliers with a model of the engine manufacturer. To create a *synthesis_agent*, the engine manufacturer searches for a *synthesis_agent* that supports models of engine manufacturing facilities that supply the automobile manufacturer. This new *synthesis_agent* then reuses the model interaction and coupling hierarchies, possibly extending it, thus allowing the newly created model to be part of previously composed composites.

If a previously created *synthesis_agent* that satisfies the requirements of a new model cannot be found, then *Model-Interaction* and *Transfer-Entity* hierarchies along with a *synthesis_agent* need to be created by the model builder.

5.3.4 Composite model building

Composite model synthesis can be guided by using the knowledge (topological, taxonomic and coupling) present in *synthesis_agents*. The composite builder illustrated in Figure [5.7] employs a blackboard based architecture. The idea of blackboard based systems was first used in the development of HEARSAY-2, a speech understanding program [Erman 80]. Blackboard based frameworks are useful in the context of systems that use distributed sources of knowledge to arrive at a solution. Blackboard systems consist of:

- A set of independent modules that incorporate various domain-specific knowledge sources.
- A blackboard that acts as a shared structure enabling the various knowledge sources to communicate with one another.
- A control system that sequences the actions of the various knowledge sources on the blackboard.

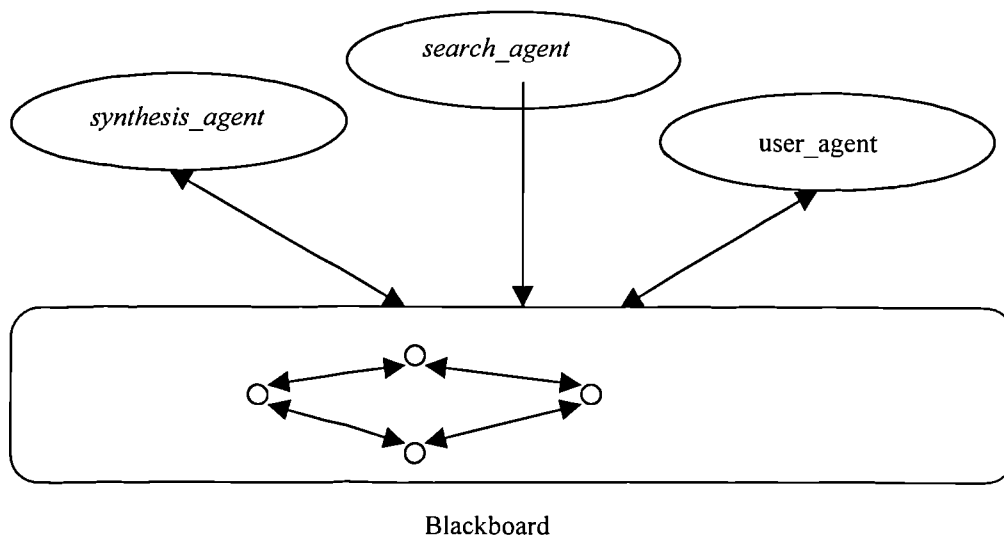


Figure 5.7 Blackboard architecture of composite modeller

In the composite modeller the knowledge sources are:

1. *synthesis_agents* which incorporate knowledge about the various simulation models they support. The *Model-Interaction* hierarchy contains knowledge about the topology of potential composite networks, the *Transfer-Entity* hierarchy contains knowledge about the coupling of various models, and the model selection methods are used to select suitable models for the composite.
2. A *search_agent* which is responsible for locating appropriate *synthesis_agents* that support the required component models of the composite model.
3. A *user_agent*, that essentially is a user (composite modeller) interacting via a graphical user interface (GUI).

The *user_agent* also acts as the blackboard controller, and co-ordinates composite model

building by activating various agents and making appropriate selections based on the information provided by the knowledge sources.

The blackboard provides a structure that is manipulated by the various knowledge sources (synthesis_agents, search_agent, user_agent) to create a composite graph that describes the various component models and the manner in which they are coupled to one another to create a composite model.

The agents (synthesis, search, user) perform operations on the blackboard by creating, deleting, and manipulating two entities, viz. nodes and edges, to create a composition graph that describes a composite model by specifying all the component models (nodes) and the manner in which they are interconnected (edges).

Nodes are described by a five-tuple structure:

Nodes $\langle \mathbf{L}, \mathbf{T}, \mathbf{S}, \mathbf{Mi}, \mathbf{M} \rangle$

Where,

L is a label that uniquely defines the node,

T is a *Model-Type* that the node represents,

S is a *synthesis_agent* that is of type **T**,

Mi is a *Model-Interaction* class that is a subclass of **T**,

M is a model that belongs to class **Mi**.

For example (see Figure [5.8]), consider a node that represents a CPU manufacturer then **T** would be a *Model-Type*, say for instance *CPU_MANF*, that represents CPU manufacturers. **S** is a *synthesis_agent* that represents models that belong to *Model-Type* **T**. In this case, if say, models of XYZ's (a hypothetical company) CPU manufacturing facilities are of interest, then **S** would be a *synthesis_agent* that represents a collection of models that model the manufacture of XYZ CPUs. The number of edges (ports) that a node has and the other nodes that these edges connect with is given by the *Model-Interaction* class **Mi**. In other words **Mi** incorporates structural information about the composite network. In the above example if it is of interest to model the relationship between a CPU manufacturer and a motherboard manufacturer, **Mi** would be a model interaction class that consists of a single port which connects with a node of type motherboard manufacturer. Knowledge about the behaviour of the node is given by **M** the model instance. Again continuing with the example, if it is required to take into account machine breakdown, then **M** will be a model instance that belongs to class **Mi** and incorporates breakdown of machines in the model of the CPU manufacturing

process.

Edges are represented on the blackboard by a two tuple structure. An edge that connects node i to node j is represented as follows:

EDGE $_{i,j} \langle \mathbf{port}_i \mathbf{port}_j \rangle$

where

port_i is an ordered pair [**inputstream outputstream**] such that

Inputstream is a set of *Transfer-Entities* that represents the input required by node i and **outputstream** is a set of *Transfer-Entity* that represent output generated by node i .

An edge **edge_{ij}** is valid if and only if for all inputstream and outputstream Transfer-Entities

Port_i . inputstream \subseteq **port_j . outputstream** and

Port_j . inputstream \subseteq **port_i . outputstream**

Thus, a valid edge guarantees that a model connected to one end of an edge gets at least all the *Transfer-Entities* it requires as input and conversely, outputs at least all *Transfer-Entities* required by the model at the other end of the edge.

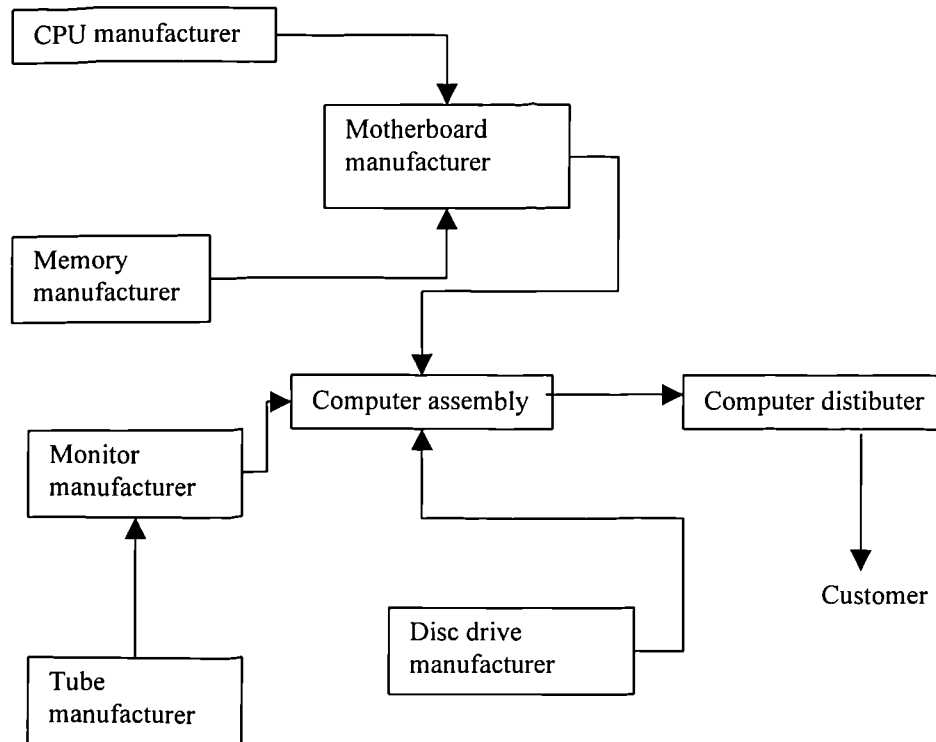


Figure 5.8 Supply chain of a computer manufacturer

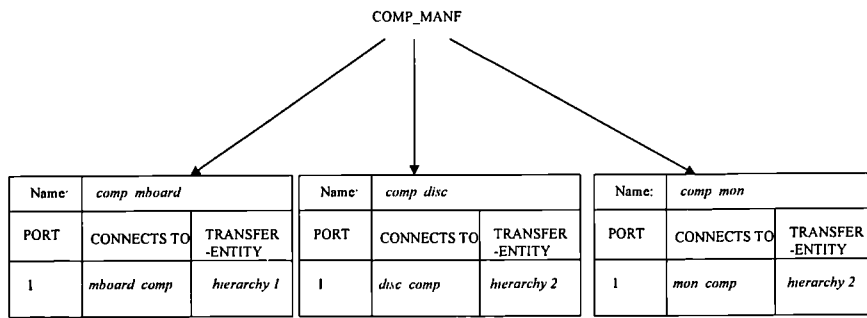
In order to explain the synthesis of a composite model an example is presented. Figure [5.8] illustrates a supply chain of a computer manufacturer. If, say, it is required to study the relationship between the computer, motherboard and CPU manufacturers. The process of synthesis is described in the following steps.

STEP 1: The synthesis process can begin with any of the component models of the composite model, although starting with the most important component may help the synthesis process. In this instance, it is assumed that synthesis begins using models of the computer manufacturer. The user begins by creating a node on the blackboard which

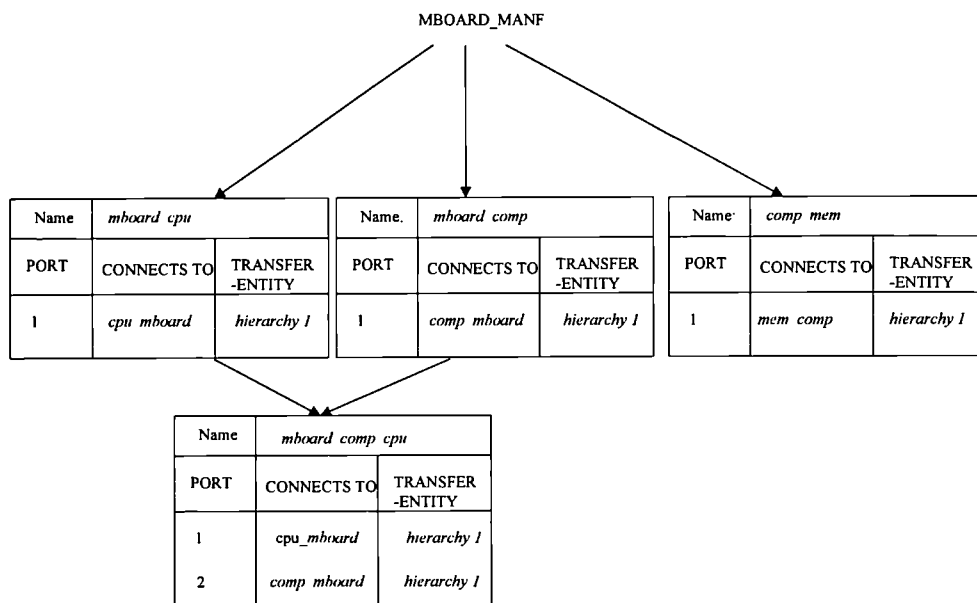
is given a label N_1 and assigned type COMP_MANF.

STEP 2: Next the user uses the search agent to locate a suitable *synthesis_agent*, of type COMP_MANF, that supports models of the computer manufacturer in question. The *synthesis_agent* is then assigned to the node, N_1 , and evoked to assign a *Model-Interaction* class to the node. As the user is interested in models that interact with a motherboard manufacturer, the user selects *Model-Interaction* class *comp_mboard* (see Figure [5.9 a]). As models of *Model-Interaction* class *comp_mboard* interact with models of motherboard manufacturers, the *synthesis_agent* creates a new node, N_2 , and assigns it with type MBOARD_MANF and *Model-Interaction* class *mboard_comp*. An edge is then created to connect nodes N_1 and N_2 . This edge is initially left unassigned as the type of transfer entities used by the model instance for the interaction is not known at this stage.

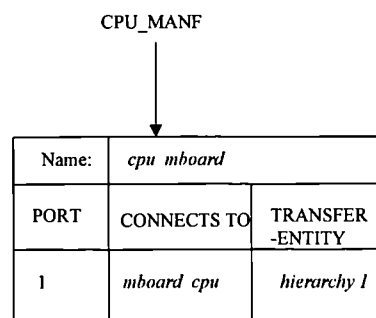
STEP 3: The user next uses the search agent to locate a *synthesis_agent* that supports models of the motherboard manufacturer, and assigns it to node N_2 , the MBOARD_MANF node. The node has already been assigned to *Model-Interaction* class *mboard_comp* in STEP 2 . However, as the user wishes to model the interaction between the motherboard manufacturer and the CPU manufacturer, the node needs to be assigned to a sub-class of *mboard_comp* that interacts with a CPU manufacturer. The node is thus assigned to *Model-Interaction* class *mboard_comp_cpu* (see Figure [5.9b]), which is a



a)



b)



c)

Figure 5.9 MI hierarchy of a. Computer assembler, b. Motherboard manufacturer, c. CPU manufacturer.

sub-class of *mboard_comp*. The *synthesis_agent* then uses the information in the *Model-Interaction* hierarchy to create a new node, N_3 and assigns it to type *CPU_MANF* and *Model-Interaction* class *cpu_mboard*. An edge is then created to connect node N_2 to node N_3 .

STEP 4: So far a composite of *Model-Interaction* classes has been created. The next step is to assign specific model instances to each of these classes to create a composite model. This is accomplished by evoking the model selection methods supported by the *synthesis_agents* in each of the nodes N_1, N_2, N_3 .

STEP 5: The next step is to verify that the model components in the composite will interact meaningfully, i.e. that they use compatible transfer entity classes for interactions. The edges are assigned the input and output *Transfer-Entity* classes of their respective model instances. All the edges are then verified to make sure that interactions are valid (See earlier definition of valid edge), thus guaranteeing the models interact meaningfully.

The synthesis process is now complete as all the nodes in the composite have been assigned with model instances and connected suitably by edges.

5.3.5 Simulation of the composite model

So far, how a composite model can be synthesised from a set of component models has been discussed. Next, the manner in which such a composite model can be simulated is presented. Two issues need to be addressed in order to develop a simulation mechanism. Firstly, the component models due to the distributed nature of modelling in HerMIS may

be stored at various locations, thus the models need to be encapsulated as mobile components that can be transported over the internet. Secondly, a distributed computer infrastructure needs to be created to allow the mapping of component models and their subsequent simulation on a network of computer workstations.

5.3.5.1 Model encapsulation and *model_agents*

Model_agents provide a mechanism for component models to be encapsulated and allow a standard interface to enable interaction with other components in the composite. *Model_agents* are based on the ideas of agent based software engineering [Genesereth 94]. The key concept behind agent based software development is to create software programs as autonomous agents, rather than stand alone programs, that interoperate with one another providing services and sharing information, and thereby solving problems that cannot be solved alone.

Applying the ideas of agent based software engineering to model building allows the modelling framework to support composites constructed from heterogeneous models. Models in a composite may be created using any combination of tools and formalisms as long as they act as agents and employ a mutually agreed communication protocol, termed in agent parlance as an agent communication language (ACL).

As in the case of *synthesis_agent*, the *model_agent* is supplied by an agent server. The server is responsible for accepting requests for the *model_agent* and transferring the agent to the appropriate destination (processing node on which the *model_agent* is simulated). The *model_agent* consists of four components:

1. **Simulator:** It performs the actual simulation of the model. It maintains all the entities in the model, their state, event list, and an executive to schedule events and manage the simulation process.
2. **Model:** It contains the model representation that the simulator simulates. This is akin to a ‘model file’ in a traditional simulation environment.
3. **Agent interface:** The interface provides a means for other entities involved in the simulation to interact with the *model_agent*. The interface provides a mechanism to (1) send and receive event messages, and (2) send and receive control messages for co-ordinating the simulation.
4. **Agent control:** This component is responsible for the overall behaviour of the agent. It uses the messages received through the interface to control the execution of the simulation. It interacts with the simulator executive for this purpose. Event messages are introduced and removed from the simulator.

The architecture of the *model_agent* described above is not suitable when using simulations based on tools that do not support the agent based programming paradigm. As the majority of simulation tools are static pieces of software and do not support mobile operation, an alternate agent architecture is required to support legacy simulations. One option is to create a ‘wrapper’ that encapsulates the simulator to allow its ‘agentification’. Alternatively, the simulator can be separated from the rest of the *model_agent* and interaction between the two can occur through a client-server relationship.

5.3.5.2 Distributed computer infrastructure

Figure [5.10] illustrates the architecture of the simulation mechanism. Simulation of a composite model is performed by mapping the component models onto a network of computer workstations.

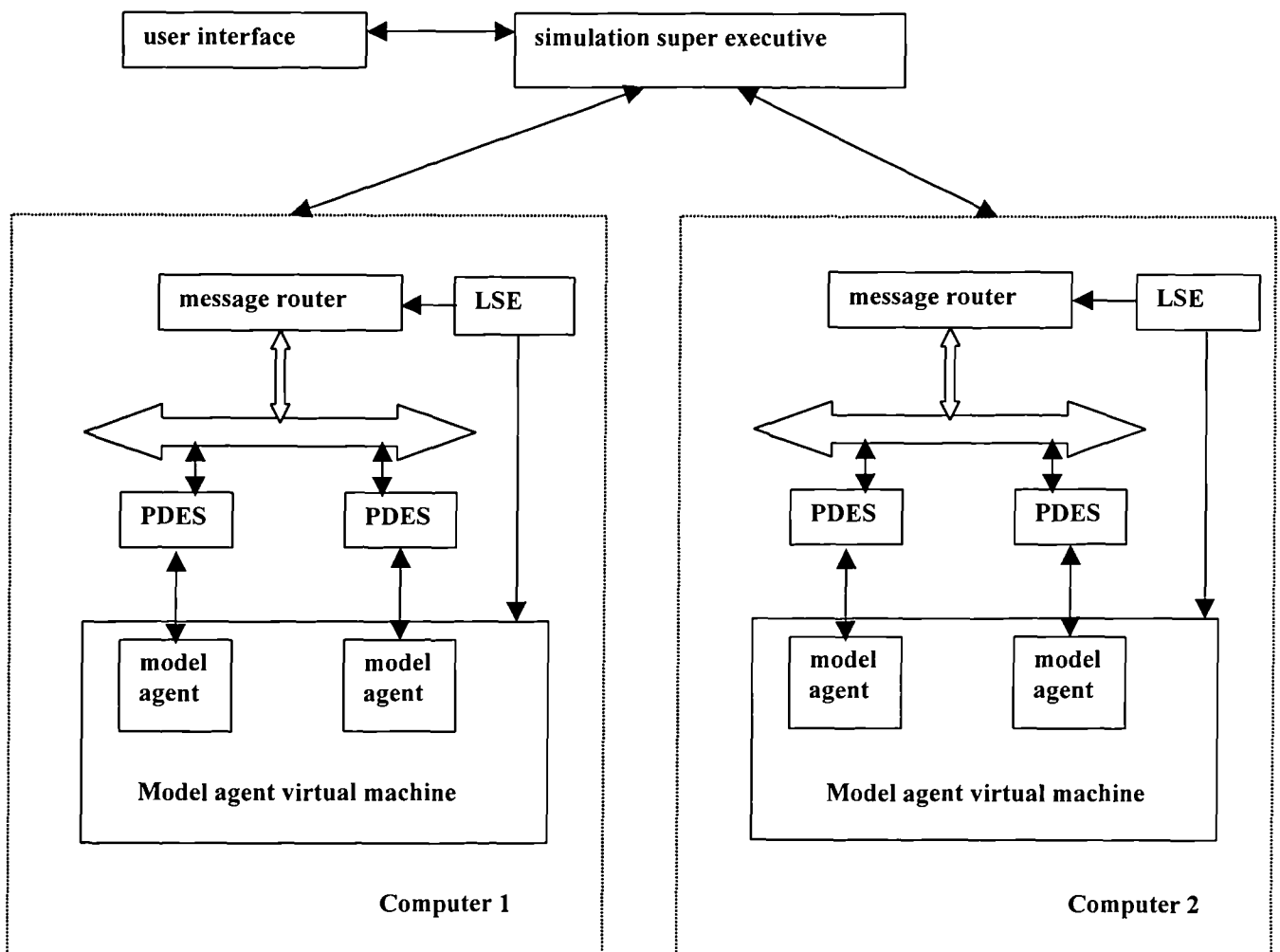


Figure 5.10 Architecture of simulation mechanism

The component models, for instance, could be mapped on to a single computer for simulation, or alternatively onto a number of workstations. The simulation mechanism consists of a centralised facility and a number of distributed facilities. The centralised facility consists of a user interface and the simulation super executive (SSE), while each

of the distributed facilities includes a local simulation executive (LSE), a message router (MR), a model agent virtual machine (MAVM), and a PDES (parallel discrete event simulation) controller. The simulation super-executive co-operates with local simulation executives to control and co-ordinate the set-up and execution of composite models. The local simulation executive in turn uses the message router, MAVM, and the PDES controller to perform their functions.

5.3.5.3 Simulation super executive

The simulation super executive (SSE) is responsible for managing the simulation process of the composite model. A single SSE is in operation for each invocation of HerMIS. The SSE performs its activities by interacting with the user interface and the various local simulation executives (LSE).

The user interface is an independent process that can be hosted on a separate computer. The separation of the user interface from the SSE has the advantage of allowing a user (model builder) with minimal computational resources to simulate a composite model over a distributed network of workstations. Further, by allowing multiple user interfaces to interact with a single SSE, a number of users can participate in the simulation of a composite model.

The SSE and the user interface share a client-server relationship. The user interface acts as a client by initiating the SSE to perform the simulation of a composite model and the SSE in turn acts as the server that performs the simulation and provides the results of the simulation to the user interface. The simulation process begins by the user interface

specifying the composition graph (the creation of which was discussed in the preceding section) of a composite model that is to be simulated. Each of the component models, which are represented by nodes in the composition graph, is then assigned to a local simulation executive (LSE). Each workstation on the network is controlled by a LSE. If, for example, all the component models are to be simulated on a single workstation then all the models will be assigned to the same LSE. For each component model in the composite, the SSE sends a component configuration packet (CCP) to the LSE that the model is mapped onto. The CCP contains three fields:

1. **Label:** A unique label by which each component model in a composite can be identified. The label may be the same as the label in the composition graph or assigned something different.
2. **Model agent:** A link to the component model agent server that supplies the appropriate model agent for the component model.
3. **Interconnections:** A list of labels (component models) with which the component model interacts, and a link to the LSE that supports these component models.

This information is used by the various LSEs to instantiate a composite model on the network of workstations. The SSE in collaboration with the user creates an experimental frame for the simulation of the composite model. The experimental frame consists of the initialisation parameters, the various performance measures to be derived from the simulation, and conditions for termination. A template of the experimental frame of each of the component models is provided by the component model builder to the composite model builder. The experimental frame of the composite is linked to all the individual

component experimental frames. The experimental frame of the component models are then sent to the appropriate LSE. The SSE then signals all the LSEs to start the simulation. Once the simulation is terminated, all the LSEs return the experimental frames with values of performance measures generated by the simulation run to the SSE. The SSE combines these results, updates the experimental frame of the composite, and uses the user interface to present results of the simulation run.

5.3.5.4 Local simulation executive

The Local simulation executive (LSE) is the principal component of the distributed facility of HerMIS. Each simulation node (workstation) has a LSE assigned to it. The LSE manages the resources available on the workstation for simulation. The LSE consists of a Simulation Controller and a Communications Manager. The Simulation Controller is responsible for initialising and co-ordinating the various resources available on the simulation node. The Communications Manager on the other hand is responsible for handling the message passing between the various component models of the composite. The SSE initiates actions on the LSE by sending it a component configuration packet that describes all the component models that are simulated by the LSE. The LSE responds by requesting the PDES controller to assign a PDES engine to each of the model agents in the CCP. In addition, the LSE assigns a set of ports, one for each PDES engine – model agent pair to enable the exchange of messages between the model agents. The communications manager manages these ports by using information present in the CCP (interconnections field). As mentioned earlier, the interconnections field details the coupling between the component models in the composite. The Communications Manager incorporates two mechanisms for communication, viz. local

communication mechanism and network communication mechanism. The local communication mechanism is used in communications between component models simulated on the same simulation node, while component models on different simulation nodes use the network communication mechanism to communicate.

5.3.5.5 PDES controller

The PDES controller implements the PDES algorithm that is used in the simulation of the composite model. The function of the PDES controller is twofold. Firstly, to provide a mechanism to simulate the composite model as an aggregate; it does this by coordinating the execution of events across all the component models. Secondly, to try and exploit possible parallelism present by virtue of the distributed simulation of the component models.

The PDES controller manages the simulation process by associating each of the component models with a PDES engine. All interactions (messages) between two component models are routed via its PDES engine. The PDES engine and its associated *model_agent* (component model) share a client-server relationship. The PDES engine acts as a message server that supplies messages, which represent interactions, to the *model_agent* such that it can guarantee that the model agent does not violate the causal relationship it shares with the other component models in the composite. The details of the workings of the PDES controller are deferred to the next chapter (Chapter 6), where a PDES algorithm for the message controller is presented.

5.4 Conclusion

In this chapter a composite modelling framework that satisfies two of the three requirements, viz. composite modelling and distributed modelling has been presented. In the next chapter (Chapter 6) the framework is extended by including a parallel simulation engine, the third requirement for the modelling framework. A set of requirements for a PDES algorithm is developed and a new algorithm is presented.

CHAPTER 6

PDES controller for HerMIS

In the previous chapter the HerMIS modelling framework was presented. The framework specified how composite models were synthesised from component models and a distributed computing platform on which the composite model is to be simulated was developed. The various components of the simulation infrastructure were identified and a procedure for mapping the *model_agents* on to the processing nodes was defined. However, a key aspect of the infrastructure, viz. the mechanism for synchronising the execution of events, was deferred. In this chapter this issue is tackled and a PDES mechanism for HerMIS is developed.

In chapter four a number of PDES techniques were reviewed. From this discussion it is apparent that, just as using a single modelling formalism or methodology is not sufficient for all modelling needs, the choice of a PDES algorithm needs to be made based on the model and objectives of the simulation. For example, at one extreme, run-times of the simulation may not be a consideration at all and all that is required is a mechanism to execute the composite model. At the other extreme is a case where simulation time is of paramount importance. In the former case, synchronisation of events in the component models may be done by employing a global event list on to which all models in the composite schedule their events. The global event list can then be used to determine the

next event/events to be executed. Such a scheme would exploit minimal parallelism inherent in the models, as only events that occur at the same time would be executed in parallel. In the case of the latter, a sophisticated PDES algorithm that is optimised to maximise the degree of parallelism exploited is needed. This may require the specialist knowledge of a user who is capable of fine tuning the algorithm to suit the model.

Here a PDES scheme is developed that is somewhere between the two extremes mentioned above. A few of the operating characteristics of supply chains are taken into account and exploited, to develop an 'efficient' PDES scheme that is transparent to the user.

The Chandy- Misra framework [Chandy 79] has served as a starting point in a number of research endeavours in the past and is used here as well, with a view to understand the requirements for a PDES technique for simulation of the composite supply chain model .

An example of a framework based on the LP model [Chandy 79] is given below. The notion of a physical and modelled systems is employed here. The physical system represents the real life system that needs to be simulated. The modelled system represents the model that is used in the simulation of the physical system. In the context of this work the physical system is a supply chain and the modelled system is the LP network.

Every model in the supply chain is simulated by an LP. Communication between models are exclusively through a message passing mechanism. Each messages includes two

fields - a timestamp to represent the time at which the sending model sent the message, and a message field to describe the information to be sent from one LP to another. All LPs use a message queue to store the incoming messages. Incoming messages are processed by the LP in increasing order of timestamps.

Typically, the physical system will be composed of a network of production and distribution stages. Stages share supplier-customer relationships with one another. A supplier stage receives orders from a customer stage and based on these orders manufactures, or uses from inventory, products which are then shipped to the customer stage.

The modelled system is mapped on to a distributed simulator in order to simulate the physical system. The distributed simulator consists of a number of processors connected together via a message passing mechanism. The only method of exchanging information is via the message passing mechanism. A many to one relationship exists between LPs and processors. Each processor executes one or more LPs.

In the following two sections the suitability of conservative and optimistic PDES schemes in the simulation of the supply chain model is looked at.

6.1 The conservative approach

Using the LP framework described above in conjunction with a conservative PDES algorithm (Chandy and Misra) [Chandy 79], a number of experiments employing

computer and paper-based models were conducted with a view to study the suitability of conservative PDES schemes in simulating the supply chain model.

The notion of event-coupling ratio is introduced to help in describing a characteristic of manufacturing supply chain systems and, consequently, the effect it has on the performance of PDES algorithms. Event-coupling ratio is the ratio between the number of external events to the number of internal events generated by an LP during a typical simulation run. External events are events that are scheduled by one model on to another by passing a message, while an internal event is one that is scheduled by a model to be executed by itself. For example, external events could represent the placement of an order or a shipment of a part, from one model to another. On the other hand, the scheduling of the start of a machining cycle is an example of an internal event. A key characteristic of the entities in a manufacturing supply chain is low event coupling ratio. Typically, orders to suppliers and shipment of parts to customers occur a few times a day. Most of the time spent by the simulator is in scheduling internal events that satisfy the order. To investigate the effect of low event-coupling in terms of performance, the factors affecting performance of PDES algorithms is looked at. In order to maximise parallelism and, hence, performance, it is important that LPs spend a large proportion of time performing operations relevant to the simulation, rather than blocking (waiting for messages) to guarantee the avoidance of causal errors. Secondly, the overhead due to message passing needs to be minimised.

As mentioned earlier in chapter four conservative algorithms process events only when they can guarantee that causality errors will not occur in doing so. This is achieved by

blocking (waiting) until a message is received on all inputs to the LP. As each LP processes events in increasing timestamp order, LPs can use the arrival of a message to guarantee that a messages will not be sent later with a time stamp lower than the message just arrived. In low event-coupling systems, the dependence between LPs is limited and consequently messages are sent infrequently. As a result, LPs working under a conservative PDES scheme will spend a large proportion of time waiting for messages before the simulation can be progressed.

Another property of the supply chain modelling paradigm is the existence of multiple loops. Relationships between the entities of the supply chain tend to be cyclical (Figure [6.1]), due to supplier/customer interaction. As described earlier in section 4.2.1.1 and illustrated by Figure [4.3], conservative PDES schemes operating in LP networks that consist of loops can deadlock.

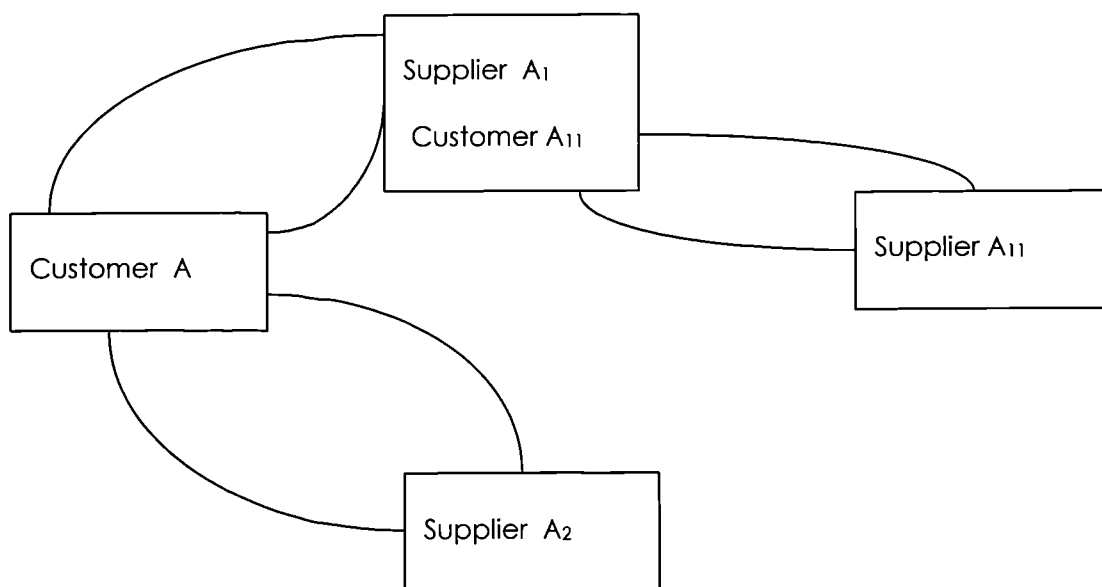


Figure 6.1 Cyclic relationship in supply chain models

Conservative PDES techniques either use null messages or a deadlock detection and recovery scheme to tackle the potential of deadlocking. In the case of null message conservative protocols, the time between actual messages is spent exchanging null messages, thus significantly increasing the communication overhead, and reducing the benefit derived from concurrent execution of models. The deadlock detection and recovery schemes also place a processing overhead on the LP as these techniques let the LPs deadlock and then try and resolve the deadlock.

Thus, applying the Chandy-Misra conservative algorithms led to the following observations:

- A large proportion of time was spent blocking due to low-event coupling ratio of supply chain models.
- The cyclic nature of supplier-customer relationships result in communication overhead in the form of null messages or, in the case of deadlock detection and avoidance schemes, a processing overhead is incurred as time is spent in detecting and recovering from the deadlock.

In the next section the appropriateness of optimistic PDES algorithms for the simulation of supply chain models is studied with a view to observe if they overcome the pitfalls suffered by the conservative approach.

6.2 Optimistic approach

Optimistic PDES algorithms, in contrast to conservative approaches, do not postpone the execution of events until the avoidance of causality errors can be guaranteed. Events are processed until causality errors are detected. Causal errors are then undone by rolling back. There are two important consequences of this mode of operation. Firstly, as LPs do not block, the low event-coupling ratio does not affect performance. Secondly, the question of deadlock does not arise as a prerequisite for deadlock to occur is for LPs to block until appropriate messages are received. However, as LPs in an optimistic scheme do not block, as a result, the cyclic nature of supplier-customer relationships does not affect the performance of optimistic PDES systems. Thus, the very nature of optimistic systems avoids the pitfalls that conservative systems suffer from.

Optimistic PDES algorithms, however, have their share of pitfalls. Performance of simulators based on the optimistic approach is affected by two factors. Firstly, the overhead caused due to rollbacks and secondly, the overhead due to message passing.

As in the case of the conservative approach, paper and computer based models were created to investigate the rollback mechanism and its effect on performance. The experiments with these models led to two important observations.

- In a number of cases wasted rollbacks occurred i.e. the rollback was not necessary to guarantee a correct state in the simulation.
- A potential for flooding of antimessages existed.

Rollbacks in optimistic simulators occur when an event message arrives at the simulator with a timestamp less than the simulation time. This is necessary as messages need to be executed in strictly increasing order of timestamps. However, consider a case where incoming messages (external events) are not processed by the simulation as and when they arrive but are batched together. For example, consider the case of an event message that represents an order for a component. Typically in a production stage, orders are not processed as and when they arrive but are batched together and processed at a few fixed points during the day. Thus, a rollback only needs to occur when the timestamp of a order message just received is less than the time at which the simulation began to process the last batch of orders.

An example is provided to better illustrate such ‘wasted rollbacks’. Consider a model M at simulation time t_{clk} . M consists of one input queue I that receives orders for the manufacture of a component C , and one output queue O that outputs the component C . Let a message $\langle o_1 t_1 \rangle$ which represents an order with timestamp t_1 arrive at I such that, $t_1 < t_{clk}$. As the timestamp of the message is less than the simulation time, the simulation will rollback to t_1 . Let another message $\langle o_2 t_2 \rangle$ such that $t_2 < t_1$ arrive next. The simulation will again rollback to simulation time t_2 . In the above example, if orders were last processed by M at time t_{batch} , and say $t_{batch} < t_2$, then the rollbacks that occurred when orders o_1 and o_2 arrived were not necessary, as they do not affect the final outcome. Further, the order of arrival of o_1 and o_2 is not important as long as they both arrive before the next t_{batch} , i.e. before the simulator processes the next batch of orders.

During a rollback an LP loads a previously saved state and dispatches antimessages to undo erroneous event messages. Jefferson [Jefferson 85] argues that in practice most applications operate in a pattern where each external event results in the generation of a few internal events and only a single external event. Consequently, the number of antimessages sent is equal to the number of external events rolled back. It is argued here that this is not necessarily true in the case of supply chain models. Consider the case of simulating a supply chain. Let each LP model a single entity of a supply chain. Messages between LPs would include orders placed by customer LPs and shipment of parts by supplier LPs. Typically, when a LP receives an order it will dispatch a number of orders to its supplier LPs. Thus, when a LP receives an external event (order), not one but a number of external events are generated by the LP. Consequently, if a rollback is required at some future time, then, a number of antimessages need to be sent to cancel all the order messages to supplier LPs. Further, the LPs receiving the antimessages may also need to rollback, resulting in a flooding of antimessages and, hence, degradation of performance.

Based on the experiments conducted using the conservative and optimistic models, the following requirements were formulated for the PDES algorithm.

- An algorithm based on the optimistic approach.
- Requires to take into consideration batched external events.
- Provide an alternative to the antimessage mechanism to prevent the occurrence of message flooding.

In the following sections such an algorithm is presented.

6.3 PDES paradigm

The issue of batched events is addressed first. In time warp, the standard implementation of an optimistic simulator, the input and output queues are part of the simulator. The timestamp of an incoming event is compared with the simulation time (clock) and a rollback is issued if the timestamp of the incoming message is found to be less than the simulation time. A modification to the LP is proposed here with the aim of avoiding wasted rollbacks. The input queues are separated from the rest of the simulator and share a client server relationship. The input queue acts as a message server to the simulator. Each input queue has a clock associated with it which stores the simulation time at which the last request for messages was made by the simulator. During the course of the simulation the simulator requests messages from the input queue and the input queue responds by serving messages. All messages that have timestamps greater than the input queue clock (initially set to zero) and less than or equal to the simulation clock are sent to the simulator. The input queue clock is then incremented to the value of the simulation clock.

Rollbacks occur when an incoming message to an input queue has a time stamp smaller than the input queue clock, i.e. only when it is certain that an input message would have been processed with other messages if it had arrived earlier. A sequential list $(t_b, t_{b-1}, t_{b-2}, \dots, t_0)$ of all previous queue clock values is maintained to determine the point of

rollback. The timestamp of the message that caused the rollback, say t_m , is compared with the list of queue clock values, in an increasing order of queue clock values until a queue clock value, say t_{b-n} , is found that is greater than or equal to t_m , the timestamp of the rollback message. The simulator is then rolled back to t_{b-n} .

An additional benefit of the modified LP has to do with the mechanism of saving state. Saving state at every point in the simulation can result in high cost, both in terms of space (memory) and time (execution speed). Thus, in practice, checkpointing or periodic saving of state is often employed. Infrequent saving of state, however, could mean excessive rollback distance. This results in LPs rolling back further than required and, consequently, a lot of the computations have to be performed again, resulting in inefficient execution. On the other hand, frequently saving state could undo the benefits of checkpointing. Thus, the frequency of state saving is an important parameter in terms of performance. In the case of clocked queue implementation, the queue clock list can be used to provide the checkpointing intervals. From the above discussion (previous paragraph) on rollbacks it can be seen that the simulator always rolls back to a point in the queue clock list. As a result, it is sufficient to perform the state saving operation only when the queue clock is updated.

The requirements for a PDES algorithm presented earlier identified the need to develop an alternative to the antimessage mechanism. A novel technique using rollback counters is presented here to overcome the disadvantages of the antimessage mechanism.

Every LP keeps track of its rollback count which is the number of times it has rolled back since the start of the simulation. The rollback count is added to every outgoing message sent by the LP. When an LP rolls back its rollback counter is incremented and a control message is sent, with the current rollback count on all its output links. Subsequent messages include the new value of rollback count. The rollback count of a message coupled with the control messages (which act as markers in the input queue) can be used to detect invalid messages. The details of this scheme are presented in the following section.

6.3.1 Modified LP architecture

Figure [6.2] illustrates the structure of the modified LP. It consists of three functional units: message controller, simulator, and state saving mechanism. The message controller and simulator share a client server relationship. The message controller is responsible for sending and receiving messages between LPs and the simulator performs the actual simulation of the physical process. In the case of HerMIS the *model_agent* acts as the simulator. A state saving mechanism is used to enable the LP to rollback. The message controller acts as a message server to the simulator. It consists of input queues (clocked queues), one for each input link, and output queues, one for each output. During the simulation the simulator client may either request messages from the message controller or send messages to the output queue. In the event of a request for messages by the simulator, the message controller furnishes the appropriate messages to the simulator. The message controller receives messages on all the input links and saves the messages in the designated input queues in increasing order of time stamps. Timestamps and rollback counts of arriving messages are used to detect any violation of causal

relationship. Erroneous messages are deleted and in case a rollback is required, the message controller signals the state saving mechanism to undertake a rollback. A clock (queue clock) keeps track of how far the message controller has progressed in simulated time, i.e. how far into simulated time the message controller has served messages to the simulator.

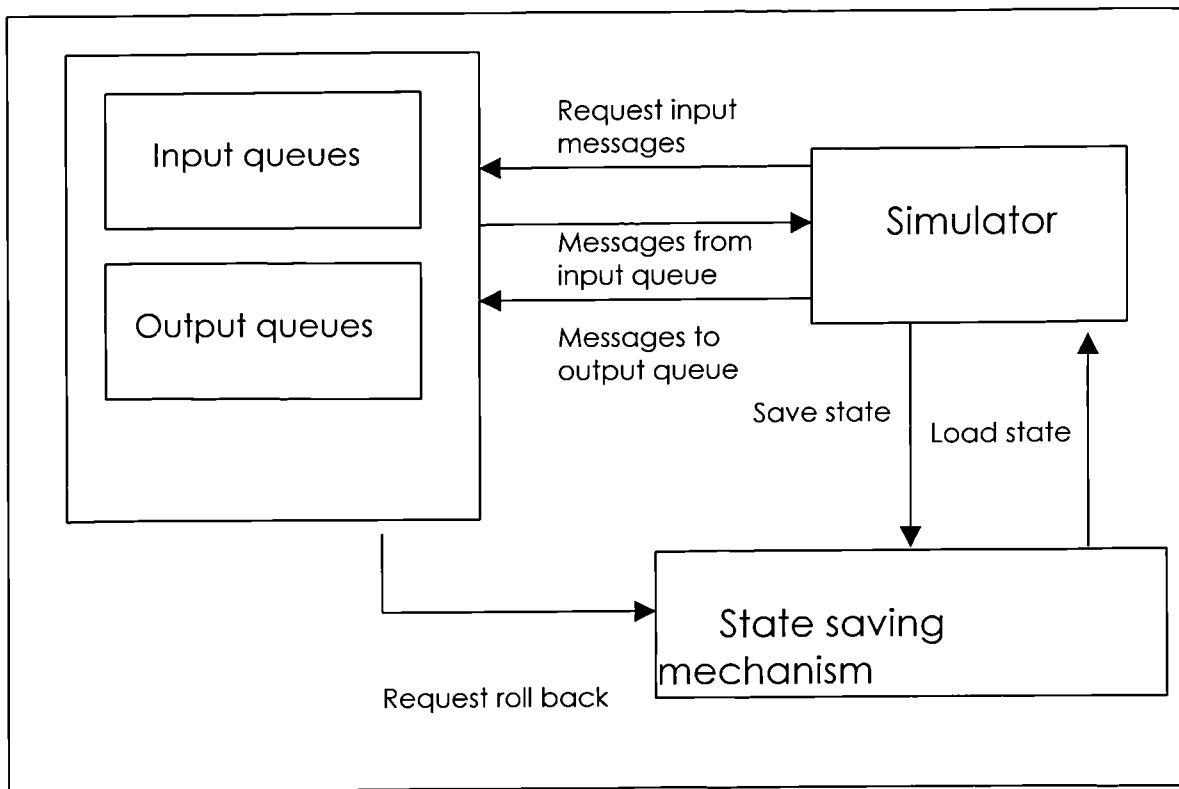


Figure 6.2 Modified LP architecture

6.3.2 PDES algorithm

The message controller performs two functions: (1) receiving messages sent by other LPs and (2) satisfying requests for messages by the simulator. The algorithms for both functions are presented below.

Receiving messages

The message controller constantly polls for messages. When a message is received the message controller needs to first determine if the message is a valid message, i.e. not the result of an erroneous computation by the transmitting LP, and secondly, if required, it needs to issue a rollback to the simulator. Message validity is performed by checking the rollback count of the incoming message. Every time an LP rolls back, it issues a rollback control message with an updated value of the rollback count on all its output links. All subsequent messages, until another rollback, sent by the LP includes this rollback count. By comparing the rollback count on a message with other messages one can determine if the message is the result of an erroneous computation or not. This is explained below

Two types of messages can be received by the message controller - normal message and rollback control message. Let us first consider the case of a normal message. When a message is received by the message controller the position i in the input queue where the message needs to be inserted is determined based on the time stamp of the message. The input queue stores messages in increasing order of time stamps. The rollback count of the message, $messg.rbc$, is compared with the rollback count of the message one before the position where the message is to be inserted in the input queue (position $i-1$). The objective is to check if there are any messages that arrived earlier (messages with smaller timestamps) and have a higher rollback count, indicating that the LP that sent the message has performed a rollback since. If $messg.rbc < messg_{i-1}.rbc$, the message is

erroneous, as the LP that transmitted the message has rolled back since it sent the message, and is not inserted in the input queue.

If the message is inserted in the queue, the next step is to check if the simulator needs to rollback. The timestamp of the message *messg.ts* is compared with the current clock value of the message controller *clk*, which indicates the last time at which the simulator requested messages from the queue. If $messg.ts < clk$, then the message should have been sent as part of an earlier request for messages by the simulator and the message controller issues a rollback. The point of rollback is determined by finding the highest value in the queue clock list that is greater than or equal to *messg.ts*.

Now let us consider if the incoming message is a rollback control message, say, *contmessg*. The position *i* in the queue based on *messg.ts* is determined and the control message is inserted. Rollback counts of all messages lying ahead, i.e. with greater than equal timestamps, of the inserted control message are compared with the rollback count of the control message to determine if they have a lower rollback count, thus indicating the messages were sent before the LP rolled back. If $messg.rbc < contmessg.rbc$ the message is deleted. The timestamp of the control message is compared with the clock of the message controller, if $messg.ts \leq clk$ a rollback is issued to the simulator. The algorithm for message processing is given below.

For normal messages

For every message m received

{

i = position in the input queue at which the message is to be inserted.

If $Q[i-1].rbc > m.rbc$ delete m

else

insert m at $Q[i]$

if $m.ts \leq clk$ issue a rollback

}

For control messages

For every control message cm

{

determine i such that

i = position in the input queue at which the message is to be inserted.

for all messages m in $Q[j]$ such that $j > i$

if $m.rbc < cm.rbc$ delete m

if $m.ts \leq clk$ issue a rollback

In case of rollback

when rollback required

{

Find smallest value of i such that

$t_{queueclk}[i] \leq m.ts$ where $t_{queueclk}$ is the queue clock list.

rollback to $t_{queueclk}[i]$

$rb++$ increment rollback count of LP by one.

broadcast a control message (t,rb) on all output links.

t = point at which the LP has rolled back, rb is the new rollback count.

}

Sending messages to simulator

Whenever the simulator requests messages, the message controller sends all the messages in the appropriate input queue with time stamps greater than the queue clock and less than equal to the simulator clock, to the simulator. The queue clock is then updated to the value of the simulator clock. In case the simulator outputs a message it is stored in the appropriate output queue and transmitted by the message controller.

The algorithm for handling message requests by the simulator to the message controller is given below. The simulator sends a request req which consists of two fields: $req.q$ (the label of the queue from which the simulator requests the messages and $req.t$ (the time the

simulator is currently in). The message controller returns a list of messages *messg* back to the simulator.

Wait until request by simulator req

send all messages messg in input queue req.q that satisfies

$$clk_{req.q} < messg.t \leq req.t$$

$$clk_{req.q} = req.t$$

6.4 Proof of correctness

Consider a physical system *PS* composed of *N* processes that communicate exclusively through message passing. *PS* is represented by a directed graph *G* consisting of *n* vertices $\{P_1 P_2 \dots P_n\}$ each representing a process in *PS*. An arc exists between P_i and P_j if process P_i communicates with P_j . In order to model *PS* a modelled system *MS* is composed such that, for every process P_i there exists an LP, LP_i , in *MS* that models the behaviour of that process. If a message passing link exists between P_i and P_j then a corresponding link exists between LP_i and LP_j in *MS*.

Let us assume that one is interested in simulating the operation of *PS* in the time interval $[0, Z]$. For some process *P* in *PS*, define $t_0 = 0$ and $t_n = Z$, i.e. the start and end of the simulation. Further, $[t_0 t_1 t_2 \dots t_n]$ represents the sequences of times at which the

process P processes incoming messages. Assume the sequence $[t_0 \ t_1 \ \dots \ t_n]$ to be monotonically increasing. Hence a positive constant ε exists such that:

$$t_{i+1} - t_i > \varepsilon \quad \text{for } i = 0 \text{ to } n-1$$

In practice all physical systems exhibit a delay in processing messages, hence the above requirement is not a restriction.

Let I_j , termed input set, contain all the messages that arrive at process P during the time interval $t_{j-1} < t \leq t_j$.

$$I_j = [\text{set of all input messages } m.t, \text{ such that } t_{j-1} < t \leq t_j. \text{ for all } j \ j= 1.....n]$$

Note in the case of $j=0$, $I_0 = \{\}$.

Similarly define O_j , termed output set, to contain all the messages that P transmits to other processes in the time interval $t_{j-1} < t \leq t_j$.

$$O_j = [\text{set of all output messages } m.t, \text{ such that } t_{j-1} < t \leq t_j. \text{ for all } j \ j= 1.....n]$$

Note in the case of $j=0$, $O_0 = \{\}$.

Let $I_p(j)$ and $O_p(j)$ define the message input and output history respectively, of process P .

$$I_p(j) = I_0 + I_1 + + I_j$$

$$O_p(j) = O_0 + O_1 + + O_j$$

Thus, $I_p(j)$ and $O_p(j)$ represent the sequence of messages received and transmitted respectively, by process P , until time t_j .

Assume a function F exists for a process such that:

$$O_p(j) = F(I_p(j-1)) \quad \text{where } j=1.....n$$

The state of process P at time t_j is represented by S_j . The state of a process depends on the input received by it, thus there exists a function G such that:

$$S_j = G(S_{j-1}, I_p(j-1)) \quad \text{where } j=1.....n$$

Define the corresponding terms in MS by using the lower case alphabet. For some LP, LP_i , in MS the following are defined.

Input set $i_j = t_{j-1} < t \leq t_j$. for $j = 1.....n$

Output set $o_j = t_{j-1} < t \leq t_j$. for $j = 1.....n$

Input history $i_p(j) = i_0 + i_1 + + i_j$

Output history $o_p(j) = o_0 + o_1 + + o_j$

An output function f exists, such that $o_p(j) = f(i_p(j-1))$ where $j=1.....n$

A state transition function g exists, such that $s_j = g(s_{j-1}, i_p(j-1))$ where $j=1.....n$

As the modelled system MS is based on an optimistic PDES algorithm, the causal relationship between messages cannot be guaranteed. Consequently, erroneous messages can exist in the message histories (input and output histories) of LPs. In addition control messages are included in message streams to detect erroneous messages.

Define input history $i_p(j)$ and output history $o_p(j)$ to be correct, i.e. $I_p(j) \equiv i_p(j)$ and $O_p(j) \equiv o_p(j)$, if all the messages they contain do not violate the following conditions:

1. All valid messages with time stamps $t_0 < t \leq t_j$ must be present.
2. A control message with a timestamp less than any valid message and a rollback count more than the message cannot exist.
3. If an invalid message exists with timestamp t and rollback r then a control message (t_c, r_c) must exist in the output stream such that $t_c \leq t$ and $r_c > r$.

Assume $s = \{\text{set of all states for some } LP_i \text{ in } MS\}$

In addition, define the following terms for an LP in MS:

Valid state: A valid state s_j is defined recursively as follows

s_0 is valid (i.e. the initial state of an LP is valid)

s_j is valid if $s_j = g(s_{j-1}, i_p(j-1))$ where $j=1, \dots, n$ and $i_p(j-1)$ is correct and

s_{j-1} is a valid state

Invalid state: A state encountered by an LP that is not valid is invalid and is denoted as s' .

The proof is structured as follows. It is first proved that if an LP is at some valid state s_j then given correct input history $i_p(j)$, it will transform to the next valid state s_{j+1} . This property is then used to prove that any LP in MS given correct input history $i_p(j)$ will generate correct output history $o_p(j)$. Finally, using the results of the previously proved theorems, the correctness of MS is proved.

Theorem 1: Assuming correct input history $i_p(j-1)$, if an LP is at some valid input state s_j then at some time in the simulation the LP can be guaranteed to be at the next valid state s_{j+1} .

Proof: Let us consider an LP at some valid input state s_j and at simulation time t_j . Now if i_j (input set) is complete, the LP will process i_j and transform to the next valid state s_{j+1} (by definition of valid state).

However if i_j is not complete or an erroneous message exists in i_j at simulation time t_j , then the LP will process i'_j (incomplete/incorrect) and transform to invalid state s'_{j+1} . Let us first consider the case of i_j being incomplete.

At some point in the future i_j will be complete as message delivery is guaranteed. The last message to complete i_j will cause the LP to rollback to s_j as all messages in i_j have timestamps in the interval $t_{j-1} < t \leq t_j$. The LP processes i_j and transforms from state s_j to s_{j+1} (NB: The LP before the rollback will be in simulation time $t_j \leq t \leq z$).

Now, consider if i_j consists of erroneous messages when the LP is at simulation time t_j . As the input is correct, from the definition of correct input it is known that a control message with a timestamp less than the erroneous message and a rollback count more than it, must exist in the input stream. From the algorithm it is known that every time a control message arrives it deletes all messages ahead of it (with timestamps greater than the timestamp of the control message) in the input queue with rollback counts smaller

than it. The last control message to arrive deletes the remaining erroneous messages in i_j and rollback to s_j as all messages in i_j have timestamps in the interval $t_{j-1} < t \leq t_j$. The LP processes i_j which is void of all erroneous computations and transforms to the next valid state s_{j+1} .

Lemma 1: The roll back count of an LP cannot decrease during the course of the simulation.

Proof: From the algorithm in the previous section note that the rollback count of an LP is initialised to zero at the start of the simulation. The only other operation performed on the rollback count is an addition operation every time a rollback occurs. Hence the rollback count of an LP cannot decrease during the course of the simulation.

Theorem 2: If an LP receives correct input during the course of a simulation $0 < t \leq z$, $i_p(t_n)$, it will generate correct output $o_p(t_n)$.

Proof: From the definitions given at the start of this section it is known that output generated by an LP must not violate the following conditions:

1. All valid output must be present.
2. A control message with a timestamp less than that of a valid message and a rollback count more than that of the message cannot exist.
3. If an invalid message exists with timestamp t and rollback r then a control message must exist in the output stream such that $t_c \leq t$ and $r_c > r$.

Next it is proven that given correct input, an LP does not violate each of the above conditions.

Condition 1

From Theorem 1 we know, given a correct input the LP will go through all valid states and as valid states generate valid output condition 1 is satisfied.

Condition 2

There are two ways an LP can send a control message with a timestamp less than that of a valid message.

1. The LP sends the control message first and then sends the message.
2. The LP sends the message first and at some point in the simulation rolls back to a time less than the timestamp of the message and transmits the control message.

From lemma 1, it is known that the rollback count cannot decrease during the course of the simulation. Thus, if a control message is sent before the normal message, it cannot have a greater rollback count. Its now proven that the second case is also impossible.

Let us assume that the LP is at valid state s_j at time t_j and generated output o_j . Now as s_j is valid, all inputs in the time interval $0 < t \leq t_j$ have arrived, hence the LP cannot roll back to t_j and behind. Consequently, a control message cannot be sent such that its timestamp is less than or equal to t_j after o_j has been transmitted.

Condition 3

Let us assume that the LP is at some invalid state s' and outputs an invalid message m with timestamp t and rollback count r . Then some t_j where $t_j < t$ must exist which is not complete. As it is assumed that correct input messages and message delivery is guaranteed, at some point t_j will be complete (see theorem 1) and the LP will rollback to t_j ; this will cause the rollback count to be incremented and the transmission of a control message at t_j with rollback count $r+1$, hence satisfying the condition.

Combining the proofs for condition 1,2, and 3 it can be guaranteed that given correct input an LP generates correct output.

Theorem 2 may seem to indicate the correctness of simulation of an arbitrarily connected network of LPs. However, there is a possibility that the outputs of LPs in a loop may all be incorrect. Theorem 3 proves that this is in fact not possible.

Let us define a simulation run $0 < t \leq z$ to be described by a set $R \{ [LP_1] [LP_2] \dots [LP_n] \}$, where $[LP_j]$ denotes the output history of LP_j during the course of the simulation ($0 < t \leq z$), i.e. $o_p(t_n)$. Define R_e to be such that the LP output sequences only includes the outputs generated by states s_0 to s_e where s_0 is the initial state and s_e the e^{th} valid state in the LP, i.e. $LP_j = o_p(e)$. Define R_e to be correct if for all j , LP_j is a correct output history. A simulation run R is correct if R_e is correct for all e , $0 < e \leq n$, where n is the final valid state or terminating state of the LP. This is now proved in the next theorem.

Theorem 3: For every simulation run R , R_e is correct for all e , $0 < e \leq n$.

Proof: Proof is by induction on e . For $e=0$ $R_e = \{ \}$ and, hence, the theorem holds trivially. Let us assume that for some $e>0$, R_{e-1} is correct. Now to complete the proof it needs to be shown that R_e is correct.

Consider LP_i , an LP in MS . Then, $i_p(e-1) \subseteq R_{e-1}$, as the input history at $e-1$ of LP_i is the output histories at $e-1$, of the LPs it receives input from. Now as R_e is correct all the outputs in R_e are correct as well. Hence $i_p(e-1)$ is correct. If $i_p(e-1)$ is correct then from theorem 2 it is known that $o_p(e)$ is correct. Continuing this argument for all LPs, it is concluded R_e is correct.

6.5 Discussion

In this chapter a modified optimistic PDES scheme was presented that is suitable for the distributed simulation of models which constitute a supply chain. An optimistic scheme was shown more suitable due to the loosely coupled nature of supply chain models. However, two potential drawbacks, viz. wasted rollbacks and antimessage flooding, in using a standard (time warp) optimistic simulator implementation, were identified. The problem of wasted rollbacks is dealt with by using an interrogative message delivery (messages are only delivered when the simulation requests it.) rather than an imperative delivery scheme (delivery of messages to an LP based on the timestamp of the message). Thus, messages are ‘pulled’ by the simulator as and when required rather than being ‘pushed’ by the model sending the message. A pull-based scheme has the advantage of

avoiding wasted rollbacks and minimising the number of state saves required, when operating in a system where messages are processed in batches. However, a pull-based scheme is not sufficient for handling all types of messages. For instance, emergency orders may need to be sent to the simulator immediately. Hence, messages will have to be 'pushed' to the simulator as soon as they arrive. Thus, in addition to serving messages to the simulator when demanded (pull based), the message controller also needs a mechanism to interrupt the simulation and send messages to it when required. These messages may cause a rollback and as the nature of them demands that they be processed immediately, the simulator may have to rollback to a point that is not saved. Consequently, the simulator will have to rollback to an earlier time and repeat some of the computations.

In terms of performance, although the interrogative approach results in additional processing compared to the standard implementation, it is argued that in general, state saving and retrieval operations would be significantly more expensive (computationally) than the operations required to perform message interrogation.

The idea of wasted rollbacks (termed artificial rollbacks) has also been discussed by Bagrodia [Bagrodia 90] with respect to Masie, a distributed simulation language. Bagrodia investigates the occurrence of wasted rollbacks in the context of priority servers. For example, consider a server that serves two types of messages, one with a high priority and the other with a low priority. A high priority message is allowed to preempt the server if a low priority message is being processed. Let us assume the LP receives two messages- (10, high) and (12, low). Both types of messages take 10 units of

time to be served. The first message is received by the LP at time 10. The second message can cause the LP to rollback depending on how far the LP has progressed after receiving the first message. For instance if the LP is at time 15 and the second message arrives, the LP will rollback. Bagrodia terms such a rollback as an ‘artificial rollback’, as a rollback is only necessary if the second message arrives after time 20. In Masie the message types that can interrupt the simulation are made explicit. When a message arrives with a timestamp higher than the simulation clock, it is checked to see if it is of a valid type to interrupt the simulation. Only if the message is valid (of a type that with higher priority), the simulator issues a rollback. Masie employs a Wait until structure for message acceptance. The wait statement has the following form:

Wait t until

```
{
    r1
    r2
    ..
    ..
    rn
}
```

t represents the wait time during which the message is processed, **r₁ r₂ ...r_n** describe the type of messages that can interrupt the processing of the message accessed by the wait statement.

To tackle the issue of flooding of messages, the antimessage mechanism, used in the standard optimistic PDES implementations, has been replaced by a novel technique employing rollback counts and control messages. The algorithm requires a LP to send only one message to delete all erroneous message sent by the LP due to a rollback, thus

greatly reducing message overhead incurred during rollback. The LPs that have received erroneous messages may require in turn to rollback and undo erroneous messages sent earlier, resulting in a cascade of rollbacks. However, at each stage only one message (control message) needs to be sent. The upperbound M_{\max} , the maximum number of messages sent by an LP network to undo the effects of a rollback is equal to the largest distance that can be traversed on a graph G without traversing any vertex more than once. Where, the vertices represent LPs and the arcs message passing links.

Finally, a proof of correctness is given for the PDES algorithm presented in this chapter.

CHAPTER SEVEN

Conclusion

7.1 Summary of work

The primary objective of the research presented in this thesis was to develop a framework to support the modelling and simulation of supply chains. Supply chains are an example of what has been termed ‘multifaceted systems’ by Zeigler [Zeigler 84]. The management of supply chains involves dealing with a multiplicity of objectives at various levels of decision making. Consequently, employing an exclusively reductionist methodology of model building that involves analysing the whole by studying the component parts in isolation, or, on the other hand, using a holistic approach that attempts to capture all the aspects of a supply chain in a single model, are not sufficient. In order to develop the modelling framework three areas of simulation and modelling, viz. large scale modelling, distributed modelling, and parallel discrete event simulation, were deemed to be relevant.

A key feature of large scale modelling is the ability to combine appropriate component models, that represent the various facets of a system, to create a composite model. To support a compositional approach, component models need to be modularised with a well defined interface and a coupling scheme needs to be specified that allows the various component models to be coupled so as to interact with one another as part of the

composite model. Various coupling schemes have been described in the literature. The most common scheme is a tightly coupled approach where all the component models and their respective ports are uniquely identified. Interconnections between component models are made explicit by mapping appropriate ports on to one another. The DEVS and Control flow graph (CFG) formalisms are two examples that employ a tightly coupled scheme. The advantage of such a scheme is that coupling relationships can be described unambiguously. However, the task of explicitly specifying all the model couplings in a composite model is time consuming. This is further exasperated if a top down coupling knowledge representation scheme, such as the SES, is employed, as the system to be modelled needs to be decomposed completely and all possible decompositions, aspects, and specialisations need to be made explicit.

Thus, rather than using a top down structure for model synthesis a bottom up approach may hold more value. This is possible by distributing knowledge for synthesis in each of the component models, rather than using a single structure such as the SES. A bottom up approach supports incremental model building, as all possible coupling scenarios need not be identified beforehand, as is the case in a top down representation.

Three class based taxonomies viz., *Model-Type*, model interaction hierarchy, and transfer entity hierarchy, were employed by HerMIS to provide knowledge for composite model synthesis. The *Model-Type* classification classifies models based on what part of the system they model. Each of the *Model-Type* classes is further structured into a *Model-Interaction* hierarchy. The *Model-Interaction* hierarchy classifies models based on what other *Model-Types* a model interacts with. Thus, the *Model-Interaction*

taxonomy can be viewed as providing semantic information that can be used to guide composition. The *Transfer-Entity* hierarchy, on the other hand, provides syntactic knowledge. It describes the actual manner in which the models interact in a composite. It describes the structure of the interaction between the component models.

The second requirement for the supply chain modelling environment is to operate in a distributed environment. This is essential as it is natural for the models to be developed and maintained at the various sites of the supply chain. Thus, just as the supply chain is distributed in various locations the models that represent the various components of the supply chain are also distributed. This requirement is addressed by the use of agents. The knowledge required for composition is incorporated by an interface agent termed *synthesis_agent*. Each *Model-Type* is associated with a *synthesis_agent* and includes all the information required for composition. Composite models are synthesised by using the synthesis agents in a blackboard based architecture.

A second type of agent, based on the principles of mobile agents, is used to allow distributed component models to be integrated to create a composite model. This allows the models to be transported on to the simulation node where simulation occurs.

Finally, a PDES algorithm, which takes into consideration the cyclical and loosely coupled nature of the supply chain, was presented that allows the simulation of the composite model. The algorithm is based on the optimistic approach and incorporates a novel mechanism that reduces the risk of antimessage flooding.

7.2 Conclusions

The primary objective of this work was to develop a modelling and simulation framework for the modelling of supply chains. This objective has been accomplished by the conceptualisation of HerMIS. The insights gained from the development of HerMIS allows for the following conclusions to be drawn.

- **Large scale modelling:** The creation of models of the supply chain from scratch in an ad hoc fashion to satisfy specific objectives is wasteful. Due to the complexity in terms of size and interaction of supply chains a single model is not sufficient to satisfy the various decision making objectives that arise in the management of supply chains. Alternatively, decomposing the supply chain into smaller parts and analysing them in isolation ignores potentially important interactions that occur between the components of the supply chain. Thus, there is a need for a compositional approach that allows individual components of a supply chain to be studied in detail and also to be coupled to other models so as to take into account the wider system and the interactions that consequently occur.
- **Unsuitability of SES:** The system entity structure (SES) provides a knowledge representation scheme in the DEVS formalism that aids synthesis of composite models. The SES is found to be unsuitable for the modelling of supply chains for the following reasons. The structure of a supply chain is principally a network one while the SES is a tree structure that represents the various decompositions, aspects, and specialisations of a system. Thus, a network based knowledge representation is more

suitable. In addition, the creation of a SES is an arduous task that requires the complete top down decomposition of the system, and the exposition of all possible models that represent the various facets of the decomposed system and their various couplings. Creating a SES may be worthwhile during the design of a new system, but its applicability is limited when trying to model existing systems.

- **Class based hierarchies:** As mentioned above, the task of making all the port to port couplings explicit in creating a SES (System Entity Structure) can be a time consuming activity. An alternative is to include taxonomic information in the component models, which aids synthesis. Thus, rather than employing a global structure that incorporates the knowledge required for synthesis, it is derived from the component models themselves. Such an approach, in addition to being suitable for a distributed modelling environment (which is one of the requirements of the modelling framework), allows a more incremental model development approach.
- **Conservative PDES algorithm:** Employing a conservative PDES algorithm to the simulation of composite models was found unsuitable due to the following two reasons. Firstly, the loosely coupled nature of a supply chain meant that a large proportion of time was spent by the component models waiting (blocking) to guarantee the causal relationship between the component models was maintained. This could be alleviated to some extent by incorporating 'look-ahead' information, but this requires the user to be aware of the temporal characteristics of the model. Secondly, the cyclic nature of supplier-customer relationships in a supply chain

means that the simulation is liable to deadlocking. A communication overhead is thus incurred in avoiding or detecting and resolving the deadlocks.

- **Optimistic PDES algorithm:** Optimistic PDES algorithms do not require to block and consequently are also not liable to deadlocking. However, two potential pitfalls of the optimistic approach were identified. Firstly, there is a potential for ‘wasted rollback’ to occur if message processing occurs in a batched fashion. Secondly, a rollbacks in the simulation is likely to cause a flooding of rollback messages.

7.3 Contribution

The major contribution of this work is in the conceptualisation of HerMIS, a framework that enables a compositional approach to discrete event modelling of supply chains. In doing so a number of smaller contributions were made to the area of modelling methodology.

- A model taxonomy scheme was developed that allows a representation of composite building knowledge. A scheme such as this can also be used for modelling other systems that incorporate a network structure such as models of the internet etc.
- The use of agents was proposed to allow composition to occur in a distributed framework.

- A novel PDES algorithm was proposed for the simulation of the composite model. This again is not restricted to use in supply chain modelling but may find application in other domains that reflect a similar structure, for example in the modelling of computer networks.

7.4 Suggestions for future work

The obvious first step in extending the work presented in this thesis is to develop a complete prototype tool based on HerMIS. By applying this to various modelling case studies the efficacy of the framework can be determined. In addition, the following are some other suggestions for extending this work.

- One of the issues that was not addressed in detail by the HerMIS framework was the development of a mechanism to aid the selection of a particular component model instance based on the modelling objective. The development of a rule based knowledge system may help in this regard.
- With the increasing adoption of HLA as a standard for model interoperability, integrating it with HerMIS may be of some value. HLA could provide the syntactic knowledge for coupling while the *synthesis_agent* in conjunction with the blackboard based architecture could aid the composition task.

- The PDES algorithm presented does not take into account the existence of emergency orders or other contingencies and operates in a purely batch processing mode. The algorithm may be improved to take this into consideration.

References

- Addanki, S., Cremonini, R., and Scott, J. (1991). "Graphs of models." *Artificial Intelligence*, 51, 145-177.
- Aesop. (1994). *Optimization by Simulation: Simple ++ Tutorial*, Aesop, Stuttgart.
- Alter, C. (1993). *Organizations Working Together*, Sage Publications, London.
- Askin, R. G., and Standridge, C. R. (1993). *Modeling and Analysis of Manufacturing Systems*, John Wiley & Sons.
- Ayani, R. (1989). "A Parallel Simulation Scheme Based on the Distance Between Objects." In *Proceedings of the 1989 SCS Multiconference on Distributed Simulation*, 113-118.
- Bagrodia, R. L., Chandy, K. M., and Misra, J. (1987). "A Message-Based Approach to Discrete-Event Simulation." *IEEE Transactions on Software Engineering*, SE-13(6), 654--665.
- Bagrodia, R. L., and Liao, W.-T. (1990). "Maisie: A Language and Optimizing Environment for Distributed Simulation." In *Proceedings of the 1990 SCS MultiConference on Distributed Simulation*, 205-210.
- Ball, P., and Love, D. (1994). "Expanding the Capabilities of Manufacturing Simulators Through Application of Object-Oriented Principles." *Journal of Manufacturing Systems*, 13(6), 412-423.
- Ballou, R. H. (1992). *Business Logistics Management*, Prentice Hall, Engelwood Cliffs, NJ.
- Barros, F. J., Mendes, M. T., and Zeigler, B. P. (1994). "Variable DEVS - Variable Structure Modeling Formalism: An adaptive Computer Architecture Application." In *Proceedings of the 5th Annual AI Planning in High Autonomy Systems Conference*, 185-191.
- Basnet, C., Farrington, P. A., Pratt, D. B., Kamath, M., Karacal, S. C., and Beaumariage, T. G. (1990). "Experiences in Developing an Object-Oriented Modeling Environment for Manufacturing System." In *Proceedings of the 1990 Winter Simulation Conference*, 477-481.
- Belanger, R. (1990). "MODSIM II - A Modular, Object-Oriented Language." In *Proceedings of the 1990 Winter Simulation Conference*, 118-122.
- Bellenot, S. (1990). "Global Virtual Time Algorithms." In *Proceedings of the 1990 SCS Multiconference on Distributed Simulation*, 122-127.

- Benjamin, J. (1990). "Analysis of Mode Choice for Shippers in a Constrained Network with Applications to Just - In -Time Inventory." *Transportation Research*, B 24B/3, 229-245.
- Bhuskute, H. C., Duse, M. N., Gharpure, J. t., Pratt, D. D., Kamath, M., and Mize, J. H. (1992). "Design and Implementation of a Highly Reusable Modeling and Simulation Framework for Discrete Part Manufacturing Systems." In *Proceedings of the 1992 Winter Simulation Conference*, 680-688.
- Bhuskute, H. C., and Mize, J. H. (1993). "Parallel Discrete Event Simulation of Manufacturing Systems." In *Proceedings of the 2'nd Industrial Engineering Research Conference*, USA, 705-709.
- Birtwistle, G. M. (1979). *Discrete Event Modelling on Simula*, Macmillan.
- Bleecker, S. (1994). "The Virtual Organisation." *Futurist*, 28(2), 9-14.
- Blunden, G. P. (1967). "Implicit Interaction in Process Models." *Simulation Programming Languages*, J. N. Buxton, ed., North Holland, 283-287.
- National Research Council. (1995). *Information Technology for Manufacturing*, National Academy Press.
- Bryan, O. F. J. (1989). "MODSIM II: An Object Oriented Simulation Language for Sequential and Parallel Processors." In *Proceedings of the 1989 Winter Simulation Conference*, 172-177.
- Bryant, R. E. (1977). "Simulation of Packet Communication Architecture Computer Systems," MS Thesis, MIT.
- Burbidge, J. L. (1984). "Automated Production Control with a Simulation Capability." In *Proceedings of the IFIP Conference*, Copenhagen, 1-14.
- Buss, A. H., and Stork, K. A. (1996). "Discrete Event Simulation on the World Wide Wed Using Java." In *Proceedings of the 1996 Winter Simulation Conference*, 780-785.
- Buxton, J. N., and Laski, J. G. (1962). "Control and Simulation Language." *The Computer Journal*(5), 194-199.
- CACI. (1993). *MODSIM II: Reference Manual*, CACI Products Inc, La Jolla, California.
- Chandy, K. M., Bagrodia, R., and Liao, W. T. (1993). "Author's Response: Concurrency and Discrete-Event Simulation." *ACM Transactions on Modeling and Computer Simulation*, 3(4), 284-285.

- Chandy, K. M., and Misra, J. (1979). "Distributed Simulation: A Case Study in Design and Verification of Distributed Programs." *IEEE Transactions on Software Engineering*, SE-5(5), 440-451.
- Chandy, K. M., and Misra, J. (1981). "Asynchronous Distributed Simulation via a Sequence of Parallel Computations." *Communications of the ACM*, 24(11), 198-206.
- Christopher, M. (1992). *Logistics and Supply Chain Management - Strategies for Reducing Costs and Improving Services*, Pitman Publishing.
- Clark, A. J., and Scarf, H. (1960). "Optimal Policies for a Multi - Echelon Inventory Problem." *Management Science*, 6, 475-490.
- Cohen, M. A., and Lee, H. L. (1988). "Strategic Analysis of Integrated Production-Distributed Systems: Models and Methods." *Operations Research*, 36(2), 216-228.
- Colvin, K., and Beaumariage, T. (1998). "Network-Centric Simulation Using NCSOS." *Simulation*, 70(6), 396-409.
- Committee, D. S. (1994). *The DIS Vision, A Map to the Future of Distributed Simulation*, Institute for Simulation and Training.
- Cota, B. A., Fritz, D. G., and Sargent, R. G. (1994). "Control Flow Graphs as a Representation Language." In *Proceedings of the 1994 Winter Simulation Conference*, 555-559.
- Cota, B. A., and Sargent, R. G. (1990). "A Framework for Automatic Look-Ahead Computation in Conservative Distributed Simulations." In *Proceedings of the 1990 SCS Multiconference on Distributed Simulation*, 56-59.
- Cota, B. A., and Sargent, R. G. (1992). "A Modification of the Process Interaction World View." *ACM Transactions on Modeling and Computer Simulation*, 2(2), 109-129.
- Crookes, J. G. (1982). "Simulation in 1981." *European Journal of Operational Research*(9), 1-7.
- Cubert, R. M., and Fishwick, P. A. (1998). "OOPM: An Object-Oriented Multimodeling and Simulation Application Framework." *Simulation*, 70(6), 379-395.
- Dahmann, J. S., Kuhl, F., and Weatherly, R. (1998). "Standards for Simulation: As Simple As Possible But Not Simpler - The Higher Level Architecture For Simulation." *Simulation*, 71(6), 387-387.
- Das, S. R., and Fujimoto, R. M. (1994). "A Performance Study of the Cancel back Protocol for Time Warp." In *Proceedings of the 1994 SCS Multiconference on Distributed Simulation*, 135-142.

- Daum, T., and Sargent, R. G. (1997). "A Java Based System for Specifying Hierarchical Control Flow Graph Models." In *Proceedings of the 1997 Winter Simulation Conference*, 150-157.
- Davis, N. D. (1996). "A Compositional Hybrid Approach to the Design of Manufacturing Simulators," Ph.D. thesis, Warwick Manufacturing Group, Department of Engineering, University of Warwick.
- De Meter, E. C., and Deisenroth, M. P. (1991). "GIBBS: A Model Specification Framework for Multi-Stage Manufacturing System Design." *Simulation*, 56(6), 413-421.
- DeKok, A. G., and Bertand, J. W. M. (1995). "Performance measurements and performance control in supply chain management." *Research Report*(June 1995), 1-18.
- Delen, D., Pratt, D. B., and Kamath, M. (1996). "A new paradigm for manufacturing enterprise modeling: Reusable, multi-tool modeling." In *Proceedings of the 1996 Winter Simulation Conference*, 985-992.
- DIS Steering Committee. (1994). *The DIS Vision, A Map to the Future of Distributed Simulation*, Institute for Simulation and Training, Orlando, FL.
- Erman, L. D., Hayes-Roth, F., Lesser, V. R., and Reddy, R. D. (1980). "The Hearsay II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty." *ACM Computer Surveys*, 12(2).
- Fiddy, E., Bright, J. G., and Hurron, R. D. (1981). "See Why: Interactive Simulation on the Screen." In *Proceedings of the Institute of Mechanical Engineers*, 167-172.
- Fishman, G. S. (1973). *Principles of Discrete Event Simulation*, John Wiley & Sons.
- Fishwick, P. A. (1993). "A Simulation Environment for Multimodelling." *Discrete Event Dynamic Systems: Theory and Applications*, 3, 151-171.
- Fishwick, P. A. (1997). "Web-Based Simulation." In *Proceedings of the 1997 Winter Simulation Conference*, 100-102.
- Forrester, J. W. (1961). *Industrial Dynamics*, MIT Press, Cambridge, Mass.
- Franta, W. R. (1979). *The Process View of Simulation*, North Holland.
- Fritz, D. G., and Sargent, R. G. (1995). "An Overview of Hierarchical Control Flow Graph Models." In *Proceedings of the 1995 Winter Simulation Conference*, 1347-1355.
- Fritz, D. G., Sargent, R. G., and Daum, T. (1995). "An Overview of HI-MASS (Hierarchical Modeling and Simulation System)." In *Proceedings of the Winter Simulation Conference*, 1356-1363.

- Fujii, S., Tsunoda, H., Ogita, A., and Kidani, Y. (1994). "Distributed Simulation Model for Computer Integrated Manufacturing." In *Proceedings of the 1994 Winter Simulation Conference*, 946-953.
- Fujii, S., Tsunoda, H., Yamane, M., Hirashima, Y., and Hirano, M. (1994). "A Study on Distributed Simulation System for Design and Operation of a Large Manufacturing System." In *Proceedings of the Japan-USA Symposium on Flexible Automation*, 769-772.
- Fujimoto, R. (1990). "Parallel Discrete Event Simulation." *Communications of the ACM*, 33(10), 31-53.
- Fujimoto, R., and Nicol, D. (1992). "State of the Art In Parallel Simulation." In *Proceedings of the 1992 Winter Simulation Conference*, 246-254.
- Fujimoto, R. M. (1988). "Performance Measurements of Distributed Simulation Strategies." In *Proceedings of the 1988 SCS Multiconference on Distributed Simulation*, 14-19.
- Fujimoto, R. M. (1993). "Parallel Discrete Event Simulation: Will the Field Survive?" *ORSA Journal on Computing*, 5(3), 213-248.
- Fujimoto, R. M. (1998). "Time Management in The High Level Architecture." *Simulation*, 71(6), 388-400.
- Gafni, A. (1988). "Rollback Mechanisms for Optimistic Distributed Simulation Systems." In *Proceedings of the 1988 SCS Multiconference on Distributed Simulation*, 61-67.
- Galliers, R. D. (1987). "Choosing Appropriate Information Systems Research Methodologies." *Communications of the ACM*, 30(11), 900-902.
- Geller, T. L., Lammers, S. E., and Mackulak, G. T. (1995). "Methodology for Simulation Application to Virtual Manufacturing Environments." In *Proceedings of the 1995 Winter Simulation Conference*, 909-916.
- Genesereth, M. R., and Ketchpel, S. P. (1994). "Software Agents." *Communications of the ACM*, 37(7), 48-147.
- Geuder, D. F. (1995). "Object Oriented Modeling with Simple++." In *Proceedings of the 1995 Winter Simulation Conference*, 534-540.
- Goble, J. (1991). "Introduction to SIMFACTORY II.5." In *Proceedings of the 1991 Winter Simulation Conference*, Phoenix, Arizona, 404-408.
- Goldberg, A., and Robson, D. (1989). *Smalltalk - 80: The Language*, Addison Wesley.

- Gordon, G. (1979). "The Design of the GPSS Language." Current Issues in Computer Simulation, N. R. Adam and A. Dogramci, eds., Academic Press, New York.
- Gordon, R. F., Gordon, K. J., MacNair, E. A., and Kurose, J. F. (1990). "Hierarchical Modeling in a Graphical Simulation System." In *Proceedings of the 1990 Winter Simulation Conference*, 499-503.
- Gosling, A. K. (1996). *The Java Programming Language*, Addison-Wesley.
- Greenberg, S. (1972). *GPSS Primer*, Wiley, New York.
- Groble, J. (1997). "MODSIM III - A Tutorial." In *Proceedings of the 1997 Winter Simulation Conference*, 601-605.
- Haq, A. N., Vrat, P., and Canda, A. (1991). "An Integrated Production - Inventory - Distribution Model for Manufacture of Urea: A Case." *International Journal of Production Economics*, 39, 39-49.
- Healy, K. J., and Kilgore, R. A. (1997). "Silk TM: A Java-Based Process Simulation Language." In *Proceedings of the 1997 Winter Simulation Conference*, 476-482.
- Heim, J. A. (1994). "Integrating Distributed Models: The Architecture of ENVISION." *International Journal of Computer Integrated Manufacturing*, 7(1), 47-60.
- Heim, J. A. (1997). "Integrating Distributed Simulation Objects." In *Proceedings of the 1997 Winter Simulation Conference*, 532-538.
- Henrikson, J. O. (1981). "GPSS- Finding the Appropriate World View." In *Proceedings of the 1981 Winter Simulation Conference*, Atlanta, 505-516.
- Hills, P. R. (1965). "SIMON - A Simulation Language in Algol." Simulation in Operational Research, S. M. Hollingdale, ed., English Universities Press, London.
- Hills, P. R. (1971). *HOCUS*, P-E Group, Egham, Surrey.
- Hon, K. K. B., and Ismail, H. S. (1991). "Application of Transputers for simulation of Manufacturing Systems- a Preliminary Study." *Proceedings of Institution of Mechanical Engineers -Part B- Journal of Engineering Manufacture*, 205(B1), 19-23.
- Hooper, J. W. (1986). "Strategy-related characteristics of discrete-event languages and models." *Simulation*, 46(4), 153-159.
- Houlihan, J. B. (1987). "International Supply Chain Management." *International Journal of Physical Distribution and Materials Management*, 17(2), 51-66.
- ISTEL. (1996). *WITNESS User Manual*, ISTEL, Redditch.

- Iwata, K., Onsato, M., Teramoto, K., and Osaki, S. (1995). "A Modelling and Simulation Architecture for Virtual Manufacturing Systems." *Annals of the CIRP*, 44(1), 399-402.
- Jefferson, D. R. (1985). "Virtual Time." *ACM Transactions on Programming Languages and Systems*, 7(3), 404-425.
- Joines, J. A., and Roberts, S. D. (1997). "An Introduction to Object-Oriented Simulation in C++." In *Proceedings of the 1997 Winter Simulation Conference*, 78-85.
- Karacal, S. C., and Mize, J. H. (1996). "A formal structure for discrete event simulation. Part I: Modeling multiple level systems." *IIE Transactions*, 28, 753-760.
- Kateel, G., Kamath, M., and Pratt, D. (1996). "An overview of CIM enterprise modeling methodologies." In *Proceedings of the 1996 Winter Simulation Conference*, 1000-1006.
- Kiviat, P. J. (1969). "Digital Computer Simulation: Computer Programming Languages." *RAND Technical Report*(RM-5883PR).
- Kohli, R., and Park, H. (1994). "Coordinating Buyer - Seller Transactions Across Multiple Products." *Management Science*, 40(9), 45-50.
- Konas, P. (1996). "Where Object-Oriented Technology Meets Parallel Simulation." In *Proceedings of the 29th Annual Hawaii International Conference on System Sciences*, 360-363.
- Laski, J. G. (1965). "On Time Structures in Simulations." *Operational Research Quarterly*, 16(3), 329-339.
- Lau, H. S., and Lau, A. A. H. (1994). "Coordinating Two Suppliers with Offsetting Lead Time and Price Performance." *Journal of Operations Management*, 11, 327-337.
- Law, A. M., and Kelton, W. D. (1991). *Simulation Modelling and Analysis*, McGraw Hill.
- Lee, H. L., and Billington, C. (1993). "Material Management in Decentralized Supply Chains." *Operations Research*, 41(5), 835-847.
- Lee, H. L., Billington, C., and Carter, B. (1993). "Hewlett-Packard Gains Control of Inventory and Service through Design for Localization." *Interfaces*, 23(4), 1-11.
- Lee, H. L., Padmanabahn, V., and Whang, S. (1997). "Information Distortion in a Supply Chain: The Bull Wip Effect." *Management Science*, 43(4), 546-558.
- Lee, H. L., and Rosenblatt, M. J. (1986). "A Generalised Quantity Discount Pricing Model to INcrease Supplier Profits." *Management Science*, 32(9), 1177-1185.

- Liao, C., Motaabbed, A., Kim, D., and Ziegler, B. P. (1993). "Distributed Simulation Algorithm for Sparse Output DEVS." In *Proceedings of the 4th Annual AI Simulation and Planning in High Autonomy Systems*, 171-177.
- Liles, D. H., and Presley, A. R. (1996). "Enterprise Modeling within an Enterprise Engineering Framework." In *Proceedings of the 1996 Winter Simulation Conference*, 993-999.
- Lin, Y. B., and Lazowska, E. (1989). "Determining the Global Virtual Time in a Distributed Simulation." 90-01-02, Department of Computer Science, University of Washington, Seattle.
- Login. (1993). *Cadence User Guide*, Cesson-Sevinge, France.
- Lomow, G., and Baezner, D. (1990). "A Tutorial - Introduction to Object Oriented Simulation and SIM++." In *Proceedings of the 1990 Winter Simulation Conference*.
- Lorenz, P., Dorwarth, H., and Ritter, K.-C. (1997). "Towards a Web Based Simulation Environment." In *Proceedings of the 1997 Winter Simulation Conference*, 1338-1344.
- Lubachevsky, B. D. (1989). "Efficient Distributed Event-Driven Simulations of Multiple-Loop Networks." *Communications of ACM*, 32(January), 111-123.
- Luna, J. J. (1992). "Hierarchical, Modular Concepts Applied to an Object-Oriented Simulation Model Development Environment." In *Proceedings of the 1992 Winter Simulation Conference*, 694-699.
- Lutz, R. (1998). "High Level Architecture Object Model Development And Supporting Tools." *Simulation*, 71(6), 401-409.
- Markowitz, H. M., Hausner, B., and Karr, H. W. (1963). *SIMSCRIPT, A Simulation Programming Language*, Prentice Hall, New Jersey.
- Melman, M., and Livny, M. (1984). "The DISS Methodology of Distributed System Simulation." *Simulation*(April), 163-176.
- Minsky, M. (1986). *Society of the mind*, Simon and Schuster, New York.
- Misra, J. (1986). "Distributed Discrete Event Simulation." *ACM Computer Survey*, 18(1), 39-65.
- Monahan, J. P. (1984). "The Quantity Discount Pricing Model to Increase Vendors Profits." *Management Science*, 30, 720-726.
- Nance, R. E. (1971). "On time flow mechanisms for discrete system simulation." *Management Science*, 18(1), 59-73.

- Nance, R. E. (1981). "The Time and State Relationships in Simulation Modeling." *Communications of the ACM*, 24(4), 173-179.
- Naylor, T. J., Balintfy, J. L., Burdick, D. S., and Chu, K. (1966). *Computer Simulation Techniques*, Wiley, New York.
- Nevison, C. (1990). "Parallel Simulation of Manufacturing Systems: Structural Factors." In *Proceedings of the 1990 SCS Multiconference on Distributed Simulation*, 17-19.
- Nicol, D., and Heidelberger, P. (1996). "Parallel Execution for Serial Simulators." *ACM Transactions on Modeling and Computer Simulation*, 6(3), 210-242.
- Nwana, H. S. (1996). "Software Agents: An Overview." *The Knowledge Engineering Review*, 11(3), 205-244.
- O'Keefe, R. M. (1986). "The three-phase approach: A comment on "strategy-related characteristics of discrete-event languages and models." *Simulation*, 47(5), 208-211.
- Oren, T. I. (1975). "Simulation of Time-Varying Systems." *Advances in Cybernetics and Systems*, J. Rose, ed., Gordon and Beach Science Publications, England, 1229-1238.
- Oren, T. I., and Zeigler, B. P. (1979). "Concepts for Advanced Simulation Methodologies." *Simulation*, 32(3), 69-82.
- Overstreet, C. M., and Nance, R. E. (1985). "A Specification Language to Assist in Analysis of Discrete Event Simulation Models." *Communications of the ACM*, 28(2), 190-201.
- Overstreet, C. M., and Nance, R. E. (1986). "World View Based Discrete Event Model Simplification." In *Proceedings of the Modeling and Simulation Methodology in the Artificial Intelligence Era*, Amsterdam, 165-179.
- Page, E. H. (1993). "Technical Response: In Defense of Discrete-Event Simulation." *ACM Transactions on Modeling and Computer Simulation*, 3(4), 281-286.
- Page, E. H. (1998). "The Rise of Web-Based Simulation: Implications for the High Level Architecture." In *Proceedings of the 1998 MITRE Software Engineering and Economics Conference*.
- Page, E. H., Griffin, S. P., and Rother, S. L. (1998). "Providing Conceptual Framework Support for Distributed Web-Based Simulation within the High Level Architecture." In *Proceedings of the SPIE*, Orlando.
- Page, E. H., Moose, R. L. J., and Griffin, S. P. (1997). "Web-Based Simulation in Simjava Using Remote Method Invocation." In *Proceedings of the 1997 Winter Simulation Conference*, 468-473.

- Page, E. H., and Nance, R. E. (1994). "Parallel Discrete Event Simulation: A Modeling Methodological Perspective." In *Proceedings of the 1994 SCS Multiconference on Distributed Simulation*, 88-93.
- Pawletta, T., Lampe, B., Pawletta, S., and Drewelow, W. (1996). "An Object Oriented Framework for Modeling and Simulation of Variable Structure Systems." In *Proceedings of the Summer Simulation Conference 1996*, Portland, Oregon.
- Pegden, C. D., Shannon, R. E., and Sadowski, R. P. (1990). *Introduction to Simulation using SIMAN*, McGraw Hill, New York.
- Peng, C., and Chen, F. F. (1996). "Parallel Discrete Event Simulation of Manufacturing Systems: A Technology Survey." *Computers and Industrial Engineering*, 31(1), 327-330.
- Pflug, G. C., and Prohaska, M. (1990). "The Entity-Connection Approach to Modelling and Simulation." *Simulation*, 226-235.
- Pidd, M. (1988). *Computer Simulation in Management Science*, John Wiley & Sons.
- Pidd, M. (1994). *Computer Simulation in Management Science*, John Wiley & Sons.
- Pidd, M. (1995). "Object-orientation, Discrete Simulation and the Three-Phase Approach." *Journal of the Operational Research Society*, 46, 362-374.
- Popken, D. A. (1992). "An Object-Oriented Simulation Environment for Airbase Logistics." *Simulation*, 59(5), 328-338.
- Popplewell, K., and Yu, B. (1994). "Approximate Factory Modelling in an Integrated Multi-View Modelling Environment." In *Proceedings of the Factory 2000 - Advanced Automation*, 630-637.
- McNab, R., and Howell, F. W. (1996). "Using Java for Discrete Event Simulation." In *Proceedings of the 12'th UK Computer and Telecommunications Performance Engineering Workshop (UKPEW)*, University of Edinburgh, 219-228.
- Roberts, C. A., and Dessouky, Y. M. (1998). "An Overview of Object-Oriented Simulation." *Simulation*, 70(6), 359-368.
- Rosenblit, J. W. (1989). "Model Development in Knowledge Based System Design and Simulation." In *Proceedings of the 1989 Winter Simulation Conference*, 167-172.
- Rosenbluth, A., and Wiener, N. (1945). "The role of models in science." *Philosophical Science*, 12(4), 316-321.

- Roy, R., and Meikle, S. E. (1995). "The Role of Discrete Event Simulation Techniques in Finite Capacity Scheduling." *Journal of the Operational Research Society*, 46, 1310-1321.
- Rubinstein, R. Y., and Melamed, B. (1998). *Modern Simulation and Modeling*, John Wiley & Sons.
- Samadi, B. (1985). "Distributed Simulation, Algorithms and Performance Analysis," Ph.D., University of California, Los Angeles.
- Sarjoughian, H. S., and Zeigler, B. P. (1998). "DEVSTJAVA: Basis for a DEVS-based Collaborative M&S Environment." In *Proceedings of the 1998 International Conference on Web-Based Modelling and Simulation*, 21-31.
- Schwarz, L. B. (1981). "Introduction." Multi-level Production/Inventory Control Systems: Theory and Practice, L. B. Schwarz, ed., North Holland.
- Seethalaksmi, M. (1978). "Performance Analysis of Distributed Simulation," MS, University of Texas, Austin, Texas.
- Shires, R. (1984). "Parallel execution of a FMS Simulation." *Simulation*, 70(6), 359-368.
- Slats, P. A., Bhola, B., Evers, J. J. M., and Dijkhuizen, G. (1995). "Logistic Chain Modelling." *European Journal of Operational Research*, 87, 1-20.
- SMC. (1993). *ARENA User Guide*, Systems Modelling Corporation, Sewickley, PA.
- Sokol, L. M., Briscoe, D. P., and Wieland, A. P. (1988). "MTW: A Strategy for Scheduling Discrete Simulation Events for Concurrent Execution." In *Proceedings of the 1988 SCS Multiconference on Distributed Simulation*, 34-42.
- Southall, J. T., Mirbagheri, S., and Wyatt, M. D. (1988). "Investigations Using Simulation Models into Manufacturing - Distribution Chain Relationships." *BPICS Control*(April/May), 29-34.
- Stroustrup, B. (1988). *The C++ Programming Language*, Addison Wesley.
- Thomas, C. (1994). "Interface-Oriented Classification of DEVS Models." In *Proceedings of the 5th Annual AI Planning in High Autonomy Systems Conference*, 208-213.
- Thomas, D. J., and Griffin, P. M. (1996). "Coordinated supply chain management." *European Journal of Operational Research*, 94, 1-15.
- Tocher, K. D. (1963). *The Art of Simulation*, The English Universities Press, London.

- Tocher, K. D. (1965). "Review of Simulation Languages." *Operational Research Quarterly*, 16, 189-217.
- Tomlinson, A. I., and Garg, V. K. (1994). "An Algorithm for Minimally Latent Global Virtual Time." In *Proceedings of the 1994 SCS MultiConference on Distributed Simulation*, 35-41.
- Towill, D. R. (1991). "Supply chain dynamics." *International Journal of Computer Integrated Manufacturing*, 4(4), 197-208.
- Towill, D. R. (1993). "System dynamics - background, methodology, and applications." *Computer & Control Engineering Journal*, 261-268.
- Turner, J. R. (1993). "Integrated Supply Chain Management: What's Wrong with This Picture?" *Industrial Engineering*, 25, 52-55.
- Uhrmacher, A. M. (1993). "Variable Structure Models: Autonomy and Control - Answers from To Different Modeling Approaches." In *Proceedings of the 4th Annual AI Planning in High Autonomy Systems Conference*, 133-139.
- Ulgen, O., and Gunal, A. (1998). "Simulation in the Automobile Industry." Handbook of Simulation, J. Banks, ed., Engineering & Management Press, John Wiley & Sons, 547-570.
- Ulgen, O. M., and Thomasma, T. (1990). "SmartSim: An Object Oriented Simulation Program Generator for Manufacturing Systems." *International Journal of Production Research*, 28(9), 1713-1730.
- Urmacher, A. M., and Arnold, R. (1994). "Distributing and Maintaining Knowledge: Agents in Variable Structure Environments." In *Proceedings of the 5th Annual AI Planning in High Autonomy Systems Conference*, 178-184.
- VanDuin, J. H. R., TerBrugge, R., Geilleit, R. A. A. M., and Klaren-Polman, M. G. (1992). "The TASTE Program Report." *TNO-IPL, Eindhoven*.
- Vidal, C. J., and Goetschlckx, M. (1997). "Strategic production-distribution models: A critical review with emphasis on global supply chain models." *European Journal of Operational Research*, 98, 1-18.
- Waldorf, J., and Bagrodia, R. (1994). "MOOSE: The Concurrent Object-Oriented Language for Simulation." *International Journal of Computer Simulation*, 4(2), 235-257.
- West, J., and Mullarney, A. (1988). "ModSim: A Language for Distributed Simulation." In *Proceedings of the 1988 SCS Multiconference on Distributed Simulation*, 155-159.

Wilson, A. L., and Weatherly, R. M. (1994). "The Aggregate Level Simulation Protocol: An Evolving System." In *Proceedings of the 1994 Winter Simulation Conference*, 781-787.

Wonnacott, P., and Bruce, D. (1996). "The APOSTLE Simulation Language: Granularity Control and Performance Data." In *Proceedings of the 1996 SCS Multiconference on Distributed Simulation*, 114-123.

Wu, B. (1994). *Manufacturing Systems Design and Analysis - Context and Techniques*, Chapman & Hall.

Young, C. H., and Wilsey, P. A. (1996). "Optimistic Fossil Collection for Time Warp Simulation." In *Proceedings of the 29th Annual Hawaii International Conference on System Sciences*, 364-372.

Yu, L. C., Steinman, J. S., and Blank, G. E. (1998). "Adapting Your Simulation for HLA." *Simulation*, 71(6), 410-420.

Zeigler, B. P. (1984). *Multifaceted Modelling and Discrete Event Simulations*, Academic Press.

Zeigler, B. P., and Praehofer, H. (1989). *System Theory Challenges in the Simulation of Variable Structure and Intelligence Systems*, Springer Verlag, Berlin.

Zeigler, B. P., and Sarjoughian, H. S. (1999). "Support for Hierarchical Modular Component-based Model Construction in DEVS/HLA." In *Proceedings of the 1999 Spring Simulation Interoperability Workshop*, 30-37.

Zeigler, B. P. (1990). *Object-Oriented Simulation with Hierarchical, Modular Models*, Academic Press.

APPENDIX A

JAVA source code for PDES controller

The source code of an implementation of the PDES algorithm in Java is included below. The program represents the queue controller used in the PDES engine.

```
import java.util.*;
import java.io.*;

public class Controller {

// Instantiate a new Controller
    public Controller(int num) {

        queues = new Vector[num];
        for (int i=0;i<num;i++) {
            queues[i] = new Vector(initial_queue_length);
        }
        // we could just initialize qc_clk_values to a Vector of
        // of length 0 but this will cause too many resizings as the
        // vector grows.
        qc_clk_values = new Vector(initial_qc_clk_length);
        qc_clk_values.insertElementAt(new Integer(0),0); // Time 0
        qc_clk_index = 0;
        qc_clk = 0;
        num_queues = num;
    }

// Locate a message X in queues[msg.link] such that the following
// invariant holds:
// if op = 0 then
//     prev(X).timestamp <= X.timestamp < next(X).timestamp
// and if op = 1
//     prev(X).timestamp < X.timestamp <= next(X).timestamp
// The function returns the index of message X in queues[msg.link]
    private int locate(int op) {
        int i;
        Vector Q = queues[msg.link_type];
        // Take care of the base case
        if (Q.size() == 0) return -1;

        for (i=0;i<Q.size();i++) {
            if (op == 0) {
                // op = 0 i.e 'x <= m < y'
                if (msg.timestamp <
                    ((Message)Q.elementAt(i)).timestamp)
                    break;
            } else {
                if (msg.timestamp <=
                    ((Message)Q.elementAt(i)).timestamp)
                    break;
            }
        }
    }
}
```

```

    }
    if (i<0) return 0;
    return i;
}

// return the index of the first element in the queue that is
smaller than value
private int findIndex(int value, Vector Q) {
    if (Q.size() == 0) return -1;
    int i;
    for (i=0; i<Q.size(); i++) {
        if (((Message)Q.elementAt(i)).timestamp > value) break;
    }
    if (i == Q.size()) return -1; // No element was found
    return i; // else return index
}

synchronized public int processMessage(Message mg) {
    int i;
    msg = mg;
    Vector Q = queues[msg.link_type];
    if (msg.message_type == 'N') {
        // Normal message
        i = locate(0);
        if ( (i-1 > 0) &&
            (((Message) Q.elementAt(i-1)).rollback_count >
msg.rollback_count)) {
            msg = null;
            // delete msg
        } else {
            // insert msg at Q[i]
            if (i < 0) Q.insertElementAt(msg, 0);
            else Q.insertElementAt(msg, i);
        }
    } // end 'N' specific process
    else if (msg.message_type == 'C') {
        // Control message
        i = locate(1);
        Q.insertElementAt(msg, i);
        // if i < 0 or i is the last element then we can safely
        // skip the following block
        if ( (i >= 0) && (i < Q.size())) {

            int j; int count = 0; // count = # of elements deleted
            int size = Q.size();
            for (j=i+1; j<size; j++) {
                if (((Message) Q.elementAt(j-
count)).rollback_count < msg.rollback_count) {
                    Q.removeElementAt(j-count); // delete m
                    count++;
                }
            }
        } // endif

    } // end 'C' specific process
}

```

```

        if (msg == null) return 0; // No chance of rollback here bcoz
        message was deleted
        if (msg.timestamp <= qc_clk) { // Check for rollback

            // Determine the what value qc_clk should be reset to
            int found = -1;
            for (i=0;i<qc_clk_values.size();i++) {
                if (((Integer)qc_clk_values.elementAt(i)).intValue() <
msg.timestamp) continue;
                found = i;
                break;
            }
            if (found < 0) {
                System.out.println("Fatal System Error");
                System.exit(-1);
            }
            // found > 0
            int count = 0;
            int size = qc_clk_values.size();
            // Remove all time values that are greater than the
            selected timestamp(i.e qc_clk_values.elementAt(found))
            for (i=found+1;i<size;i++) {
                qc_clk_values.removeElementAt(i-count);
                count++;
            }
            qc_clk =
            ((Integer)qc_clk_values.elementAt(found)).intValue();
            qc_clk_index = qc_clk_values.size()-1;

            // Now remove all elements from the referenced queue that
            have timestamp greater than qc_clk
            //if (Q.size() > 0) {
            //    for (i=Q.size()-1;i>=0;i--) {
            //        if ( ((Message)Q.elementAt(i)).timestamp <=
qc_clk) break; // done searching
            //        Q.removeElementAt(i); // timestamp > qc_clk
            //    }
            //}
            // Call the rollback function
            if (msg.timestamp <= qc_clk) rollback(qc_clk);
        }

        msg = null; // msg in not pending anymore
        return 0;
    }

    public Vector getMessages(int req_timestamp){

        int i;
        Vector return_list = new Vector(10,10);

        for (i=0;i<num_queues;i++) {
            Vector Q = queues[i];
            int index = findIndex(qc_clk,Q);
            int j;
            if (index == -1) index = Q.size(); // this will ensure the
            loop won't execute

```

```

        for (j=index;j<Q.size();j++) {
            if (((Message)Q.elementAt(j)).timestamp >
req_timestamp) break;
            else return_list.addElement(Q.elementAt(j));
        }
    }
    qc_clk_values.addElement(new Integer(req_timestamp));
    qc_clk_index = qc_clk_values.size() - 1;
    qc_clk = req_timestamp;

    return return_list;
}

public int rollback(int timestamp){
    System.out.println("Issuing rollback on message
"+msg.toString()+ " timestamp = "+timestamp);
    // Issue a rollback
    return 0;
}

public void printInfo()
{
    System.out.println("Number of queues = " + num_queues);
    int i;
    for (i=0;i<num_queues;i++) {
        System.out.println("_____ Queue #" + i + " _____");
        int j;
        for (j=0;j<queues[i].size();j++)
        {
            System.out.println(" Element #"+j);
            System.out.println(((Message)queues[i].elementAt(j)).
toString());
        }
        System.out.println();
    }
    System.out.println("clk = "+qc_clk);
    System.out.println("Clock history =
"+qc_clk_values.toString());
}

synchronized public int processMessage(InputStream istream) throws
java.io.IOException {
    DataInputStream din = new DataInputStream(istream);
    msg = new Message(din);
    processMessage(msg);
    return 0;
}

private final static int initial_queue_length = 100; // Default
Vector size
private final static int initial_qc_clk_length = 100;

```

```

        private final int num_queues;
        private Message msg; // Pending message, yet to be processed
        Vector[] queues;
        Vector qc_clk_values;
        int qc_clk_index; // index into qc_clk_values.
        int qc_clk; // qc_clk = qc_clk_values[qc_clk_index]
    }

```

```

import java.io.*;
import java.lang.*;

```

```

public class Message {

```

```

    public Message(int link_type, char message_type, int timestamp,
                   int rollback_count, byte[] info) {
        this.link_type = link_type;
        this.message_type = message_type;
        this.timestamp = timestamp;
        this.rollback_count = rollback_count;
        this.info = info;
    }

```

```

}

```

```

public Message(DataInputStream din) throws java.io.IOException {
    link_type = din.readInt();
    message_type = din.readChar();
    timestamp = din.readInt();
    rollback_count = din.readInt();
    info = new byte[info_size];
    din.read(info);
    System.out.println("Printing Message");
    System.out.println(toString());
}

```

```

public String toString() {
    String str = " link_type = ";
    str += link_type;
    str += "\n message_type = ";
    str += message_type;
    str += "\n timestamp = ";
    str += timestamp;
    str += "\n rollback_count = ";
    str += rollback_count;
    return str;
}

```

```

public int link_type;
public char message_type;
public int timestamp;
public int rollback_count;
public byte[] info;
public final int info_size = 10;
}

```

```

/*
    A basic Java class stub for a Win32 Console Application.
*/

import java.io.*;
import java.net.*;
import java.util.*;

public class SimplCon {

    public SimplCon () {

    }

    public static void testController() {
        Vector return_list;
        int i;
        byte[] bt = new byte[64];
        Controller con = new Controller(2);
        for (i=0;i<64;i++) bt[i] = 'X';
        Message msg = new Message(0, 'N', 10, 0, bt);
        con.processMessage(msg);
        msg = new Message(0, 'N', 15, 0, bt);
        con.processMessage(msg);
        msg = new Message(1, 'N', 5, 0, bt);
        con.processMessage(msg);
        msg = new Message(0, 'N', 13, 1, bt);
        con.processMessage(msg);
        msg = new Message(0, 'N', 20, 1, bt);
        con.processMessage(msg);

        return_list = con.getMessages(10);
        System.out.println(return_list.toString());
        return_list = con.getMessages(12);
        System.out.println(return_list.toString());

        msg = new Message(0, 'C', 12, 2, bt);
        con.processMessage(msg);
        msg = new Message(0, 'N', 8, 0, bt);
        con.processMessage(msg);
        msg = new Message(0, 'N', 15, 0, bt);
        con.processMessage(msg);
        msg = new Message(0, 'C', 8, 3, bt);
        con.processMessage(msg);
        con.printInfo();
        return_list = con.getMessages(30);
        System.out.println(return_list.toString());
        msg = new Message(1, 'C', 4, 1, bt);
        con.processMessage(msg);
        con.printInfo();
    }
}

```

```

public static void testCase1(Controller con,byte[] bt) {
    int i;
    Vector return_list;
    for (i=0;i<10;i++) {
        Message msg = new Message(0,'N',1000-i,5,bt);
        con.processMessage(msg);
    }
    con.printInfo();
    return_list = con.getMessages(5);
    System.out.println(return_list.toString());
    con.printInfo();
    con.processMessage(new Message(0,'N',1001,5,bt));
    con.printInfo();
    con.processMessage(new Message(0,'C',995,7,bt));
    con.printInfo();
}

static public void main(String args[]) throws java.io.IOException{

    System.out.println("Hello World");
    //Message msg = new Message(10,'N',11,12,new byte[100]);
    //System.out.println(msg.toString());
    Controller con = new Controller(5);

    //con.printInfo();
    testController();
    System.in.read();
    // Setup a server socket that listens on port portnum
    //ServerSocket server = new ServerSocket(portnum,100);
    //while(true) {
        // Socket sock = server.accept();
        // con.processMessage(sock.getInputStream());
    //}

    System.in.read();
}
final static int portnum = 23;
}

```

7

191