AN OBJECT ORIENTED/DEVS FRAMEWORK FOR STRATEGIC MODELLING AND INDUSTRY SIMULATION

PANAGIOTIS NINIOS

LONDON BUSINESS SCHOOL

Thesis submitted to the University of London
for the degree of Doctor of Philosophy
in the Faculty of Economics

JULY 1994

ABSTRACT

The use of simulation modelling for the development of business strategy models, at an industry level, focusing on the exploration of different scenarios and future policy, has been gaining increased acceptance and popularity over the last decade.

This thesis develops a modelling and simulation framework for industry simulation, extending the approach of System Dynamics, by integrating recent concepts from software engineering and mathematical formalisms for discrete event system modelling.

The current modelling view of industry simulation, based on System Dynamics, is reviewed. A critique of the capabilities of System Dynamics is presented, regarding the ability of the System Dynamics core technology to address the broad requirements of industry modelling. We focus the critique and develop a research agenda around the issues of natural model building, model structure and focus, model reusability and time representation.

An overview of manufacturing simulation and the research directions in that area, is presented with the objective of identifying possible areas of cross-fertilization which can be used in modelling at the industry level in a more effective way.

A review of Object Orientation is presented, along with a general review of mathematical formalisms for the description of discrete event systems, with particular focus on the Discrete Event System Specification formalism (DEVS) [Zeigler (1976, 1984)]. An innovative synthesis of Object Orientation and DEVS is proposed in order to address the research questions which resulted from our critique of System Dynamics.

A Smalltalk implementation of the concepts supported by the synthesis, called OO/DEVS, has been developed. Using as a point of reference the requirements of industry simulation, we build upon a critique of previous DEVS implementations (placed within the manufacturing simulation problem domain), by presenting an innovative implementation view of DEVS, which exploits fully the concepts supported by Object

Orientation.

The issues related to graphical model specification within OO/DEVS, and its comparison to the modern System Dynamics graphical user interfaces, are explored. A OO/DEVS Graphical User Interface and its implementation are explored and presented.

Two case studies have been employed, in order to test the capabilities of OO/DEVS as an alternative to System Dynamics, as well as to demonstrate the modelling characteristics of the framework and its implementation.

A comparative study is presented, where a capacity investment model of the post-privatised UK Electricity Industry is developed in both frameworks. The model is used as a vehicle for assessing the modelling characteristics of OO/DEVS versus System Dynamics. Our initial conclusion is that the modelling properties of OO/DEVS can address at a sufficient level the research issues related to the System Dynamics core technology.

Finally, a large scale modelling case study is carried out, within one of the UK Electricity Distribution companies, where a OO/DEVS model of the Electricity Markets is developed jointly with a management team. This real application establishes the value of OO/DEVS, and its modelling characteristics, as a powerful platform for building decision support industry models.

PREFACE/ACKNOWLEDGMENTS

The research work presented in this thesis, was developed within the Electricity Planning Project, at the Decision Sciences area of the London Business School, and began in early 1991. My research agenda has been influenced on one hand by the general research directions in the area of System Dynamics strategic industry modelling, and on the other hand by the real modelling problems faced by the Electricity Industry after its privatisation.

The research questions presented and explored in this thesis, were initially triggered by the research work of Professor Derek Bunn and Dr Erik Larsen on the investment behaviour in post-privatised UK electricity industry [see Bunn & Larsen (1992a)(1992b)]. The outcome of their research efforts was a family of System Dynamics models, which provided, in a true System Dynamics fashion, a number of interesting insights into the problem. Most importantly, these models inspired a number of questions regarding the 'modelling' power and structure provided by System Dynamics, in relation to the requirements of modelling industries at a strategic level. As a result, I am grateful to both Professor Derek Bunn and Dr Erik Larsen for their stimulating research work, as well at for the comments and help they provided throughout my work.

I am deeply indebted to my supervisor Professor Derek Bunn for his comments, ideas and support throughout my PhD work, without which it could not have been realised. Working with him was a valuable experience that I will never forget.

I also owe great thanks to Dr Kiriakos Vlahos for his suggestions as well as all the time and effort that he devoted to my project. His ideas, and the debates that we had on simulation modelling, provided some of the most crucial elements in formulating my research thought and paths.

I would also like to thank Dr John Morecroft for his constructive comments in the earlier stages of this work. His suggestions have provided me with a valuable conceptual perspective of System Dynamics modelling.

I would like to thank in particular Dr Brian Smith and Peter Dasey at Southern Electric, for sharing with me their knowledge and expertise of the UK electricity industry.

I owe thanks to the Economic and Social Research Council and the London Business School for their financial support throughout the three years of my PhD thesis.

I am thankful to all the lecturers at the London Business School Decision Science subject area, as well as to the PhD programme, for providing a stimulating research environment, as well as the required resources.

I owe special thanks to Akin Adamson for the effort that has put in testing the simulation software and developing further my research ideas.

I would also like to thank my PhD student colleagues, for the inspiring and exciting time that I had with them at the London Business School.

Least but not last I am thankful to my friends that supported me at the difficult times through these three years, especially to Giannis Giannikos and Catherine Gallagher.

Finally, I am most grateful to my parents for all their support throughout my studies. This PhD thesis is devoted to them.

TABLE OF CONTENTS

TABLE OF DIAGRAMS

Chapter 1

Introduction

Contents:

1.1 Modelling Industry Structures and Policies

Modern industrial organizations have become a complex interlocking network of information channels. Information channels emerge at various points to control physical processes, such as building of capacity and production of goods, and strategic decisions such as investment strategy and market development. As a result, in today's world the location of production, R&D and marketing are increasingly becoming, among others, conscious managerial decisions, rather than historical precedents. In that organizational and industry environment "If management is the process of converting information into action, then management success depends primarily on what information is chosen and how the conversion is executed... Every person has available a number of information sources. But each selects and uses only a small fraction of all the available information... A manager's success depends on both selecting the most relevant information and on using that information effectively" [Forrester (1992)]. This is particularly true, if the focal point is not the organization as an entity, but the organization within its industrial environment.

The issues of effective information utilization, bring into light the questions related to the ability of today's organization's senior managers to absorb the developments in business environments, and act on that information with appropriate business moves, in other words the dependence of management on learning. The rapid change that underlines our business world emphasises the importance of institutional learning, which is "the process whereby management teams change their shared mental models of their company, their markets and their competitors" [De Geus (1988)], while as it has been pointed out, "The ability to learn faster than our competitors may be the only sustainable competitive advantage" [De Geus (1988)]. In that respect, the need for institutional learning underlines the importance of modelling tools, that can be used to play back and forth management's view of its market, the environment, or the competition.

In addition to the need of effective information utilization and learning, the current movement towards competition, economic liberalization and privatization has introduced issues of competitive strategy, which are 'soft' in nature, and turn our attention to the multiplicity of players and objectives within an industry structure. From a business modelling perspective, organizations participating in today's competitive industries, need the tools to discover new concepts that enable the organization to become more vigilant in recognizing significant industry trends, emerging business problems and preparing the managers to deal with them. Issues of broader financial objectives, flexibility and increased risk have to be addressed. Modelling methodologies are needed, that have the qualities of flexible decision support tools which can serve as a vehicle for scenario development, communication and debate.

Over the last decade, simulation modelling has attempted to address the modelling problems related to the evolving industrial structures, by extending its influence beyond the manufacturing and operational problems, into the boardroom and as a platform to support strategic thinking, group discussion and learning in management teams.

The use of simulation for the development of business strategy models, at an industry level, and the facilitation of executive debate, focusing on the exploration of different scenarios and the formulation of future policy has been gaining acceptance and popularity [for example see Morecroft & van der Heijden (1992), Bunn & Larsen (1992), Merten et al (1987)]. **Industry simulation** is the term that we will be utilising in this thesis to describe this type of simulation.

The participants of an industry simulation exercise are invited to think about the structure of the industry analyzed and the behaviour of the main players. A model of the industry is developed, which represents the shared understanding of the functioning of the industry. This is usually followed by the elaboration of credible industry scenarios which are simulated in order to explore the main uncertainties. Interest focuses on the dynamic behaviour resulting from

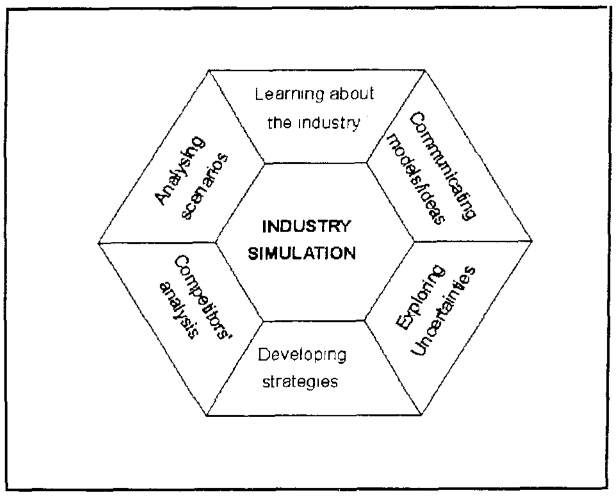


Figure 1: The aims of industry simulation.

the industry structure and its relation to different adopted strategies. The whole process aims at enhancing the group's understanding of the main issues in the industry and will hopefully lead to the development of robust strategies. Finally, the models developed document the shared understanding of the industry and facilitate communication with other people in the organisation. Figure 1 summarises the main objectives of industry simulation.

1.2 Motivation & Thesis Objective

The requirements for robust modelling of the structure and decision rules that govern an industry, scenario development, strategy exploration and learning, point towards the need for modelling methodologies that facilitate natural model building, versatility in decision rule specification, model modularity, fast model reformulation, and the ability to look at the same model from multiple perspectives.

The System Dynamics modelling philosophy [see chapters 2 & 6], has proven to be an adequate starting point for industry modelling and simulation, as we have just described it. Researchers within the System Dynamics area have addressed a number of research questions regarding this approach to model building, management participation and knowledge elicitation, model representation and communication, as well as the problem of model utilization as a management learning tool. Nevertheless, the issues that industry analysts want to address at a modelling context, are multifaceted and broader than the System Dynamics core technology can address.

Our research goal is to address the problem of industry modelling by revisiting the core technology of System Dynamics, focusing on one hand on the modelling requirements that we have just described, and on the other hand on research developments in adjacent areas of simulation modelling. Specifically, our aim is to explore recent developments in software engineering, as they stem from the evolution of Object Oriented Analysis Design and Programming, as well as overview the area of simulation modelling and exploit the advances in discrete event mathematical formalisms.

The research agenda of this work, is to develop a modelling and simulation framework, that can meet the challenges of industry modelling in the '90s and beyond. Such a framework should recognise, and be based on, the main assumptions of System Dynamics. In that respect, the framework should capitalize on the System Dynamics experience in knowledge elicitation, model building and model use. Additionally, the framework should address the limitations of the System Dynamics' core technology, in such a way that its own core technology could be technically viewed as a super-set of that of System Dynamics. Similarly, the framework should be broad enough to accommodate modelling of industries with a variety of structural characteristics. In particular, the framework should address (i) a more realistic modelling of the industry's players, (ii) the multiplicity of players and policies, (iii) a structured and manageable way to incorporate detail where it is needed, (iv) the issue of model evolution.

1.3 Thesis Organization

A brief description of the remaining chapters of this thesis follows:

Chapter 2:

System Dynamics in Perspective

The chapter discusses the main characteristics of the System Dynamics model building methodology, and core technology. A brief review of its application areas follows, with emphasis on industry simulation. Through a conceptual classification framework, we compare industry simulation and System Dynamics with manufacturing simulation. Finally, we elaborate on our research agenda by pinpointing a set of System Dynamics characteristics and limitations, that should be addressed in respect to industry modelling.

Chapter 3:

Object Orientation & the Discrete Event System Specification formalism

In this chapter we present the characteristics and the value of object oriented programming and design, and discuss how these characteristics can be used to address some of the research questions that we set in chapter 1. We also discuss the evolution of discrete event mathematical formalisms in general, and the Discrete Event System Specification (DEVS) in particular. We finally, sketch an overall Object Oriented/DEVS approach and its relative advantages.

Chapter 4:

The Smalltalk implementation of OO/DEVS

In this chapter we discuss the computer implementation of the OO/DEVS framework based on the theoretical concepts, presented in Chapter 3. We discuss the characteristics of our implementation and we support a number of design choices in relation to our research objectives. We also demonstrate the use of the framework through a small OO/DEVS model. We close the chapter, by referring back to our research agenda and evaluating OO/DEVS in respect to the critique that we present in chapters 2 and 3.

Chapter 5:

The OO/DEVS GUI

The research issues related to graphical model specification within OO/DEVS, as well as the implementation of the OO/DEVS Graphical User Interface (GUI), are presented in this chapter. The 'Beer Game', a classic System Dynamics model, is used at the end of the chapter to demonstrate the use of the GUI, as well as the modelling paradigm provided by OO/DEVS.

Chapter 6:

Modelling the Investments in the UK Electricity Industry: A comparative study

A System Dynamics model of the investing behaviour in the industry, as presented by Bunn & Larsen (1992a), is re-implemented for the purposes of a comparative study. The problem area and the associated issues are presented. The problem is approached again, this time within the Object Oriented/DEVS framework and the model is recreated. Finally, a comparison of the two models is presented, and the Object Oriented/DEVS is evaluated in relation to the System Dynamics core technology.

Chapter 7:

The Electricity Markets Model: The development of a OO/DEVS model in a business environment

In the last chapter of this thesis, we present the development of a large-scale, real world, OO/DEVS model in a business environment. The model maps the electricity industry as a whole, concentrating on the electricity purchasing and selling behaviour in the industry. We present our experience from modelling the industry with a management modelling team, the nature and elements of the model as well as sample results. Finally, we discuss the evaluation of the framework, based on the assessment of the management modelling team.

Chapter 8: Conclusions & Future Research Directions

In the last chapter of this thesis we summarize the research questions set in chapters 1 and 2, as well as the way in which we attempted to address them through the development of the OO/DEVS framework. We conclude, by discussing the questions that remain open in relation to (i) the modelling relationship between OO/DEVS and the System Dynamics core technology, (ii) the modelling characteristics provided by OO/DEVS, (ii) the development of a number of tools that can be used within the framework to facilitate model development, utilization, and evolution.

Chapter 2

System Dynamics in Perspective

Contents:

2.1. The emergence of System Dynamics

In this chapter we concentrate on System Dynamics (SD) as a modelling and simulation framework. The reason for doing so is that System Dynamics is the main framework widely used for Industry Simulations. It should be stressed that our aim is to provide an overview of the field and the way that it has developed, and we will not therefore attempt to concentrate on the details.

Coyle (1977), defines System Dynamics as "that branch of control theory which deals with socioeconomic systems, and that branch of management science which deals with problems of controllability". The origin of System Dynamics can be traced to engineering control systems and the theory of information feedback systems. Initially the subject was heavily mathematically flavoured, and the applications were tied up to the engineering field. However, during the 60's the concept of control theory was reshaped in order to be applied to modelling and analysis within the business/social arena. 'Industrial Dynamics', by J. Forrester, was the first, and a very influential, text on the subject. The underlying motive behind this work, and the subsequent SD research and modelling, was the search for a better comprehension of social and economic systems. As Forrester (1987) manifests "The great challenge for the next several decades will be to advance understanding of social systems in the same way that the past century has advanced the understanding of the physical world, ... which can provide a foundation for effectively dealing with economic and social stresses". Over the years, SD has contributed substantially to managerial insights. In addition, work like the 'World Dynamics' [Forrester (1971)] and 'The Limits to Growth' [Meadows et al. (1972)], among others, have addressed issues of worldwide concern, and their contentions have provoked much political and environmental debate. Overall, SD has evolved to something more than a modelling framework. It became a paradigm for perceiving and analysing the world, which is quite distinct from the other fields dealing with the behaviour of systems.

In this chapter we aim to review the main concepts behind System Dynamics by examining its core technology, looking closer at the application domains that have been employed, and finally reviewing the main research areas within the field.

2.2. The System Dynamics view of the world:

As we have discussed, the notion of the control system lies in the core of System Dynamics. Therefore, we ought to embark with the definition of a control system:

The Control System: Control is a way of influencing a system to behave in a desired way. The system may be a technological one, an economic one or even an ecological one. A typical control system contains the *controlled system* (also called process, plant, object, environment) and the *control unit* (also controller, decision unit). The controlled system has some *manipulated inputs* which may cause a change in the *outputs*. The controlled system is also subject to another group of inputs, which are called *disturbance inputs* and are beyond our influence. Disturbances should essentially be considered random variables. The task of the control unit is to achieve a certain goal; and we refer to the values determined by the control unit as *control decisions*.

Forrester (1968), building up on the notion of the control system gives a very vivid description of the a system through the lenses of system dynamics. A *System* is defined as a structure of interactive functions where both the separate functions and the interrelationships, as defined by the structure, contribute to the system behaviour. Therefore, in order to describe a system one should not only describe the separate functions but their method of interconnection, as well. If such a system is viewed trough time, then we arrive at the concept of the *Dynamic System*, which is one which changes with the progress of time. In such a system the parts interact to create progression of system conditions.

Forrester (1961, 1968) describes the theory of system structure in terms of four steps in the hierarchy depicted in Figure 1. The fourth step in this hierarchy resembles the definition of a control system, that we have formerly given. The control system is formally expressed as rates and flows (which constitute a set of difference equations), these levels and rates represent the feedback loops within a system. Finally, the system itself is defined by a closed boundary which marks the problem under consideration.

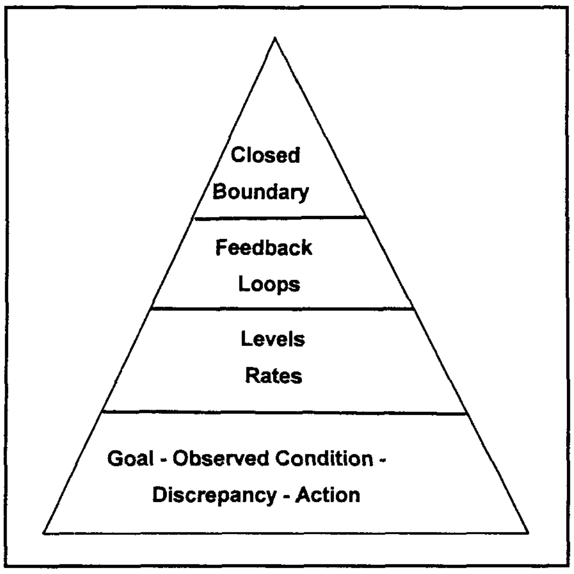


Figure 1:
The hierarchy of Systems Dynamics

The *closed boundary*, in the hierarchy defines the system of interest. It states that the modes of behaviour under study are created by the interaction of the system components within the boundary. The boundary implies that no influences from outside of it are necessary for generating the particular behaviour being investigated. The concept of the closed boundary implies that one starts not with the construction of a model of a system but by identifying a problem , a set of symptoms, and a behaviour mode which is the subject of study. The implication is that without a purpose, it is impossible to define a system boundary.[1]

Inside the closed boundary one tries to map cause and effect. The system dynamics way to do this is through a structure of interacting *feedback loops.* The feedback loop is a structural setting within which all decisions are made. A decision is based on the

[1] It should be noted that the concept of the closed boundary, is intrinsic to every modelling process. Therefore, the above definition applies to every type of simulation.

observed state of a system and produces action which alters the state of a system and the new state gives rise to new information as the input to further decision. One has not properly identified the structure surrounding a decision point until the loops are closed between the consequences of the decision and the influence of those consequences on future decisions.

The next level in the hierarchy contains the *level and rate variables.* These variables represent the activity within a feedback system. The levels (or state variables) represent accumulations within a system. Mathematically they are integrations. The rate variables represent the system's activity. The consequence of the latter definitions is that the level variables are the integrations of those rates of flow which cause the particular level to change. Therefore, a level variable depends only on the associated rates and never on any other level variable. Similarly, no rate can depend on the simultaneous value of any other rate (this is something that is checked in a syntactical way by any System Dynamics language compiler). Rates depend only on the values of the level variables. The implication of the latterly described structure is that any path through the structure of a system will encounter alternating level and rate variables.

The most classic example of a feedback loop (a negative one) is the one that maps a central heating system. If the room temperature (represented as a level of heat in the room) is below a desired temperature, the central heating system will switch on to supply heating and to correct the discrepancy over a period of time. The heat input will be represented by a rate which influences the room temperature; and is influenced by the room temperature in such a way that the heat input would be progressively reduced as the room temperature reaches the desired level.

Ultimately, a substructure exists within the equation that defines a rate variable. A rate equation defining a rate variable is a statement of system policy. The term policy coincides in many ways to what has been referred in the literature as 'decision rule'. Policy is defined as a formal statement giving the relationship between information sources and resulting decision flows. A policy statement [Forrester (1968)] incorporates four components (i) the goal of the decision point, (ii) the observed condition as a basis for decision, (iii) the discrepancy between goal and observed conditions, and (iv) the desired action based on the discrepancy. For example, applying this definition to the thermostat case: (i) the goal would be the desired temperature; (ii) the observed condition would be the room temperature at time t; (iii) the discrepancy between goal and observed condition would be the temperature difference; (iv) the desired action would be an equation that shows how we would like to reach the desired goal. It is suggested, by Forrester (1961), that the decision functions should be perceived as divided into two categories, depending on whether they are ordinarily conscious human decisions or whether they arise from the physical condition of the system. The former are defined as overt decisions while the latter as implicit decisions.

A quite distinctive characteristic of the SD modelling approach is the specific attention paid to the concept of the *aggregation of variables*. The idea is that similar items should be combined into a single aggregate. The issue of aggregation has been discussed at length within the SD literature. For example, Rahn (1985) examines the effects of both internal fluctuations in the variables and external fluctuations affecting system parameters. Allen (1988), considering the origin and nature of evolutionary processes, assesses the concept of the aggregation of variables and comments on the effects of microscopic diversity emphasising how microscopic variability confers on a system the ability to learn and hence to adapt. Forrester (1961) provides a set of conditions under which aggregation of items (variables) is permitted:

(1) The items can be assumed to be controlled by the same identical function;

(2) The controlled outputs are assumed to be used for identical purposes elsewhere in the model.

Overall, System Dynamics [eg. see Forrester (1987), Vennix et al. (1990), Graham & Senge (1992)] can be viewed as a way of clarifying, structuring and finally unifying knowledge. The elementary model components originate from information about structure and policies. Such information is reliable and is usually shared by the mental models of several people. Nevertheless, people's mental models are often logically incorrect. Furthermore, assumed resulting dynamic behaviour is likely to be contrary to that implied by the assumptions being made about system structure and policies. This is a self-evident drawback as, for example, even a first order differential equation is unsolvable by intuitive inspection. It is therefore expected, that attempts to deal with nonlinear dynamic systems, using ordinary intuitive description and debate would lead to internal inconsistencies. In that respect, SD modelling can be effective because it builds on the reliable part of our understanding of the system (i.e. structure, policies) while separates consideration on these underlying assumptions form the implied and expected system behaviour.

2.3. System Dynamics Simulation

Having defined a system structure hierarchy of a particular system, we have in hand a sufficiently rigorous and concise representation that can be directly translated into a set of mathematical equations capable of being handled by a computer. The medium for doing so is to use *difference equations* to approximate the process of integrating rates into levels. The simulation method is essentially a time slicing simulation applied to continuous variables and incorporating continuously adaptive information feedback facilities.

Although a number of SD languages have been developed since the 1960's, the most influential packages have been the DYNAMO [eg. see Professional Dynamo Plus (1986)], and the Stella (1987) with its upgraded version *iThink* (1990). These packages use common numerical analysis integration methods as Euler's and Runge-Kutt. Specifically Professional Dynamo Plus uses a fixed-step size (constant Δt) first order Euler integration method; or a variable step-size third order Runge-Kutta method. It is usually the case that the integration error is monitored in order to achieve a specified accuracy.

Optimization of parameters is a quite distinct feature offered by some SD languages [eg DYNAMOC, Jiuqiang (1991)]. The main principle behind the optimization is that the user sets an objective function, which contains the parameters to be optimised. An optimization routine is usually implementing some optimization method like the *steepest descent*.

2.4. Applications of System Dynamics

As has been already pointed out, control theory found its initial applications in engineering. However, through the conception and development of systems dynamics, as a broader view of control theory, application work was carried out in areas like inventory control, labour and advertising policy. Forrester (1961), demonstrates in his 'Industrial Dynamics' quite a few case studies. A typical one is the Customer-Producer-Employment case study. The "purpose" that defined the closed boundary of the fore-mentioned case study was the search for causes of a fluctuating employment level, of a company, that varied significantly with peak to peak intervals. The model shows that the observed employment instability can result from interactions between the purchasing practices of the customers and the inventory, production and employment practice of the company.

Models like the latter capture industry characteristics in a microeconomic sense. SD modelling has also concentrated on macro issues, like market behaviour in relation to the growth and stagnation of a new product [eg see Forrester (1968)]. In a similar strand, Sterman (1985), has looked at the economic long wave through a system dynamics model and has argued that the principle of capital self-ordering is shown to be sufficient to generate long waves. Forrester, Graham, Senge and Sterman (1983) have provided an increasingly rich theory of the long wave, their system dynamics model relates capital investment, employment and workforce participation, monetary and fiscal policy, inflation, productivity and innovation, and even political values. Probably, one of the largest SD models is the System Dynamics National Model [eg see Forrester et al. (1976)]. The model contains over 200 integrations and more that 1500 equations, which in mathematical terms means that one is dealing with a 200th order, non-linear, differential equation. The objective of the National Model is to generate behaviour as observed in the actual economy from the interactions of local structures and decision-making policies. As a result, "the model builds a bridge joining microstructure and macro-behaviour" [Forrester (1989)]. Even though the latest macro-economic policy models are much smaller, they are still quite popular among System Dynamists. Such models are frequently reported in the literature [for example see Wang et al (1989) for the functions of the economic leverages in China's economy; Arif & Saeed (1989) for a model that attempts to identify how specific policies can procure sustainable growth in an economy dependent on non-renewable natural resources].

It should be noted that, although SD models have provided useful insights in many economic problems, economists have strongly criticized them as unscientific, and system dynamists as "boy economists" [eg see Greenberger et al. (1976)]. Radzicki (1990), has investigated the spirit of these criticisms, by positioning the dispute within an epistemological framework, and contrasting the logical empiricism followed by most economists, to the pragmatic instrumentalism that characterizes System Dynamics models.

Overall, System Dynamics applications have stretched from defence analysis [for example see Coyle (1981), Wolsenholme and Al-Alusi (1987)], looking at problems as broad as the dynamics of a third word war, to problems as specific as Research & Development project modelling. For instance, Roberts (1974) has used an SD model to examine the key factors affecting R&D performance, and assessed how these factors are affected by different policies. Shtub (1992), uses an SD model as an evaluation platform, to explore the ability of two types of R&D schedule control systems to cope with unscheduled events.

Nowadays, it is becoming increasingly common to see SD simulation models being used to support business strategy and executive debate concerned with future policies and scenarios. These "strategy support models" [*sic,*Morecroft (1984)] seek to organise managerial judgement on the structure and behaviour of organisations within a market so as to facilitate insight and group learning about the effects of various strategies. Merten, Löffler & Wiedmann (1987), present a quantitative portfolio simulation model which incorporates features of the Boston Consultancy Group growth-share matrix approach to the allocation of investment funds in multi-business firms. The systems dynamics model helped to explain the evolution of multi-business firms in duopoly markets and demonstrated the fact that the BCG strategy does not take into account a dynamic competitive environment. Foschiani (1989), discusses the use of an SD model as a tool for strategic planning of flexible assembly systems. The modelling originates from the hypothesis that the firm's need for flexibility arises from the difference between its supply and demand within the market. The model accesses various investment policies based on profit-cost related decision rules.

Our interest is focused in these strategy related models. However, we are particularly interested in the modelling of a problem domain that exhibits a particular structural characteristic. That is the existence of a set of companies in a given market place. In such an *industry level* structural setting, companies may compete for market share and resources, or may cooperate against an external agent (eg a regulatory body) that seeks to influence their market-place. Applications like these, are usually centred around concepts like capacity investment, demand growth, price strategies, regulation and production. The Merten at al (1987) model is a good, though simplified by assuming a duopoly, example of such a situation. A more elaborated example is provided by Morecroft and van der Heijden (1992), through a model of the oil producing industry. Their model incorporates the world oil producers by classifying them in different "camps" with different objectives and decision rules. The aim behind the model is to explore questions like the instability of oil price, and the prospects for introduction of new capacity. Bunn and Larsen (1992) present a model that addresses the investment policies in the U.K. privatized electricity industry. The model maps the main players in the industry and their decision rules in investing for new capacity, as well as the ways they interact with each other in their market place. In a similar fashion, Ford & Yabroff (1979) explore strategy issues in the model of an investor-owned electric utility in the U.S.

2.5. Developments in System Dynamics

Looking at the field of System Dynamics, we can identify that since the 60's three major streams of developments have emerged:

1st in the model building area, as a method of system enquiry;

2nd in the software and methodology area;

3rd in the behavioural area.

It can be argued that the first of these streams has many common links with the third; however, we would like to perceive it as a separate one because it is associated with Wolstenholmes' (1990) ideas about quantitative and qualitative systems dynamics. It is also related to the modelling process, rather than the completed model's behaviour.

Wostenholme, (1982, 1990) makes a very clear split between the system description/qualitative mode of System Dynamics, and the quantitative analysis mode using simulation. He argues that the process of analysis of influence diagrams has much to offer in its own right to the methodological dilemma in the field of the system enquiry. This process can be considered, particularly in soft systems modelling, as an end by itself. It should be mentioned that the split under discussion was not inherent in the original approach. Forrester himself does not present a single influence diagram in his 'Industrial Dynamics'. Nevertheless, Wolstenholme points out that influence diagram analysis provides guidance in moving from "what is" to "what should be", by assisting both in the generation of alternatives for improvement and their assessment.

However, it should be noted that Morecroft (1982) takes a slightly different point of view. He argues that influence diagrams (or causal loop diagrams) are weak tools for conceptualization, and do not correspond closely to common mental models of social and industrial systems. Consequently he suggests two different diagramming tools, namely the subsystem diagram and the policy structure diagram [for details see Morecroft (1982)]. Nonetheless, the point is that recently a whole view of the conceptualization has emerged, which is not directly linked to the formalization (through equations) nor the actual simulation of the system.

In that strand, modellers have been recently attempting to involve management teams in the System Dynamics modelling exercise as an end in itself [Morecroft (1991, 1987)]. These attempts have provided the building blocks for a research agenda that is attempting to understand the essence of organizational learning and improve decision-making [Vennix et al (1990), Senge et al (1990), Senge (1990)].

The second major strand of development is in the software area. No need to say, that in the first place, the application of systems dynamics as a methodology was made feasible with the introduction of computers that could solve sets of equations in a numerical analysis sense. Nowadays, however, the evolution of computer graphics and graphical interfaces have made it feasible to model systems directly through rates and flows diagramming (Stella, *iThink*). This development opened up the use of system dynamics to people with very limited computing experience.

Concerning the software area, a brief research agenda, based on a stream of critique, that has emerged from within the SD community, has also been set. That stream of critique is related to the methodology itself, and argues that the SD core technology, is sufficient as far as the situation in hand can be quantified and expressed in numerical equations, while there is no way to incorporate qualitative variables into an SD model. Subsequently researchers have focused on incorporating multidimensional variables into SD. For instance, Tu (1992) has discussed how linguistic representation like "low", "medium" and "high" can be incorporated into SD models. Camara et al (1990,1987) have developed an integrated simulation approach based on logical rules, which incorporates causal diagrams and feedback loop concepts, and can handle linguistic and pictorial variables as well as numerical. Within that agenda, another drawback that has been recognised, is the inability to model uncertainty in terms of relationships among elements. In that direction Tu (1992) has used certainty factors to model situations where, for example, we could only be about 80% certain about the relationship A=2B+C and 60% about A=0.5B+C. By addressing these issues, he has tried to broaden the modelling capacity of SD through a rule based reasoning mechanism.

Probably the most significant methodological development in SD came in the eighties. When chaotic behaviour was detected even in very basic SD models [eg see Rausmussen & Mosekilde (1988)], the SD research community turned its attention to chaos. As a result, research has been focusing on the technical analysis of chaotic models of social, economic and biological systems [eg see Mosekilde & Larsen (1988)], the notions of self organizing structures and structural evolutions in non-linear systems [eg see Allen (1988)], and the techniques and use the qualitative theory of non-linear dynamic systems [eg see Toro & Aracil (1988, 1989)].

The third major stream of development relates system dynamics to the behavioural decision theory. Morecroft (1988), argues that two main inputs from theory go into a systems dynamics model. The first input, from information feedback theory, provides symbols for diagramming a business or social systems and rules for mapping (as has already been mentioned). The second input, from behavioural decision theory, improves the integrity of the models. Morecroft suggests that system dynamics models can be described as "behavioural simulation models" that portray bounded rationality[2] in organisations. It is argued that the feedback structure of the models emerges from the assumptions one makes about decision-makers' access to information, while dynamic behaviour is a consequence of the feedback structure. Overall, the models represent organisations as decision-making/information processing systems involving many players, with multiple and often conflicting goals and limited information processing capability. It should be noted that system dynamics has been influenced in the behavioural side by Simon (1969), and the Carnegie School [for more about behavioural simulation models see Sterman (1987)].

Progress has also been made, in the behavioral side of system dynamics, by the introduction and use of workshops and role-playing simulation games [see Morecroft (1988) and Sterman (1987)]. The purpose of these game models is to create a "learning environment" where policy-makers can debate and relate their own experience more closely to the model. In that sense, in Morecroft (1988), we find a discussion about 'microworlds' for policy debates. It is argued that the debate leads to clarification of the problem or issue and essentially recommendations for action. The important point in this discussion is that all these views can be legitimately facilitated by the existence of the relevant software (see MicroWorld Creator).

Finally, a very interesting advancement, is the introduction of generic policy models. These are models which display important dynamic processes that occur frequently in business and social systems. Example of generic policy models can be found in the

[2] The concept of bounded rationality in human behaviour identifies cognitive limitations in the perception and processing of information and the organisational strategies people devise to deal with them. The idea is that people use heuristics that lead them to sub-optimal or biased decisions.

Cookbook Appendix of Stella software. Distinctive types are the "External Recourse Process", "The compounding process", "The stock adjustment process", the "Implicit Goal seeking process", etc. In the 'Fifth Discipline', Senge (1990), has stressed the importance of many *archetype feedback loops* like the "limits to growth", the "eroding goals", the "growth and under-investment" and many more, by demonstrating how theses archetypes apply to every day situations.

2.6 System Dynamics Summary

In the first five sections of this chapter we concentrated on System Dynamics with the aim to provide an overview of the area in terms of its modelling view, simulation capabilities, application areas and reserach developments. The following table provides a summary of the System Dynamics modelling and simulation view:

SYSTEM DESCRIPTION QUALITATIVE ANALYSIS	QUANTIFIED ANALYSIS USING CONTINUOUS SIMULATION TECHNIQUES		
1.Of existing/proposed systems 2.In terms of system flows 3.Using physical, cash and information flows. 4.To examine feedback loop structure	STAGE 1 1.To examine the behaviour of the system variables over time 2.To examine the sensitivity of the model to changes in: (i)structure; (ii)policies; (iii)delays/ uncertainties	STAGE 2 To examine alternative structures and control policies based on: (i)intuitive ideas (ii)control theory analogies (iii)control theory algorithms	STAGE 3 To optimise system parameters
TO PROVIDE: (i)a perspective on the observed problem or symptom; (ii)a qualitative analysis on which to base recommendations for change	TO PROVIDE: A quantified assessment of alternative ways of improving system performance.		

Table 1: Summary of System Dynamics [Wolsenholme (1982)]

In what follows we extend our discussion of System Dynamics, by placing it within the broader area of simulation modelling. Our objective, is to overview the developments in other simulation areas, and identify possible areas of cross-fertilization for industry simulation.

As we have pointed out in the introduction (Chapter 1) and the 'application of System Dynamics' section, we are particularly interested in what we have called industry models and the way that these models have been implemented within the System Dynamics framework. Having in mind the nature of *industry simulation models* and System Dynamics as the framework for building such simulation models, we intend to perceive industry simulation through the development of a simulation model's classification scheme. We will use that scheme, as a vehicle to provide a more general view of the SD field, and underline the differences between manufacturing and industry simulation from a "core technology" point of view. We are doing this by reviewing the main classification schemes within the literature, and by suggesting a scheme based on different levels of model conceptualisation.

2.7 A Classification of Simulation Models

Although computer simulation is a well studied research area, with origins as old as computing itself, there seems to be little agreement on the classification of simulation models. Many different simulation modelling classification schemes, appear in the literature, attempting to classify simulation models from a number of different perspectives. Nonetheless, classification schemes have been mainly concentrating on discrete event simulation where the multiplicity of strategy-related characteristics of simulation languages or models, constitutes a rich basis for classification.

For instance, Hooper (1982) presents an algorithmic analysis of the three discrete event simulation strategies: event scheduling, activity scanning and process interaction [for a discussion about the discrete simulation strategies see also Paul (1991)]. Highland (1977) proposes additional, more general, classification criteria based on characteristics of simulation models as :

- purpose of simulation (eg. prediction, understanding, optimisation)
- model characteristics (eg. time frame, system size, environmental interaction)
- relationships among entities (eg. symbiotic, antithetic)
- attribute characteristics (eg. time relationships)
- variable characteristics (eg. statistical nature of variables, etc.)

Another common way of classification is by application area. Highland (1977) suggests six main classes:

- Computer systems (eg. VLSI design)
- Governmental and social systems (eg. national economy)
- World modelling (eg. Forrester's world model)
- Industry, businesses (eg. USA car manufacturing industry)
- Ecological and environmental systems
- Biosciences

Ozdemirel et al (1988) classify four different schools of thought, with regard to model construction:

- Hierarchical modular model development [eg. Zeigler's DEVS Scheme (1984)]
- Object oriented simulation [eg. ISIS Fox (1984)]
- Rule based modelling [eg. T-Prolog, a goal oriented simulation language, Adelsberger (1984)]
- Intelligent user interfaces.

Ozdemirel's classification objective, is to put emphasis on research regarding the development of a generic simulation model base, which could be able to assist the user in developing the appropriate specific model for his/her purposes. Probably, the most basic classification is discrete vs continuous, static vs dynamic, and deterministic vs stochastic. Ören & Zeigler (1979), suggest a functional decomposition, and argue that a taxonomy based on such a decomposition generalises significantly the taxonomy according to which simulations are classified by characteristics like time set, etc.

The classification that we present, categorizes simulation models in accordance to different levels of abstraction in model conceptualisation. By using this perspective we aim to provide a "vertical" classification scheme (or classification of classifications), in contrast to the fore-mentioned schemes that provide "horizontal" taxonomies. Our classification looks at applications at the bottom level, as computerized models of specific parts of the world, and simulation formalisms at the top level, as generic vehicles for expressing any part of the world. By doing so, we aim to contrast and compare industry to manufacturing simulation models. The following table depicts such a classification scheme:

Level 5	Mathematical Simulation Formalisms
Level 4	Functional Decomposition
Level 3	Time Set
Level 2	World view (executive type)
Level 1	Simulation Languages
Level 0	Specific Applications

Table 2: A classification scheme of conceptual simulation models.

Within the scheme different levels represent:

Level 0 : Models of specific parts of the real world; eg. a production process, a chemical process, a labour market, an inventory system.

Level 1 : Specific simulation languages: syntactic consistent forms of expressing and applying certain world and time views (eg. GPSS, DYNAMO).

Level 2 : World view (executive[3] type), as: differential and difference equations(with an associated integration method), event scheduling, process interaction, activity scanning (see Pidd (1988) for a detailed account). An interesting approach to different world views can be found in Henriksen (1987), where

[3] Executive: a control program which is responsible for sequencing the operations which occur as the simulation proceeds.

a model of a battle between two armies is presented from several perspectives.

Level 3 : Time set: continuous vs discrete views of time (again Herinksen's (1977) example is a good demonstration).

Level 4 : Functional decomposition: at that level we view a simulation model as functional parts, using the Oren & Zeigler (1979) classification. The primary functional elements of simulation programs are considered as: model structure, model outputs, input scheduling, initialisation, termination, interpretation and display.

Level 5 : Formalisms: generic forms that can accommodate in a consistent way the specification of any real world system.

The scheme is essentially a pyramid. A simulation model of a specific part of the world (application) may satisfy only one world view and time set and may not be expressed in a certain simulation language or be clearly functionally decomposable. It should be pointed out that such classification can be viewed from a historical perspective due to the fact that the simulation enterprise started from ad hoc applications and developed into specific world and time views, simulation languages, well defined functional decompositions and formalisms.

2.8 Manufacturing Simulation vs Industry Simulation

In this part our aim is to view the simulation/modelling enterprise as it is utilised both in the world of manufacturing and industry simulation. We attempt this comparison due to the fact that the world of manufacturing simulation represents one of the most representative parts of simulation modelling both at a research and an application level. The fore-mentioned classification scheme will facilitate us in doing so.

Level 0:

Given that simulation modelling started its life as ad-hoc applications, we would like to start our discussion at classification level 0. Looking at different applications at both types of simulation, the following characteristics can be identified:

Broad Objectives: at that level there seems to be a point of agreement among all the different types of simulation modelling. Paul (1991) points out that manufacturing simulation modelling is mainly used as a means for understanding a problem, and that the tendency is to use simulation modelling as a vehicle for debate about the problem. In a similar strand, Forrester (1988) argues that while the information in people's heads is rich with regard to structure of a system and the policies within it that govern decisions; the mental processes are not reliable in deducing the future dynamic implications of the known structure and policies. Simulation, therefore, can serve as a tool for understanding dynamic behaviour. Nonetheless, in manufacturing simulation modelling, as soon as an understanding of the system has been established, the modeller tries to choose, out of a set of alternatives, a near optimal system configuration, given a set of specific objectives.

Model Construction: In manufacturing simulation the structure of the system to be modelled is usually well defined and well understood. In a typical manufacturing model, where a network oriented language is used, the real system can be expressed in queuing situations (queues or buffers) and processing situations (workstations: a common way of classifying activities on a shop floor) [eg Law (1987)]. The participants can be represented as jobs [eg see Pidd (1988)].

On the other hand in industry simulations (which are essentially System Dynamics applications) what is usually understood is the current behaviour of the system, while its internal structure is in most cases semi-understood. Forrester (1961), points out that an SD model contains policies that are constant for the duration of the model simulation. These policies[4] are laws of human behaviour, for the circumstances within the model. The part of the structure that is usually understood is: sets of formal decision rules that could represent strategies and policies, decisions that are made in accordance to the output of an algorithmic process, physical structure of the sector of the industry (eg. plants, their

[4] The term policy in the SD context is a rule that states how the day-to-day operating decisions are made.

characteristics, etc), resources, communication between the industry participants that is expressed through influences and decision rules [for example the Merten et al (1987) model].

In both cases diagramming techniques are used during the model construction. For example, Activity Cycle Diagrams, flow-charts or special symbols (as in GPSS) [eg see Pidd (1988), Balmer & Paul (1986)] are very popular in manufacturing simulation and Influence Diagrams in System Dynamics. The former concentrate on queuing and manufacturing activities while the latter on behavioural influences.

Specific Objectives: In manufacturing simulation applications the concentration is usually on reduced in-process inventories, increased utilisation of machines and workers, increased on-time deliveries, reduced capital requirements, etc. The issues that are addressed here are the needs for quantity and quality of equipment, performance evaluation (eg. bottleneck analysis), evaluation of operational procedures, etc. [for a detailed approach to manufacturing simulation see Law (1987)].

In industry simulation applications, the concentration is on what the model tells us about the future implications of certain policies, and the factors that will cause changes in the behaviour of the model. Such factors include financing, "product" demand, lack of perfect foresight, market growth and market shares, changes and delays of price regulation, delays in new capacity approval and placement, use of different plant or production technologies and strategies [see references in the chapter about System Dynamics industry modelling]. What the modellers and users are interested in is the impact of a specific policy in the dynamic behaviour of the whole model, with aim to study "how a system can be defended against, or made to benefit from the shocks which fall upon it from the outside world" [Coyle (1977)].

Experimentation: This is the process of experimenting with the simulation model for a specific purpose. Some purposes of experimentation are (i) comparison of different operating policies, (ii) evaluation of system behaviour, (iii) sensitivity analysis, (iv) forecasting, (v) optimization, and (vi) determination of functional relations [Balci (1990)].

Both types of simulation aim to study the dynamic behaviour of the system trough time. However, in manufacturing simulation the behaviour is propagated through system structure, in an operational sense. During the experimentation phase, the concentration is on how different configurations of the basic components of the system, or different values of their attributes, change the system's behaviour. The aim is to find a near optimum configuration (given a finite set of testable system configurations) under which the system could operate.

On the other hand in SD configuration is set (in terms of participating entities), and what we are looking for is a near optimal operating strategy (given the current behaviour and the constraints built into the model in terms of policies, and participating organisations). Therefore, regarding the latter experimentation purposes, industry simulations are basic related to (i)-(iii).

Users: An important factor that distinguishes the two simulation methods, at the application level, is the one that is related to the actual users of the simulation system. Manufacturing simulations are addressed to engineer managers, therefore, people who have a good view of the system from within. Industry simulation applications are usually addressed to top level managers, who in most cases have a more general and aggregated view of the system that is represented in the model.

Level 1 :

We would not like to go into details at this level due to the fact (i) that there is a vast amount of implementation languages [see Paul (1991) and Mathewson (1989) for a review of the discrete simulation languages], and (ii) their features are not particularly relevant to our comparison. In discussing about manufacturing and industry simulation languages, we can identify two types of language environment (which exist in both types of simulation). The first is the programming environment and the second is the graphical user interface (GUI) environment.

The former is well developed. Simulation languages that belong in this type of environment provide the user with a set of primitives (commands or instructions) from which a simulation program can be constructed [for example in DYNAMO]. The main difference, between the two types of simulation that we have been looking at, is that manufacturing simulation languages resemble conventional programming languages in terms of variety and type of instructions, while industry simulations are written as sets of equations.

Graphical user interface environments have been quite well developed for the SD modelling framework [as in Stella and *iThink* for example], probably due to the concise nature of the approach. On the other hand, in manufacturing simulation graphical interfaces, under the name Visual Interactive Simulation, still partially constitute a research area [see Paul (1989), Vujosevic (1990)]. Nevertheless, in both cases the aim is to provide the user with a user friendly and "easy to use" tool, where he/she can build simulations without the need for programming skills.

Levels 2 & 3 :

Our aim is to discuss those two levels concurrently. This is due to the fact that while they are quite distinct in terms of model conceptualisation, they are connected in terms of model building. What we mean by that, is that when a modeller builds a model, he/she links a specific executive type to a specific time set, i.e. discrete or continuous time.

In manufacturing simulations we have three executive types (that have already been mentioned). In industry simulation, the continuous time executive is basically implemented through an integration method. An important point to make, is that in industry simulation the executive is directly related to the actual expressions (equations) within the language; while the same does not hold for manufacturing simulation languages, where different executives may be supported by the same language [for example see Balmer (1987)].

The world view supported in manufacturing simulations by a *process interaction* executive [Kiviat (1969), Fishman (1973), Derrick (1989)] concentrates on the sequence of operations through which an entity passes during its life within the system. As an entity moves through its process, it may experience certain delays and be blocked in its movement. Entities experience periods of activity during process execution and periods of inactivity or delay. Such delays are incurred and execution is shifted (to another entity) at interaction points. If we consider a single server queuing system, the customer process is as follows: (i) customer arrives, (ii) waits until head of queue, (iii) moves into the service channel, (iv) remains there until end of service and leaves the system. On the other hand an *activity scanning* executive [for a good description see Pidd (1988) and Kreutzer (1986)] requires that the modeller identifies the various types of entities in the system to be modelled, the activities which the entities perform, and the conditions under which the activities take place. The single server queue can be represented by three activities: (i) arrival of a new customer, (ii) begin a new service, (iii) end of service. Finally, in the *event scheduling* world view [see Nance (1981)], a simulation program is made up of a set of event routines, each of which describes the operations in which entities engage when the system changes state. The approach specifies that some event is to take place at a determined time in the future and can be scheduled. In our example, we would have two event routines: (i) customer arrival, and (ii) end of service.

The reason why we dwelt on the three discrete-event executives, is that we would like to point out the most commonly used perspectives in which they view the system. In the queue example the process interaction approach views the system through the eyes of the active-customer. In the activity based approach the view is from someone that is participating into the system. Finally, in the event based world view, the view is from above, from someone that is an observer of the system, but still has an interest in the "discrete" events within it.

In a quite similar way, in System Dynamics (and its industry simulation applications) the view is from above, but now the observer views the system through an important simplification: for him time and events become irrelevant. In our example the queue would be a level which accumulates customers through a specific rate. The time that a customer arrives or service starts are of no interest.

Level 4 :

As has been already stressed, at this level we look at simulation models as functional parts

[Ören & Zeigler (1979)]. Such a taxonomy can accommodate both manufacturing and industry simulations. Nonetheless, while in manufacturing simulation much research effort focuses on functional decomposition, in industry simulation and SD such decomposition is more a result of program design than of software support. The functional parts of a simulation model, as presented in Oren & Zeigler, are:

(i) Model structure: the modeller specifies in some form (machine independent or dependent) the static and dynamic structure of the model. Static characteristics refer to the component models which make up the overall model and the variables which describe the states, inputs and outputs of the component models. For example in a manufacturing simulation a component model could be a queuing process with specific Poisson arrivals and Exponential services. In an industry simulation a component model could be a demand accumulation (level) and its associated rates. The input could be growth in demand, and the output satisfaction of demand through a specific rate.

The dynamic characteristics are fixed by the rules of interactions among component models; such rules in a manufacturing simulation are dictated by the structure of the actual system; whereas in an industry simulation by the policies that govern the system.

(ii) Output variables and output functions: the output variables are those that are of interest to the modeller; eg. market growth, average time, in a queue.

(iii) Input scheduling: the modeller specifies the external inputs to the model; i.e. variables that are not controlled by the model; eg. set regulatory conditions, workstation capacity.

(iv) Initialisation: the desired initial states; eg. initial demand, initial number of jobs in a queue.

(v) Termination: the conditions under which a simulation run is to be stopped; eg. run length.

(vi) Collection of simulation data: the modeller specifies the output trajectories to be plotted or statistical summaries of them; eg. profit per time unit, utilisation of a workstation.

Level 5 :

Ultimately the top level in our classification accommodates mathematical formalisms that

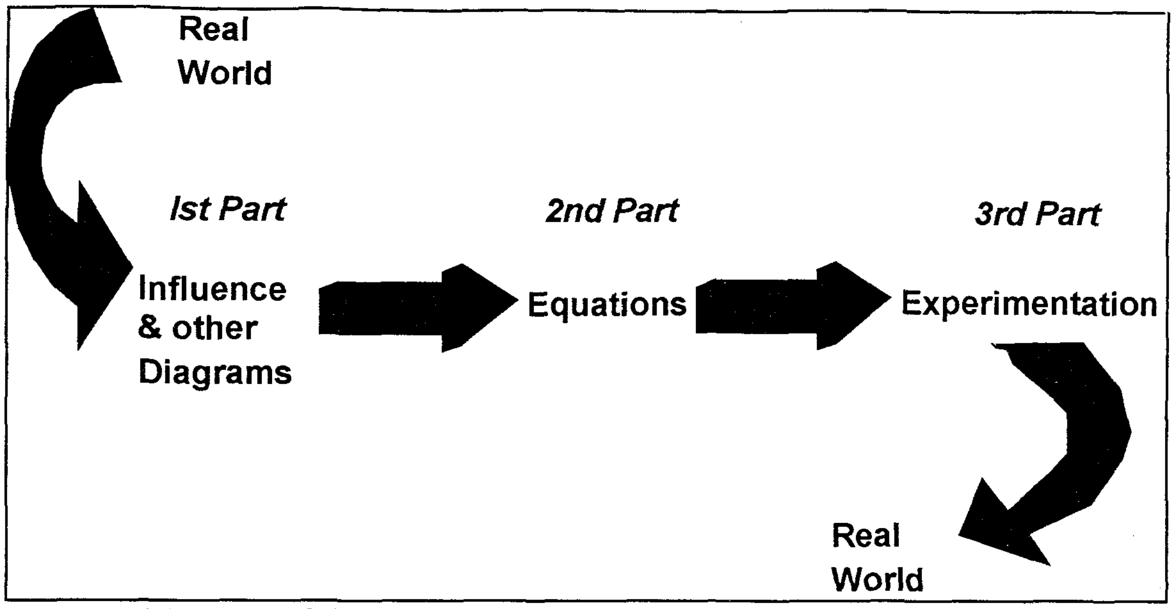


Figure 2: The Parts of the System Dynamics Model Building Process

aim to represent in a concise, consistent and generic way a series of decompositions, time sets and world views. By specific instantiations of a formalism one can move downwards within the classification scheme. Examples of such formalisms are Petri networks, Zeigler's System Modelling Formalism (1984) and Discrete Event System Specification (1976). Such formalisms will be reviewed in a more detailed manner in the following chapter.

By looking at the summary of the comparison of industry and manufacturing simulation (Table 3), we can identify a number of interesting points. The most noticeable difference is that the core technology (System Dynamics) used for industry simulation model building is very well established whereas in manufacturing simulation we can identify a multiplicity of views (mathematical formalisms, time-views). In addition, in contrast to industry simulations, in manufacturing simulations an enormous amount of attention is paid to the quality of the models from a software engineering point of view (eg. in functional decompositions, language constructs). Overall, it apears that the SD core technology for industry simulations has developed to a mature level. The question is whether or not this is true. If the answer is not, then two issues arise: why the core technology for industry simulations is not fully developed up to the requirements of the problem domain, and then what type of technology should be provided to address these requirements. We will attempt to explore further these issues in the next section.

		Manufacturing Simulation	**Industry Simulation**
Level 0	Broad Objectives	System Understanding	System Understanding
	Model Construction	Physical Structure	Policies/some Structure
	Specific Objectives	Optimal configurations	Scenario building
Level 1	Languages	Multiplicity	Standard approach
	GUI's	Research theme	Implemented
Level 2&3	Time view/Executives	Attention to events	View from above
Level 4	Functional Decomposition	Supported at software level	Supported at design level
Level 5	Mathematical Formalisms	Research theme, many views	Control theory

Table 3: Summary of Manufacturing vs Industry Simulation within the classification scheme.

2.9 System Dynamics from a Critical Perspective - Issues for Research into its core technology.

As has been pointed out in the introduction, System Dynamics is viewed throughout this work through industry simulation applications. System Dynamics is the main modelling methodology that has provided us with concepts and tools for modelling and simulating social-economic systems in general and industries in particular. Taking over from the issues that we have opened up in the previous section, in this section we focus on its core technology and underline related research issues. While our comments are fairly general, we mainly base our critique in respect to industry simulations.

From our discussion, earlier in this chapter, we can describe System Dynamics as composed of three distinct parts (Figure 2):

(1) the model building part (Influence Diagrams and other diagramming techniques)

(2) the core technology (equations, integration methods)

(3) the experimentation part (gaming, microworlds)

We can view each of the above parts as involving a certain conceptualization of the system to be modelled, and some conceptual leap that bridges the parts.

It is apparent, from what we have been discussing in the previous section, that a significant amount of research has been devoted in parts (1) and (3), while a substantial amount of work has been focusing on applying SD methodology in many application areas. As a result, as Toyoda & Mawatari (1991) emphasize "evaluation of system dynamics has been obscured by inconclusive debate about particular models... .But its methodology needs further development and codification for revealing general characteristics of complex systems". From a similar viewpoint Tu (1992) observes "the methodology itself (core technology) seems to have progressed less than its applications".

Looking at System Dynamics from a historical perspective, we can identify that the initial input was control theory. Forrester used concepts from control theory in generating the notion of policy, in order to identify and perceive models of social systems. However, the bulk of research in System Dynamics has moved from the actual core technology to softer, behavioural aspects. We would like to argue that the notion of the control system itself (which is good in demonstrating behaviour) helped in moving towards that direction, and helped in the development of the notion of influence diagrams for model conceptualisation. As we have pointed out, a number of System Dynamics researchers have identified the need to re-focus research efforts on the SD core technology. In what follows, we aim at stressing this need, by presenting a number of modelling issues that the current technology cannot readily address. The aim of our critique is to point towards a new core technology, which takes advantage of the current System Dynamics tradition, while provides enhanced modelling capabilities, as well as uniformity in model conceptualisation throughout the three fore-mentioned stages.

Natural Representation:

As Peterson (1992), one of the developers of *iThink*, observes "The stock/flow framework for model conceptualisation is rigorous and precise. It is also abstract, and in many cases not a 'natural' way for the less-proficient modeller to think about system structure".In agreement with the latter, we would argue that System Dynamics, as a modelling framework, does not correspond directly to people's mental models about the world; this drawback exists both at the influence diagram level, and the levels and rates (equation) level.

It is the case, as we will discuss in Chapter 3, that people perceive the real world in terms of entities which have certain attributes. Establishing influences between entities is a secondary process, which actually changes over time. In other words, entity perception is more permanent while the perception of its influences is more transient. Given the latter, it would be extremely useful to have a core technology that maps directly people's mental models. That would help significantly not only the modelling part of the simulation process, but the validation/verification parts as well. As a result, an entity based approach will provide us with a uniform view of the system throughout model development, simulation and experimentation.

Structure:

System Dynamics is a purely structurally based approach. SD models are causal (theory-like) models, i.e. they base their mathematical expressions on postulated causal relations within the modelled system. As Ansoff & Slevin (1968) state, in Industrial Dynamics (i.e. Forrester's initial book) emphasis is placed on "making models 'true to life' the first time, by observing carefully, on testing boundaries, on testing the internal logic of the model, on obtaining parameters from real-life applications". Forrester (1979) indicates that "System Dynamics focuses on policy and how policy determines behaviour. ... One must describe the setting of interrelated policies and therefore the structure of the system".

As we have already discussed research on the SD core technology has concentrated on improving the modelling capability of influences, as components of structure (i.e. multidimensional influences, uncertainty in connections). Nevertheless, even in this advanced form, the core technology is essentially concerned with association relationships within a system, through its emphasis on influence modelling. These association relationships constitute the basis for the system equations that represent the decision rules within the system. It should be pointed out that the equations are somewhat a limited way of expressing decision rules. In addition, we should take into consideration that system relationships can be grouped into three categories [see Blaha et al (1988) p.416] viz., generalisations, aggregations, and associations. As Pracht (1990) points out, generalisation relationships (A is a kind of B) and aggregation relationships (A is part of B), serve the purpose of forming hierarchies and separating entities into their structural components.

The current core technology lacks the ability to provide aggregated and disagregated views of model components within a model[5]. As a result a considerable amount of SD models have been too complex to understand and manipulate, because the modellers have attempted to incorporate over-abundant levels of detail. This is one of the reasons for the current trend of emphasis on small transparent models [eg. Morecroft et al (1989), Larsen et al (1992)] which contain a highly aggregated view of the problem space. Of course it should be stressed that, as Larsen et al. (1992) argue, small models can still produce an interesting behaviour and be useful to the management, while they can be easily communicated to the decision-makers and allow a variety of issues to be explored due to the advantage of yielding fast 'results'. In that respect, contemporary applications of industry simulation through system dynamics have favoured simple, discardable models produced to facilitate strategy meetings, making in that way a virtue out of necessity. On the other hand, as Ford & Bull (1989) observe "managers are excited about the prospect of conducting a wide variety of studies with quick turnaround, but at the same time are suspicious of models that do not possess the level of detail present in the existing corporate models". The question of 'small models versus large models' has troubled the SD community [see Forrester (1987)], and while it is a valid consideration for any modelling exercise, we believe that is should also be addressed at a 'technology' level. In that respect, the provision of aggregation/dissagregation relationship modelling, can provide a more structured way to deal with detail.

[5] The only SD software platform that provides some kind of aggregation is iThink, through the use of 'sector frames'. Nevertheless, this is an ad-hoc addition which has a number of drawbacks which will discuss later on in this thesis, and demonstrates the practical requirement for such type of relationships.

Another factor that adds to the weakness of SD in supporting big and complex models, is the lack of support for generalisation relationships. This form of knowledge organization corresponds to the ability of the human mind to perceive similarities and differences in objects and organisms. Generalisation relationships enable the humans to reduce the entities in a given problem scenario to a manageable proportion [Pracht (1990)]. Therefore, such relationships ought to be supported by a modelling framework that attempts to model socio-economic systems in general, and industries in particular.

Following the tradition of modelling the structure of a system, we would like to identify the need for a core technology that can facilitate the development of hierarchical models, where decisions can be represented at multiple levels. An important point is that disagregation should be supported by preserving at the same time the ability to have aggregated views of the system. A model built in such an environment could facilitate managers in looking at the implications of their decisions at different levels of the modelled system. Overall, in thinking about a natural and comprehensive structure for industry modelling, it would seem necessary to first focus upon entities, their level of aggregation and the way they are organized into hierarchies, before going on to consider the various sorts of influences.

Reusability:

As we have indicated above, contemporary applications of industry simulation through system dynamics have favoured simple, discardable models. Nonetheless, the structure of the system dynamics core technology does not favour major reformulations of the model once produced. However, as Walter & Lopilato (1992) stress, "Socioeconomic-models are likely to be characterised by rapid change and great complexity. Conversational inquiry and update capabilities are necessary for anybody to be able to know what is in the model and make changes as required". In industry simulations in particular, we encounter many situations where we wish to speculate on a different market behaviour or set of competitive strategies, we need to change the pattern of influences (and hence in system dynamics the structure of the model), which, if our modelling structure had been based upon entities, would not be a major editing change. Generalisation relationships, apart from their modelling power, can add to the reusability of a model, through the development of generic modules. In that respect, modularity in the structure of the model, would add to the reusability of valuable models and extend their use from the strategy laboratory into decision (or executive) support.

The concept of reusability brings also into focus model-base and knowledge-base issues, as well as the ways that a model can be treated as a form of expressed knowledge about a problem domain. From that perspective, SD models accumulate essential model-knowledge in the model equations. This is a result of the fact that simulation engine and model are bound together. On the other hand, if the knowledge built into the model could be separated and stored in a knowledge-base, it could be used in different types of models and even accessed in a data-base fashion. Given the time consumed to elicit the knowledge required to build an SD model, as well as the importance of this knowledge, the separation of model knowledge from the model itself becomes an important goal. The problem of separating the model-knowledge from the equations has been identified in the SD literature [eg see Kleinhans (1986, 1989)]. For example, the use of a "knowledge extractor" that is able to make a description of the model, i.e. is able to treat the model as data, that can then be stored, has also been suggested.

Time-Representation:
Finally, we should question the continuous time view of System Dynamics. As we have demonstrated in the first part of this paper such a time (and executive) view, provides a view from above in a way that time and events become obscured. Two points can be stressed. First, people do not think in continuous time, on the contrary they view the world through a discrete time frame (see Chapter 3.5). It is, therefore, apparent that such a continuous time view undermines conceptual uniformity between people's mental models and the simulation model. In addition, in reference to industry modelling it should be pointed out that while the time representation provided by SD, makes sense for macroeconomic or environmental dynamic systems where the "view from above" holds, most corporations and industries are man-made dynamic systems where the evolution of the system in time depends on the complex interactions of the timing of various discrete events. Second, the fact that recent software (eg *ithink*) has introduced discrete delays in

an *ad hoc* way shows this to be a practical requirement.

2.10 Summary

In this chapter we have attempted to provide an overview of System Dynamics as the framework for building industry simulations. We started out by looking at the main elements and concepts of the approach, as well as its main application areas and research themes. We then attempted to place industry simulations and System Dynamics in the more general context of simulation modelling, by providing a classification scheme for simulation models, and comparing industry to manufacturing simulations. By doing that we set a platform which triggered a number of research questions, regarding the core technology of System Dynamics and its ability to support the modelling requirements of industry simulations.

In conclusion, it should be stressed that the critique that we initiated in the very last part of the chapter points towards a core technology that should preserve the top 'horizontal' behavioural aspects of System Dynamics that are supported by association relationships. At the same time, such a core technology, should provide a 'vertical' representation of structure through the incorporation of aggregation and generalization relationships. Overall, the methodological needs that we have identified can be classified as related to:

- Natural Model Representation
- Structure
- Reusability
- Time-Representation

The task that we should set at this point is to explore the feasibility of such core technology, and investigate whether or not any of the concepts that can support it, have been explored within the adjacent areas of manufacturing simulation, software engineering and data modelling.

Chapter 3

Object Oriented Programming & The Discrete Event System Specification Formalism

Contents:

3.1 Introduction - The Research Issues

In the two previous chapters we have defined industry simulation and its modelling requirements. We have also compared manufacturing simulation models to industry ones, and discussed in detail System Dynamics: the methodological framework on which traditionally industry models have been built. Finally, we produced a critique of System Dynamics as a vehicle for building industry models, which opens up a series of research questions. These are the questions that we attempt to address in this chapter.

In our previous discussion, we have identified two basic principles on which System Dynamics has been based. The first one is that a system is being viewed as a structure of interactive functions, where both the separate functions and the interrelationships contribute to system behaviour [Forrester (1968)]. The second one is based on the fact that people (managers) have a good perception of how different policies work, but cannot project consistently the dynamic implications of their policies, into the future. What follows attempts to build upon, as well as extend these two fundamental principles of SD in particular and dynamic systems modelling in general.

Given the fore-mentioned critique we are in a position to formulate the questions that we would like to address as follows:

Entity Based Modelling:

(1) Can we provide a technology that addresses the question of what the entities are in the real system and not what the entities in the simulation language represent in the real system? The objective is to minimize the cognitive leap between the way people perceive the world and the way the simulation technology models it.

Modularity, Reusability, Extendability:

(2) Can we provide a technology where we do not have to make a presumption of what is relevant (by choosing the system's boundary) in the early stage of modelling, so that we can provide for extendability of the model?

(3) Can we provide a technology that can model explicitly the physical structure of a system as well as the policies that govern the structure? The objective will be to maximize stability and reusability of the model, and enhance the decision support

capabilities of models.

Association, Aggregation, Generalization Modelling

(4) Can we provide a technology which supports aggregation, generalization, and association relationships, in a way that a model of a complex system can be built, and at the same time to provide a unitary, concise and intuitively understandable description of the system?

(5) Can we provide a technology in which aggregation/disaggregation can be used in accordance to the model's purpose and modules can be aggregated/disaggregated in different stages of the model's life?

Time Representation:

(6) And finally, can we model time in terms of events, providing at the same time the view from above of System Dynamics as well as the possibility to investigate the implications of a specific event?

In what follows we try to address these questions bearing in mind three important modelling aspects of industry simulation: **structure** (physical - eg. plants in a production system, or abstract - eg. legislation that governs the system), **policies** (that players within an industry may adopt) and **time** (through which structure and policies evolve). We aim to discuss the structure-policy modelling through a software engineering perspective, and compare System Dynamics to the way software engineering and data modelling has evolved, by focusing on Object Oriented Programming. We describe the main characteristics of Object Orientation and we discuss the naturalness of the approach. We also describe how object orientation has been used within the manufacturing simulation world, as well as organizational modelling. Finally we discuss the modelling of time through discrete event system formalisms and concentrate on one of them, the discrete event system specification.

3.2 Object Oriented Programming & Analysis

Object orientation started as an extension of structured programming techniques which aimed through the naturalness of its concepts to improve productivity and software reuse.

Nevertheless, its concepts proved so powerful that object oriented analysis and design [Coad & Yourdon (1989)] were born and even object oriented organizational behaviour modelling has been suggested [Blanning (1987)]. Many proposals for object oriented designs have their origins in the concept of the abstract data type, implemented with Simula, a simulation language developed in the mid-1960's, as well as the ideas about system modularity [first presented by Parnas (1972)]. Many of the concepts supported in Simula exist in ADA (1980), a language developed as a response to the 1970's software crisis from the U.S. department of defence. ADA is what is called object-based [Wegner (1989)]. Nonetheless, true object orientation came with Smalltalk (1980) which is what is considered to be a pure object oriented language. In addition, a series of hybrid languages, like C++ (1986), have emerged. A list of languages in the continuum from object-based to object-oriented, as well as a good discussion of the evolution of object orientation, is presented by Wegner (1989).

In what follows we will review the main concepts of object oriented programming (OOP), and Object Orientation (OO) in general, and suggest how these concepts can be used in addressing some of the questions that we have set, for industry modelling and simulation. We will briefly compare OOP with previous programming views and contrast its world view with that of System Dynamics. Finally, we will provide evidence, from the area of cognitive psycology, in support of the naturalness of the technique in modelling.

3.3 Object Orientation: A Paradigm Shift

As Meyer (1988) points out, when laying out the architecture of a system, the software designer is confronted with a fundamental choice: should the structure be based on the actions or on the data? In answering this question we effectively make a choice between traditional design methods and the object-oriented approach.

Traditional design methods have been based on the notion of "function", "action" or "process"; most of the structured design methodologies [see Birrell & Ould (1985)] tend to place an enormous emphasis on the modelling of functions with less emphasis on the as are procedures or functions (this is the case in System Dynamics where we model with functions). For this reason, we distinguish objects from mere processes, which are input output mappings.

The key concepts that make software modules more understandable, modifiable and reusable are: encapsulation, inheritance, late binding, message passing and polymorphism.

- **Encapsulation** is a technique for minimizing interdependence among separately written modules by defining strict external interfaces[2], and therefore achieving information hiding [see Meyer (1988)]. It is the result of the very notion of packaging data and procedures (methods) together. The external interface of an object serves as a contract between the module and its client modules. The implication is that data abstraction is achieved. This means that the user of an object does not need to understand how these operations are implemented or how the object is represented, so a module can be re-implemented without affecting its clients.

- **Message Passing** is the way through which objects communicate. The idea is that an object can affect the internal condition of another object. This can be achieved by an object requesting (by sending a message) from another object to execute one of its methods. This is what is called a client-server relationship between objects. Message passing is the way to implement what we have already called association relationships, as a means to associate two or more independent objects.

- **Inheritance** is probably the most powerful concept in OOP. It provides for software reuse at a low level, through the provision of OOP classes that are defined in a hierarchical tree-like structure. Each class in the tree inherits the methods and data structures of all its superclasses in the branch of the tree. Inheritance allows the construction of new objects from existing ones by extending, reducing or modifying (by overloading) their functionality. Reusability

[2] i.e. the set of operations defined upon and can be applied to the data of the object

in relation to the concept of inheritance in object orientation is discussed in depth by Cox (1990). An example of inheritance in the database context is presented in Altair (1988). In that example, an object 'Person', which has attributes name and age, is defined, with methods 'die' and 'marry'. Consequently, objects 'Employee' and 'Student' can be both defined as Persons (having something in common). In addition, they also have specific characteristics. Thus an Employee is a special type of Person who inherits attributes and methods, but also has the special attribute salary and method pay. In a similar fashion a Student can be defined by extension from Person. In contrast, in a relational data base system, the designer defines a relation for Employee, a relation for Student and then writes the code for their operations. As a result, the code related to Person is written twice, whereas the use of inheritance helps code reusability because every operation is at the level at which the largest number of objects can share it. In addition, OOP proves itself a powerful modelling tool, because it gives a concise and precise description of the world. Through inheritance "concepts can be rigorously organised because natural mechanisms such as specialization, abstraction, approximation and evolution can be captured" [Wegner (1989)].

- **Polymorphism** means the ability to take several forms [Meyer (1988, p.224)]. This refers to the ability of an entity to refer at run-time to instances of various classes. The notion of polymorphism is related to inheritance, and can also be labelled as feature redefinition. A polygon, for example, can show itself but a line can also show itself. Polymorphism allows both line and polygon objects to contain a method 'show' which is, however, implemented in a different way for each case. In a similar fashion, in a model of factory floor, where both a robot and a crane can fetch material, polymorphism can be used to model the similar behaviour of both objects under the method 'fetch'.

- **Late Binding**, finally, is a rather technical characteristic, which nevertheless has important implications, due to the fact that it provides the advantages of high modularity, operator and method overloading[3], as well as the ability to change variable data types during execution. Late binding refers to the time when a procedure and the data on which it is to operate are related. In contrast to early binding (i.e. at the time of software construction) in traditional procedural languages, late binding in OOP delays the binding process until the software is actually running. In the above example, a program can send the message 'show' to any graphical object, and at run time the appropriate code is executed depending on wheter the object is a polygon or a line.

The advantages of OOP over procedural programming have been documented in Cox (1986) and Meyer (1988). As Meyer (1988) argues, data structures if viewed over time, at a sufficient level of abstraction, are the really stable aspects of the system; while functions tend to change through the system's life cycle. Therefore, by focusing on the data structures we can greatly enhance compatibility and reusability. Compatibility is the ease with which software products may be combined with others, and it is obvious that it is difficult to combine actions if the data structures are not taken into consideration. Reusability is the case where software modules can be used in similar systems without changes. It is therefore evident, that it is difficult to built reusable components if they embody actions alone and ignore the data part.

Another complaint about traditional structured methods has been pointed out by Yourdon (1990). That is related to the fact that structured methods provide little or no guidance in developing the user interface of the system. This is a quite important disadvantage given that as much as 75% of today's window-based, mouse driven, icon oriented systems is associated with the user interface [Byte (October 1990), p.258]. However, object oriented analysis addresses the problem of the user interface in the early stages of analysis.

"Thinking about objects is fundamentally different from thinking about functions" [Jourdon (1990)]. Many have stressed the fact that object orientation represents a

[3] Overloading may be defined as the ability to attach more than one meaning to a name appearing in a program. This is a facility for client modules: they may use the same name requesting different implementations of the same operation [Meyer (1988, p.37)].

paradigm shift in computer programming and modelling. Given the fact that whole generations of software engineers have to be retrained, that very fact can be regarded as a drawback of the approach, while the sobering reality is that 75% of the business data applications are still written in COBOL, a 50's language [Byte (October 1990), p.260]. Of course there are many advantages in object oriented technologies, but there are costs as well. OOP provides semantics that are easily understandable, and closer to the way people create mental models about the world (we will support this argument later). However, the world view of an OOP system increases the semantic gap between the language and the current actual hardware[4], which means that more computing power is needed, while porting systems between different machines can be more difficult. In addition, extensive class libraries should be understood by analysts and programmers, which results in a steep learning curve. From a hardware point of view Object Oriented systems usually need a substantial amount of RAM to run, while late binding has some run time cost. Persistence[5] problems also need to be resolved. The issue of object persistence is particularly important in object oriented database design where no clear consensus has emerged [see Altair (1988) and Stone & Hentchel (1990)]. In general object orientation should be considered as a new technology and the object oriented languages that have emerged, or are emerging, provide different shades of the main object oriented programming concepts.

3.4 Object Orientation in Manufacturing Simulation

The fact that the first concepts of object orientation appeared in Simula, a simulation language, is probably the best argument on how well the object oriented paradigm is

[4] Object Oriented architectures and operating systems are very few. The Next machines represent a fundamental step forward Object Oriented operating systems [see Thomson (1989)]. Apple machines provide object orientation at the user interface level. Finally, the new OS/2 by IBM provide some object oriented features.

[5] Persistence is a property of data that determines how long it should be kept. In traditional procedural languages, the lifetime of data usually does not transcend the life time of a particular program. In order to support persistence in OOP you need a strong notion of object identity that persists across programs and projects. In Smalltalk this problem has been attacked by saving the whole "image" [Goldberg (1984)] which contains every object in the environment.

suited to simulation modelling and in manufacturing simulation in particular.

In such a 'manufacturing type' setting, when we describe a situation, we define 'things' that should be modelled. In a machine shop we define the machines of interest and the pieces to be produced. We also declare what each of these 'things' can do, and what their condition is before, during and after each of these operations. So in that machine shop the operations of each machine are defined and the states of the machines are described before, during and after the part process. However, in most simulation environments (until at least the mid 1980's) the modeller had to translate the fore-mentioned view into a different world view that defined the simulation environment. So, if a network oriented language is used, the problem must be transformed into simulation entities like transactions, queues, resources, sinks, etc. Naturally, it would be better to describe our simulation model using the same terminology that we used in describing the actual system. This would minimize the cognitive leap that should be made between the physical system and its computer model, and would reflect more faithfully the way the system is being viewed. If we slightly change our terminology and use objects in the place of "things", methods instead of "operations" and message passing for the way that these "things" interact, we can easily adopt the object orientated perspective.

Simulation is about representing real systems. In modelling with objects the question is not what the objects in a simulation language represent in the real system, but what the objects in the real system are. If you see your system as composed of general entities, you make them classes (eg. class workstation, transporter, robot, storage facility, etc.), and you specify the specific operations that they can perform (encapsulation). If you need specific instances you refer to the corresponding objects with certain attributes. If some objects appear to group together they can be a subclass of a more general class. For example, a printing machine and a binding machine can be subclasses of machine in the same fashion that student and employee can be subclasses of person. In general, by creating subclasses you may refine general methods for more specific operations.

A system can be modelled in terms of the principal components of which it is made, while their interaction can be identified in a second stage and modelled through message passing. By encapsulating the characteristics (data structure) and methods (operations) within the objects, the objects can be viewed as the fundamental components of the system, yielding a very natural and furthermore stable decomposition. Stability is a result of the fact that the principal components in a manufacturing environment (eg. workstations) remain stable while the interactions between them may change. In this respect OOP provides a clear advantage as compared to process based methods. The result is that old models become reusable because they are conceptually more stable and do not change.

Reusing software components also means reduced code size which in turn means that a single analyst can handle more complexity. Managers, on the other hand, can gain a better understanding of a model and its dynamic behaviour, through the naturalness of the approach; which can also provide the basis for pictorial (iconic) representation [for example see Vujosevic (1990), Thomasma & Ulgen (1988), Guasch (1991)], which can be animated, improving substantially the understanding of the real system. Finally, intelligence in the form of facts and rules can be built directly into the object's functionality.

A good discussion of the OOP advantages in the simulation modelling context can be found in Roberts (1988). A substantial amount of research in simulation based on OOP concepts has been carried out the last few years. For instance, Basnet et al. (1990) have suggested a simulation environment in Smalltalk based on a formalism for manufacturing systems. Knapp (1987) presents Simtalk, an extension of Smalltalk that supports queuing, statistics gathering and simulation oriented graphics for discrete event simulation. Ulgen & Mao (1988) describe SmarterSim, an object oriented program generator for hierarchical, modular modelling. Bryan (1989) discusses the use of object oriented programming in proving a language (Modsim II) capable of exploiting parallel and sequential processing. In a similar fashion Lomow & Baezer (1990) present Sim++, an object oriented language, based on C++, capable of parallel processing. Goldberg (1984) in his Smalltalk-80 book discusses the implementation of a discrete event simulation environment and a series of applications. Zeigler (1987) has implemented his Discrete Event Specification (DEVS) formalism, and built a simulation environment in an object oriented version of Lisp. Kim

(1990), has further described how polymorphism can be exploited in developing new model classes in the DEVS environment. Burns & Morgeson (1988) have produced a simulation world view that can simulate systems involving intelligent decision-making entities, following the object oriented and actor paradigms.

3.5 The Naturalness of the Approach

Up to this point we have discussed the advantages and disadvantages of object orientation strictly from a software engineering point of view. Nevertheless, it appears that the most important characteristic of the approach is its naturalness, and its importance becomes apparent as we descend from general software systems to simulation ones. One of the aims of this research work is to test the hypothesis that this characteristic makes the approach attractive from a modelling perspective. However, before we make any assessment about the usefulness of the technique in industry simulations, we should address two questions: (a) Do people create mental models of the world in terms of entities and the attributes they possess?, and if (a) is true then (b) how do people model mentally the way that entities interact with each other?

It seems that for the software engineering community the comparison between traditional techniques and OOP leans so much towards the latter that cognitive evidence in support of the approach is rarely provided. Instead, enthusiastic statements like the following are often given: "... object-oriented designers usually do not spend their time in academic discussions of methods to find the objects: in the physical or abstract reality being modelled, the objects are just there for picking. The software objects will simply reflect these external objects." [Meyer (1988), p.51].

However, theoretical evidence can be found in the area of cognitive psychology, and has been used in a number of papers on the teaching of the approach [for example see Beck & Cunningham (1989), Gibson (1990)]. The fundamental view of cognitive psychology is that people organize the world through concepts, as Smith & Medin (1981 p.1) point out "Without concepts mental life would be chaotic. If we perceive each entity as unique, we would be overwhelmed by the sheer diversity of what we experience and be unable to remember more than a minute fraction of what we encounter... Fortunately, we do not perceive, remember and talk about each object and event as unique, but rather as an instance of a class or concept that we already know something about. When entering a new room, we experience one particular object as a member of the class of chair, another as an instance of desks, and so on... In sort, concepts are critical for perceiving, remembering, talking and thinking about objects and events in the world.". It should be noticeable how close these ideas are to the concepts provided by OOP.

A significant part of psychology deals with the way people acquire and use concepts (or classes of objects!). The classical view dates back to Aristoteles and advocates that all instances of a concept share common properties, and that these common properties are necessary and sufficient to define the concept. A list of contemporary sources on the classical view can be found in Smith & Medin (1981 p.22). The Aristotelian view, however, has its critiques, the most prominent of which assumes that instances of a concept vary in the degree to which they share certain properties, and consequently vary in the degree to which they represent the concept. According to Labov (1973) one needs a view that posits a unitary description of objects, but where the properties in this description are true for most though not all members. Therefore, the representation of a concept can not be restricted to a set of necessary and sufficient conditions. Finally, the most extreme departure from the classical view advocates that there is no single representation of an entire class or concept, but only specific representations of the class' exemplars. Thus, we view and understand the world through examples of the concepts that we develop. A complete definition of these views, as well as experimental evidence in support or comparison of the fore-mentioned views, can be found in 'Categories and Concepts' by Smith & Medin (1988). They also provide a distinction between component and holistic properties of an object concept. A component property helps to describe the object but it does not constitute a complete definition, while in contrast a holistic property provides a holistic one. A characterization of components is also provided as dimensions (quantitative components) or features (qualitative components). Nevertheless, the fact is that people organize the world around concepts (objects) that possess certain attributes. In addition, as Woods (1981) argues, people use concepts both to provide a taxonomy of things in the world and to express relations between classes in that taxonomy. Object Orientation is in a position to model the world even if we accept the exemplar view given that our exemplar objects can evolve through inheritance and polymorphism. In addition, both dimensional and featural properties can be modelled in an OOP environment.

As a result of the current discussion a positive answer to the first question can be supported and therefore we should try to investigate the second question. While there is some consensus on the way people perceive entities, the exact way that people build taxonomies of concepts and define their relationships has not yet been defined. One possible way discussed by Smith & Medin (1981) is the categorical one. It is suggested that the categorization function involves determining that a specific instance is a member of a concept or that one particular concept is a subset of another (eg. the stock market is a part of the financial markets). Certainly such categorization can be modelled in OOP through inheritance.

However, as Mandler (1984) points out "categorical organization lacks principal relationships among the members of a given class. Each member is only guaranteed to have one relation to the other members, and that there is only the vertical relationship in the hierarchy of class inclusion. ...More relations may exist in idiosyncratic ways". Of course, "idiosyncratic" relations may be modelled through what we have called a "client-server" relationship and message passing. However, the fact that OOP is heavily based on categorical (hierarchical) relationships is probably a drawback given the fact that up to a point it obliges the modeller to think in such a way. Madler (1984), discusses what is called a schematic structure. Such a structure has a part-whole nature which results in "connections among the items in a given unit". In the case of an event schema[6] such connections are temporal. Madler also discusses the concept of the "script" which is a kind of event schema and serves as a way of organizing event sequences (for example going to a restaurant is a script, as is the event sequence in the beer game [see Senge (1991)]). In the case of scene-schemas physical objects are connected and the

[6] An event schema is a hierarchically organised set of units describing generalized knowledge about and event sequence [Madler (1984, p.14)].

connections become spatial (eg. when we have an overall schema for a bedroom, a supermarket or a dealing room). Again the organization is hierarchical in respect to the fact that individual parts are governed by schemas of their own (eg. we know what check-out counters or terminal look like in addition to the scenes in which they are found). It has been experimentally proven [see Madler (1984, p.87)] that when people are asked to list the parts in an ordinary scene, such as an office, most of the things that they list are basic level objects; in addition, people remember the fairly accurate spacial relationship of objects as they remember the temporal relation in scripts. However, people have difficulties in remembering objects in a categorical fashion unless a special attention is drawn to it.

As we have already discussed, the object oriented paradigm is particularly suited for mapping categorical relations such as: Robin and Sparrow are special intances of Bird. However, spatial or temporal relations can be modelled through attributes and methods of an object. For example in a windowing environment [see Goldberg (1984)] graphical objects can identify their relative position on the screen (through coordinates) and communicate it to other objects through message passing. Nevertheless, mapping such relations under the object oriented paradigm can be done as a result of relevant design more than as a result of following the paradigm.

Overall, the answer to the first question set at the beggining of this section, is that people perceive the world through entities and their attributes, while in regard to the second question experimental evidence points towards the direction of 'scematic structures' which can be, for example, an 'event schema' or a 'scene-scema'.

3.6 From Manufacturing to Industry Simulation

The modelling of socioeconomic systems has been traditionally carried out in continuous simulation frameworks using differential (or difference) equations. This makes sense for macroeconomic or environmental models where what we have called the "view from above" holds. In industry simulations, however, we may want to model organizations and their policies in a sufficient level of detail. This is an area that the SD core technology can not address sufficiently, as there are no explicit and robust modelling facilities for incorporating different levels of detail, in different sections of the same model. More importantly, SD fails to minimize the cognitive leap between the way people view the system, and the model. This is one area where concepts of object orientation can be most helpful. As we have already discussed people organize the world in terms of objects and therefore a modelling technique that recognises this fact in the area of industry modelling and simulation will enormously improve the understanding of the models by managers, as well as their use.

As we have discussed in previous chapters System Dynamics models the world as a set of interactive functions; the goal-action pair is at the heart of its core technology. A parallel can be drawn to the traditional software engineering modelling where the modeller models the functions of the system. As we have already discussed such a view does not produce a stable, reusable and maintainable system. Therefore, it should not be surprising that System Dynamics has evolved as a one-off modelling tool where models have a limited life span and are used as a learning environment[7].

Naturally, one can model complex systems in System Dynamics. However, this results in huge and uncomprehensible models. In general, from a software engineering point of view all the arguments that we have previously presented hold for the core technology of System Dynamics, even when a graphical interface is provided (in *iThink* for example).

By suggesting the use of object orientation in modelling industry structures we practically take the use of OOP in organisational modelling one level further. Blanning (1987), has suggested the use of the object oriented paradigm as an extension of the information processing paradigm for organizational behaviour. It is suggested that object orientation can be used to model and simulate in detail the types of messages between organizational subunits and the conditions under which they send or receive messages. The feature of

[7] It should be noted that this is true for most but not all SD models. Learning microworlds, like 'People Express' represent models that demonstrate interesting behaviour, that can be seen in a number of similar business situations. Therefore models like these can be used as 'case studies'.

OOP that is regarded as especially useful is the ability to create and delete objects during execution and simulation. Thus organizations can restructure themselves, patterns of information flow can change, liaison roles, task forces, etc. can be created and abolished dynamically. In addition, it is argued that inheritance can support the modelling of organizational structures which can change dynamically following the environmental uncertainty on internal information processing requirements. Finally, Blanning (1987) views the simulation of an OOP model as a way to generate and refine a series of hypotheses about the management of decision processes or the way in which individuals form communication groups within an organization.

In a similar strand McIntyre & Higgins (1988, 1989) discuss the architecture of an object oriented environment for organizational modelling. In particular in [McIntyre & Higgins (1989)] knowledge based representations of a stakeholders' positions are used, and simulated decision scenarios are formulated in order to access their impact. It is argued that OOP can enhance the construction not only of descriptive, data-oriented models, but also of active models which can be used to simulate target organizational activities. In [McIntyre & Higgins (1989)] the organization is represented as a hierarchy of personnel entities, business functions, information systems and projects. These are class objects, the instances of which model data, relationships and activities, associated to a specific organization. In the same work, the 'definition-simulation cycle' is defined. The idea is that during model definition, target systems entities are modelled as objects, while during simulation the model is instantiated and the specific decision environment is evaluated. When inadequacies of the model are found, redefinition and refinement of the objects take place, followed by more simulation.

In our view many of these ideas can be used in the area of industry simulation. Physical entities like competing manufacturing organizations, service organizations or government interventions can be modeled directly. Capacity in terms of production plants, vehicle or airplane fleets or even salesmen can be naturally represented. Chains of command and internal policies within an organization can be modelled to many levels of refinement; while sufficient level of detail can be naturally modelled, aggregated or disaggregated where necessary. General classes which represent types of organizations can be built, and specific views (eg. an investing organization, or a customer services organization) can be developed through inheritance and polymorphism. Aggregation can also be used, to model market structure, in a natural way. It can provide taxonomies that help the managers to use the model in terms of different levels of abstraction starting from the market as a whole at the top level and ending to specific plant subunits as a subassembly system or a specific managerial information chain.

Dynamic binding can be used to create dynamically new entrants in the market or allocate new capacity to existing organizations. Withdrawal of organisations from the market can be naturally modelled when specific market conditions hold (eg. a recession) and existing capacity can be taken away (for example under a specific decision rule or obsolete technology). External organizational policies can be modelled through message passing and the impact of specific inter-organizational policies can be investigated.

3.7 Addressing the Modelling of Time

Having discussed a framework for modelling structure and policies, we now need a concise way to represent time.

System Dynamics and its mathematical framework, control theory, are founded on differential equations, which is the formalism for describing continuous systems. In contrast discrete event simulations were based until now on the prime requirement of being able to program the computer appropriately and very often in an ad-hoc manner. Computer independent model description formalisms for discrete event systems paralleling the differential equations for continuous systems were late in coming.

However, our understanding of complex systems can be greatly enhanced if a mathematical formalism is used as a basis to model the system in hand. A formalism can provide a concise and structured basis upon which systems can be modeled and validated, while it can serve as a context in which initially vague solutions to a problem can be precisely specified.

Discrete event systems have many characteristics that differentiate them and make them more complex in nature than continuous ones. They evolve through a sequence of events, each occurring at a specific time. The intervals between events are not likely to be identical. At each event, changes take place in only part of the variables describing the state of the system, leaving the others unchanged. Although several events can occur at the same point in simulated time, they will actually occur one after the other in simulation. Therefore concepts like system states, events and the temporal aspects must be dealt properly requiring a sound mathematical formulation. The following table is an initial attempt to compare discrete and continuous systems (as System Dynamics views them).

CONTINUOUS SYSTEMS	DISCRETE SYSTEMS
levels	state variables
flows (rate of change)	transition functions
continuous change	events
global change	local change
input: piecewise continuous functions of time	input: arbitrary spaced events

Table 1: Continuous vs Discrete System Characteristics

In the following paragraphs we attempt to address the last question that we set in the introduction: i.e. the modelling of time under our specific objectives. The system dynamics continuous time representation makes sense for macroeconomic or environmental dynamic systems where the "view from above" holds. Nevertheless, most corporations and industries are man-made dynamic systems where the evolution of the system in time depends on the complex interactions of the timing of various discrete events. As a matter of fact, decisions within an industry are being taken in a discrete fashion and a number of decision support systems in the literature [for example see Moore & Whinston (1986), Widmeyer (1988), Sebastian (1990)], take this view in modelling in a fundamental way. In an industry simulation model, we see all the characteristics of a discrete event system, i.e. events happen in a discrete fashion when a participant in the industry makes a decision, and decision could only have localized effects in the system. We would like to argue that industry simulations (as have been defined previously) can be modelled more realistically in a discrete event framework, exploiting the advantages of a discrete event formalism, instead of the traditional System Dynamics one (we will support this argument further in the next chapter). In what follows we will review in general discrete event formalisms and discuss in detail DEVS [Zeigler (1976, 1984)]. We then suggest the reasons why DEVS can be advantageous to use in the place of a purely continuous time representation, within an Object Oriented industry simulation framework.

3.8 Discrete Event Formalisms

The quest for the development and use of discrete event formalisms in simulations has been intensified during the '80's. One approach has been inspired by the systems theory concepts [for example see Mesarovic and Takahara (1975)] and was elaborated in the work published by Zeigler [Discrete Event System Specification (DEVS) formalism (1976, 1984)]. Closely related formalisms have emerged, under the framework of Generalised Semi-Markov Processes (GMSP), which as formulated by Glyn (1989) as well as Cassandras & Stickland (1989), provide a formalism of Discrete event systems, as Markov processes with countable state sets. Zeigler (1990), however, has shown that DEVS is more powerful in terms of expressive power, than GMSP.

Another group of approaches has been based on Petri Networks. Van Hee et al (1991), present a modelling environment for decision support systems, based on a framework for formal description of discrete event systems called DES. DES is characterized by a finite state space and a behaviour that can be described by a succession of states. The formalism distinguishes three aspects of a discrete event system, the state space of the components, the state transformations of components and the interaction structure. These aspects are embodied in the system through a coloured Petri net [see Jensen (1987)]. It is suggested that the advantage of using Petri net theory, is that the structural properties of the systems modeled in such a framework, can be verified. It is claimed [Van Hee et al (1991)], without any proof however, that DES has at least as expressive power as DEVS. It should be noted that their approach provides a similar system static structure with DEVS based on the notion of the automaton[8]. However, the two approaches differ in the way the basic automata are coupled together to describe a dynamic system.

The graphical properties of Petri nets have also been used to provide an underlying framework for model building [see Kyratzoglou (1991) on how a Petri Net can be used to model decision-making and the structural attributes of an organizational], as well as their stochastic generalisations [Sanders & Meyer (1988)].

Another approach for modelling discrete event systems [Cohen (1990)], has been used for evaluating the performance of flexible manufacturing systems. This technique is based on minimax algebra in which the sum of two numbers is defined as the larger of the two, and their product as their sum. Since discrete event systems can be characterized by the starting and ending times of various activities, the minimax algebra can be used to calculate the times required for various manufacturing operations.

From the fore-mentioned approaches DEVS and DES have been implemented in a complete fashion, i.e. have been coupled with a language for model specification, and a software environment for editing and validating system descriptions.

3.9 The DEVS formalism

Discrete Event System Specification is one of the basic formalisms for discrete event modelling. It provides a formal representation of discrete event systems capable of mathematical manipulation just as differential equations serve this role for continuous systems. The formalism has been used to support the design of computer architectures, communication networks and multi-robotic systems [Rosenblit et al (1990)]. The compatibility between object orientation and discrete event world view formalisms has been discussed elsewhere [O'Keefe (1986)]. Up to date applications of the formalism

[8]Automata are characterized by a state space, an input output set and a transition function.

have been concentrated on engineering type applications where simulation is used as a tool for assessing design choices. In addition, DEVS models can be used as a basis for event-based system control [Zeigler (1989)].

In the DEVS formalism one must specify: (a) the basic models (atomic models) from which larger one could be build, and (b) how these models are connected together in a hierarchical fashion (coupling procedure). Atomic models are defined as mathematical structures:

$$M = < X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta >$$

where:

X	:the set of external input values,
Y	:the set of output events,
S	:the sequential state set,
δ_{int}	:the internal transition function, dictating state changes due to internal events,
δ_{ext}	:external transition function, dictating state changes due to external events,
λ	:the output function, generating external events at the output,
ta	:the time advance function.

Under the constraints:

(i) ta is a mapping from S to the non-negative real numbers with infinity:

$$ta: S \rightarrow R^{+}_{0,\infty}$$

(ii) the total state set of the system specified by M is

$$Q = \{ (s,e) \mid s \in S, 0 \leq e \leq ta(s) \}$$

where s the sequential state, and e the elapsed time spend in this state.

(iii) $\delta_{int}: S \rightarrow S$

(iv) $\delta_{ext}: Q \times X \rightarrow S$

Atomic models can be coupled to form a multicomponent DEVS which is defined as a structure:

$$DS = < D, \{M_i\}, \{I_i\}, \{Z_i\}, SELECT >$$

where:

D : is a set, the component names,

for each i in D:

M_i : an component DEVS model,

I_i : a set of influencees of i,

and for each influencee i in I_i, and j in $\{M_i\}$:

Z_{ij} is the transition function (output function) from i to j,

SELECT finally is the tie breaking selector (i.e. selects which of the next events will be executed first in the case that more than one have the same scheduled time).

Intuitively, we can perceive an atomic model as a box with input and output ports through which all the interaction with the environment is mediated. A coupled model specifies how these ports of a number of atomic models are connected to each other in order to form a new model which can be employed itself as a component to a bigger model, and so on. If changes of a state of a component A can cause changes of state of component B, then A is an influencer of B and B is an influencee of A; an event is now associated with the state change of a component A. It should be pointed out that we are allowed to construct such hierarchical models because DEVS is closed under coupling[9] [for proof see Zeigler (1984) chapter 3]. Further insights into the DEVS formalism in relation to system theory ideas can be obtained by consulting Zeigler (1976, 1984).

It should be pointed out that (as we have mentioned elsewhere) the typical world views can be easily expressed as subsets of DEVS. The event scheduling word view can be most naturally mapped under DEVS. The DEVS state set S is represented into component states that consist of sets (s_i, σ_i) where s_i a state and σ_i the time left component. Each σ_i is non-negative and represents an event scheduled corresponding to component i. If $\sigma_i=\infty$ then component i is passive. In terms of implementation we can view this as having a single next events list ordered by time, where the time advance function is $ta(s)=\min\{\sigma_i\}$. Each time an internal event occurs $\delta_{int}(s_i)$, actions corresponding to processing the corresponding component occur. Actions may cause changes to the influencees of the active component. Each time an external event $x \in X$ occurs when the model has been in state (s_i, σ_i) for elapsed time e, and cause a transition to (s', σ') where $(s',$

[9] i.e. any composite system obtained by coupling components specified by the formalism, is itself specified by the formalism. This is the property that allows hierarchical model construction by recursive application of the coupling procedure.

$\sigma')=\delta_{ext}((s,\sigma),e,x)$, the event is said to be ignored if $s'=s$ and $\sigma'=\sigma-e$ (the only result is to update the time left component to account for the passing of elapsed time e), i.e. the model remains scheduled to undergo a transition from the same state as it was before. On the contrary an event which is not ignored is said to cause an interrupt and causes a state change and/or a rescheduling of the model's next internal transition.

3.10 Comparison Between Differential Equations and DEVS in the Simulation Context

Briefly, a differential equation system specification is a structure:

$$D = < X, Q, Y, f, \lambda >$$

where X: the input value set, Q: the state set, Y: the output value set, f: the rate of change function and λ the output function; subject to the constraints: X,Y, Q real finite dimensional spaces, f: $Q \times Y \rightarrow Y$, λ: $Q \rightarrow Y$. Composite models can be specified due to the fact that differential equations are closed under coupling.

In the computer simulation context, an obvious advantage of DEVS, is that the input, output and state sets do not have to be real numbers (numerical in general). This is an advantage in terms of naturalness because we can model directly qualitative types of states, input and output (especially under a symbolic manipulation language like Lisp).

When a differential equation system is simulated, it is in practice discretized for numerical integration. The time step must be carefully chosen to conform to the rates of propagation expressed in the original model. In particular, when a naturally discrete system is modelled in differential equations (or difference equations for discretization), too small a discretization will result in unnecessary recomputation of states that essentially do not change; too large a value risks missing events that could have occurred in the original system. In the System Dynamics context, Forrester (1968), has suggested the heuristic that dt must be less that one half, but greater than one fifth, of the system's shortest first order delay. Such a problem does not exist when the system is modelled as a DEVS.

Zeigler (1984) [see chapter 6] has compared the cellular automata[10], which provide the formal basis for the usual representation of partial differential equations models for computer simulation, with DEVCS[11]. The conclusion is that DEVCS has more expressive power than a cellular automaton, i.e. every cellular automaton can be simulated[12] by some DEVCS, but the converse does not hold. For the full proof and mathematical documentation of the fore-mentioned concepts see Zeigler [1984, Chapter 6.4].

3.11 The DEVS Simulation Environment

The DEVS formalism underlies a general purpose simulation environment based on the principles of the abstract simulation developed by Conception & Zeigler (1988), as part of the DEVS theory. The environment is based on the formalism for model description, a user language for model specification and a software environment for model editing and validating system descriptions. A detailed discussion this environment implemented in an Object Oriented version of Lisp (DEVS-Scheme) is reported in [Zeigler (1990)]. Three more implementations have been reported in the literature, one in Modula-2 by Linvy (1987), one in Smalltalk [Thomasma & Ulgen (1987)], and another in C++ by Kim & Park (1992). The similarities (dis-similarities) of the above with DEVS-Scheme, as well as the application areas and characteristics of these implementations, will be discussed in the next chapter.

It should be mentioned that one of the main advantages of the DEVS environment,

[10] A cellular automaton is represented by a triple <S, N, T>. Intuitively it can be seen as infinite check-board such that at each square is located a cell with state S. The neighbours of a cell located in square (i,j) are determined in the neighbourhood N, N is a finite ordered set of the neighbours. If we let a global state of the system, at time t, by assigning to each cell a state from S. T is a transition function which if applied simultaneously to each cell at time t, the system will move to a new global state at time t+1.

[11] Discrete Event Cell Space model (DEVCS) can be associated with a DEVS in a one-to-one correspondence.

[12] The definition of simulation that we use in this context, is that simulation related systems must be "close" in terms of behaviour and structure. Thus, in this case we require that there is a one-to-one correspondence between the cells of the simulator and the simulatee, and a mapping from the local state set of the one into the other.

especially in the context of building industry models, is that it provides a convenient basis for development of evolutionary models which adapt or change their internal structure. The reasons for that will be apparent as we sketch the DEVS simulation environment.

The DEVS environment is a hierarchical structure, the root of the structure is a DEVS entity, every entity has two types of children: models and processors. Models are further specialized to atomic models (atomic DEVS) and coupled models. Processors are specialized to Simulators, Co-ordinators and Root-co-ordinators. Simulations are carried out as follow: Simulators and coordinators are assigned to handle atomic models and coupled models in a one-to-one fashion, respectively. A root-co-ordinator manages the overall simulation and is linked to the root co-ordinator at the outermost coupled model. Simulation is carried out by message passing among the processors which carry information concerning internal and external events, as well as data need for synchronization.

As Zeigler (1987) points out the model/simulator separation is very advantageous because:

- (a) any model can be simulated by the simulator, since the interaction takes place only at interface level, achieving separation of the time handling mechanism and the actual model,

- (b) due to the separation of the simulator and the models, we can create a model base; the implication is that coupled models can be created easily in many different configurations, while models can be treated as knowledge (facts) in the A.I. sense, and

- (c) atomic models can be tested in a stand alone fashion, so that validation and verification are enhanced, and greater understanding of the model's behaviour can be gained.

It should be mentioned that there is now a clear trend towards simulation languages that separate model and simulator [eg. Kettenis (1992)].

Simulation is initiated by initializing the states of the atomic models and specifying the influencees of each atomic model (the result is a series of coupled models). Then the processor-model pair are defined. Processors are simulators in the case of atomic models and co-ordinators in the case of coupled models. The root-co-ordinator is assigned to the outermost co-ordinator in the hierarchy. As has been mentioned the model and the simulation can be defined through a user oriented language.

3.12 An Object Oriented/DEVS framework in Smalltalk

We have implemented the concepts of the DEVS theory in Smalltalk/V for *Windows*. Smalltalk and its pure object oriented features have allowed us to produce a natural synthesis of OO and DEVS. Two distinct Smalltalk implementations are presented in detail in the next chapter. The first implementation, consists of the hierarchy of models and processors discussed in the last paragraph, and follows the concepts discussed by Zeigler (1987). A new novel implementation has also been carried out, in such a way so that the object oriented paradigm is exploited in its full potential (the motivation for embarking into a new implementation view is discussed in the next chapter).

The environment under discussion can be viewed through two dimensions, the object oriented one and the DEVS one. We should point out that while we have used the DEVS-Scheme concepts, in our framework we place more emphasis on the use of object orientation for modelling the system domain, than Zeigler (1987, 1990) does. The main difference is that Zeigler has used the object oriented paradigm as an implementation platform for the DEVS-Scheme, while we use this paradigm in order to model inter-organisational decision-making.

3.13 Summary

We started off this chapter focusing on the research theme that we presented in the previous chapter, by identifying six key requirements related to a simulation environment for industry modelling. Using these requirements, we motivated the discussion about Object Orientation and discrete event formalisms (with a particular focus on DEVS), based on a number of open research questions regarding industry modelling. These questions evolved around (i) the ability to provide entity based modelling, (ii) the need for modular, reusable and extendible models, (iii) the requirement of modelling aggregation and generalization relationships along with the association relationships supported by System Dynamics, and finally (iv) the ability to represent time as discrete events.

We addressed some of these questions by exploring the latest advancement in software engineering and data modelling, and by discussing and assessing object orientation, as a natural and robust way to model the world. In the last part, we have attempted to address the fifth question that we set in the introduction, i.e. the modelling of time. We have argued that in simulating industry structures and policies, a discrete event formalism provides a more natural framework, both for the model builder and the user. Therefore, we have reviewed the main discrete event formalisms and we have sketched DEVS, one of the main discrete event simulation formalisms. We have also contrasted DEVS with the continuous systems formalism and discussed its advantages in terms of expressive power. It should be mentioned that the fact that a continuous system can be simulated by a DEVS means that even continuous situations can be modeled in DEVS, if that is necessary by the nature of the problem domain.

Our analysis has suggested that a synthesis between Object Orientation and DEVS can address the research questions set at the beggining of this chapter, from a theoretical point of view. Furthermore, our analysis suggests that such a synthesis can be most natural, in bringing together the complementary modelling powers of OO and DEVS. In the following chapters of this thesis, we put into practice these theoretical concepts, with the aim of evaluating their practical value in answering the questions of our research agenda.

Chapter 4

OO/DEVS:
A Smalltalk Implementation of the DEVS Formalism

Contents:

4.1 DEVS Implementation Views

Zeigler (1987, 1990) [see also (Rosenblit et al 1990)] has used the DEVS formalism as the foundation for a general purpose simulation environment, based on the principles of the abstract simulator developed by Concepcion & Zeigler (1988), as part of the DEVS theory. This DEVS-Scheme simulation environment, developed in a Lisp dialect, separates models (entities of the system modelled) from the mechanics of carrying out the simulation. It introduces two generic classes of objects, *models* and *processors*. Figure 4.1. depicts the object hierarchy diagram of the DEVS-Scheme implementation (a more elaborate version of this diagram can be found in Zeigler (1990) p. 60). *Atomic* and *coupled-model* are specialisations of *model,* whilst *simulator* and *co-ordinator* are specialisations of *processor*. Further specializations of *atomic* and *coupled-model* (that cover special modelling requirements) have also been developed under the DEVS-Scheme implementation (see Zeigler (1990) Chapters 8 & 9). Simulations are carried out as follows: Simulators and coordinators are assigned to handle atomic models and coupled models respectively, in a one-to-one fashion. A root-co-ordinator, a special type of processor, manages the overall simulation and is linked to the co-ordinator of the outermost coupled model. Thomasma & Ulgen (1987) take a similar view in their Smalltalk-80 implementation, while in the Linvy (1987) implementation no distinction is made between atomic and coupled models.

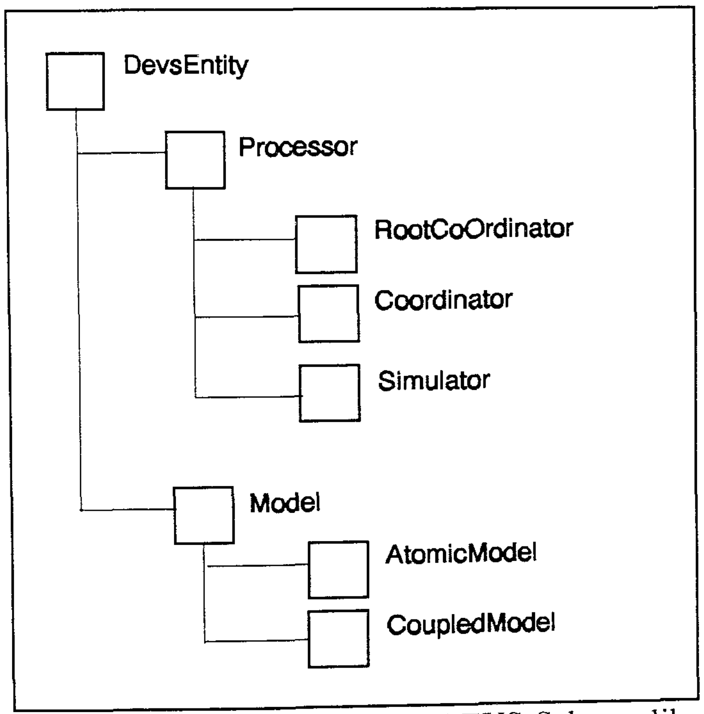


Figure 1: Object Hierarchy in DEVS-Scheme like implementation

As we have discussed in the previous chapter, simulation is carried out by message passing among the processors which carry information concerning internal and external events. Time is represented through the event scheduling world view. The DEVS state set is represented into component states that consist of sets (s_i, σ_i) where s_i a state and σ_i the time-left component in that state. Each σ_i is non-negative and represents a scheduled event corresponding to component i. If $\sigma_i=\infty$ then the component i is passive. Each time an internal event occurs $\delta_{int}(s_i)$, the corresponding model component is processed. Actions may cause changes to the influencees of the active component. Each time an external event $x \in X$ occurs, the event is said to be ignored if the model remains scheduled to undergo a transition from the same state as it was before, otherwise it is said to cause an interrupt and causes a state change and/or a rescheduling of the model's next internal transition.

In Zeigler's implementation (1990), every DEVS-model communicates with its world through a set of input and output ports. The addition of the concept of a port, represents an extension of the original Discrete Event System Specification as formally defined by Zeigler (1976), and was introduced by Linvy (1987), who has produced a DEVS implementation using this concept. The advantage of the port structure, is that it enables the modeller to represent the coupling specification within two DEVS as a mapping from the output port of one DEVS to the input port of the other. Such a mapping preserves the autonomy and structural independence of the two systems and thus leads to modular and extensible multi-component models. However, the drawback of the port structure is that while it is a natural representation for computer (networks, etc) modelling, it is not ideal for other systems.

In Zeigler's (1990) implementation models communicate through messages that are triples of the form: Message=< source, time, <port, value> >. The fields of these triples correspond respectively to the source of the message (a DEVS model), the time that it was send, the port that it was send from and a value that the message is carrying. Four types of message facilitate the message passing between *Processors*: *, x, y and done message. When a *Coordinator* of a *Coupled Model* receives a message the coupling scheme is consulted, and the message is translated and dispatched accordingly. When a *Simulator* of an *Atomic Model* receives a message two things can happen. If the message is a *-*message* it means that the model is scheduled to undergo its next internal transition. This causes the triggering of the *output function* that produces output into the output ports, and the triggering of the *internal transition function* that changes the state of the model. If the message is an *x-message* then the model is about to receive input in one of its input ports. Then the *external transition function* is triggered and in accordance to the port that the input is placed, some specific operation is performed. Overall, the external transition, internal transition and output functions are handled by the simulator of an atomic model. They provide a mapping between the ports of the model and the operations that the model can carry out. It should be noted that all the characteristics of the DEVS-Scheme have been faithfully transferred in a C++ implementation by Kim & Park (1992).

In what follows we will discuss our Smalltalk implementation of the DEVS formalism to which we will refer from now on as OO/DEVS. In section 4.2 we discuss the reason that we have chosen Smalltalk as our implementation platform. In section 4.3 we demonstrate an initial DEVS-Scheme like implementation that has been used in the U.K electricity investments model and presented in Ninios et al (1993). In section 4.4 we discuss the drawbacks of the DEVS-Scheme like view, and we introduce our final OO/DEVS implementation, which attempts to address them, by exploiting fully the concepts of object orientation, and the characteristics of Smalltalk in particular. We conclude this chapter by demonstrating in a small example how the constructs provided by OO/DEVS can be used in model building.

4.2 The Use of Smalltalk

Given the acceptance of the compatibility between OO and discrete event world view formalisms [O'Keefe (1986)], as well as our research objective of testing the usefulness of OO concepts in industry modelling and simulation, the choice of an object oriented language as the implementation platform for OO/DEVS becomes evident. However, it is important to explain the selection of Smalltalk, as the implementation platform. As we have discussed at the beginning of Chapter 3, there are several Object Oriented Languages that we could have used. However, Smalltalk is the purest object oriented language. The designers of Smalltalk have pushed the paradigm to the limits, in a way that everything within Smalltalk (and Smalltalk itself) can be seen as an object. In that way, the basic concepts of encapsulation, message passing, inheritance and polymorphism are enforced by the language itself. As a result Smalltalk provides an ideal validation platform regarding the proper use of these concepts in modelling within DEVS.

In addition, we can be satisfied that we use in full the modelling of generalization (taxonomic) relationships. It should be mentioned that, as we will see latter in this chapter, the entity modelling and message passing can prove ideally compatible with the DEVS modelling view. In that respect we aim to use the Smalltalk message passing protocol to model communication between DEVS models.

One of the great advantages of Smalltalk is that it provides a complete windowing environment, which can be tailored in order to build a user-friendly user interface. The current version of OO/DEVS has been developed in Smalltalk/V for Windows. Full Windows compatibility provides the additional advantage, of being able to link code, written in Smalltalk, to other windows applications. Finally, Smalltalk is a powerful prototyping environment, an important fact for research projects, as application prototypes can be easily built and altered.

4.3 A Faithful DEVS-Scheme Implementation in Smalltalk

4.3.1 Class AtomicModel

As we have discussed in the previous chapter, in the DEVS formalism one must specify: (a) the basic models (atomic models) from which larger ones could be build, and (b) how these models are connected together in a hierarchical fashion (coupling procedure).

In what follows we discuss how DEVS atomic models are realized in a Smalltalk implementation, that takes a similar view to Zeigler's (1990) DEVS-Scheme. Following the concept of encapsulation (see Chapter 3.2), we view each atomic model as a black box that receives input and produces output, both through a specified interface which mediates the interaction of that black box with its environment. A good example is that of a radio receiver. As users we do not know its inner workings, nevertheless, we can send input to the receiver by turning its station and volume buttons, and we can receive output from its speakers. Therefore, the buttons and the speakers constitute the interface of the radio receiver.

Regarding an atomic-model, when external events, arising outside the model, are received on its input ports, the internal model description must determine how the model responds to them. In addition, internal events arising within the atomic-model, change its state, as well as manifest themselves as events in the output ports to be transmitted to other model components. In the Smalltalk implementation, a basic atomic-model contains the following:

- A set of input ports, that constitute its interface regarding external events. This set is associated to a number of its methods in a one-to-one fashion.

- A set of output ports, that constitute its interface regarding the way that the atomic model can communicate with the rest of its model world. This set is associated to a number of its methods in a one-to-one fashion.

- The set of state variables and parameters. This set contains at least two state variables: phase and sigma, and a third variable e. The ordered pair (phase, sigma) represents the state of the atomic-model at any point in time. Phase represents the *current phase* of the atomic-model. For example, in a simple atomic-model, the variable phase may take the two values: active and passive. The variable sigma represents the *time left* in the current phase. For example, in the absence of external events the system stays in the current phase for the time given by the time left component, sigma. Finally, e represents the *elapsed time* that the atomic-model has been in a certain state (phase, sigma).

- The time advance function, which controls time regarding internal transitions. When the atomic-model is in a specific active state then the time advance function returns sigma, otherwise returns infinity[1].

- The external transition function, which specifies state changes after an external event has occurred. This is where the relationships input port-to-method are specified.

- The internal transition function, which specifies how the atomic-model will change state, after the time specified by the time advance function, has elapsed.

- The output function, that generates an external output, through the output ports, just before the internal transition function is activated. This is where the relationships method-to-output port are specified.

We have implemented an atomic-model through the class *AtomicModel.* This is an abstract class, that contains the fore-mentioned functionality. In that respect, we never use direct instances of the class, but we apply inheritance to create problem specific atomic-model subclasses, we then instantiate. In what follows we describe the main variables and methods of the class *AtomicModel*:

Variables: *phase, sigma, e* (as above)

Methods:

new
Creates a new instance of an atomic-model.

initialize

[1] When the state variable sigma is set to infinity, that means that the atomic-model is in a passive state, and is not scheduled to carry out an internal transition. If this is the case, its state can only change with an external transition.

Initializes the atomic model to the state (passive, infinity).

continue

Continues the current phase that the model is in. The only state change is on the sigma state component, that changes to: sigma - elapsed time.

passivate

Change the state variables of the model to: (passive, infinity).

holdIn: aPhase for: aTime

Sets the atomic-model to the state (aPhase, aTime).

timeAdvance

Returns the current sigma (i.e. a Real value or infinity).

sete: aValue

Resets the elapsed time of the current state to aValue.

internalTransition

Implemented by a subclass

externalTransition

Implemented by a subclass

4.3.2 Implementation of Coupled Model

As we have seen in chapter 3, atomic-models may be coupled to form coupled-models. A coupled model, contains the instructions on how to connect several component models (atomic or coupled) in order to form a new multicomponent model. A multicomponent DEVS is defined as a structure:

$$DS = < D, \{M_i\}, \{I_i\}, \{Z_i\}, SELECT >$$

where:

D : is a set, the component names,

for each i in D:

M_i : a component DEVS model,

I_i : a set of influencees of i,

and for each influencee j in I_i:

Z_{ij} is the transition function (output function) from i to j,

SELECT finally is the tie breaking selector (i.e. selects which of the next events will be executed first in the case that more than one have the same scheduled time).

As we can see by the above definition, a coupled model contains the following information:

- A set of DEVS models (atomic or/and coupled) and their names

- For each of these models its influencees

- A set of input ports through which external events are received, and a set of output ports where the model places its output after a transition.

The coupling specification, which consists of:

- The *external input coupling*, which specifies the connections (a set of influences) between the external input interface and the input interfaces of the coupled model components.

- The *external output coupling*, which specifies the connections (a set of influences) between the external interfaces of the component models, and the external interface of the coupled model. This allows output generated by a component model to be transmitted externally.

- The *internal coupling*, which specifies the connections (influences) between the coupled model components. This allows internal communication between the model components of a coupled model.

Within a coupled model time management is based on the principle that the component that has minimum time of next event should carry out its transition first. Nevertheless, there are cases that two or more component models may have the same time of next event. To resolve this conflict Zeigler (1990) has introduced a SELECT function. If a conflict exists, this function embodies the rules that direct which component should go first. In our Smalltalk implementation we follow the same route and we also use such a select function.

We have implemented a coupled model through the class *CoupledModel.* We should point out that Zeigler and his colleagues have created a number of problem specific coupled models (see Zeigler 1990, for eg. p 60). In our work we have used the *DiagraphModel* which is the most general and flexible type of coupled model. As a result class CoupledModel represents an abstract class, that we have only added for generality. The main functions of a coupled model, as we have described above, are implemented in the class *DiagraphModel.* This class provides the constructs to build coupled models, and in contrast to the class AtomicModel, it is actually instantiated so that every coupled model, in a specific model, is an instance of this class. The class *DiagraphModel* contains the following variables and methods:

Object DiagraphModel

Variables: *compositionTree, influenceDiagram, priorityList*

Variable *compositionTree* contains a set of pointers that refer to all the model sub-component models. Variable *influenceDiagram* contains quadruples of the form: (Influencer, Output Interface Specification, Influencee, Input Interface Specification). The variable *priorityList* is an array that contains the priorities between models, and is used by the select function when it is activated.

Methods:

new

Creates a new instance of a diagraph-model (i.e. a type of coupled-model).

initialize

Initialises the above variables and data structures

buildCompositionTree: anArray

Builds the compositionTree data structure from the model names contained in the array anArray.

setInfluenceFrom: aModel1 to: aModel2 from: port1 to:port2

Specifies the influences (internal coupling) from aModel1 to aModel2.

inputCoupling: aModel from: port1 to: port2

Specifies the input coupling, regarding a subordinate model aModel, from the port1 of the coupled model, to the port2 of aModel.

outputCoupling: aModel from: port1 to: port2

Specifies the output coupling, regarding aModel from the port1 of the model aModel, to the port2 of the coupled model.

getChildren

Returns a list of the subordinate model names

getInfluenceesOf: aModel atIterface: aPort

Returns the names of the influencees (internal coupling) of the subordinate model aModel, regarding aPort.

getReceivers: aPort

Returns the set of subordinate models that are linked to the port aPort of the coupled model, and should get its input.

translate

Provides translation from the coupled-model to the atomic-model interface *priorityList: anArray*

Specify the priority list

select

The SELECT function as specified above

4.3.3 The Class Model

All classes in the Smalltalk DEVS implementation are subclasses of the class *DevsEntity* (see Figure 4.1), which is a direct subclass of the outermost (in the class hierarchy) class Object. This is a class that contains the variables name (representing the name of the model) and infinity (corresponding to a number that represents infinity within the DEVS classes). Class *DevsEntity* has two direct subclasses, the classes *Model* and *Processor*. The classes *AtomicModel* and *CoupledModel* are subclasses of the class *Model*. In what follows we sketch the class *Model*.

Object Model

Variables: *clock, inports, outports, parent, position, processor, monitoredVariables*

Variable *clock* provides time monitoring facilities. Variables *inports* and *outports* provide the definition of the ports (interface) of models. Variable *parent* contains a pointer to the parent model (atomic-models have coupled-models as parents, coupled-models have other coupled-models as parents, and the outermost coupled-model has no parent). Variable *processor* contains a pointer to a the 'simulator' of the model (see Chapter 3.11 as well as the next paragraph). Finally, variable monitoredVariables contain a list of variables to be monitored during a simulation run.

Methods: This object contains a set of methods to access and assign the fore-mentioned variables. The most interesting ones are: the *monitorVariables*, which is used to specify which instance variables of a *Model* should be monitored, during a simulation run, and the method *viewMonitoredVariables* which creates a graphical representation of the monitored values of a specific variable.

4.3.4 The Processor Classes

The class *Processor* is a subclass of the class *DevsEntity*. As we have pointed out in Chapter 3.11 one of the advantages of the DEVS simulation environment, is that it separates models and simulation engine. The simulation engine is composed of three Smalltalk classes: *Simulator, Coordinator,* and *RootCoordinator.* All three classes are subclasses of the class processor. When a simulation model is initialized, instances of the classes *Simulator* and *Coordinator* are assigned to handle each atomic-model and each coupled-model respectively. An instance of the class *RootCoordinator* manages the overall simulation, and is assigned to the outermost coordinator. The only parts of an atomic model that a processor requires to know the existence of, are the internal, external and output functions. In what follows we present the basic characteristics of these objects:

Object Processor

Variables *devsComponent, parent, timeOfLastEvent, timeOfNextEvent*

The variable *devsComponent* contains a pointer to the DEVS model that the processor is assigned. Variable *parent* contains a pointer to the parent processor. Variables *timeOfLastEvent* and *timeOfNextEvent* contain the times of the last and next events within the scope of the processor, respectively.

Methods

new

Creates a new instance of an object processor *assign: aModel*

Assigns a specific model aModel, to an instance of the processor

parent: aProcessor

Sets the parent of the processor

Object Simulator

Variables none (all inherited by *Processor*)

Methods

initialize

Initializes the timeOfLastEvent and timeOfNextEvent, and triggers the initialization of the states of the assigned atomic model. Simulation is initiated by determining the timeOfNextEvent for each atomic-model.

star: aMessage[2]

A star method can be evoked by the parent processor (coordinator). When the star method is evoked, that means that the next internal event should be carried out within the scope of the atomic-model that is handled by the *Simulator* instance. The code within the star method: (i) Checks if the time carried by the *aMessage*, agrees with the timeOfNextEvent. If this is not true an Error Window is placed within Smalltalk, and the simulation stops. (ii) Calls the output function of the atomic-model that handles. (iii) Updates the times of last and next events as:

timeOfLastEvent := aMessage time[3]

[2] aMessage is an object that is an instance of the class MessageContent. This is a 'auxiliary' class that we have created in order to handle the message passing between different DEVS models. This class has variables *portName, value, source and time*, as well as methods to assign and return the values of these variables.

[3] method time returns the time value of the object *aMessage*

timeOfNextEvent := timeOfLastEvent + (devsComponent timeAdvance)[4]

(vi) Embeds the result of the output function to a *yMessage* that sends to the next higher level. (v) Finally, a *doneMessage* instance of the class *MessageContent* is created and is returned to the processor that has evoked the star message. This *doneMessage* carries the time the next event will be carried out within the scope of the atomic-model handles by the *Simulator* instance.

x: aMessage

The parent coordinator can evoke the *x* method. The x method represents the arrival of an external event within the scope of the simulator. When it is evoked the code in the x method: (i) Checks if the time carried by the *aMessage* lies between the *timeOfLastEvent* and *timeOfNextEvent*. This should be so because an external event should arrive before the next internal event and after the last external (or internal) event. (ii) Computes the elapsed time

e = aMessage time - timeOfLastEvent

(iii) Triggers the external transition function of its atomic-model. (iv) updates the times *timeOfLastEvent* and *timeOfNextEvent*, as above. (v) a *doneMessage* instance of the class *MessageContent*, which indicates that the state transition has been carried out, is created. This object reports the new *timeOfNextEvent* and is transmitted to the next level.

Object Coordinator

Variables *imminentChild, waitList*

Methods

initialize

assign: aModel

[4] As we have discussed earlier in this chapter, *timeAdvance* is one of the methods of atomic-model

x: aMessage

y: aMessage

star: aMessage

done: aMessage

Object RootCoordinator

Variables *clock* (this is the global simulation clock)

coordinator (a pointer to the outermost coordinator)

Methods

initialize

done: aMessage

In what follows we demonstrate how simulation is carried out, and the roles of the above *Coordinator* and *RootCoordinator* methods. Simulation is initiated through the initialization of the atomic-models. Their simulators calculate their *timeOfNextEvent*, and this time is propagated to the next level coordinator. This is where the role of the variable *waitList* comes in. Every coordinator keeps track of the times of next event propagated by its components by storing these times in its waitList. The variable *waitList* is a Smalltalk OrderedCollection of pairs <model, time>. When every subordinate model has reported its *timeOfNextEvent* then the *done:* method is evoked and the model with the minimum time, in the *waitList*, becomes the *imminentChild.* This is a variable that contains a pointer to the model with the minimum time of next event. Then the coordinator itself sends its *timeOfNextEvent* (i.e. the time of its imminent child) to its next level. The processes is repeated until the outermost level is reached. At that level, we have information regarding the *timeOfNextEvent* (this will be the minimum time within the whole model), and information concerning the identity of the imminent child at each level.

This is the point where the simulation cycle starts, as the star method of each imminent child at every level is evoked. Every coordinator responds to its star method by transmitting it to its imminent child. The coordinator places the imminent child in its (initialized) wait list. When an atomic-model is reached it responds to the star message (see the implementation of the object *AtomicModel*) mainly by transmitting a *yMessage* and a *doneMessage*. The *doneMessage represents* the time of next event to be entered to the *waitList*. The *yMessage* represents the 'result' of applying the internal transition function of the atomic-model.

This result is transmitted to other atomic or coupled models within the system by consulting the coupling specifications within the atomic-model's coupled model. So, when a coupled model receives a y-message, it employs the methods *translate* and *getInfluenceesOf: aModel atIterface: aPort* to obtain its children and its respective interfaces where the message should be sent. As a result, the *x: aMessage* method is evoked for each influencee. The argument *aMessage*, of the *x:* method, is a MessageContent instance, identical with the *yMessage* in all but the *source* variable which now contains a reference to the coordinator (i.e. the translated message is coming form the coordinator). The coordinator adds to its wait list all the influencees that sends x messages to. The atomic models that receive these x messages, are also placed in the *waitList*. These atomic-models also schedule themselves (see the definition of the class AtomicModel) to undergo a state transition due to the external event corresponding to the *xMessage*. As a result a series of *doneMessage*s return to the *waitList* of the coordinator. When every model contained within the coordinator has responded with a *doneMessage* the minimum time of next event is calculated and a new *imminentChild* is set. This new *timeOfNextEvent* is propagated to the next level by each coordinator in the model, and a new simulation cycle starts. Simulation can be stopped when all subordinate models have been passivated (this is done by assigning infinity to the sigma state variable of each model). As a result, the *timeOfNextEvent* of the outermost coordinator is infinity and the *RootCoordinator* ends the simulation.

4.3.5 The Class SimulationPlatform

This is the class that manages the overall simulation environment. It contains methods to support the following facilities:

- Model Building
- Model Browsing
- Model Saving & Retrieving
- Graphical Model Representation
- Graphical display of the results

Its method *open*, opens up the main window which can display the graphical representation of a model. With its browse methods one can browse through the subordinate models of a model, by double-clicking their iconic representations, and evoking windows that contain their instances or their class representation. This class supports also, the graphical representation of results, because it is linked to the subordinate models that can evoke their *monitorVariables* and *displayMonitoredVariables* methods. It should be mentioned that the graphical representation of results is supported by the auxiliary class *Diagram* that takes care of window display and graph re-scaling and drawing.

4.3.6 The Class DevsModel

As we have already shown, an overall DEVS model is a set of Smalltalk objects, that are either atomic or coupled models. As soon as we have specified the basic atomic-models we can start connecting them, creating aggregations, by placing them in a coupled model. In addition, we can also specify influences (association relationships) between them. As a result a DEVS model should be viewed in two dimensions, the first one is this set of basic-atomic models and/or their aggregated versions as coupled models. The second one is the specification of the set of influences between these models. An instance of the class *DevsModel* contains within its methods the necessary code that specifies the instances of these atomic models (with their initial instance variable values) and the influences between these models. For that reason we will alternatively refer to the subclasses of *DevsModel* as the topology object, in order to denote the fact that it contains the instance and influence specifications that define a DEVS model, as a whole. The class *DevsModel* contains two methods:

doIt

is the method that contains (i) the code that specifies the instantiation of the atomic-model subclasses (eg. instantiation of the class company), (ii) the specification of the coupled models (aggregation relationships), and (iii) the specification of the influences between subordinate models. In the current implementation all these influences are specified in a procedural fashion. Our research aim is to provide the tools that can facilitate graphical specification and representation of these influences. As we will show in Chapter 5, we have developed a set of diagramming tools, which can facilitate graphical modelling of aggregation, generalization and association relationships.

runModel

This is the method that is responsible for initializing the simulators within the model. As we have discussed above, this the way to start running a simulation. This method is evoked by the object *SimulationPlatform* when we choose to simulate a model.

4.4 Modelling Object Oriented Message Passing within DEVS

4.4.1 The DEVS-Scheme Implementation from a Critical Perspective

In Chapter 3 we discussed extensively the nature of OO design, setting as one of our research tasks to explore how OO can contribute to industry modelling and simulation. Taking a critical view in the design of a DEVS-Scheme like simulation environment, we can identify a certain incompatibility with the classic OO view.

This incompatibility, is driven by the design of the interface of a DEVS model. This interface can be viewed in two dimensions: (a) as an interface between the model and its simulator, and (b) as an interface between the model and its external model world. In dimension (a) the elements that constitute the interface are the internal transition, external transition and output functions. The simulator of an object has only knowledge of these three functions and manipulates them in accordance to the type of event is handling, at each point in time. In dimension (b) the components of the interface are the input and output ports, that are specified for each model (atomic or coupled). Within the external transition and output functions, the modeller has to specify port-to-method mappings. These mappings should be explicitly constrained by the state set.

From an OO design viewpoint, this modelling approach: (i) perplexes the issue of the 'software' interface of an object (which should only be its publicly accessed methods), as it introduces two more semantically different types of interface; (ii) confuses the visibility of the 'user' interface of the object, as the definition of what the object does is specified procedurally within the three DEVS functions, and not by its methods; finally, (iii) the concept of the port is redundant, as it coincides with the concept of the method selector in OO environments. As a result a number of fundamental drawbacks can be identified in the DEVS-Scheme like simulation environment:

(a) ***The reusability of a model is limited***, since is sacrificed in order to follow faithfully the semantics of the formalism. As far as the object's functionality is practically defined in the external transition, internal transition and output functions, it is quite difficult to use inheritance (one of the most powerful concepts of OO) as the means of producing subclasses of the model. The modeller has to practically rewrite the above functions each time a subclass of a model is defined. It should be noted that this is a drawback that has been identified by Zeigler (1990) who has addressed the problem by producing a subclass of AtomicModel called ForwardModel. A Forward Model is a rule based model that exploits the forward chaining paradigm to evaluate its rules. As it is pointed out: "The ability to write models using rules provides an *additional level of decomposition or granularity to model specification.* Until now the smallest meaningful chunks of a model were the basic functions: internal transition, external transition and output function. *Atomic Models could share these functions but they could not share smaller parts of them.* Rules as more granular knowledge units, make it possible to ''mix and match'' specifications among models." [Zeigler (1990), p. 210, the emphasis is ours]. Our proposition is that a redesign of a DEVS model, that exploits the characteristics of OO will alleviate this problem, as the smallest meaningful chunks of a model (object) will be its methods. Rule based models could then be build, but with the objective to exploit the intrinsic characteristics of the rule based paradigm.

(b) ***The property of polymorhism is violated*** due to the distinctions between coupled and atomic models, as the modeller ends up using two 'model' object, which however have a completely different interface, and different factionality through out the model building process. Within the DEVS theory every coupled model is a DEVS model (closure under coupling property). If this concept is interpreted in a true OO fashion, it can be implemented in such a way that the modeller can use a coupled model in exactly the same way that uses an atomic model, and vise versa.

(c) ***The naturalness of OO is compromised***, as the interface of the object is not any more its method, but the internal transition, external transition, and output functions. This makes modelling less transparent, as the user has to examine the code written in these functions to understand the behaviour of the model (object). Nevertheless, a pure object oriented design would identify the main actions (responsibilities) of each object and model them through methods and message passing [see Wirfs-Brock & Wilkerson (1989); Gibson (1990)]. Moreover, the main attraction of such an approach is that a DEVS-model would be more transparent, as its main functions would be identifiable by looking at the interface and not the specifications of the three fore-mentioned functions.

It is the objectives set by these drawbacks that have mainly driven our second (and final) Smalltalk implementation, where in contrast to earlier implementations, a fundamental objective was to exploit fully the naturalness and modularity provided by OO. The main view that we have taken, in satisfying our objectives, was to exploit fully the message passing concept of OO.

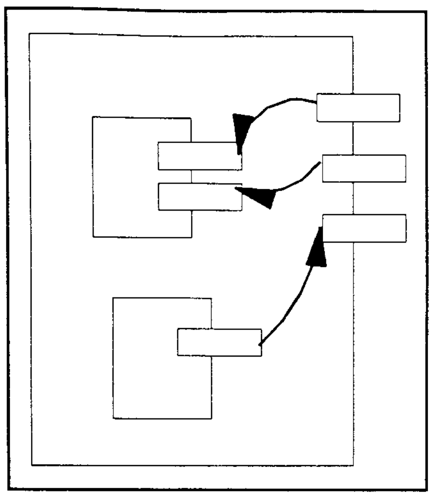


Figure 2.: Aggregation in OO

As we mentioned above, the way that objects communicate in object oriented environments is message passing. Each object has a set of methods that are made up of selectors and arguments. A message expression describes a receiver, a selector and possibly some arguments. Every object can request from another object to execute one of its methods through a message expression [Goldberg (1983)]. However, using the message passing paradigm within a DEVS implementation is not that straight-forward. While we would like objects to communicate with each other, the communication should be performed through the coupling mechanism. This restriction effectively means, that some objects should be able to invoke only some methods of some other objects in a model.

The modelling question here is whether or not we can exploit directly the object oriented view and remove the DEVS functions without loosing the DEVS functionality. In addition, the fore-mentioned restriction posses the additional design issue of how to model object oriented like message passing, at the coupled-model level (aggregation level).

4.4.2 Towards an Object Oriented DEVS Implementation

From now on by referring to 'the implementation' we imply our second OO/DEVS implementation. References to our first DEVS-Scheme like implementation, will be distinguished as the 'first implementation'.

In our OO/DEVS implementation we first tried to address the issue of using OO type message passing, skipping the three types of functions implicit in DEVS, as well as

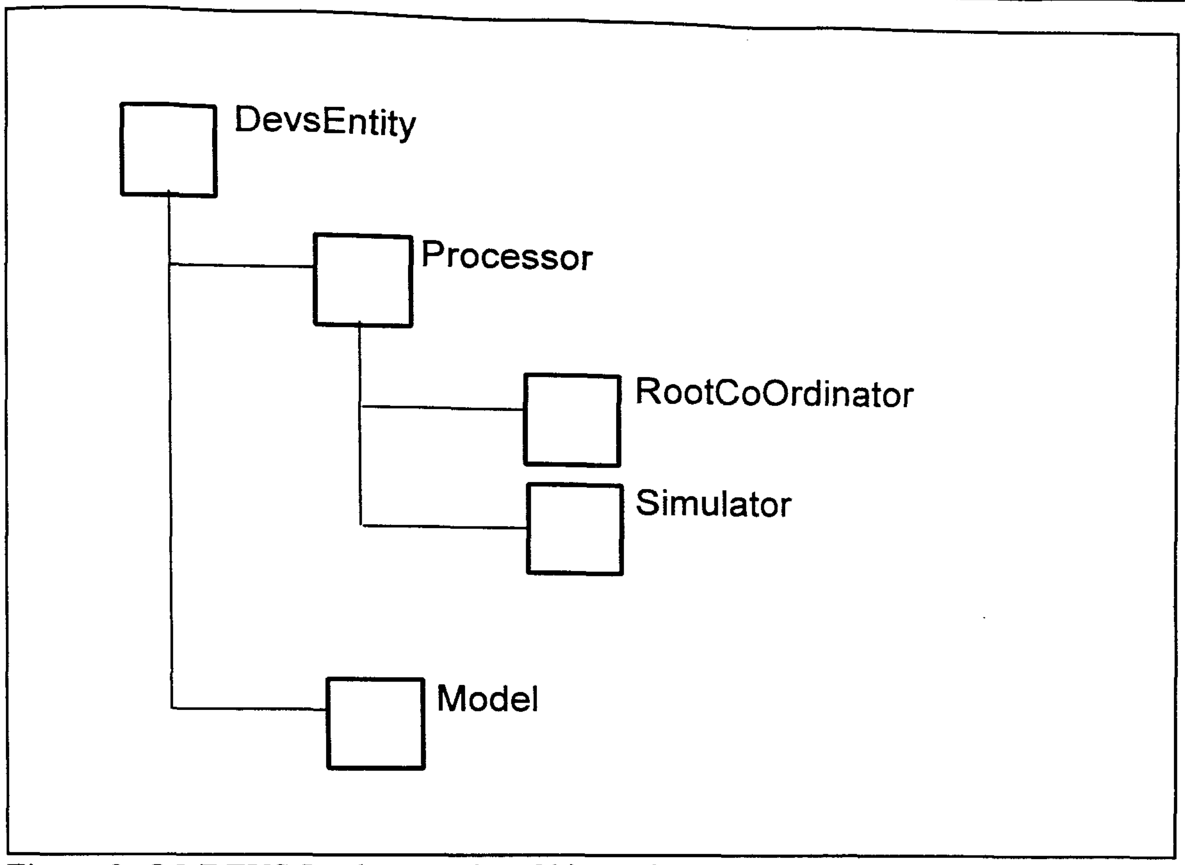


Figure 3: OO/DEVS Implementation Object Hierarchy Diagram

bypassing completely the concept of the port. This objective resulted into a design that literally transfers the responsibilities of the internal transition, external transition and output functions to the simulator of an atomic-model. It should be pointed out, that a fundamental design consideration, in order to achieve this, is related to the way we represent messages within the OO/DEVS environment. Our design decision was to model messages as instances of the class *SimulationMessage* which is a subclass of class *Message* of Smalltalk. As a result, a SimulationMessage can be represented by a structure:

<receiver, selector, arguments, source, method, time>.

Variables selector, receiver and arguments are inherited from class *Message*. The variable source contains a reference to the DEVS-model that is the sender of the message. The variable method contains the name of the method that send the message. And finally, time is a variable that contains the simulated time that a message was sent.

4.4.3 The class Model

Figure 4.3. depicts the main objects of the OO/DEVS implementation. As can be seen in figure 4.3. the top level object, in the hierarchy, is the object *DevsEntity*, this object is identical to the *DevsEntity* object of the previous implementation (see section 4.3.3 for details). Two subclasses stem from this object, subclass *Model* and subclass *Processor*. Object Processor has two more subclasses *Simulator* and *RootCoordinator*.

Under this implementation, objects *Model* and *Simulator* represent the most important object classes in the system. The advantage of providing a unified view of Model (by retaining the functionality of atomic and coupled model at the same level) is that the framework benefits by the property of being able to operate at any level using the same set of constructs, and thus utilizing the concept of polymorphism. This view is also consistent with the concept of aggregation, due to the fact that a coupled model is also a model. The modeller should have the ability, to use the same operations at the model and coupled model levels.

Class *Model* provides the constructs to specify new methods, influences between two models (by specifying method-to-method relationships), and message structures for model output. This is a class that can be either instantiated, in order to produce problem specific models (objects), or used directly to represents model aggregates. Each instance of Model also contains a *messageList* that accommodates a set of messages to be triggered by the simulator, when a specific method of the Model has been triggered. As we will discuss and demonstrate further on, the contents of the *messageList* are specified by the modeller upon model specification in the initialization method.

Every instance of Model (or of a subclass of *Model*) is an object with some specific functionality (for example a company that invests and retires production capacity), that has a (possibly empty) list of children. The modeller is given the ability to specify which methods of the subordinate models are methods of the aggregate model, and therefore provide selected functionality (methods) of the sub-models at the aggregate model level. This view of aggregation of DEVS models is compatible with the emerging view of aggregation in object orientation [see Graham (1993) for the layers concept in object oriented design]. For example in figure 4.2. objects A and B represent DEVS-models, where B has two methods that process input, and A has one method that process output. Model C is an aggregate version of A and B that can respond to messages which have the same receivers as the methods of A and B.

It should be pointed out that such a view does not violate the DEVS definition of either the atomic or the coupled model. In the case of an atomic model (which will be an instance of a subclass of *Model*) two disjoint sets of methods will correspond to the external and internal transition functions, while their selectors will accommodate the X and Y sets correspondingly. At the coupled model level, M_i is an object corresponding to a component DEVS-model, I_i is a set of messages that the coupled model can respond to (that would belong to the influencees of i, and finally Z_{ij} will be the i-to-j output translation and would be specified as method-to-method relationships in the form of quadruples: <from_Model, from_Method, to_Model, to_method>. In what follows we describe the main variables and methods of the class.

Object Model

Variables: sigma, e, compositionTree, influenceDiagram, priorityList, parent, processor, clock, position, monitoredVariables, selectors, messageList

Variables *sigma* and *e* are exactly the same as in the first implementation (see section 4.3.1). It should be noticed variable *phase* becomes redundant, as the phase that the model is in, is represented by the imminent method (i.e. the next method of the object to be triggered). Variable *compositionTree* contains pointers to the subordinate models of the instance of *Model*. An instance of Model that has a nil compositionTree corresponds to an AtomicModel of the first implementation, while an instance that contains entities, corresponds to a CoupledModel of the first implementation. Variables *influenceDiagram* and *prioriryList* are the same as the corresponding variables in object CoupledModel (see section 4.3.3), the only difference is that in this implementation the *influenceDiagram* contains object-to-object, method-to-method relationships, as specified in the previous paragraph. Variables *parent, processor, position, clock* and *monitoredVariables* contain references to the parent of the model, the processor attached to the model, the position of the model on the screen, the current clock time, and a list of model variables to be monitored through a simulation run, respectively. Variable *selectors* contains all the selectors that the model can respond to. Finally, variable messageList contains a list of messages, as have been specified above. This list of messages is visited as soon as a method is performed, and specifies which methods, within the scope as well as outside the scope of the model, should be triggered next.

Methods

Note: Methods that are followed by ***(interface)*** constitute the modelling interface of the object and should be used in building OO/DEVS models. Methods that are followed by ***(overload)*** can be redefined (overloaded) in the subclasses of *Model.* The rest of the methods are private.

addAllSelectors

This method should be used in a composite model in order to add all the selectors of its sub-models to its selector list. The result is that the model can understand and respond to incoming messages that carry one of its selectors.***(interface)***

addSelectors: aModel

Adds all the selectors of a subordinate model aModel to the Model.***(interface)***

addSelector: aSymbol

Adds the selector aSymbol to the Model.***(interface)***

addMessage: aMessage

Adds a simulation message, aMessage, to the messageList.***(interface)***

buildCompositionTree: anArray

Builds the compositionTree data structure which contains references to the subordinate models, as specified in the array anArray. ***(interface)*** *canTranslate: aMessage*

Checks whether of not the parent model can understand the method that triggered the message aMessage, and if aMessage can be translated in the super-ordinate model.

getInfluencees: aMessage

Returns a list of the influences that can understand aMessage, if translated.

getReceivers: aMessage

Returns a list of models that understand the selector specified in aMessage.

initialize

This is one of the most important interface methods of *Model*. At the level of *Model* it initialises the fore-mentioned instance variables (sets the clock to zero, etc). It should always be overloaded in subclasses of Model. In the overloaded method the statement: s*uper initialize* should be executed first. Then local variables should be initialised and the message protocol should be specified. An example of this process, will be given in the last section of this chapter.***(interface) (overload)***

priorityList: anArray

This method should be used to specify the execution priority of the subordinate models (when they exist). The array provides references to the models and the indexation of the array provides the priority [priority of 1 (index 1) is greater that priority of 2 (index 2)].

processor: aProcessor at: aPoint

Assigns a processor to the model (so that it can be simulated), and places the model at the point aPoint on the screen.***(interface)***

respondsTo: aSymbol

Returns true if the model has as one of its selectors aSymbol, else returns false.

select: aMessage1 comparingTo: aMessage2

This is the DEVS select function. Selects the imminent message between two messages. ***(overload)***

setInfluenceFrom: aSelector1 to: aModel withSelector: aSelector2

Sets an influence (variable influenceDiagram) from the instance of the class Model to another instance: aModel, and from the method with selector aSelector1 to the method with selector aSelector2 of aModel. ***(interface)***

timeAdvance

This the DEVS time advance function, its functionality remain the same as in the first implementation.

A number of additional private methods are also declared within the scope of the object. However, their presentation does not provide any further understanding of the nature of the object *Model*, and we do not therefore present them herein.

4.4.4 The Object Processor

Figure 4.3. depicts the object hierarchy of the OO/DEVS implementation. The simulation capabilities of the framework are built in the object Processor and its subclasses: Simulator and RootCoordinator. Object RootCoordinator has exactly the same specification as in the first implementation and we do not therefore present it in this section. It should be noted that no object coordinator exists, as in the first implementation. This steams from the design decision to encapsulate the functionality of atomic and coupled models in the object Model. Consequently, there is no need for two different types of processor to handle the two types of models.

Variables parent, devsComponent, timeOfLastEvent, timeOfNextEvent

Variable *Parent* contains a reference to the parent processor, while variable *devsComponent* contains a reference to the DEVS model (an instance of class Model), that is associated to the processor. Finally, variables *timeOfLastEvent* and *timeOfNextEvent* contain the time that the simulator performed its last event and is scheduled to perform its next event, respectively.

The methods of the processor are all private and are designed to manipulate the above variables, as well as deal with the notion of infinity in time, that represents the notional time that a passive object is scheduled to perform its next event.

4.4.5 The class Simulator

This class, along with class Model, represent the most important classes of the OO/DEVS implementation. In the first implementation the interface between an atomic model and its simulator was its external transition and output functions. The simulator was dispatching information to the model (object) by triggering these functions. In the OO/DEVS implementation, the simulation capabilities encapsulated in these functions, have been transferred to the simulator of a Model (an instance of class *Simulator*). As in the first implementation, and in the spirit of the concepts of the abstract simulator within the DEVS theory, *Simulator* is a generic object, instances of which can be coupled to any instances of class Model or its subclasses. In what follows we examine the functionality of these methods, as well as the main variables of the object *Simulator*.

variables waitList, imminentMessage

Variable *waitList* is used in a similar manner to the waitList variable of the object Coordinator of the first implementation. However, in contrast to the Coordinator one, it does not include pairs of the type <model, time>, but instances of *SimulationMessage*. As we have already discussed the fundamental difference between the two implementations is that the final OO/DEVS implementation is based on the notion of the SimulationMessage. Objects communicate by sending simulation messages to each other. In that respect the simulators have to deal with these messages, distinguish their senders and receivers, calculate which message should be send at every point in time, and dispatch the messages accordingly. As a result the concept of the *imminentChild* (see section 4.3.4), which was a reference to the imminent subordinate model of each model, has been substituted by the *imminentMessage*. This is a reference to the message that should be triggered first (naturally by the specification of the SimulationMessage this reference includes a reference to the model that will perform the message).

methods

Note: all methods are private. The most important methods of object *Simulator* are:

initialize: aMessage

Triggers the initialize method of object *Processor* and initializes the devsComponent of the *Simulator* instance, so that it will start its simulation life-cycle by sending the message *aMessage.*

perform: aMessage

This is triggered when its DEVS-model is imminent, i.e. the message aMessage should be triggered next in the scope of the devsModel linked to the simulator. Perform carries out the following sequence: (i) Checks if the time reference in the message aMessage is the same as the timeOfNextEvent. (ii) If that devsModel has sub-models, perform removes the aMessage from the waitList and triggers the perform message of the sub-models that can understand the aMessage. (iii) If the devsModel has no children, perform triggers the imminent method, then (vi) traverses the *messageList* to output all the messages linked to the triggered method, and (v) changes the timeOfLastEvent to the time carried by the message aMessage, and the timeOfNextEvent using the timeAdvance function of the devsComponent, finally (vi) done is triggered. Notice that the expressive power provided by such a view is equivalent to the atomic-model internal transition function, as discussed in the DEVS-Scheme. This can be shown by scheduling a message to self at time zero, this can produce the same effect as the sequence *output function - internal transition function*, ie. change the state of the model after output.

input: aMessage

Is triggered to dispatch an incoming message to the associated DEVS-model. The code within the input method: (i) checks that the time carried by aMessage is between the timeOfLastEvent and the timeOfNextEvent. (ii) checks whether or not the devsComponent of the Simulator has sub-components, if it has (iii) sends the aMessage to the valid receiver sub-components, or (iv) if it has no sub-components, lets its devsComponent perform the message aMessage, (v) traverses the messageList of its devsComponent so that any relevant messages (as a consequence of triggering a method of the devsComponent) will be send, and (vi) changes the timeOfLastEvent and timeOfNextEvent (using the timeAdvance function of its devsComponent). Finally, (vii) method done is triggered.

output: aMessage

Is triggered when its DEVS-Model sends an output message. An output message can be either a message to other objects in the overall model, or a message to self. In that way objects can schedule themselves to trigger one (or more) of their methods in the future. The code within the output method carries out the following: (i) checks whether or not the receiver of the message aMessage is the devsComponent of the Simulator, if it is the aMessage it is placed in its waitList, if it is not (ii) checks if its compositionTree is empty, if this is the case the output: aMessage method of the parent Simulator is triggered, (iii) if the devsComponent has sub-models, then output checks if any of these sub-models can receive aMessage, if this is the case the aMessage is dispatched accordingly. Finally, (iv) the aMessage is dispatched to the parent Simulator (if it can understand it and translate it).

done

This method is triggered every time that one of the three latter methods is triggered. Its main function is to traverse the *waitList* of the Simulator instance, with aim of finding the imminent message and passing it to the parent Simulator.

All three methods handle and dispatch instances of class SimulationMessage. It should be pointed out that while in the first (DEVS-Scheme like) implementation, only coupled models have wait lists that store messages of their subordinate models, in our implementation every model has a wait list. As a result, an 'atomic model' object can schedule itself to trigger more than one of its methods in the future, and can also remove messages from its waitList as it has direct access to it. In addition, σ, and as a consequence the ta function (time-advance), are specified by the time of the imminent message.

Finally, it should be noted that classes *SimulationPlatform* and *DevsModel* remain (in terms of interface) as they are in our first implementation (see sections 4.3.5, 4.3.6 respectively).

4.4.6 Realisation of the DEVS fundamentals within OO/DEVS

Even-though we have criticised the DEVS-Scheme like implementation, particularly from an OO design perspective, it should be noted that OO/DEVS follows closely the fundamental concepts of the DEVS formalism. Indeed there is a one-to-one correspondence between the OO/DEVS constructs and the mathematical representation of the formalism, as follows:

Ports: these are the publicly assessed methods of a Model subclass that take arguments, specifically:

X: the set of symbols that represent arguments

Y: the set of symbols that represent variables that are passed as arguments in the messageList (to be passed through instances of simulationMessage to the rest of the object world).

States: the method selectors of the object that have been placed in its method list.

Total State Set: the Q set - these are the method selectors of the object that exist in instances of SimulationMessage in its waitList.

δ_{int}: the state-to-state transition, which is specified in the messageList

δ_{ext}: this a mapping from the (total state set Q) x (input set X) to (the state set S), and is also specified in the messageList.

λ: a disjoint set of object's methods.

4.5 A Simple Processor Example

At this point we will borrow a case-example from Zeigler (1990, Chapter 4) to demonstrate the functionality of the framework. Lets assume a rather simplistic model of a computer architecture, which consists of a processor (P) coupled together with an experimental frame. The experimental frame (EF) consists of a job generator object, and an object that gathers statistics. In modelling under OO/DEVS we should create three subclasses of class *Model* i.e. classes ComputerProcessor, JobGenerator and Transducer. (EF) could be modelled as an instance of *Model*, describing an aggregate model similar to the one in figure 4.2 with model (A) representing the statistics gathering object with methods *recordGeneratedJob: aJobName* and *recordProcessedJob: aJobName*, and model (B) representing the job generator with a method *generateJob*. The aggregation diagram one level up is depicted in figure 4.5., where object P represents the job processor with methods *receiveJob: aJobName* and *processJob: aJobName*. Figure 4.7. depicts the system decomposition diagram for the overall model.

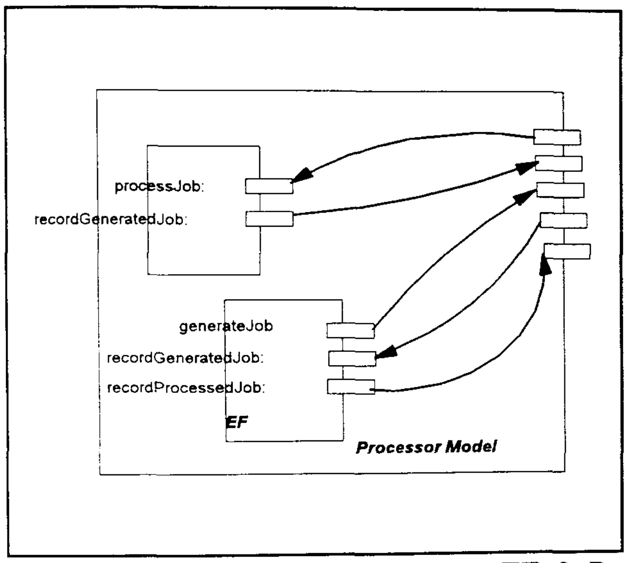


Figure 5:Message Passing between EF & P

```
doIt
    | c1 c2 c3 c4 c5 gen trans proc ef efg anArray1 |
    "------------ Generator ------------"
    gen := JobGenerator new initialize; name:'Generator'.

    "------------ Processor -----------"
    proc := ComputerProcessor new initialize; name:'Job Processor';
                                setInfluenceFrom:   #receiveJob:  to:  ( proc ) withSelector:
#processJob:.

    "------------ Transducer -----------"
    trans := Transducer new initialize; name: 'Transducer'.

    "------------ Experimental Frame -----------"
    gen  setInfluenceFrom: #generateJob  to: ( trans) withSelector: #receiveJob:.

    anArray1 := Array new:2.
    anArray1  at: 1 put: gen;
              at: 2 put:  trans.
    ef := TModel new initialize; name: 'ef';
            setInfluenceFrom: #generateJob to: (proc) withSelector: #receiveJob:;
            buildCompositionTree: anArray1;
            addSelectors: gen ;
            addSelectors: trans ;
            priorityList: anArray1.
    network at: ef put: anArray1.

    proc setInfluenceFrom: #processJob:  to: ( ef ) withSelector: #recordJob:.

    "--------------------- Processor Model as a whole -------------------"
    anArray1 := Array new:2.
    anArray1 at: 2 put: ef;
              at: 1 put: proc.

    efg := TModel new initialize; name:'Test Model';
          buildCompositionTree: anArray1;
          priorityList: anArray1.
    network at: efg put: anArray1.

    "Assign  simulators to the above models"
    s1 := TSimulator new assign: gen at: (300@250); parent: c2.
    s2 := TSimulator new assign: trans at: (400@250); parent: c2.
    s3 := TSimulator new assign: proc at: (100@150); parent: c1.
    c2 := TSimulator new assign: ef at: (350@150); parent: c1.
    c1 := TSimulator new assign: efg at: (200@20).
    root := TRootCoOrdinator new attach: c1.
    c1 parent: root.
```

Figure 6: The topology object for the simple processor example - Method *doIt*

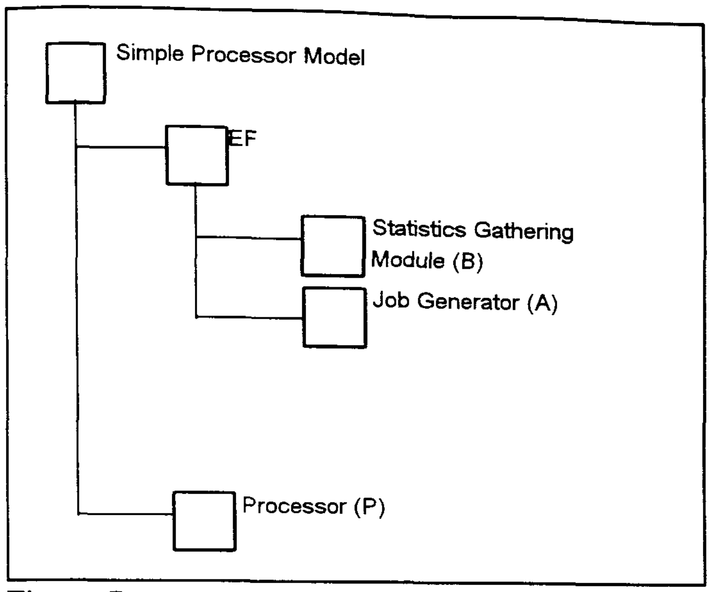


Figure 7:
Processor Model - System Decomposition Diagram

As we have discussed earlier on (section 4.3.6), as soon as the model objects have been defined, one needs to define a subclass of the *DevsModel* class, in order to specify the instances of the model objects, as well as their topological relationships (influences). This subclass of DevsModel has two methods: *doIt* and *runModel*. The former method, regarding the example in hand, is depicted in Figure 6, while the latter specifies the initial messages to be send to the model objects, and is discussed further on, as we describe the simulation cycle. As can be observed in Figure 6, the modeller has first to specify the instances of the model objects. In this case we have four instances of the class Model and its subclasses, which are refered by the variables *proc* (the instance of the ComputerProcessor class), *gen* (the instance of the JobGenerator class), *trans* (the instance of the Transducer class - the statistics gatherin object), anf finally ef (a direct instance of class Model that represent the aggegrate model of (A) and (B)). As soon as instances are specified, influences between them are established with statements like:

gen setInfluenceFrom: #generateJob to: (trans) withSelector: #receiveJob:.

which sets an influnce from the JobGenerator to the Transducer. Finally, the last sets of statements assign simulators to the instances of models that have been allready created.

As soon as we couple models to instances of Simulator, the simulation starts with each one of the Models initializing their wait list of messages and selecting their imminent message. This will result to the jobGenerator sending the message: <self, generateJob, nil, self, generateJob, 0.0 > at time 0.0, while the rest of the objects send messages to self at time infinity (this indicates that they will start at a passive state).

```
Model variableSubclass:
#ComputerProcessor
  instanceVariableNames:
    'currentJob '

initialize
    | aMessage |

    super initialize.

    processTime := 20. "time units"
    currentJob := Array new: 1.
    currentJob at: 1 put: 0.

    aMessage := SimulationMessage new
                            source: [ self ];
                  method: [ #processJob: ] ;
                            receiver: [ nil ];
                             time: [ clock ];
                   selector: [ #recordJob: ];
                  arguments: [ currentJob  ] .
    self addMessage: ( aMessage  ).

    aMessage := SimulationMessage new
                            source: [ self ];
                    method: [ #receiveJob: ];
                           receiver: [ self ];
                             time: [ clock ];
                  selector: [ #processJob: ];
                  arguments: [ currentJob ].
    self addMessage: ( aMessage  ).

processJob: aJobName
    "has no code
    - only outputs a message from the
      message  list  to the transducer"

receiveJob: aJobName
    clock := clock + processTime.
    currentJob at: 1 put: (aJobName)  .
```

Figure 8:
Methods initialize, processJob *and* receiveJob
- object JobProcessor

```
Model variableSubclass: #JobGenerator
  instanceVariableNames:
    'interArrivalTime jobName '

initialize
    "initialize the special state variables"
    | aMessage |

    interArrivalTime := 10.
    jobName := ( Array new: 1 ).
    jobName at: 1 put: 0.
    super initialize.
    aMessage := SimulationMessage new
                                 arguments: [ nil ];
                             source: [ self yourself ];
                           method: [ #generateJob ];
                           receiver: [ self yourself ];
          time: [ clock + interArrivalTime ];
                          selector: [ #generateJob ].
     self addMessage: ( aMessage  ).

    aMessage := SimulationMessage new
                                     source: [ self ];
                           method: [ #generateJob ];
                                  receiver: [ nil ];
                                  time: [ clock ];
                          selector: [ #receiveJob: ];
                           arguments: [ jobName ] .
    self addMessage: ( aMessage  ).

generateJob

    jobName at: 1 put: ( (jobName at:1)+ 1 ).
    clock := clock + interArrivalTime.
```

Figure 9: Methods initialize and *generateJob* of *JobGenerator*

As the initialization proceeds the aggregate models initialize themselves and a series of *perform* methods are triggered at the simulators of the imminent models. Finally the *perform* method, of the simulator of the job generator (B), is triggered which in turn calls the *generateJob* method. At completion the *messageList* is scanned by the simulator of (B), and the messages associated to generateJob are sent through the Simulators' output method. In this case two messages are send: (i) the message <nil, receiveJob, aJobName, self, generateJob, 0.0>, and (b) a message to self so that the *generateJob* method in (B) will be triggered after some elapsed time.

The first message goes to the EF aggregate model and dispatched to all subordinate and super-ordinate models that can translate its selector. This results to the triggering of the *input: aMessage* method of the Simulator of (EF) which sends the method accordingly to the overall model and the statistics gathering object (A). The overall model consults the list of selectors that can respond to and sends the message to the processor object (P). As soon as (P) receives the message, it logs the job name. Finally, its simulator visits its *messageList* and outputs a message to self triggering its *processJob: aJobName* method and putting the message in its wait list.

As soon as all the messages have been dispatched, done methods are triggered at each simulator and a new imminent message is calculated, for all of them. As a result an overall imminent message is produced, and the simulation cycle starts again. Figures 8 & 9, contain the Smalltalk code corresponding to the objects ComputerProcessor (P) and JobGenerator (A), respectively .

4.6 Discussion

Overall modelling under the Object Oriented/DEVS framework, and the associated Smalltalk implementation, appears to be able to address at a sufficient level the questions that we have set at the beginning of Chapter 3, by providing:

- *Entity based Modelling:* encapsulation and message passing allow the modeller to think and model the industry in terms of the main players, their strategies and the way they interact. Each player can be viewed as an object with specific attributes and methods that represent decision rules. The OO/DEVS paradigm allows this natural type of thinking to be directly mapped into a representation that can then be simulated, in a way that the cognitive leap between model and real system can be sufficiently minimised, especially in comparison to the System Dynamics core technology. This is due to the fact that it is a technology that addresses what the entities are in the real system and not the opposite.

- *Specialisation/Generalisation:* through inheritance, objects that share common attributes and behaviour can be modelled in generic classes and organised in a tree-like structure. For example, all companies have balance sheets and a share price. So they can be members of the same class 'company'.
- *Time representation:* as we have already discussed DEVS represents an attractive and concise way to represents time as events within the system and furthermore to bound decision rules (that are object methods) to time.
- *Aggregation/Disaggregation:* the ability of the DEVS formalism, to construct coupled models from a set of atomic models, allows the modeller to develop detailed decision support systems by modelling the required level of detail in atomic models. At the same time a strategic 'view from above' can be maintained by monitoring behaviour at the coupled model level, at different aggregation levels. The advantages of aggregation will become evident in Chapters 6 and 7, where elaborate OO/DEVS models are discussed.
- *Separation of models and simulation engine:* this is achieved through the ability to have generic simulator and co-ordinator modules that are attached to models. This is particularly attractive because it provides the basis for treating models as knowledge and creating model-bases. The advantage of this can be even demonstrated in the above small example, where our knowledge of the processors functionality (modelled in the object processor), can be stored separately in a data-base.
- *Modularity, reusability, extensibility:* due to the separation of models and simulation engine, objects can be stored in a model base. In addition, inheritance and encapsulation provide the means of extending, modifying and reusing old

model components. This ability has the additional attraction, that we do not have to presume what is relevant in the initial stages of modelling. In addition, we do not have to model, at the initial stages of model building, the way that organizations interact, while such influences can be easily remodelled through the model's life cycle. Overall we can achieve high modularity and reusability of the model. For example, in a model of the Electricity Industry, if a model of a customer is created, further special types of customers can be created by extending the functionality of the existing one. The new customer can be combined with models of generators, distribution companies and other customers and a completely new simulation model can be created.

In addition, the OO/DEVS framework, at its current implementation stage posses more open research questions. These questions can be summarised as:

- *Graphical Model Specification:* The approach lends itself to extensive use of graphical model specification, manipulation and synthesis. As a result, we have devised three types of diagram, the class hierarchy diagram, the system decomposition diagram and the level diagram (the semantics of which and their use will be discussed extensively in the next three chapters). The issues of implementing these tools within the OO/DEVS framework, as well as the naturalness of these diagramming tools will be discussed in the next chapter.
- *The DEVS Model Specification within the Smalltalk Implementation:* OO/DEVS has been implemented in Smalltalk/V for Windows [Digitalk (1991)]. Apart from the excellent facilities for user interface development that Smalltalk provides, the Windows version would give the modeller access to code developed to other, procedural, languages. As a result, the issue of modelling the decision rules of the players within a model, through previously developed algorithms or even an alternative modelling environment like a spreadsheet or an expert system shell, should be investigated. The foundations for the exploration of these issues are set in the next chapters.

4.7 Summary

In this chapter we have presented a Smalltalk implementation of the DEVS formalism, called OO/DEVS. We started off by discussing previous DEVS implementation views, and by presenting a DEVS-Scheme like implementation in Smalltalk, that we produced and used as a vehicle to assess the practical compatibility between the DEVS and OO modelling views. We then presented a critique of the previous DEVS implementation views, from an Object Oriented perspective. This critique has been based on three points reagarding DEVS-Scheme like model building, within an OO environmnent. These points have been summarised as:

- The reusability of a model is limited
- The concept of polymorphism is violated
- The naturalness of OO is compromised

We then embarked on the task of addressing the reserach questions associated to these drawbacks, through an implementation view that exploits fully the modelling characteristics of object orientation. The result was to produce an overall OO/DEVS simulation framework.

The advantages of the final OO/DEVS implementation, mainly in terms of the above drawbacks as they have been discussed in our critique in section 4.4.1, have been partially demonstrated through the example in section 4.5. Finally, in the previous section we discussed how OO/DEVS and its implementation address the research issues discussed at the end of Chapter 2 and the beginning of Chapter 3. Overall, what we have achieved in this chapter is to show that the OO/DEVS ideas can be implemented and used at a practical level.

However, in order to perform a comparable evaluation of OO/DEVS versus System Dynamics platforms like *iThink*, one has to address a number of research questions related to graphical model building. It should be pointed out that as a significant part of the value of SD is seen at the front end, OO/DEVS needs a user interface of similar friendliness, in order to make any comparative studies of practical use between the two frameworks. As a result in the next chapter, we embark on the task of addressing graphical model bulding issues within OO/DEVS, and discuss how a Graphical User Interface has been implemented within the framework.

Chapter 5

The OO/DEVS GUI

Contents:

5.1 Introduction

In chapter 4, we discussed the structure of OO/DEVS models and how models can be built by writing Smalltalk code (a) within the scope of the objects that represent the model components, in order to specify object behaviour, as well as (b) by developing a topology object, in order to specify the aggregation and association relationships between model components, using the constructs provided by the OO/DEVS shell. Nevertheless, as one of our research objectives is to address the issue of natural model building, the question of visual interactive modelling has to be addressed within the OO/DEVS framework

In this chapter we discuss how the fundamental OO/DEVS concepts have been used in the development of a graphical user interface (GUI), which aims to take much of the programming effort away from the modeller. Overall, the GUI caters for the graphical specification of model components, the assembly of such components into aggregate models and the creation of influences between them, by providing an interface which permits models to be built graphically, with little or no knowledge of programming syntax. A number of 'dialogue boxes' facilitate behaviour specification within the objects, influence specification, variable monitoring and simulation runs. Finally, in addition to the fundamental ability to specify object behaviour by writing Smalltalk code, decision rules can be also specified within the GUI, by exploiting the *Windows* DDE interface, within a spreadsheet environment.

We initiate the discussion by presenting the current views on graphical model building within the SD community as well as the broader software engineering perspective. We then address the question of supporting visually the fundamental OO/DEVS concepts, and we examine the basic functionality of the GUI and its supporting Smalltalk classes. Finally, we use the 'Beer Game' [for a description of the game see Sterman (1989)] as an example, for demonstrating the use of the OO/DEVS GUI.

5.2 Graphical Support for Model Building

The need for model visualization, and the advantages that represents over textual modelling languages has been stressed by a number of researchers and computer practitioners [for example see Pracht (1990), Frangini (1991)]. One of the biggest advantages of GUI's is that: 'GUI's help people stretch their computer expertise and extend their reach to draw in resources more quickly and easily than would otherwise be possible' [Seymour (1991)].

'Visual' software has proved these advantages in a number of simulation areas, with discrete event simulation the primary field for the applications of model visualization [for example see Ulgen et al (1989), Zhang & Mourant (1990), Mourant (1992)]. Within the SD modelling community, *iThink* [Richmond et al (1990)] (and its predecessor Stella) is the most distinct example of 'visual' software. It should be stressed that even before the arrival of Stella and *iThink*, with their revolutionary GUI's, the use of graphical tools within the SD community has been considered almost synonymous with model building, given that each model is usually represented through an influence diagram, a rate and flow diagram, or both.

Nevertheless, in SD the transition from influence diagrams to stock-flow ones is often problematic, since the two representations are quite different, and even when intermediate graphical tools are used there is no one-to-one relationship between them and the software constructs. In addition, stock-flow diagrams become difficult for managers to understand as problems get large (see for example figure 9 in Chapter 6, which shows a section of a systems dynamics investment model in the Electricity Industry in the UK, as has been described in [Bunn & Larsen (1992)]). As a result, a number of graphical tools have emerged within the SD community, with the objective of dealing with the fore-mentioned issue. For instance [Morecroft (1982)] has suggested the use of the 'Subsystem Diagram' for showing the major organizational divisions in the social or industrial system under modelling. In a similar fashion, many SD modellers use the concept of the 'sector', to model in an aggregate fashion distinct parts a system, and the way they influence each other. For example, see [Mashayekhi (1992) fig.1] for a 'subsector diagram' of a solid

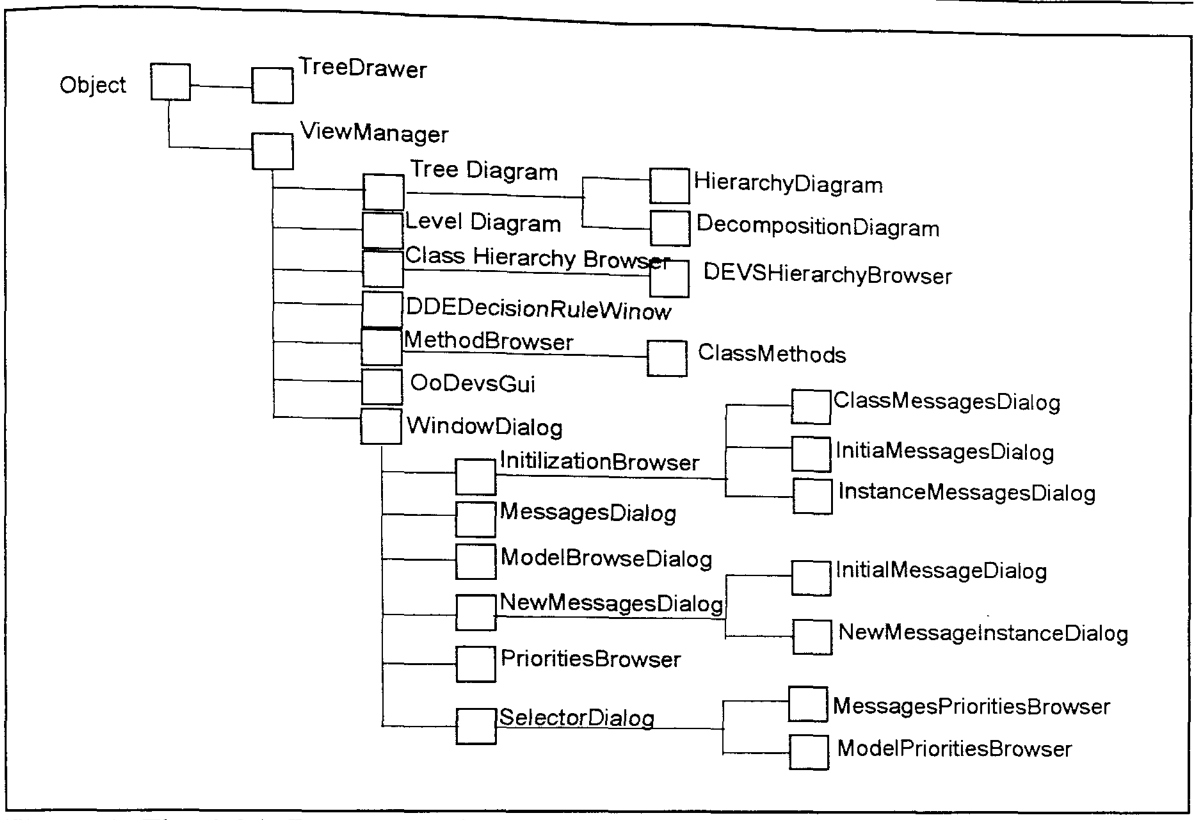


Figure 1: The OO/DEVS GUI Classes

waste management sector. Semantically richer versions of sector diagrams, that depict influences between sectors and the main variables in the system, are also widely used [see Homer (1992) fig. 6].

The existence and use of all these graphical tools, demonstrate the recognition by the SD community that aggregation relationship modelling is an important part of model conceptualization. Nevertheless, this part of modelling has only recently been supported on the software front by *iThink,* through the provision of 'sector frames'. These however, can not be used in a recursive fashion to model sectors within sectors, and do not therefore correspond directly to the DEVS coupled models, that provide the constructs for hierarchical model building within the OO/DEVS technology.

Within the broader modelling community, it is the case that most modelling methodologies usually provide a set of diagramming tools that facilitate model building and understanding. Structured Design Approaches, for example, use Structure Diagrams and Data Flow graphs [eg see Birrel & Ould (1985) for a discussion of the major techniques and notations in the area]. Within the Object Oriented Analysis & Design world, a typical set of tools is the one developed by [Booch (1981, 1986)]. The Booch method has started as an ADA language targeting design technique, and therefore it provided initially an object based view which has evolved to object oriented. The method incorporates notations such as the Class Diagram and Class Specification, icons that can represent a number of relationships between classes, state transition diagrams, and object diagrams. In general, the research area of graphical notation for object oriented analysis and design is particularly active and good examples of methods under development can be found in the SOMA method [Graham (1993)] as well as in [Edwards & Henderson-Sellers (1993)].

Within the DEVS research area, Zeigler (1980, 1990), has developed, as part of the DEVS theory toolbox, a graphical system structuring tool called System Entity Structure. This tool consists of a set of axioms and rules that can be used to construct, for any given system, a diagram that incorporates decomposition, taxonomic and coupling relationships[1]. The System Entity Structure, is used within DEVS-Scheme for hierarchical model specification and model reuse.

The research question of developing graphical model building facilities within OO/DEVS, was one that we also had to address as soon as we started to use the framework for model building. Trying to model under OO/DEVS, we discovered that a set of diagramming tools, could help us in understanding the structure of the problem, and more importantly share this understanding with decision-makers and modellers within the industry under modelling. The research goal of OO/DEVS GUI is to allow the modeller to form clear mental images of the model's structure and function. Nevertheless, the success of a GUI is very much based on the model structuring tools that it provides. In this respect the intrinsic characteristics OO/DEVS provide the basis for a semantically rich model visualization platform which can fulfil the above goal.

The modeller, and indeed the user, can build and view a OO/DEVS model through three

[1] These correspond to what we have called, aggregation, generalization and association relationships respectively.

types of diagram, which depict the specialization, aggregation and association relationships within the model. These diagrams, are represented within corresponding windows in the GUI:

(i) the *Class Hierarchy Diagram* (see figure 2), which is the equivalent to the class hierarchy diagram of an OO language, but for the model hierarchy within OO/DEVS. This diagram depicts generalization/specialization relationships among the model components (objects), which can be used to build an OO/DEVS model. These objects are subclasses of TModel which is the object that carries the essential functionality for a model to be simulated. The subclasses of TModel contain methods that represent decision rules that will be utilised during a simulation run. The Class Hierarchy Diagram is particularly useful when inheritance is used heavily, and the modeller wishes to track the ancestors of a specific model-object, their attributes and methods.

(ii) the *Model Decomposition Diagram* (see figure 7 for the Beer Game in 5.5.5) which provides a platform for model conceptualization, as it allows the user to view and (re)structure the model at different levels of detail. Model building takes place within the Model Decomposition Diagram window, as the user picks with the mouse model components from the Class Hierarchy Diagram and pastes them on existing model components within the Model Decomposition Diagram. The user has the ability not only to paste the newly selected models but to cut models previously added to the decomposition, and consequently paste them to other models. This quality of the interface is a direct consequence of the underlying modelling paradigm, and can be used as a powerful tool for experimentation with different model structures.

(iii) the *Level Diagram* (see figure 10 for the Beer Game) represents graphically association relationships within a model. It provides a view of the decomposition diagram, from the top, allowing the user, to zoom in and out of model aggregates, as well as to create and view influences between different sub-models as he/she dissects the Model Decomposition Diagram.

It should be pointed out that we view the OO/DEVS GUI in terms of visual layers. The top layer is composed of the Model Hierarchy Diagram and the Decomposition Diagram. The second layer is composed by the set of Level Diagrams that correspond to any given Decomposition Diagram, while there is a third layer that is comprised by a set of dialogue boxes that facilitate influence modelling and message sequencing.

5.3 The GUI Smalltalk Implementation

In Chapter 4.2 we have argued for the use of Smalltalk as the implementation platform for OO/DEVS, claiming that one of its advantages is its graphical user interface. Given that the OO/DEVS GUI has been developed on top of the OO/DEVS modelling and simulation engine in Smalltalk/V for Windows. In designing the GUI, and in order to address the research issues discussed earlier in this chapter, a number of design objectives was set, these objectives primarily are:

- The use of distinct windows for the representation of the three fore-mentioned diagram types.
- Ease of transition among the different diagramming representations of a model.
- The development of menu driven tools, and the use of context specific menus in particular.
- Increased level of 'visual' programming regarding model behaviour.
- The ability to hide completely the simulation mechanism from the user.

The OO/DEVS GUI is based on a number of classes, and their subclasses, as they are depicted in figure 1. The most important GUI class is the *TreeDiagram*, an abstract class that provides the functionality for drawing hierarchical tree structures. This is a class that provides the generic functionality for two subclasses: *DecompositionDiagram* and *HierarchyDiagram*. These two classes cater respectively, for the aggregation and generalization relationship modelling of the model components.

The *HierarchyDiagram* class provides the functionality specific to the Model Hierarchy

Window. Within this Window, the user may:

- Open a new decomposition diagram
- Add or remove model components (objects), i.e. create or delete subclasses of TModel
- Browse the methods of a model (object), add or change its decision rules (methods) and instance or class variables
- Create and name uniquely, new instances of a model class that can be used as components for the construction of a larger OO/DEVS model.
- Specify class message protocols (i.e. as we have described in Chapter 4, the methods that a model component may evoke within a simulation run).

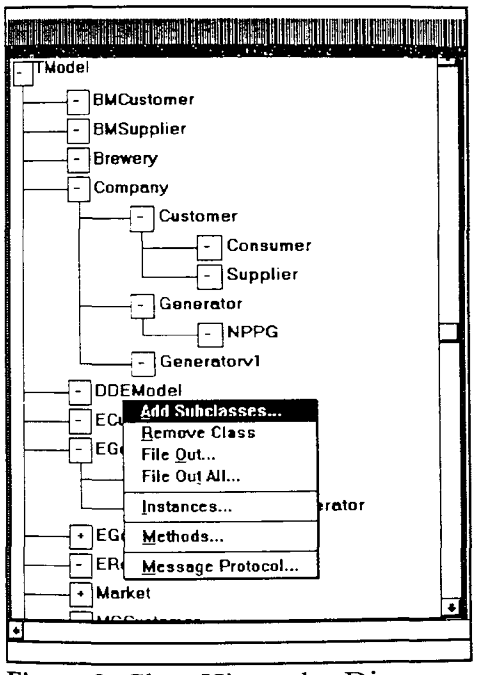


Figure 2. Class Hierarchy Diagram.

Model Components (instances) may be passed into the Decomposition Diagram Window as they are created. The *Decomposition Diagram* class, provides a number of graphical tools that remove completely from the user the need to create the topology object that we presented in Chapter 4. Overall the Decomposition Diagram class provides access to the second and the third layers of the user interface:

- Allows the model components to be aggregated by cutting and pasting the tree nodes
- Facilitates access to the third layer of the GUI in the form of a dialogue for model sequencing. This dialogue has been implemented through the class *PrioritiesBrowser* and its subclasses, and corresponds to the graphical implementation of the DEVS SELECT function for tie-breaking within a simulation run (see Chapters 3&4 for details about the DEVS SELECT function).
- Provides access to the second layer of the GUI i.e. the level diagrams that

correspond to aggregate models.

- Supports facilities for storage and retrieval of the current OO/DEVS model, by utilizing the Smalltalk Object Filer.
- Provides tools for model simulation and simulation time settings.

As we have already pointed out, one of the goals of the GUI is to hide completely the simulation mechanism from the user. As we have indicted in Chapter 4, the objects responsible for the simulation process (i.e. *Processor* and its subclasses) as well as their public functionality, have to be known to the user. This is necessary, due to the fact that models have to be coupled to simulators, within the scope of the topology object, so that a simulation run can be initiated. Class *DecompositionDiagram* removes the need for such an explicit coupling, due to the fact that models are now coupled to simulators automatically prior to a simulation run. In that respect, the OO/DEVS GUI promotes further the fundamental DEVS property of distinction between model and simulator, by hiding completely the simulators and their functionality from the user, who is now only aware of the model and its structure.

The second layer of the OO/DEVS GUI is supported through the class *LevelDiagram*, which provides the graphical tools for dissecting the Decomposition Diagram at any given level. The class provides the following facilities:

- Permits links to be created and removed between the components of aggregate models by giving access to the third layer of the GUI, in order to specify the message protocol between two models (objects). These links are displayed graphically as lines between the object entities on the screen, and represent relationships of the type <modelFrom, methodFrom, modelTo, methodTo> between the models within a level diagram. The presence of a link denotes the existence of a message protocol between two model objects.
- Gives access to the message sequencing facility (see next paragraph).
- Allows for initialization of models (objects), i.e. the modeller can set a model so that it performs a number of its methods at the beginning of the simulation.
- Provides for instance and class browsing.

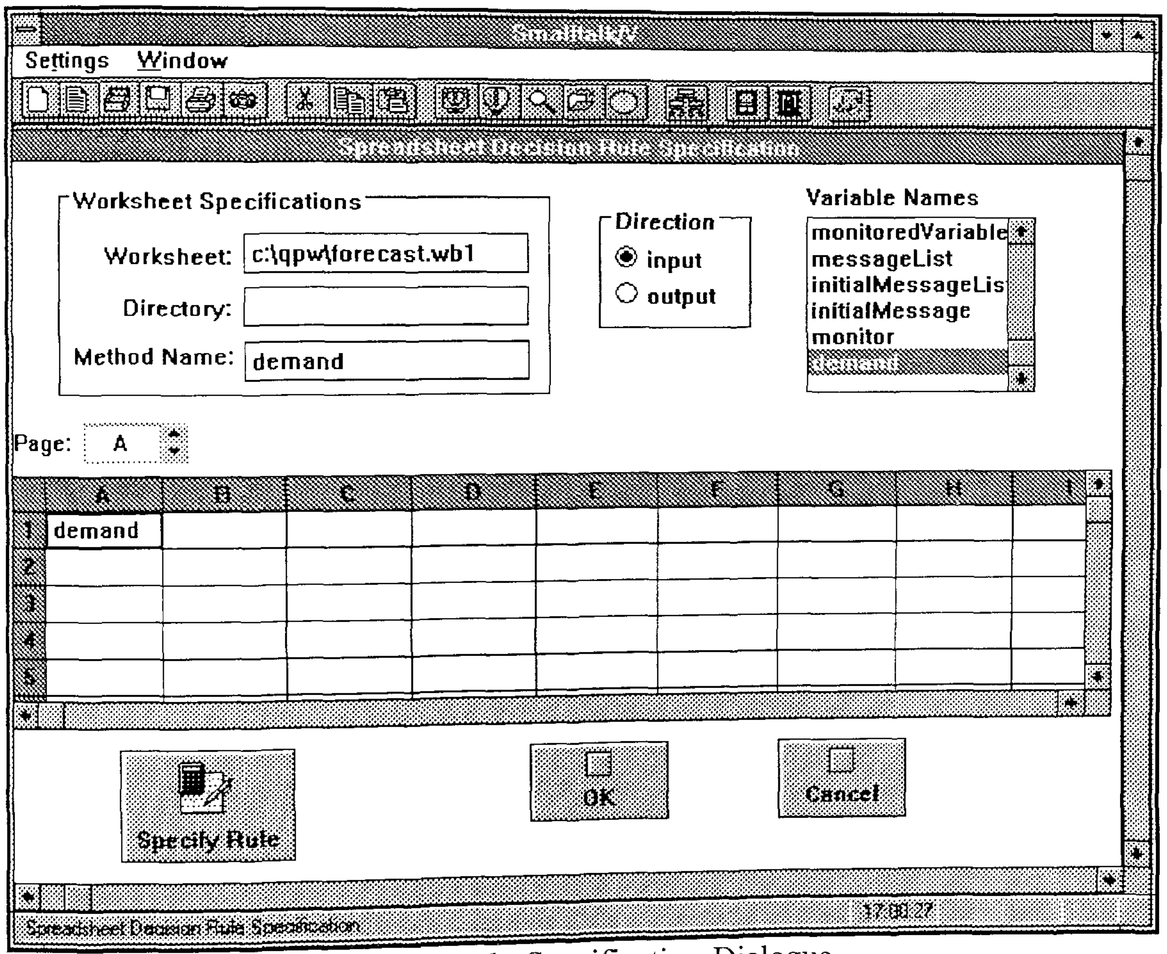
- Supports variable monitoring.

It should be stressed that a common characteristic of the window panes, of each of the fore-mentioned three objects, is the provision of context specific menus. As a result, when the user clicks the right mouse button on the iconic representation of the model components receives a specific pop-up menu. Similarly the menu differs when the mouse is placed within the general window pane area. This feature serves the objective of a purely menu driven interface, facilitating easy and natural access to every model component.

The third layer of the GUI can be accessed in the form of a set of Message protocol dialogues that facilitate:
(i) message specification, addition, deletion and editing, and
(ii) message sequencing, which provides a graphical way of specifying the sequence that the messages in the messageList will be triggered. This sequence was expressed explicitly in the non-GUI implementation by the order of messages in the method *initialize* (for example see figures 4.6 & 4.7 in Chapter 4).
These dialogues and their functionality have been implemented through the classes *MessagesDialog* and its subclasses and *MessagePrioritiesBrowser*, respectively.

The definition of the message passing protocol between OO/DEVS models, is facilitated at two levels within the GUI environment. The first one is within the Model Hierarchy Diagram. At this level, the modeller can define, for a specific model class, that any of its methods will evoke any other of its methods, at a specific clock time. In addition, any method of the class may in turn evoke methods of any other object which is a subclass of TModel. Nevertheless, even though the modeller can specify the name of the method to be evoked, he/she can not specify the actual object that owns this method. This approach enhances model reusability as models (and their subclasses, as the message protocol is inherited in an OO fashion) know how they may behave in a simulation run, but they do not know yet which are the other models within an overall model space.

The Level Diagram represents the second entry point into the third layer of the GUI. At this point the modeller can specify association relationships between objects by defining the message protocol between two instances. The modeller can use the class message protocol, while maintains the ability to specify new messages in the message protocol of the instance. The use of the class message protocol can speed up model development significantly, especially in models that contain many instances of the same class. This ability is provided through the Message Specification dialoque (see figure 11 for the outlay of the corresponding dialogue box) where the modeller can automatically select the messages that contain the methods of the influencee object to be entered into the instance messageList.


SmalltalkV
Settings
Window
Spreadsheet Decision Rule Specification
Page:
A
B
C
D
E
F
G
H
I
1
2
3
4
5
Specify Rule
OK
Cancel
Spreadsheet Decision Rule Specification
17:08:27

Worksheet Specifications
Worksheet: c:\qpw\forecast.wb1
Directory:
Method Name: demand

Direction
[✓ input]
output

Variable Names
monitoredVariable
messageList
initialMessageLis
initialMessage
monitor
demand

Page: A

	A	B	C	D	E	F	G	H	I
1	demand								
2									
3									
4									
5									

Figure 3: Spreadsheet Decision Rule Specification Dialogue

5.4 Decision Rule Modelling

In terms of decision rule modelling, our objective is to provide the modeller with a number of tools that can be used in accordance to the specific problem in hand. Within the current OO/DEVS implementation, two ways of decision rule modelling are supported. The first one is to write Smalltalk code, while the second is to link an object method to a spreadsheet, through the *Windows* DDE interface. This requires the modeller to specify a number of input and a number of output cells within the client spreadsheet, which provide the interface to the OO/DEVS object. Given the definition of the interface, which is facilitated by a specific dialogue box (see figure 3), the modeller can use straightforward spreadsheet modelling in specifying the required decision rules. In that way a decision rule is viewed as an input/output relationship and the transformation process can take place in a spreadsheet environment. Classes *MethodBrowser* and *DDEDecisionRule*, provide the facilities for Smalltalk code specification and spreadsheet decision rule specification, correspondingly. In addition, class *DDESpreadsheetLink* (a subclass of class DDEClient) provide the specific functionality for a spreadsheet as a DDE client.

The advantage of this approach lies on the fact that it utilises the experience of most modellers in using spreadsheets, while at the same time provides a consistent interface to a broad base of models (for example financial analyses) that can exist independently outside the scope of the simulation model. It should be noted, that the same principle can be used to provide access to databases or other *Windows* applications that support DDE, in a way that an OO/DEVS model can be viewed as an integrator of information.

5.5. Using the OO/DEVS GUI: The Beer Game Example

5.5.1 Case Background

The Beer Game is a classic System Dynamics model that explores the behaviour of a dynamic feedback system. The "game" involves a distribution chain, the constituents of which are customers, a retailer, a distributer and a brewery.

Decisions have to be made at each level of the chain about the demand for beer from a lower level and hence the size of an order that should be placed from a higher level.

The model explores the effects of time lags within the hierarchical beer ordering and distribution system. Supplies of beer are transmitted through the system and the effects of fluctuations in demand are modelled. The aim of the game is to demonstrate the dynamics of a distribution system, i.e. how after a shock in the system (sudden demand increase), the initial quantities of beer ordered, retained as inventory and backlogged are amplified at each link within the distribution chain, given ordering and distribution delays. A number of decision rules on ordering, formulating demand expectations and maintaining effective inventories control how the system behaves (for a detailed description of the Beer Game see [Sterman, 1989]).

5.5.2 Assumptions

The following assumptions have been applied to the model and are described below.

- All un-met demands are backlogged and accumulated. This backlog will be cleared as soon as sufficient stock becomes available.
- Previous orders and/or deliveries may not be cancelled or returned.
- No level in the chain may bypass another level when ordering or supplying goods.
- "Traders" have sound local knowledge of their level but have no global knowledge of the system.
- There is only one brand of beer and one "trader" at each level.

```
BMCustomer initializeVariables method

initializeVariables

    demand := 4.


BMSupplier initializeVariables method

initializeVariables

    backLog := 0.0.
    inventory := 12.0.
    supplyLine := 12.0.
    placeOrder := 4.
    receiveGoods := 4.
    receiveOrder := 4.
    dispatchGoods := 4.
    expectedDemand := 4.
    desiredInventory := 12.
    desiredSupplyLine := 12.
    phi := 0.5.
```

Figure 4: BMCustomer and BMSupplier *InitializeVariables* methods.

- The manufactures have unlimited capacity with only the set up time being influential.
- There is no natural loss or wastage.
- The backlogging costs are significantly higher than the inventory holding costs.
- Demand follows a simple step function increasing from four to eight units after period two.
- Communication is restricted solely to the processes of placing and receiving orders and deliveries.

The system is arranged in a cascade production-distribution structure to insulate the brewery from any short term, random fluctuations in demand. Thus only long term trends should affect the production of the brewery.

5.5.3 Entity Modelling

Model building begins by identifying the main objects in the problem domain. In the case of the beer game this is quite straight forward as there are four model components to be simulated, the customer, the retailer, the wholesaler and the brewery. However, we note that the last three have similar, if not identical, functionality in that they receive a demand for beer and place an order at a higher level (even though the brewery effectively sends an order to itself). They can therefore be grouped as one class, BMSupplier. This takes full advantage of the inheritance properties of Smalltalk and the object oriented paradigm. This means that we only need to write methods for BMSupplier as all the model components that we create will be instances of this class and will therefore share the same functionality.

```
demand

    clock >= 1
    ifTrue: [ demand := 8.]
    ifFalse: [ demand := 4.]
    self monitor:'demand' value: demand.
```

Figure 5: BMCustomer demand.

Object modelling takes place in the Class Hierarchy window (see figure 2). New OO/DEVS model components are created as subclasses of TModel (or one of its subclasses) by activating the node menu over the node representing the object which is to be the superclass.

5.5.4 Decision Rule Modelling

As previously stated, objects have a number of variables and methods which define their behaviour. Therefore, in order for the modeller to fully specify the model components to be simulated these variables and methods must be created first.

For example the only instance variable required for the BMCustomer class is demand where as the BMSupplier class has many instance variables including backlog, inventory, etc. Instance variables may be initialized in the #initializeVariables method. For the BMCustomer class the #initializeVariables method simply consists of two lines of code (see figure 4).

Where as the same method for the BMSupplier class is much larger as there are far more instance variables to be set up. Notice, also, that the method is only declared once as all the Retailer, Wholesaler and Brewery have the same initial values for all the variables.

By altering the values of these initial values it easy to see how the simulation model would be affected. For example, if the desiredInventory value was to be raised, to say 20, each of the suppliers would place larger initial orders to compensate, thus affecting the inventory and the behaviour of the suppliers higher up the chain.

Once these methods have been set up, the other methods which provide the model components with their behaviour and functionality can be built and tested.
For Customer there is only one such method, #demand which calculates the demand that will be sent to the Retailer every period. Note that for period one the demand is set at four but is eight there after. Note, also, the self monitor: statement which keeps a record of the value of the specified variable at this point in the simulation. These values can be used to trace (plot) the value of the variable at the end of the simulation run (see figure 5).

The BMSupplier class has a more detailed behaviour which is modelled by the following methods:

#dispatchGoods - Dispatches goods to a lower level in the ordering structure. The amount dispatched is either the amount ordered, or if this is not available then the value of the inventory. If the full order cannot be sent then the backlog is updated to include the difference.

#placeOrder - Makes adjustments to the desired inventory and supplyline to calculate the value to be ordered next period. The alpha and beta parameters can be adjusted to alter the desired inventory and supplyline.#receiveGoods - Adds the amount of goods received to the inventory and updates the supplyline value.

#receiveOrder - Adds the order received to the backlog and calculates the expected demand for the next period.

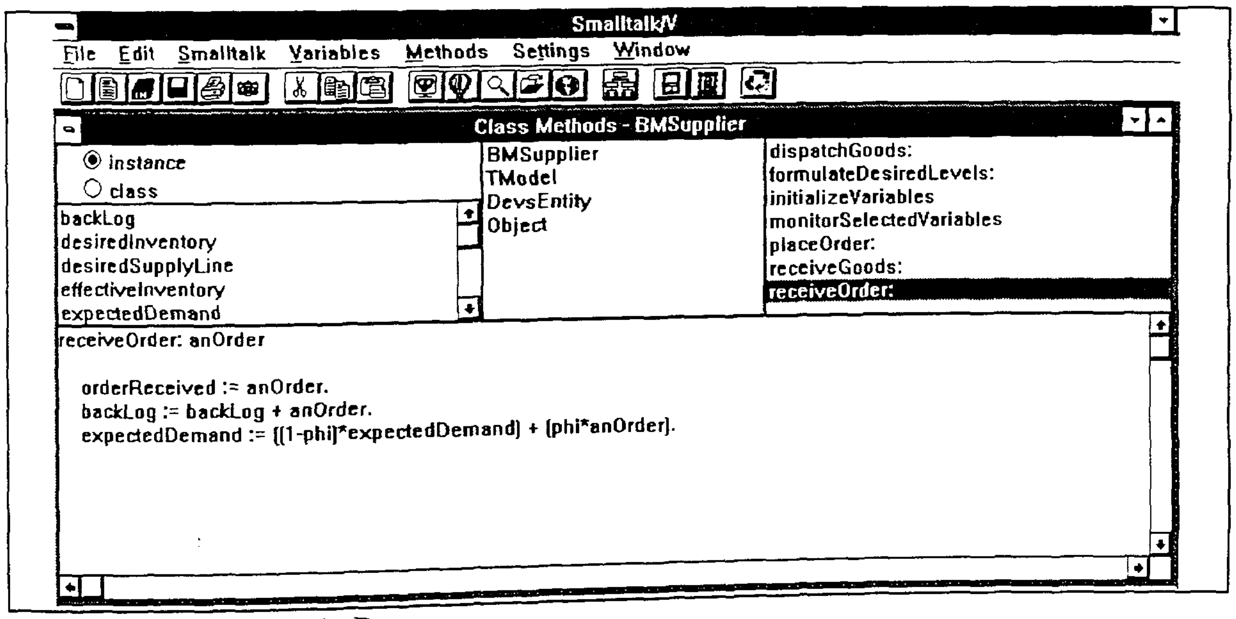
There are two other BMSupplier methods. The first, called #formulateDesiredLevels, sets up and alters the values for the desired inventory and the desired supplyline throughout the simulation. The second, #monitorSelectedVariables, simply saves the values of all the variables listed every time period so they can be plotted at the end of the simulation run.

All these methods can be entered by using the appropriate menus and menu selections within the Model Hierarchy diagram in order to bring up the Class Methods Browser (see figure 6).

5.5.5 Model Organization

The next stage in modelling is to select instances of the objects that have been created and place them in the Decomposition Window. This is achieved by selecting the object in the Model Hierarchy Diagram, and using its menu items. The first instance to be created should be an instance of TModel which will automatically appear in the Decomposition Window. Subsequent instances which are created, will sit in a paste buffer until they are pasted onto a node in the Decomposition Window or are overwritten.

To paste a node on the current Decomposition Diagram, is simply a matter of selecting the node to be pasted on and use the appropriate menu item of that node (see figure 7). Nodes can be cut and pasted, once they have been entered onto the Decomposition Diagram, so that the user has the ability to experiment with alternative model structures. A number of consintency checks have been incorporated, so that, for example, the user is warned if he/she tries to cut a node which is linked by messages to another node.



```
receiveOrder: anOrder

    orderReceived := anOrder.
    backLog := backLog + anOrder.
    expectedDemand := ((1-phi)*expectedDemand) + (phi*anOrder).
```

Figure 6. Class Methods Browser.

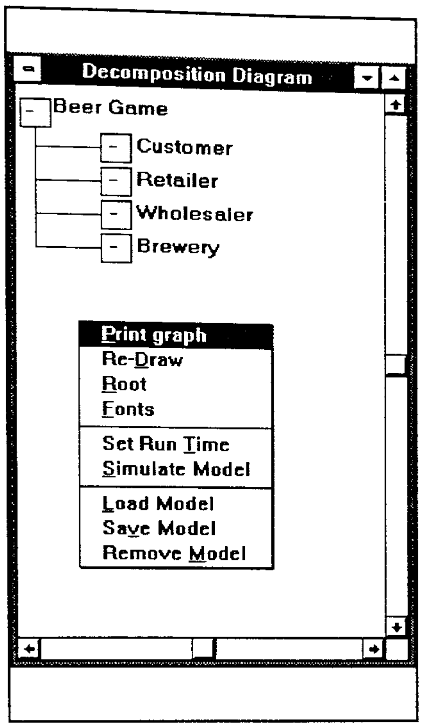


Figure 7. Model Decomposition Diagram.

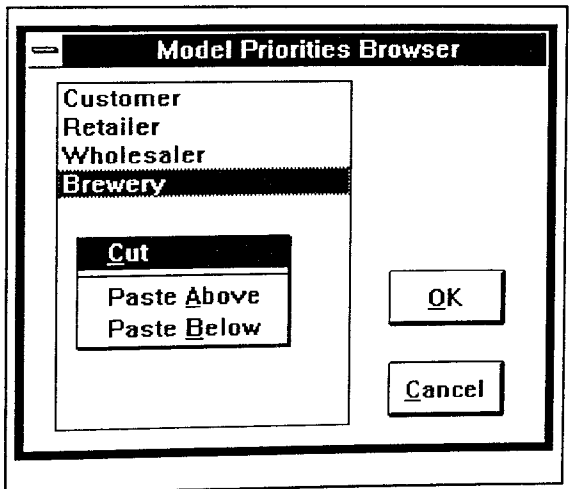
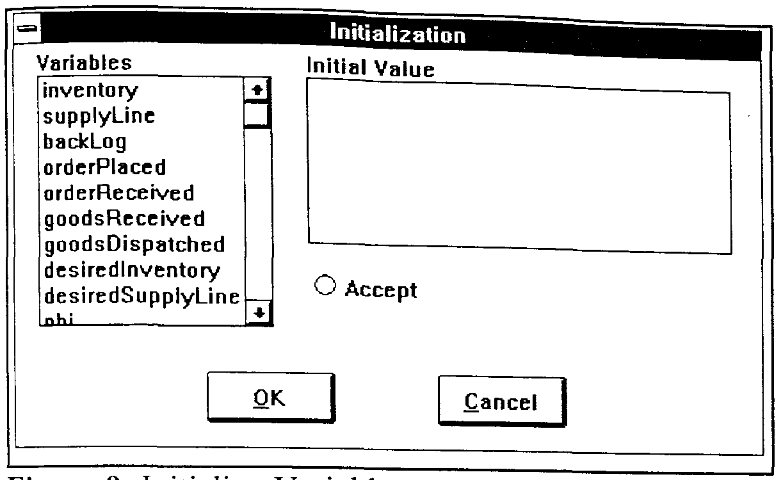


Figure 8. Model Priorities Browser.



inventory
supplyLine
backLog
orderPlaced
orderReceived
goodsReceived
goodsDispatched
desiredInventory
desiredSupplyLine

Figure 9. Initialize Variables.

Once all the required instances have been created and arranged in the Decomposition Diagram the following steps need to be taken using the appropriate node menu options.

• The level priorities should be set up within each aggregate model. This ensures that in the event that two instances within an aggregate model are scheduled to send message at the same time the one which is higher in the list will send first. This can be achieved by cutting and pasting the instance names within the table. The user selects a model by clicking over it and pulls up the pane menu using the right hand mouse button (see figure 8). A second model is then selected and the menu activated again. The first model selected may then be pasted above or below the current model. The instances at the top of the table will send their message first.

- The level selectors should be set up. This is a way of declaring which of an instances methods are visible to other instances. For example, the #placeOrder method of one instance will need to be able to send a message to the #receiveOrder method of another so these methods should be declared as 'public' messages. This can be achieved readily through the node menus of the decomposition diagram.

- As well as the #initializeVariables method which initializes the variables for every instance of a class there is also the ability to initialize the variables of individual instances (see figure 9 for the corresponding dialogue box). This is achieved by, once again, selecting the relevant option from the Decomposition Diagram node menu, selecting and assigning the required values. Caution should be taken, however, as the #initializeVariables will overwrite any values set in this box.

For the Beer Game one instance of BMCustomer (named Customer) and three instances of BMSupplier (named Retailer, Wholesaler and Brewery) need to be created and arranged as sub nodes of an instance of TModel say BeerModel in the structure shown in figure 8. The Customer should have priority over the Retailer and the Retailer over the Wholesaler etc. All methods should be declared public except the #initializeVariables, #formulateDesiredLevels and #monitorSelectedVariables methods.

5.5.6 Influence Relationship Modelling

Influences are created in the Level Diagram Window (see figure 10) which is obtainable for all aggregate models by selecting the appropriate node menu option from the Decomposition Diagram. Note that level diagrams may not be created at terminal nodes.

Once in the Level Diagram Window. the iconised representations (rectangles) of the instances may be moved around the window using a "drag and drop" technique. By pressing down the left hand mouse button over a rectangle, the cross hair cursor appears, drag the cursor across the screen and release the button at the desired position for the rectangle. The rectangle redraws itself at the new location.
A popup menu can be activated by clicking the right hand mouse button over a rectangle. The menu displays three options as follows:

- The Message Priorities option allows all the messages of an instance to be viewed (see figure 12). They can be cut and pasted to place them in order of priority (in the same way that models are cut and pasted in the model priorities browser from the Decomposition Diagram). Once again in the event that two messages are to be sent at the same time the one which is placed higher in the list will be sent first. If you are having difficulty running a simulation

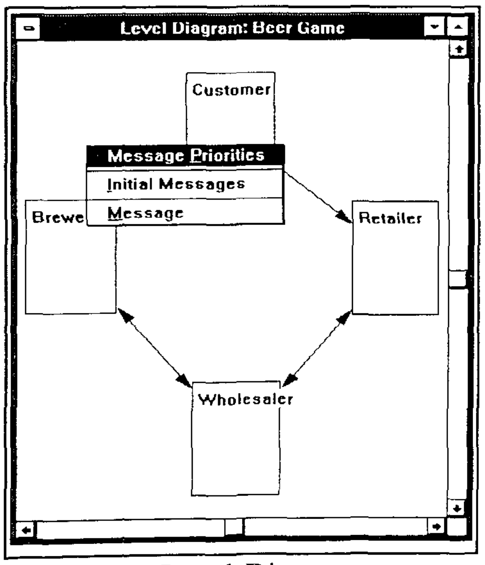


Figure 10. Level Diagram.

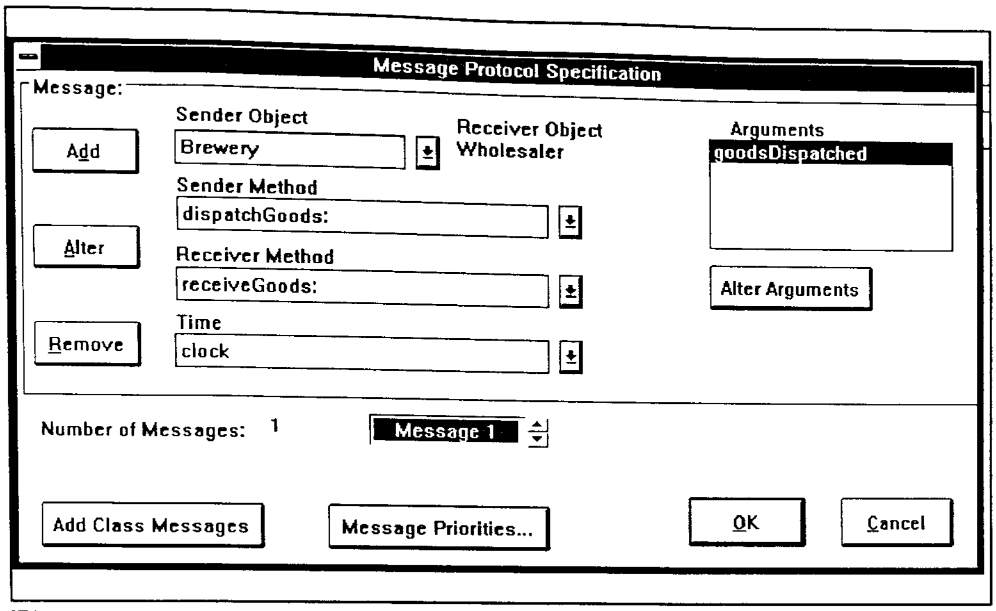
model check that your messages are ordered correctly.

- In order to run a simulation, one or more instances need to have an initial message set up. By default, all instances are initialised to do nothing unless one of the methods is triggered at some point in time. As a consequence it is vital that at least one of the instances of the model is initialised to perform one of its methods. This facility is provided by the Initial Message option on the popup menu.

- It should be noted, that this initial message is particularly important as in order for any method in the simulation to send a message it must first be called by another simulation message.

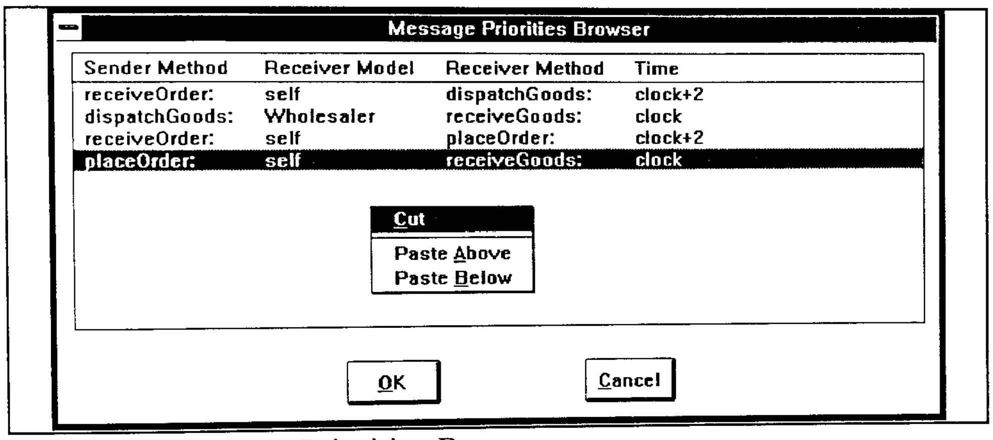
- Finally the Messages option allows new messages to be created, altered and removed. Messages may be set up between different instances or within the same instance. On selecting this option the cursor changes to the cross hair and can be moved to the receiver of the message (which may be the same as the sender). On clicking the left hand mouse button over the receiver a message dialog box appears.

Some of the fields in the Message Protocol Specification Window (see figure 11) may be pre-filled. Those that are not need to be completed by selecting one of the options in the pull down boxes. Arguments are specified by pressing the Alter Argument button which opens an Argument Window (see figure 14). The required argument can be selected, accepted and the window closed. The selected argument should now appear in the top right hand box of the



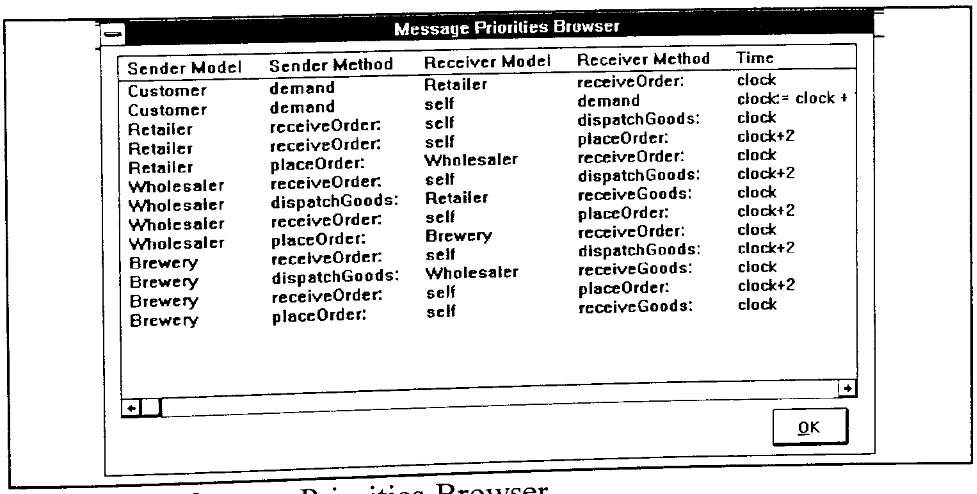
Sender Object: Brewery
Receiver Object: Wholesaler
Arguments: goodsDispatched
Sender Method: dispatchGoods:
Receiver Method: receiveGoods:
Time: clock
Number of Messages: 1
Message 1

Figure 11. Message Protocol Specification Dialog Box.



Sender Method	Receiver Model	Receiver Method	Time
receiveOrder:	self	dispatchGoods:	clock+2
dispatchGoods:	Wholesaler	receiveGoods:	clock
receiveOrder:	self	placeOrder:	clock+2
placeOrder:	self	receiveGoods:	clock

Figure 12. Message Priorities Browser.



Sender Model	Sender Method	Receiver Model	Receiver Method	Time
Customer	demand	Retailer	receiveOrder:	clock
Customer	demand	self	demand	clock:= clock +
Retailer	receiveOrder:	self	dispatchGoods:	clock
Retailer	receiveOrder:	self	placeOrder:	clock+2
Retailer	placeOrder:	Wholesaler	receiveOrder:	clock
Wholesaler	receiveOrder:	self	dispatchGoods:	clock+2
Wholesaler	dispatchGoods:	Retailer	receiveGoods:	clock
Wholesaler	receiveOrder:	self	placeOrder:	clock+2
Wholesaler	placeOrder:	Brewery	receiveOrder:	clock
Brewery	receiveOrder:	self	dispatchGoods:	clock+2
Brewery	dispatchGoods:	Wholesaler	receiveGoods:	clock
Brewery	receiveOrder:	self	placeOrder:	clock+2
Brewery	placeOrder:	self	receiveGoods:	clock

Figure 13. Message Priorities Browser.

Message Specification Window. The message can now be added by pressing the Add button. The number of message set up should now increase by one. Messages may be subsequently altered or removed. It is important when creating new messages that the top right hand arguments box is initially empty.

Once all the message in the system have been specified, an easy and effective way to check them is by returning to the Decomposition Diagram and to select the aggregate node. Pull up the node menu and chose the Level Message Priorities option. This brings up the diagram seen in figure 13. Each of the instances at that aggregate level are displayed, in order, along with a description of their messages. This clearly shows the receiver object, the sender and receiver methods and the time at which the message is to sent.

5.5.7 Running a Simulation and Obtaining Results

Before simulating a model it is necessary to select those variables that will need to be tracked throughout the simulation run. A variable can be tracked only if it is monitored within one of the methods of an instance and it has been selected in the Level Diagram Window. Variables are selected by holding down the shift key and pressing the left hand mouse button simultaneously. This opens a box with three buttons. A Select button brings up the a list of variables to choose. Clicking on a variable in the list denotes that it has been chosen to be monitored. They will appear in the adjacent window and can be removed by double clicking the left hand mouse button over them.

After the simulation has been run this same box can be used to Plot the variables that were selected.

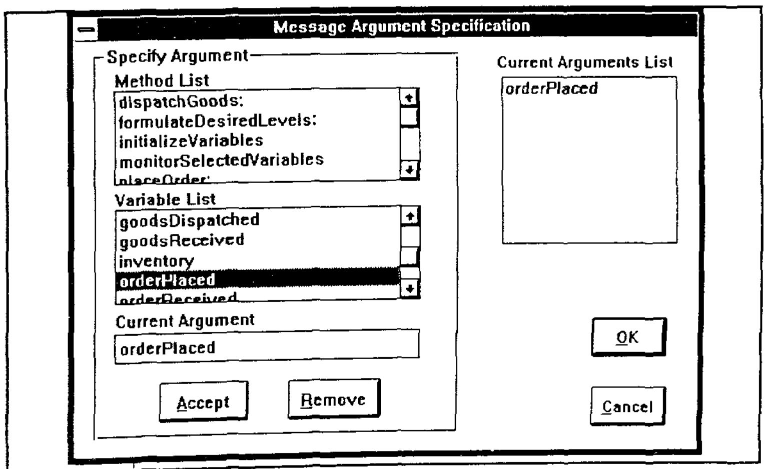


Figure 14. Argument Specification Dialog Box.

Simulating an OO/DEVS model is achieved by setting the Run Time option from the pane menu in the Decomposition Window and then the Simulate Model option.

5.6. Conclusions

In this chapter we have presented a Smalltalk implementation of the GUI for use within the OO/DEVS simulation framework. Our research task was to create an efficient graphical modelling environment, that can be used effectively with little knowledge of the Smalltalk programming language and the OO/DEVS model building constructs, under the hypothesis that such an environment will facilitate fast and natural model building.

We have supplied a small example of how the GUI may be applied using the Beer Game example. Our initial experience by testing the GUI on small models, such as the Beer Game, has shown us that the approach has the following benefits:

- increased speed and ease of modelling owing to the naturalness of the graphical representation of the model components and the speed at which influences can be attributed to these components via the use of dialogue boxes.
- high interactiveness, as the modeller can easily move back and forth during the various stages of the model bulding process.
- ease of model modification without reworking large sections of code.
- versatility in decision rule specification, as Smalltalk coding or as spreadsheet formulas representing decision rules within the same model.
- increased model accuracy as the user not only has a more complete picture of the entire model but is also guided through those processes which have previously proved difficult to code.

Earlier on in this thesis we made the point that any comparison of OO/DEVS with SD would require a 'state-of-the-art' GUI, in order to achive an even assesment. Having addressed the questions related to graphical model specification within OO/DEVS, in chapter 7 we will demonstrate further the use of the GUI, and discuss its role in using OO/DEVS with a management team. Before doing this, we first present (in the next chapter) a practical comparison between OO/DEVS and SD, in terms of their modelling concepts.

Chapter 6

Modelling Capacity Investments in the U.K Electricity Industry

A comparative modelling study between System Dynamics & OO/DEVS

Contents:

6.1 Introduction

In this thesis we present a view on industry simulation, which is applicable to a broad spectrum of industries, with the aim to develop and demonstrate a number of modelling concepts related to the modelling of industry structures and policies. In the last two chapters, we have presented two small models, a simple processor model and the Beer Game, in order to test the functionality of the OO/DEVS framework and its Graphical User Interface.

In order to provide a more realistic test of the concepts and methodology that we have previously developed, we now focus on the UK electricity industry and discuss how the framework was used to model capacity investment behaviour in the industry. In what follows, we describe a System Dynamics model of capacity investment and its equivalent OO/DEVS model. The two models are used as a platform for comparison between the two frameworks, addressing the questions of model conceptualization and structuring, use of diagramming tools for model building, model modularity and reusability.

6.2 Industry Background

The UK electricity industry is a clear example of the need to model a radically new industrial structure, as a result of the privatization of the Central Electricity Generating Board (CEGB), which has been operated as a monopoly owned by the government from 1957 to 1990. Electricity was generated by CEGB and transported through a nationwide transmission system called the 'National Grid'. In addition, twelve Area Electricity Boards received power at 'bulk supply points' and delivered it to their customers through their local distribution networks. [James Capel & Co (1990)]

However, during the 1990/91 period, the electricity industry in England and Wales was fundamentally restructured, with great emphasis on competition with vertical dis-integration [see Holmes (1990,1992)]. As a result the generation business was separated from the transmission, distribution and supply. Generation was split into two privatized companies, National Power and PowerGen, while a third public sector company (Nuclear Electric) retained all the nuclear plants. The supply and distribution business was also privatised and twelve Regional Electricity Supply Companies (REC's) were formed for that reason. These twelve REC companies are able to compete independently to buy power from National Power, PowerGen, Nuclear Power, the Scottish generation companies (that were privatized separately) or any of the independent generators that might emerge into the new market structure. The transmission business was taken over by a 'National Grid Company' (NGC), which is owned collectively through a holding company, by the fore-mentioned twelve distribution companies. NGC is responsible for ensuring a secure dispatch of electricity and the operation of a daily 'power pool'. The power pool represents the market place for buying and selling electricity. In addition to the power pool, a contract market for electricity has emerged. Finally, an independent regulatory body (Office of Electricity Regulation - OFFER), ensures that monopolistic or anti-competitive behaviour is not exercised by any of the market players, and in general safeguards the rights of the industry's customers. [For more details about the structure of the industry James Capel & Co (1990), Holmes (1992)]. The following table shows the current structure of the industry.

	Generation		**Transmission**	**Supply**	**Regulation**
Players	National Power PowerGen Nuclear Power Independents	29,664* 18,712* 8,812* -	National Grid Company	12 Regional Electricity Companies (REC's)	Regulatory Body (OFFER)
Function	- Electricity Generation - Supply to 'big' customers & REC's		- Maintain & develop the transmission system - Facilitate competition through the pool	- Supply domestic & industrial customers - Transmit through its local network	- Ensure competition - Safeguard customer rights

*Initial Generating (1990) Capacity in MW [see James Capel & Co (1990) for details in types of plants]

Table 1: The new structure of the electricity industry in England and Wales

6.3 Recent Trends & Current Issues

The monopolistic character and the governmental control of CEGB, directed the focus of the company on two main groups of issues. The first group, is related to long term strategic electricity planning, and was primarily perceived as an optimisation problem, where the best plan had to be chosen under the minimum economic cost criterion. In that respect demand was usually estimated at an aggregate level using statistical techniques. Regarding this group of issues, national and social objectives were also considered. Objectives like the security and reliability of supply, the flexible and economic supply of fuel, the compliance with public health and safety regulations, the compliance with environmental standards. The second group of issues is related to operational decisions that are related to optimal daily plant scheduling, start-up costs, minimum up/down time, nuclear plant maintenance, etc.

Traditionally, these issues have been approached analytically as single, or more recently, as multiple objective optimization problems. For instance, Kavrakoglu (1985) uses the multiple objective linear programming framework for the long range capacity expansion problem. Vlahos (1991) has proposed an algorithmic framework, based on mathematical decomposition techniques, for the electricity capacity planning problem, formulated as a large scale mathematical program.

Nevertheless, the new structure of the U.K. electricity industry has presented new issues and priorities that redefine the long, as well as the short term, issues within the industry. As a result a whole new set of issues has arisen:

Many players and objectives:
The introduction of many companies, with multiple shareholders, introduces multiple centres of gravity and instability. As a result, a plethora of conflicting objectives is introduced, in a way that cost minimization is not any more the main issue. In that respect the priorities of the industry have changed dramatically, as companies now have to focus on profitability, and on how to gain and sustain a competitive edge.

Customers' role:

Demand and supply play an important factor in the new economics of the industry, as companies cannot rely any more on the captive market of the monopoly years. Therefore, the competitive strategies of the various players need to be investigated. In addition, it is important to take into consideration the fact that the responsibilities of the new companies are mainly defined through contractual relationships, and therefore a commercial attitude must be expected from their behalf. In that respect, consumers have to be taken extensively into consideration in planning. Demand will not represent a point estimate, due to the fact that customers can (and will) demand specific and probably idiosyncratic contractual arrangements.

Long term investment decisions:

Investment decisions take a new meaning within the new structure of the industry, as cost minimization is not any more the main objective. Adequate capacity margins have to be maintained and therefore regulation will be required, between the generators, regarding the long term capacity investments. In an industry with long investment lead times, questions like the choice of plant technology or the type of fuel, and its impact to short and long term profit, are bound to arise, as investors will be interested in maximizing shareholder value. The problem of overcapacity has to be addressed, as over contracting for new gas plants has occurred. The result of that is that the values of projected plant margin are well in excess of those used by CEGB, for generation planning purposes, and also of the 20% typical worldwide margins [see National Grid Company, Seven Year Statement (March 1992)].

Industry dis-integration:

The vertical dis-integration of the privatized industry is being tested as generators are allowed to contract directly with industrial customers. Also, the problem of vertical coordination among generation, transition, distribution and supply, has to be addressed, especially as the transmission and distribution business constitute a natural monopoly.

Pool versus contract market:

The long term viability of the pool market, as a spot market for electricity, has to be investigated, given that currently only 5% of electricity is actually traded through the pool, while the rest 95% is covered through contractual arrangements [see Bunn & Larsen (1992a)]. In that respect the influences between the pool (as a spot market) and the contract market (as an insurance market) has to be investigated, especially as the participants of the industry could become less risk-averse, as the industry matures.

Regulation:

The position of the regulator, as well as the emerging European Community legislation also needs attention. As the industry players embrace more and more a profit oriented attitude (in an industry that in some respects is naturally monopolistic), is more likely that the regulator will intervene in order to protect the rights of the customers [see OFFER (Dec 1991) for the reactions of the regulator to the tactical manipulation of the pool prices by the generators]. On the other hand the impact of the implementation of environmental legislation needs to be investigated, as the generators are bound to adjust their strategies in accordance.

In addition, the industry is now sensitive to political developments, such as the change of government policies, as well as market developments, like the withdrawal of the pound from the European Exchange Rate Mechanism (utility shares were particularly hit). While, on the other hand, free of direct governmental intervention, it can act independently on issues like the protective contracts for British Coal [see Financial Times (8th of June 1992)]. Finally, new technical problems, like the maintenance of electrical stability in the network, may also arise. Overall, the industry is at the beginning of a new historical cycle, as well as in a highly evolutionary process.

6.4 The Need for Industry Simulation

The movement towards competition, economic liberalization and privatization have introduced issues of competitive strategy, which are 'soft' in nature, and turn our attention to the multiplicity of players and objectives within an industry structure. Therefore, issues of broader financial objectives, flexibility and increased risk have to be addressed.

Furthermore, models should not be perceived any more, as black boxes that provide the best answer, but as flexible decision support tools that serve as a vehicle for scenario development, communication and debate.

Such a modelling view, has been applied in the U.S. electricity industry for several years. The U.S. electricity market has the structural setting that requires such a view, due to the fact, that the industry is privatized, the generation side is highly competitive and several pool systems are under operation [source: Paribas (1990)]. For example, Nail (1992) demonstrates a model of the U.S. energy demand and supply (FOSSIL2). The structural setting of the industry is viewed through three dimensions: the energy consumers with their objectives and demand policies, the energy producers with their investment, production and pricing policies, and finally the energy market. Ford & Yabroff (1979) discuss the behaviour of a hypothetical U.S. investor-owned electric utility industry. The basic operations of their model focus on capacity, expansion planning, financing, production, price regulation and demand growth. Ford & Bull (1989) present a model that has been used for conservation policy analyses. The model assumes a single utility and it mainly incorporates price regulation, capacity planning, demand and conservation investment policies. Similar models have also been built for adjacent energy areas like fuel supply [for example see Davidsen et al (1990) for a U.S. oil industry model].

All these U.S. models have been produced under the System Dynamics modelling methodology. They are models that contain policies and assumptions, as viewed by the policy-makers and aim at the investigation of the future implications of these policies, through simulation. In that respect these models are purely structural, as System Dynamics is a structurally based approach, and SD models are causal (theory-like) models [Radzicki (1990)].

After the privatization of the U.K electricity industry, the same modelling view has emerged, in this country. Bunn & Larsen (1992a, 1992b) have modelled the investment behaviour in the new industry structure. Their work [Bunn & Larsen (1992a)] models the generators and their decision rules, as well as the way the operating rules of the pool influences investment decisions. A set of scenarios, that looks at the change of the reserve margin, under different assumptions, is provided (we discuss this model in detail further on in this chapter). In [Bunn & Larsen (1992b)] the model is extended further by incorporating the regulator and its possible policies.

6.5 Model Background

As we have discussed earlier on, in the new electricity industry a National Grid Company (NGC) has taken over the transmission business, the responsibility for ensuring a secure dispatching of generation and the operation of a daily 'power pool'. The daily power-pool, operated by NGC, is the market place for buying and selling electricity. In the long term, the pool price is intended to give the incentive to invest in new capacity. This is meant to be achieved by the so-called 'capacity payment' in the price that generators will receive from the pool. The initial objective of our industry simulation study was to understand how well these capacity payments would work in signalling the required investment in capacity.

Every day, generators submit offer prices for power available from each generating unit in their company on a half-hourly basis for the following day. The NGC, using their 24-hour-ahead demand forecasts, together with these offer prices, and a large-scale optimisation model, produces a schedule for generating power in the cheapest way over the next day. The optimal schedule is produced by ranking the plants in order of bid prices, and selecting the cheapest schedule that meets the estimated demand.

For every half-hour, a SMP (*System Marginal Price*), expressed in £/MWh, is computed. This corresponds to the offer price of the most expensive plant needed and available for generation at that time. All stations selected to run in each half-hour period receive the same SMP. NGC also computes, for each half-hour, the LOLP (*Loss of Load Probability*) which takes into account demand uncertainty and the stochastic nature of generating unit failures. Together with VOLL (Value of Loss of Load), which is a measure of the price that pool customers may be willing to pay to avoid loss of supply (initially set by the regulator at £2/kWh), the product (VOLL-SMP)*LOLP is the capacity payment which generators receive in addition to SMP. It is the expected cost of unserved energy. [see Energy Committee (1992), James Capel & Co (1990)]

SMP is the cost of the most expensive plant in the system, nevertheless, what all sellers of electricity receive, is the Pool Input Price (PIP) per unit of electricity. Likewise, all buyers of electricity purchase at the same Pool Output Price (POP). The difference between PIP and POP consists of a charge called the 'uplift' covers the costs of: capacity reserve, plant availability, forecasting errors, transmission constraints, ancillary services and marginal plant adjustment. As a result, POP = SMP + capacity element + uplift and PIP = SMP + capacity element. PIP and POP are computed by the National Grid Company.

The basic idea of the capacity payments is that, in periods of excess capacity, LOLP should be relatively low, on average, and there will be little incentive to invest in new capacity. Alternatively, when there is heavy demand relative to available capacity, LOLP will rise steeply and should provide the required investment incentive. The assumption behind this is that when the discounted cost of providing new capacity is less than the expected revenue from capacity payments, i.e.:

$$\text{E(cost of unserved energy)} > \text{Cost of new Capacity}$$

then it is worth adding new capacity. Given the lead-time of at least 3 years to commission new generating plants, the uncertainty in plant retirements and the non-linearity of using a probability to signal new capacity needs, the motivation in undertaking a simulation study is easy to understand. One would expect such a system, in its simplest form at least, to produce cycles of under and over capacity in the industry. The extent to which this will indeed happen may depend upon the lead-time for construction, the uncertainty in demand, the foresight of planners (how far ahead LOLP is forecast), the degree of knowledge about the competition, the value of VOLL and the competitive nature of the industry.

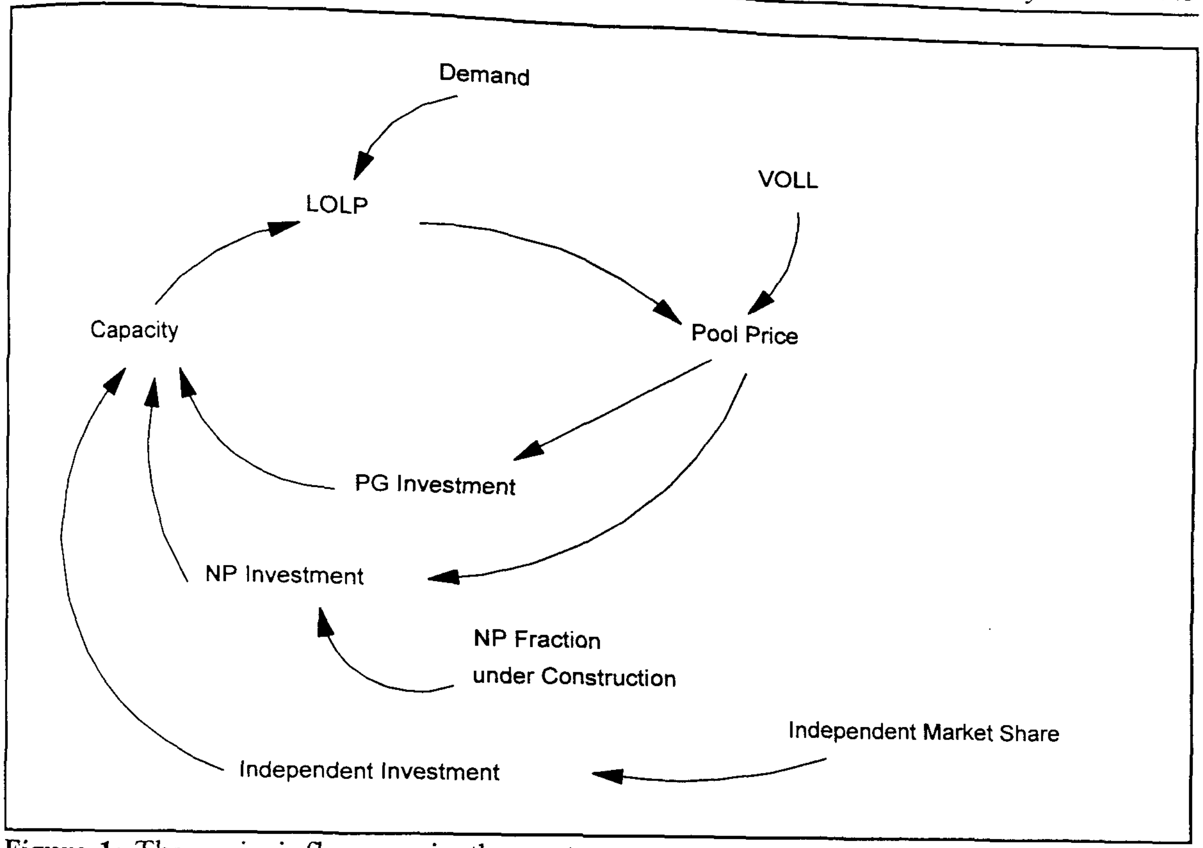


Figure 1: The main influences in the system

6.6 A System Dynamics Model

A system dynamics model to investigate these issues has been documented by Bunn & Larsen (1992a); Figure 1 summarises the main influences that the model sought to capture. The model looks at National Power, PowerGen and a third generator which represents the aggregation of future independent power producers. The equations of the model can be grouped into demand related, capacity related, and investment decision related. The expected prices provide the link between the LOLP, the expected capacity and expected demand. The LOLP is calculated as a function of the expected reserve margin in capacity. The model also covers plant retirement, in the form of retirement schedules for National Power and PowerGen. Simulations extended over 30 years with the focus of attention being the reserve margin, its potential to exhibit cycles and the way that different degrees of foresight, information exchange, uncertainty in demand, competitive and regulatory policies could affect it. The purpose was to gain some insight into how a new industrial system could be regulated, and from this initial objective the system dynamics model was successful.

Using *iThink,* the model was created quickly, with the graphical interface facilitating many re-simulations and the acquired insights concerning the potential for capacity cycles were dramatic and convincing. In that the initial perspective of that work, was one of seeking to understand the effect of various *influences*, such as LOLP, VOLL, Uncertainty, etc., on the market, going into the causal loop way of thinking about the model seemed initially quite natural. Furthermore, the model did permit some limited re-use about six months later with an updating of information on costs and retirement schedules, and some minor re-specification involving the explicit introduction of the government appointed Regulator into the system [Bunn and Larsen (1992b)].

For the purposes of this research work and thesis, the above model was duplicated, using Professional DYNAMO+. Although, the previous results were replicated, they are not reported as at this point, as we are mainly interested in the modelling exercise as such. As we have already mentioned, the equations of the model can be grouped into capacity related, demand related, and investment decision related. In what follows, we will describe briefly the structure of the model in respect to these three groups of equations.

Capacity

Capacity related equations refer to National Power, PowerGen and a third company that represents the aggregate of the independent companies that might emerge. The model recognises two different types of capacity, namely the existent capacity and the capacity under construction. An 'investment decision' is represented as a rate that is influenced by the 'capacity under construction' level, through the equation:

$$\mathit{New\_Capacity} = \mathit{Capacity\_Under\_Construction} \;/\; \mathit{Construction\_Time}$$

Finally, a rate that represents the 'Retirement Capacity' leads the flow into a sink (see Figure 2 for the *iThink* graphical representations). It should be mentioned that the model does not assume any plant retirement for the new independent companies.

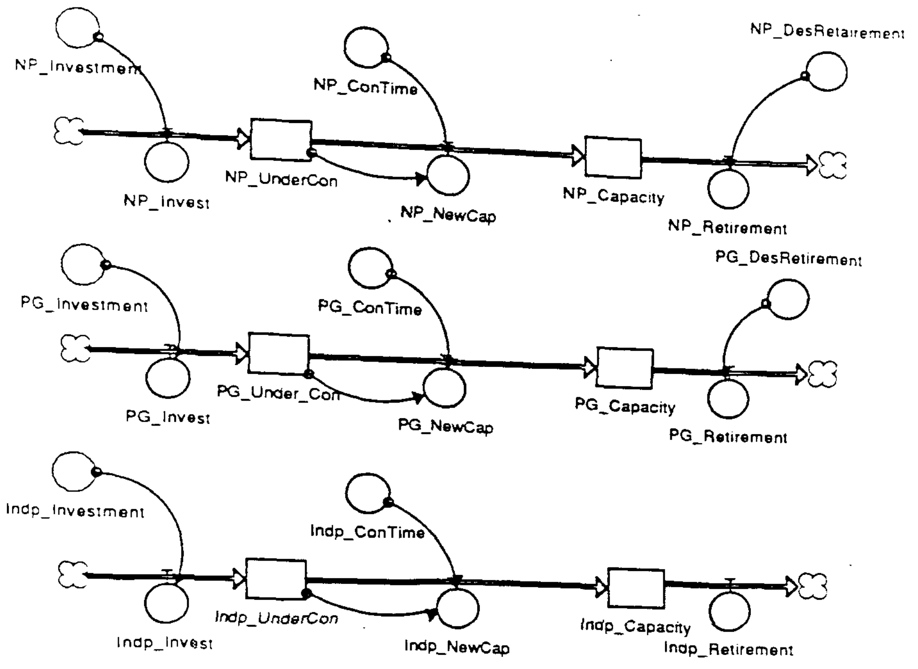


Figure 2: Capacity related rates and levels

Directly related to the capacity under construction, are the types of plant represented in the model. The underlying assumption is that all players use common plant technology and that only four types of plants are available in terms of production capacity (at 500, 1000, 1500, 2000 MW). Therefore, when new capacity is introduced into the system, one of the fore-mentioned types of plant is chosen.

Demand

As we have mentioned in the introduction of this chapter, customer demand is bound to gain a central role in the new electricity industry. Nonetheless, a rather aggregated view of the demand is taken in this model. As a result, demand modelling is based on the assumption that there is a 1% demand growth per annum[1] [which is in line with estimates, see for example UBS Phillips & Drew (1990)]. Therefore, given the initial demand the

[1] Uncertainty in the forecasted demand is also introduced in the model, by random simulation of errors in expected demand [see Bunn & Larsen (1992a)]

model computes the demand *n* periods ahead. However, demand is central in the model, as it influences directly the available capacity margin, which is defined as the rate:

$$Margin = (Total\_Capacity \,/\, Demand) - 1$$

It should be noted that the margin is computed net of the Nuclear Electric and other electricity sources.

The Investment Decision

As we have already discussed, the 'Capacity Under Construction' level, is influenced by an 'Investment Decision' rate. The investment decision[2] is represented as a series of nested IF...THEN...ELSE, as:

```
Invest = IF Expected_Price_4 > Return_Per_Half_Hour
             THEN Invest in a 2000 MW Plant
         ELSE IF Expected_Price_3 > Return_Per_Half_Hour
             THEN Invest in a 1500 MW Plant
         ELSE IF Expected_Price_2 > Return_Per_Half_Hour
             THEN Invest in a 1000 MW Plant
         ELSE IF Expected_Price_1 > Return_Per_Half_Hour
             THEN Invest in a 500 MW Plant
         ELSE Do Not Invest
```

Where 'Expected_Price_i', $i \in [1,2,3,4]$, is the expected price when the investment is related to a 500, 1000, 1500, 2000 plant correspondingly. This expected price, is the expected price of unserved energy (see chapter 4.1), and is expressed by the equation:

$$Expected\_Price\_i = LOLP_i \times VOLL$$

Where VOLL is set at £2/KWh, and the LOLP is provided by a 'table' function[3]. The 'Return_Per_Half_Hour' is computed through the yearly return, which is a function of the

[2] Bunn & Larsen (1992a) have incorporated in their model the degree of foresight which each company applies to the investment decision. This degree is varied between 0 (myopic) and full 4 years. These different degrees of foresight were tested in different simulated scenarios.

[3] Bunn & Larsen (1992a) have experimented using an LOLP curve derived from calculations reported by the Electricity Council (1985). However, they have also experimented with another, more convex, version of LOLP. Note that a more convex LOLP reflects the current trend of introducing smaller and highly available plant into the system.

acceptable return by each company, its investment cost, and the economic life of a plant. The yearly return is modelled through the equation:

$$Yearly\_Return =$$
$$Investment\_Cost \times Acceptable\_Return / (1 - (1 + Acceptable\_Return)^{-Economic\_Life})$$

which represents the fixed sum paid by the asset (plant) each year, for the specified economic life of the plant (annuity). The acceptable return is a function of the 'Investment Financial Cost' (computed through a 'table' function) and a decision variable that represents the desired return on investment. The economic life of a plant is assumed the same for all four types of plants.

Finally, plant retirement was modelled in a simple form by assuming a constant capacity retirement per generator [due to the fact that the duration of the simulation does not exceed the life of a new plant, as we have already mentioned, independents generators do not retire any plants]. The following table summarizes the main assumptions of the model:

Demand	Total 1990 demand: 48,000 MW Demand growth: 1% per annum
Capacity (1990)	National Power: 29,664 MW PowerGen: 18,712 MW Independents: 0 MW Nuclear Electric: 8,000 MW Other Sources: 2,000 MW
Retirement	National Power: 740 MW per annum PowerGen: 460 MW per annum Independents: 0 MW per annum
Plants	Plant sizes (capacity): 500, 1000, 1500, 2000 MW Common Technology Average life: 40 years Economic Life: 25 years Investment Cost: £250 per MW
Duration	Time span 40 years, starting at 1990

Table 2: The main assumptions of the model

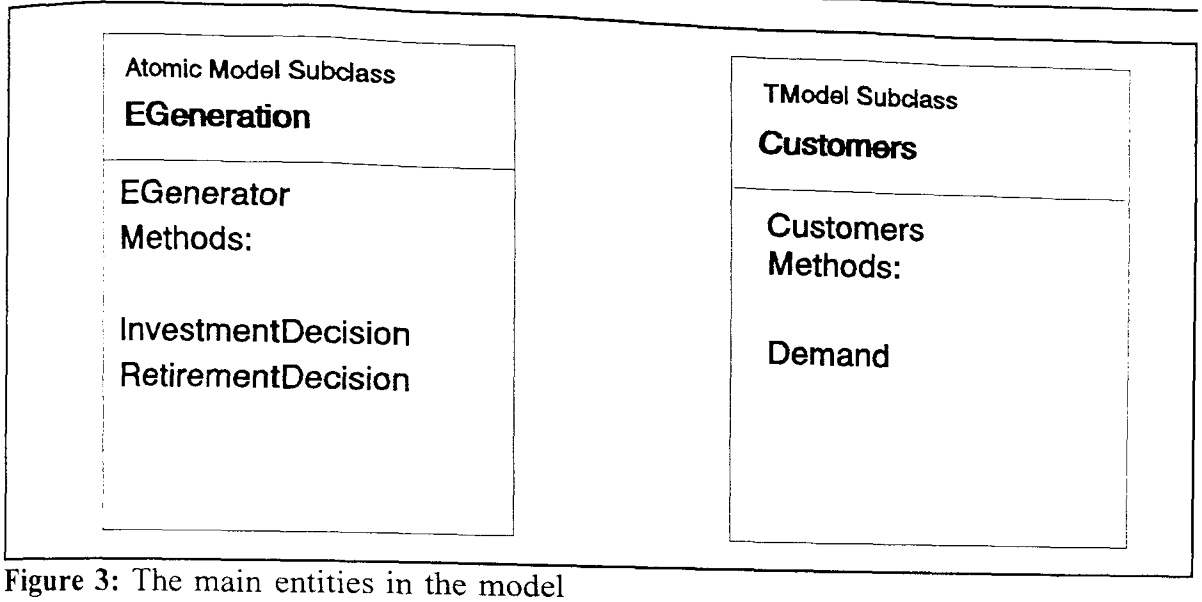


Figure 3: The main entities in the model

6.7 Modelling under the Object Oriented/DEVS framework

As we have underlined the System Dynamics model was created quickly and delivered useful insights concerning the structure of the electricity market. However, it became difficult to extend the model to deal with some major scenario changes such as an increase in the number of generators, or to introduce more realistic decision rules. As more issues were added, the model became quite large, with many replications of decision rules which one would have preferred to see represented in a more generic way (i.e. a lack of 'generalisation')[4]. Finally, when the LBS energy project research team was approached by one of the utility companies to develop an industry model to facilitate a variety of possible, but initially unspecified, simulations, with varying levels of detail, than it became appropriate to think in a new object oriented fashion, with an *entity* focus to the structure.

When we initially attempted to 'transfer' the above system dynamics model to the

[4] It should be noted that some generalization can be achieved within the current System Dynamics software. In the case of DYNAMO for example, an array capability is provided, in order to handle indexed variables. In a similar fashion, iThink provides the concept of sector frames, so that the modeller can replicate parts of the model that perform similar functions. However, in both cases it is the case that the modelling paradigm does not provide the constructs for generalization, which is left to the modeller as a strict model design choice.

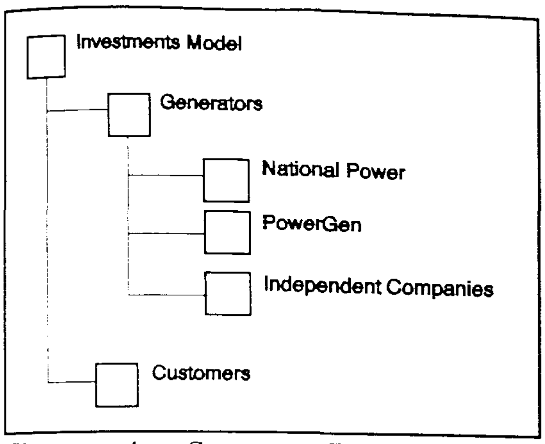


Figure 4: System Decomposition Diagram

OO/DEVS modelling environment, we experienced the fundamental difference between object orientation and an influence-based approach. Having failed to reconstruct a model by looking directly at the influences in the above system, we moved back a step and started to reconsider the main entities in it. Therefore, the modelling process started by identifying the three main generators and the entity customers. The next step was to consider the attributes of the main entities and the operations that they perform upon them. That step led to the design of two main objects: a class EGenerator and a class ECustomers.

Figure 3 depicts the interface of the class *EGenerator*. Every instance of the class *EGenerator* can *makeCapacityDecisions*, which are split up in two types of decision, i.e. *investmentDecision* and *retirementDecision*. This is in contrast to the initial system dynamics model where the investment decision rule is triplicated for each generator. In addition, class EGenerator has a *produceReturnOnInvestment: capacityUnderConstruction* method which is used within the investment decision. The object can communicate with its outside object world, through a number of methods. Method *receiveDemand: aDemand* can be triggered from any object in the outside world that has a demand for electricity and wants to let an EGenerator know about it. Methods *receiveFutureCapacity: aCapacity* and *receiveCapacity: aCapacity* are used by other generators in the system to inform an EGenerator about their capacity placement intentions within the next three year period, and about the capacity they actually bring

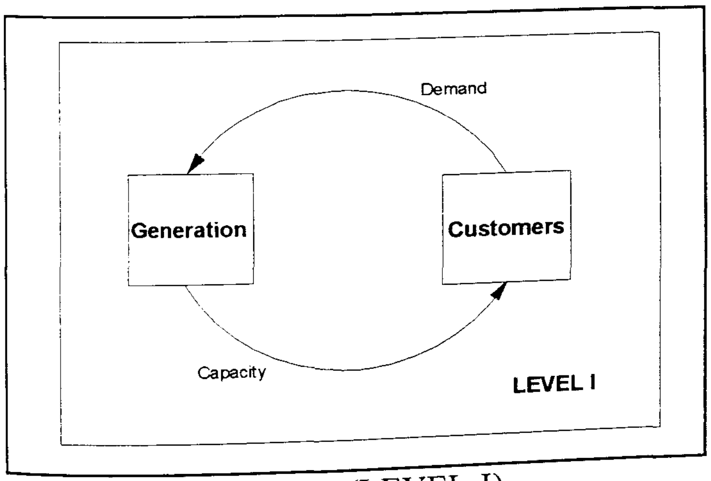


Figure 5: LevelDiagram (LEVEL I)

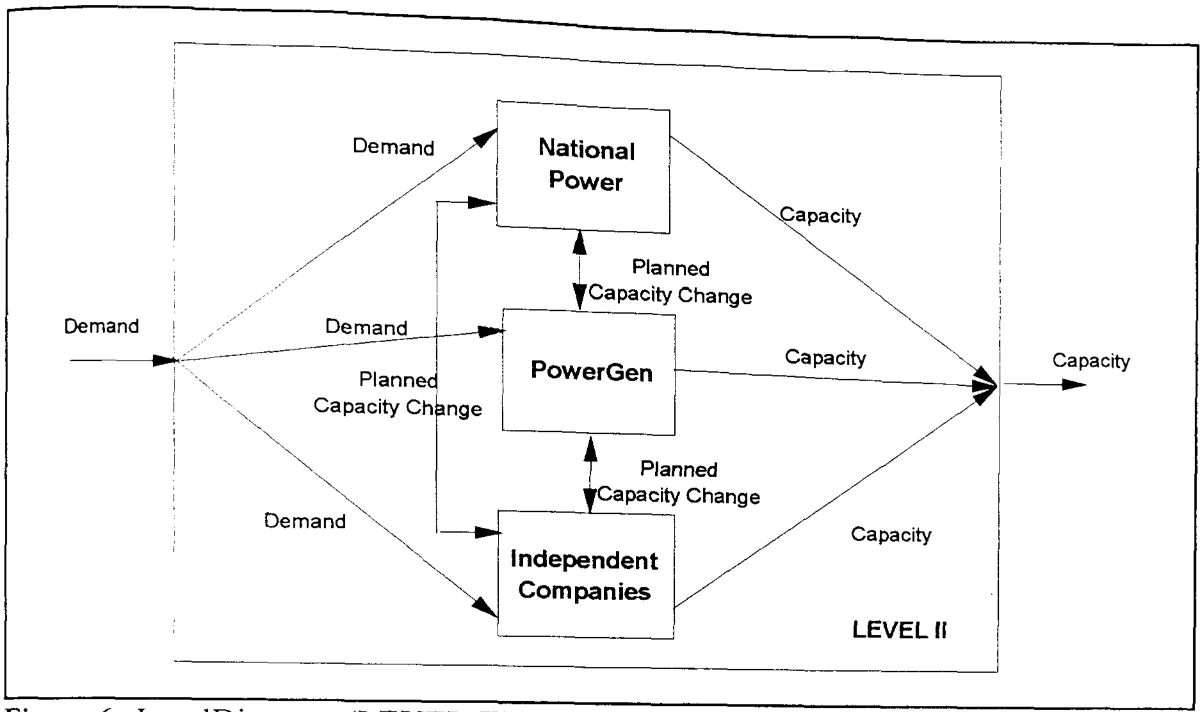


Figure 6: LevelDiagram (LEVEL II)

into the system, respectively.

The next modelling step in our environment was to lay the main entities out in a hierarchical fashion using what we call a *system decomposition diagram.* In the current example, this results in a very simple hierarchy depicted in figure 4. By dissecting the hierarchy at different levels, we can now see the influences between the fore-mentioned objects in a new type of diagram, the *level diagram.* Figure 4 represents a dissection at level I, called "Electricity Market"; at this level we can identify two main entities the atomic-model (object) Customers and the coupled-model Generators. Similarly, figure 5 represents a dissection at level II - Generators.

As has been already discussed, the DEVS formalism provides us with a concise way of describing the influences within the model. For example in level I (figure 5) we can only see two main influences, the demand influencing the generation, and the capacity influencing the customers. In level II (figure 6) the demand input influences all three generators which produce capacity which becomes an output of the coupled-model generation. In addition, information channels regarding future changes in capacity have been established between all three generators. The final step was to implement the decision rules of each object that determine their response to influences from their environment.

It should be pointed out that the two models (i.e. the one under SD and the one under OO/DEVS) are behaviourally equivalent in the sense that their observation frames[5] are compatible and they both realize the same set of I/O functions, in addition both systems incorporate the same decision rules. Overall, the implementation of OO/DEVS to this case study, via Smalltalk, was successful in that it achieved similar user insights as the earlier system dynamics model, but seems to offer greater scope for reusability.

6.8 Model reuse & expansion under OO/DEVS

The fore-mentioned investments model in its System Dynamics version [Bunn & Larsen (1992a)], was actually reused [see Bunn & Larsen (1992b)] in order to incorporate the regulatory policies within the industry, and the way these may affect investment decisions. In this section we discuss the main characteristics of the extended investments model in the form produced by Bunn & Larsen (1992b), and we present an equivalent OO/DEVS model. Our objective is to assess the level of model reusability under the OO/DEVS framework, and comment on the way that the two models were expanded.

6.8.1 Case Background

Although several empirical studies have tracked the changes in electricity prices in England and Wales since the industry was privatised in 1990/91 [eg Helm and Powell (1992)], the behaviour of the market with respect to capacity investment and the role of the regulator, remains highly speculative. As it is pointed out by Bunn & Larsen (1992b): 'Whilst the immediate focus of the Government's review [House of Commons (1993)] was

[5] An observation frame is a structure $O = < T, X, Y >$, where T the time set (R in our case), X the input value set and Y the output value set.

to find a way of subsidising British coal to make it competitive with imports, in the longer term it is essential to understand the incentives and dynamics of new capacity construction'.

As we have discussed in the first part of this chapter, the basic driving force for investments in the industry is, at least in the way that the system was designed to work, the capacity payments. The idea of the capacity payments is that, when there are periods of excess capacity, the LOLP should be relatively low, on average, and there will be little incentive to invest in new capacity. Alternatively, when there is heavy demand relative to available capacity, LOLP will rise steeply and should provide the required investment incentive.

In their first study [Bunn and Larsen (1992a)] have looked at a model where the capacity decisions are based on the capacity element, and shown that the extent to which serious capacity cycles will indeed occur depends upon the uncertainty in demand, the foresight of planners (how far ahead LOLP is forecast), the degree of knowledge about the competition and the competitive behaviour in the industry. In the simplest, "market signal", case of generating companies responding to the recent annual average value of LOLP, then, indeed severe cycles of the reserve margin were shown to result.

However, as the role of the regulator was not investigated fully in that first study, the objective of adding the regulatory policies within an investments model can be easily understood. The results of that first study point towards the direction, that if the Regulator were able to encourage more foresight and information exchange with respect to planned construction and retirement over a three year lead time, and better demand forecasts, then the "capacity payment" method of pricing appears to be capable of maintaining the reserve margin at a desired level (24% currently, but 21% is the industry target). Such 'indirect' influence by the Regulator is clearly essential for improving the efficiency of the market, but another issue of importance is whether the Regulator can control the market more directly. Can the Regulator reduce variability in the LOLP through controlling the retirement plans for old plant? What is the effect of excessive "signalling" of new capacity plans by the duopolistic generators, and how can this affect

VOLL? How much uncertainty does this produce in the market prices? These are the issues explored in Bunn & Larsen (1992b).

6.8.2 Modelling Background

In the Bunn & Larsen (1992b) study, the initial *iThink* model, as we described it earlier in this chapter, was used as a basis for a revised investments model. The revised model incorporates:

- capacity retirement policies, based on a predefined CEGB schedule,
- the policies of one more generator (a representation of Nuclear Electric), i.e. retirement only as Nuclear Electric is assumed not to invest in new reactors,
- the VOLL policy of the regulator, i.e. equations that adjust the VOLL value, in accordance to an expected margin for three years ahead.
- the perceived VOLL value change, that the generators believe it will occur if they add x new MW of capacity into the system.

It should be mentioned, that for the revised investments model, a more advanced version of iThink was used. This version, provides the advantage of being able to modularize the system into 'sector frames', and therefore gives the modeller the ability, to distinguish easily between equations that belong say to National Power versus equations that belong to PowerGen. Even though, this provides an equivalent tool to our level diagram type model specification, it fails to provide the capability to view the system at different levels of detail. As we have pointed out in chapter 5, this stems from the fact that you can only model sectors in one level, and not sectors within sectors. In addition, in contrast to OO/DEVS objects, one cannot use inheritance relationships upon sectors, and therefore sectors can be considered more as a tool for model structuring rather than a tool for model reuse.

In that respect, the first Bunn & Larsen model was used as an ad-hoc basis for model redefinition. For instance, the capacity decision equations were rewritten in order to incorporate the perceived VOLL change, that a new capacity addition will result to. The perceived VOLL change was expressed as a table of values. The behaviour of the

Regulator was also added in the model equations, and was linked to the investment and retirement decision outputs of the Generators. However, the reusability of the SD model ground practcally to a halt, when it was desided to consider the implications of a possible referal to the Monopolies and Mergers Commition the result of which could have been the spliting the genarators, or when the investment decision rule had to be changed from LOLP based to SMP based. Such changes demanded practically the rebuilding of the entire model. In contrast, as we will demonstrate in this chapter, the equivalent OO/DEVS model, by exploiting the high modularity of the framework, as well as the inheritance concept of OO, has provided a more structured basis for model reuse.

6.9 Reusing the OO/DEVS Capacity Investments Model

Figure 7, depicts the object hierarchy diagram associated with the model in hand. As we have explained in Chapter 4, class *TModel* carries all the functionality of an object that can be simulated, and therefore each OO/DEVS model-class is a subclass of *TModel.* Class *ECustomer* provides the representation of the customers as an aggregate, while classes *EGenerator* and *IndependentGenerator* provide the behaviour of the Generation Companies in terms of investment. It should be noted that the latter classes constitute the classes used in the first version of the investments model, in that respect they only map investment behaviour and a constant yearly retirement.

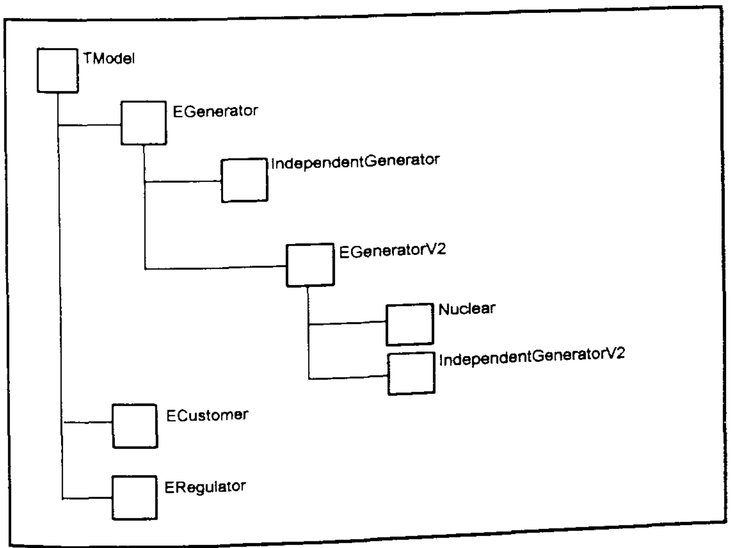


Figure 7: Class hierarchy diagram

As we discussed earlier on, in the first 'Investments Model' the class *EGenerator* can *makeCapacityDecisions*, which are split up in two types of decision, i.e. *investmentDecision* and *retirementDecision*. A number of other methods, namely *produceReturnOnInvestment : capacityUnderConstruction*

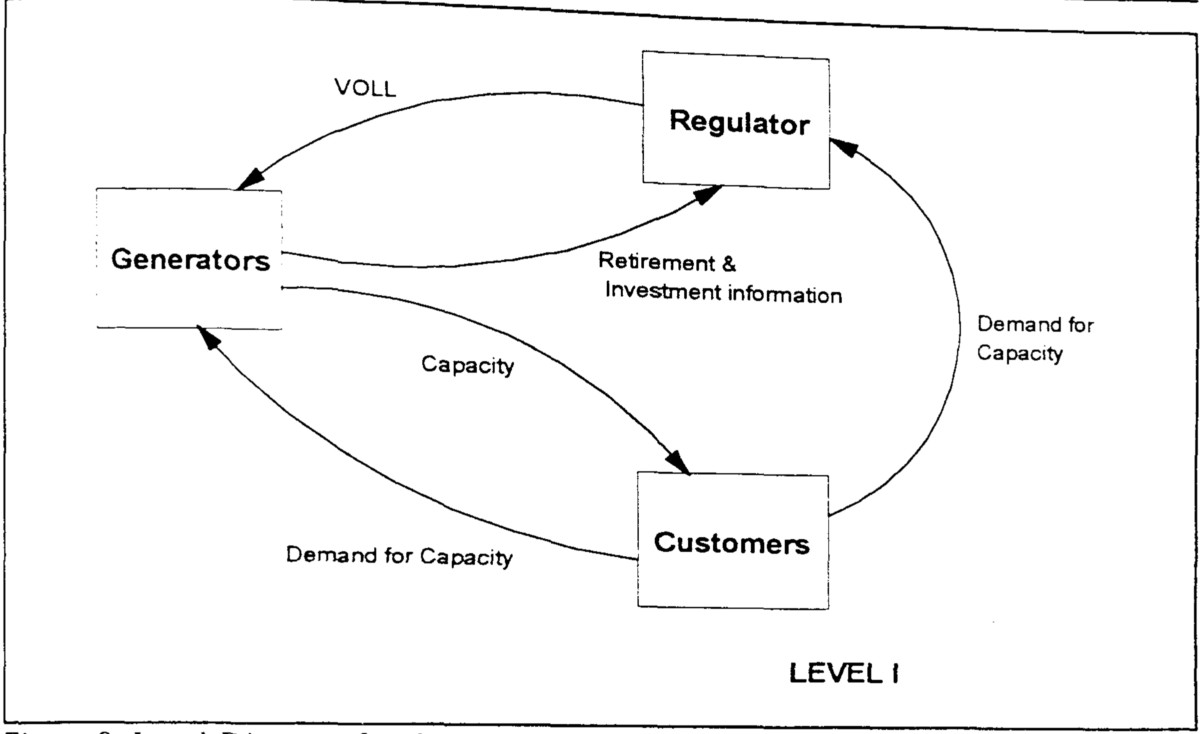


Figure 8: Level Diagram for the extended Investments Model (LEVEL I)

method, *receiveDemand: aDemand*, *receiveFutureCapacity: aCapacity* and *receiveCapacity: aCapacity* belong to the specification of the class.

All the above methods are inherited by the object EGeneratorV2, which nevertheless overloads the methods *investmentDecision* and *retirementDecision*. In the case of the former method, the investment decision is now taken by a Generator taking into account the perceived VOLL change that the decision will cause. The latter method is overloaded in order to take into account the different way of modelling the plant retirements (i.e. through a the predefined CEGB schedule, rather than a constant capacity retirement each year as it was the case in the first 'capacity investments' model). This class contains also one more method named *receiveVOLL: aVOLLvalue* to reflect the fact that the VOLL will be changed by the regulator.

Class IndependentGeneratorV2 is overloading the *retirementDecision* method is order to account for the fact that the Independent Power producers will not be retiring any capacity for the time span of the model. As a result, the fore-mentioned method does nothing in this case. In a similar fashion, class Nuclear overloads the method *investmentDecision* so that no new investments in Nuclear capacity will be made for the model's time span.

A new class ERegulator has been also created to model the industry regulator and its policies regarding the adjustment of the Value of Loss of Load (VOLL). This object has methods *receiveInvestment: aCapacity* and *receiveRetirement: aCapacity* so that can receive the investment and disinvestment decisions of the generators, as well as the method *reviewVOLL* in order to apply its VOLL reviewing decision rule. The ERegulator object can also receive the annual customer demand, through the method *receiveDemand: aCapacity*. Figure 8, depicts the level diagram of the model at the top level. The level diagram corresponding to the Generators (aggregate model) is the same as the one depicted for the first version of the investments model, with the additional influence of the VOLL value which feeds in from the Regulator.

Overall, reusability was achieved at two levels:

(a) The reuse of the model components, i.e. at the level of the model objects, where inheritance was exploited in order to overload the decision rules of the object EGenerator regarding its capacity decisions, while at the same time the rest of its methods were used in their original form. This presents a disciplined way of expanding the model components, without changing the original model (as it was the case with the SD model). This facility allowed us to test the behaviour of the generators when SMP based (instead of LOLP based) capacity investment decision rules were used. In addition to the use of inheritence, the fact that a generator is represented as an object that can be instantiated several times, provides the capability to 'split' generators, only by changing the initial values of some of their instance variables, with practically no modelling effort or redesign of the model.

(b) The original model was used as the basis for the expanded one, (i) by substituting the model components representing the generators with their new subclasses, (ii) by adding a new model component, i.e. the Regulator, and (iii) by adding some new association relationships (in the form of new messages). It should be mentioned that the latter feature of the expansion, is a direct consequence of the encapsulation property of the model objects, which allows the modeller to use the same message specification as far as the interface of the objects (model components) remains the same. For example, in this case each generator can get informed about the capacity decisions of the rest of the generators (i.e. methods *receiveInvestment: aCapacity* and *receiveRetirement: aCapacity*), irrespectively of how a generator treats the capacity information or makes its capacity decisions.

6.10 Discussion

Having modelled the system in both frameworks, it became clear that although the initial focus of the study was the core technology of System Dynamics, it was not a matter of simply replacing a differential equations engine with a discrete event simulation one. Working with the OO/DEVS framework meant that we had to substantially change:

- the way we perceive problems
- the diagramming tools for encapsulating our thinking and describing the system modelled
- the translation process from the mental model to one that can be simulated
- the way we use the model to investigate different scenarios and develop insights
- our expectations for future use of the model.

More specifically, the fundamental difference is that in system dynamics we have to translate the real world in terms of stocks and flows, whereas under OO/DEVS, we start off by modelling directly the main entities of the system and their functionality[6]. This however has created the need to devise appropriate diagramming tools, that are better suited to the entity based approach. These tools, and their semantics, have been allready presented in the previous chapter. These are *the object hierarchy, the system decomposition*, and *the level diagram*. We felt they were useful documentation and communication tools. Compared to the traditional system dynamics diagramming tools

[6] It should be noted, that as we have already discussed in section 5.2, a number of diagramming tools are used within the SD community. These tools facilitate the model building process and they provide an intermediate model representation between mental model and stock and flow equations. Nevertheless, it is the case that the final computer representation of the model is in a stock and flow equation form.

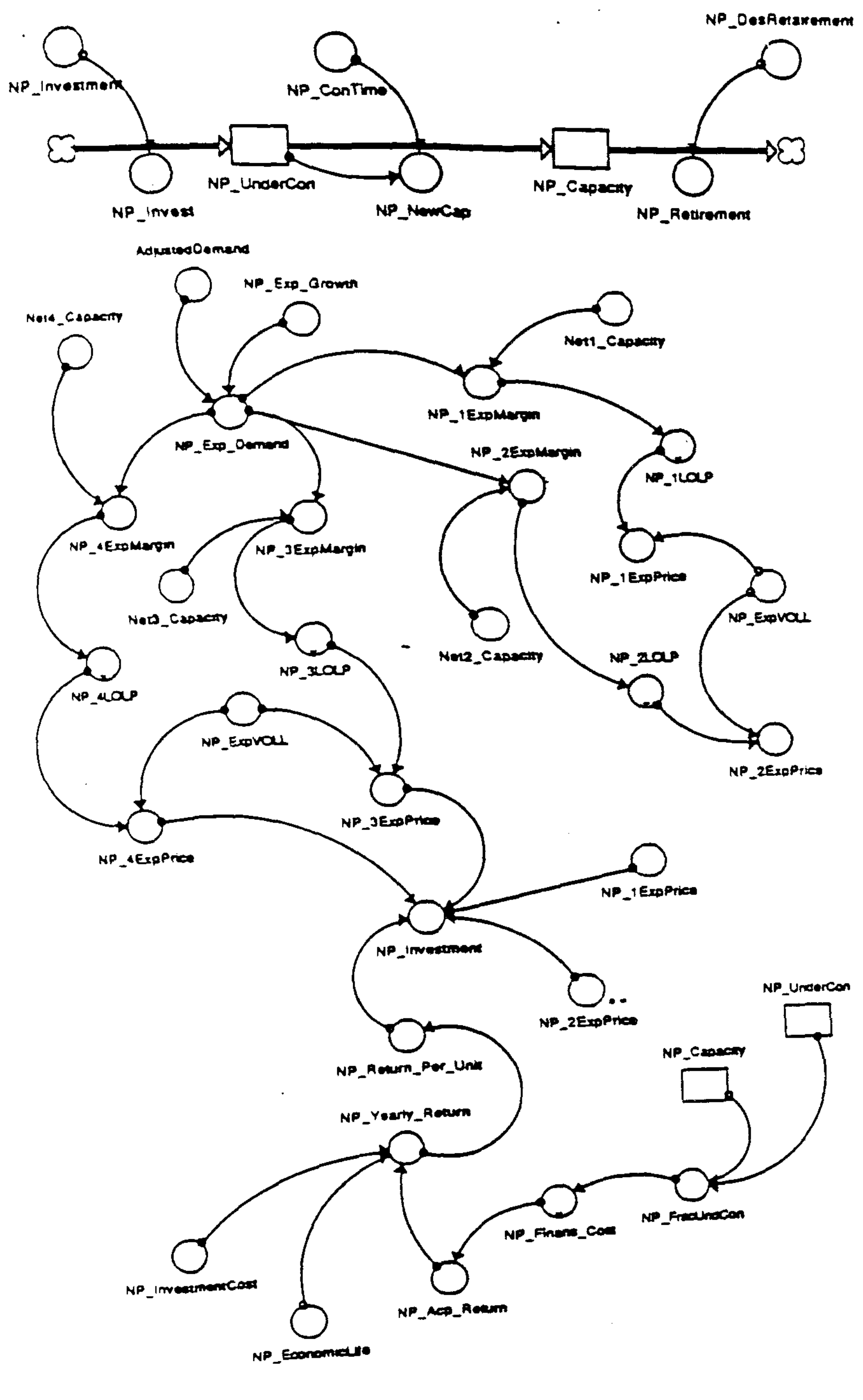


Figure 9: Level / Rate Diagrams for National Power

they exhibit certain advantages, which stem from the hierarchical representation and the information hiding. By comparing figures (9) and (3-8) the limitations of stock-flow diagrams in describing large complex becomes evident, whereas the decomposition achieved by the new types of diagrams alleviates the problems[7].

It should be pointed out, that the initial response of our sponsors, was that these diagramming tools were very natural and useful in communicating the structure of the problem and its mapping onto a OO/DEVS model. The one-to-one correspondence between the level diagrams and the entities in the real system, proved to be particularly useful, as it prompted a good basis of discussion about the way these entities influence each other. However, there was some initial confusion between the class hierarchy and the system decomposition diagrams. This was however resolved, at a conceptual level, by explaining the two different relationships that the two diagrams depict, and the practical level, by presenting the class hierarchy diagram as the model base, containing model components. Once, the class hierarchy and the system decomposition were established, it was straight-forward to move to a computer representation of the model, thanks to the close similarity between diagramming tools and program structure . On the other hand in systems dynamics the transition from influence diagrams to stock-flow ones is often problematic, since the two representations are quite different.

The System Dynamics modelling and simulation, was carried out using *ithink* and Professional DYNAMO+, both mature commercial packages, with a user-friendly graphical interface in the case of *iThink*. On the other hand, at the time of the initial OO/DEVS study of the investments model, the OO/DEVS GUI was not operational and as a result the model was developed in the fashion described in Chapter 4 (see section 4.5 for an example). In that respect this initial lack of graphical model specification, graphical output and scenario exploration tools that *iThink* and Professional DYNAMO+ supply, provided an extra leverage for the development of the OO/DEVS GUI, as it has already been presented in the previous chapter.

[7] To make the comparison fair, we should stress that the SD modeller, would try to communicate a model through the use of sector and policy maps. Nevertheless, these maps are not an inherent part of the modelling paradigm, as it is the case with the decomposition, object hierarchy and level diagrams.

Finally, our expectations for future development of the two models are very different. The system dynamics model has successfully fulfilled its initial role but further expansion or substantial modification would be awkward and time consuming. On the other hand the new modelling requirements in the electricity industry, can involve the use the OO/DEVS model in examining other issues such as the implications of competition in the newly privatised industry, the impact of environmental legislation and the possible split of National Power and PowerGen into more companies. It was also intended to use the electricity investments model, as module within a broader UK energy model, so that interactions between competing fuels can be studied.

6.11 Concluding Remarks

In the second chapter of this thesis, we motivated the development of OO/DEVS through a discussion of the issues: *structure, focus, time-representation,* and *reusability.* Having implemented OO/DEVS and tested it in this case study, we can now be more specific about these aspects. A comparative summary is provided in table (3).

Referring back to the research issues pointed out in the Chapters 2 & 3 of this thesis, we can summarise the proposed OO/DEVS simulation environment as follows:

(a) influences (association relationships) can be expressed through the use of the DEVS formalism, within a multicomponent DEVS;

(b) generalization relationships (i.e. taxonomies of model components) can be expressed through the class hierarchy of the object oriented paradigm;

(c) aggregation (and dissagregation) relationships can be expressed, within a given model, through the notion of DEVS coupled models, in such a way that the modeller can organize the structure of the system at different levels and the user can view the model at the appropriate level having all details at lower levels hidden;

(d) object orientation provides a structural separation of entities, influences and, through DEVS, the simulation "engine".

The case study presented in this chapter, aimed to evaluate in practice how well the latter four points can be used in order to produce well structured models with well defined

	OO/DEVS	**System Dynamics**
Initial Concepts	Entities	Influences
Levels of Focus	Hierarchy of Layers	Single Initial Layer
Diagramming Tools	Object Hierarchy System Decomposition Level Diagram	Causal Loop Diagram Stocks-Flows Diagram Subsystem Diagram Policy Structure Diagram
Core Technology	Object Orientation - DEVS	Difference Equations
Time View	Discrete Events	Continuous change
Simulation Engine	DEVS-Scheme simulator, co-ordinators, root-co-ordinator, Separated from model	Integration routine Not separated from model
Model Components	Objects (Atomic-Models)	Stocks, Flows Set of Difference Equations
Aggregation / Disaggregation	Supported through Coupled Models	Not Supported
Modularity	Supported	Not Supported
Reusability	Extensive	Limited
Decision Rules	Equations, Logical rules, Algorithms, Time Related Events	Equations

Table 3: A comparison of the two approaches

representational distinction between model structure and policies, and different types of model components. The practical issue is whether the attainment of a more functionally explicit structure adds value to the application of industry simulation. The case study presented here suggests that it does. Nevertheless, further exploration of this issue needs to be attempted on a more complex problem, ideally with an actual business user experiencing the need to develop and re-use a flexible, hierarchical industry simulation model.

Chapter 7

The Electricity Markets Model: The Development of a OO/DEVS Model in a Business Environment

Contents:

7.1 Introduction

In the previous chapter, we presented a comparison between System Dynamics and OO/DEVS, based on the development of the 'Investments Model'. That model was developed from an 'academic' perspective and within an academic environment. In this chapter, we discuss how OO/DEVS was transferred to a business environment, and how a management team used OO/DEVS, over a period of a year, to build a quite elaborate model of the electricity industry. We will refer to this model as the 'Electricity Markets Model'.

We start the chapter, by presenting the background of the business modelling team, as well as the phases of the project. In what follows our first objective, is to address the research question of how structured modelling, and knowledge elicitation, can be approached within OO/DEVS, by providing a structured set of eight steps to OO/DEVS model development. This set of steps, is the result of the combination of similar techniques in System Dynamics and Object Oriented Analysis, as well as our experience with the fore-mentioned management team. Our second objective in this chapter is to present the Electricity Markets Model, as it has been built in OO/DEVS. Our aim, is to demonstrate how the framework facilitated the development of a realistic model of the Electricity Industry, which contains detailed representations of the main industry players. We finally discuss, how the management modelling team viewed the framework regarding our research agenda, which we summarised (see Chapter 2) through the issues of *structure*, *focus*, *time-representation* and *reusability*.

7.2 The Background of the Modelling team

The project took place within a division of the commercial department of one of the U.K. electricity distribution companies (we will refer to it as DC from now on). One of the main issues that the division deals with, is the development of medium (six to twelve months) to long term (10 years) scenarios of the evolution of the electricity industry. The issues under consideration include the evolution of the pool market, competition, investments and regulation. Given the fact that the privatization of the industry is fairly recent, and that the market will open completely to competition over the next decade, it is clear that the plethora of issues and scenarios to be explored is formidable.

The modelling team was composed of a number of managers and analysts with prior experience within CEGB (pre-privatisation industry). Their background was engineering, in terms of education, while their modelling expertise consisted mainly of 'hard' Operational Research techniques (mainly optimization and forecasting) and spreadsheet modelling. Even though the team has been using programmers to implement mathematical models, little or no knowledge of software engineering existed within the modellers.

7.3 The Phases of the Project

The OO/DEVS project has gone through three distinct phases within the company:

The *first phase*, which initiated the project, started with the introduction to the company of the SD 'investments model' of the industry [see Chapter 6 as well as Bunn & Larsen (1992)]. At this phase the concepts of Systems' thinking were introduced and it became clear that this type of modelling has to complement the 'hard' modelling tradition that exists in the industry. It should be pointed out, however, that there was initial scepticism, on behalf of the SD management team, regarding the judgemental elements of SD modelling. Moreover, the advantages/disadvantages of System Dynamics were identified at that time, setting the primary design objectives which resulted in OO/DEVS.

The *second phase* was the introduction of the OO/DEVS modelling platform and simulation engine, as they have been presented in Chapter 4. It is interesting to observe, that as the first ideas associated to OO/DEVS were introduced to the company, we experienced considerable intial scepticism and misunderstanding. In order to aleviate these problems we developed two tutorials, which were used in a number of tutorial sessions which aimed at familiarizing the modelling team with object orientation and DEVS as well as the overall mechanics of model building within OO/DEVS. Both tutorials were based on case studies. The first utilized the OO/DEVS version of the Bunn & Larsen (1992) 'investments' model, while the second was based on a completely new model that explored the market interactions between the Generation Companies and their Customers. The latter model was later reused, forming the basis for a more elaborate model that we will present further on in this chapter.

The *third* and final phase of the project, took place over a period of four months, and consisted of the development of a model of the purchasing and selling policies within the industry (based on the fore-mentioned model, used in the second tutorial), as well as the final delivery of the software (including the GUI) to the company. This model (the Electricity Markets Model) was developed jointly with the modelling team of the company, through regular model building meetings (once or twice a week for a period of four months) at the DC offices, while a significant amount of background work, was performed in-between meetings, both at LBS and DC. It is our experience as facilitators within the model building process, that we convey further on. It should be stressed that at this stage of the project, the intially sceptical management team, became advocates of the approach, which they presented themselves (throught the Electricity Markets Model) to the their higher management and directors. Finally, it should be pointed out that this phase also triggered further development (and refinement) of the GUI, following the feedback from the users of the framework.

7.4 The Approach to Modelling

As we have pointed out in the introduction of this chapter, we aim at focusing on the last phase of the project, i.e. the transfer of the framework (and its software implementation) to the company. Our principal aim, was to assure that the modelling team felt ownership of the framework and its underlying concepts. The key in doing so, was to build a model jointly, undertaking the role of facilitators, rather than model builders. The list that follows describes the main steps that we have followed in the model building process.

STEP 1: Identify the issues under consideration and the general industry background in relation to these issues. In our case, this step was carried out through fairly unstructured brainstorming sessions.

STEP 2: Identify the main objects in the system to be modelled. These objects may be physical entities (eg. a Generation Company), aggregates of physical entities (eg. the Customer Side of the electricity industry), or notional entities which nevertheless have a specific function in the system (eg. a Contract Market). The System Decomposition Diagram may be used as a tool during this process.

STEP 3: Select the functional areas of interest within the objects in the previous step. This is the process of specifying the model boundary, by discarding any areas of the system, that are not interesting in relation to the issues in hand. The process leads to the identification of the variables of interest within each object, as well as the broad behaviour of the object. This in many respects is a labelling exercise, which in our case was, for example, the identification of the fact that a Generation Company owns a set of plants and 'supplies contracts'.

STEP 4: Specify the way that the objects, within the system in hand, influence each other. This process, effectively corresponds to the identification of information and material passing, which is also used in System Dynamics. The Level Diagram may be used as a tool during this step. The facilitator/modeller can start with a Level Diagram containing only model entities and a discussion can be carried out on how these entities interact with each other.

STEP 5: Identify alternative model structures in terms of aggregation and disaggregation relationships among the model components. This step should lead to a set of alternative System Decomposition Diagrams.

STEP 6: Specify the decision rules of the objects within the model space. This step requires the detailed description of 'the way that our objects get things done' and may bring us back to step 2 as more 'secondary' objects might be identified.

STEP 7: Having identified the objects in our problem space, and their detailed behaviour, it is important to identify the hierarchical relationships among them. Our objective is to arrive at objects at a sufficient level of abstraction, in order to either identify (in our model base) objects previously build, or to build new reusable model components.

STEP 8: As soon as the OO/DEVS have been built and tested, a model can be put together and steps 4 to 7 can be repeated within the OO/DEVS environment, so that alternative model structures can be tested.

As can be observed the process of model building can be highly iterative, as the modeller (or sometimes the user) can move back and forth on the steps of model building. It should be noted, that the first iteration of steps 1 to 7 correspond to the initial model conceptualization, while step 8 represents a design choice. Model design and building takes place in consequent iterations of steps 4 to 8, as the modeller makes design choices by alternating the structure of the model in the decomposition diagram, the decision rules and variables of the model objects (in the method browser provided in the Class Hierarchy Diagram), as well as the set and sequence of interactions between model entities (within the level diagrams, through the use of the message specification dialogues).

While the above eight steps, reflect our model building experience within OO/DEVS, it should be pointed out that they are by no means a unique way of approaching knowledge elicitation and model building within the framework. Areas like cognitive psychology, small group processes and System Dynamics have approached the problem of knowledge elicitation from many perspectives [for example see Richardson et al (1989), Vennix et al (1990), Larsen et al (1991)], and can possibly offer a number of useful techniques that can be blended with the above process. Indeed, the proposed eight steps contain ideas already found in the fore-mentioned areas. For instance cognitive psychologists [for example see Hackman & Morris (1975)] have distinguished three main types of cognitive tasks: eliciting information, exploring courses of action, and evaluating situations. Step 1, corresponds to the first of these cognitive tasks, which within the SD community is referred as brainstorming. Step 2, is very similar to Duke's (1981) structured workshop technique where the participants write down on small pieces of paper all concepts that come to mind when thinking about the policy problem under study. This step, also draws from the example of the Object Behaviour Analysis approach [see Gibson (1990)] for object oriented design. Step 3, is similar to what SD modellers refer to as deciding what variables may be included or excluded from the model's boundary. Step 4 corresponds to the definition of the responsibilities of each object in the responsibility driven approach. Finally, steps 5 and 6 can be classified in the area of 'evaluating situations'.

In what follows, we describe through a model how the latter eight steps were carried out in modelling within DC.

7.5 Background of the Model & Issues to be Explored

The first step (STEP 1 in the above list), was to discuss during unstructured workshops the background of the model, and the broad issues to be investigated, as follows:
Given the March 1990 restructuring of the UK electricity industry from a single integrated public utility to several competing companies, two distinct markets for buying and selling electricity have emerged. The first is the pool market, that produces electricity prices on a half-hourly basis (see chapter 6 for a detailed presentation of the function of the Pool Market). The second is a contract market, which has emerged due to the fact that the buyers from the pool, or indeed the generators, do not wish to be wholly dependent on fluctuating pool prices. Therefore, they enter into contracts that reduce the pool price-induced variability of electricity purchase costs. These contracts can be conceived as financial instruments with a cash-flow determined by reference to the pool. The main

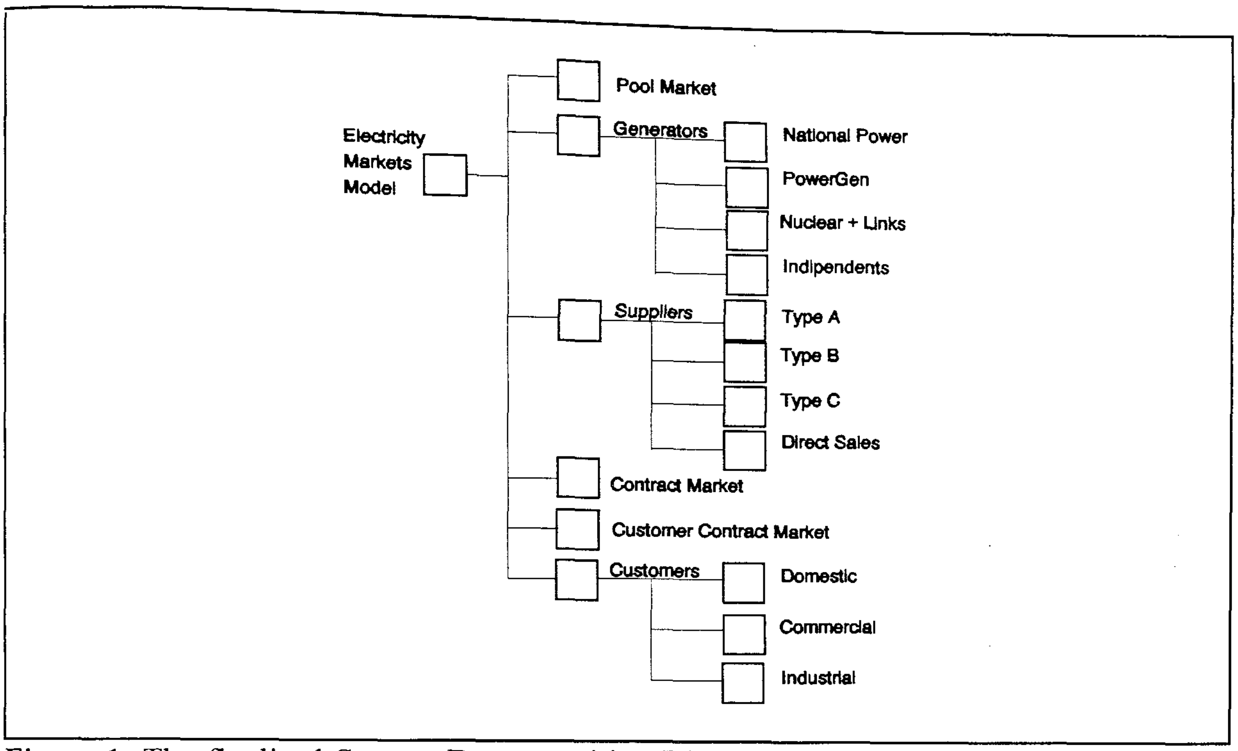


Figure 1: The finalised System Decomposition Diagram

suppliers of contracts are electricity generators, and the main buyers are the twelve distribution companies. Distribution companies are simultaneously buyers and sellers of electricity contracts. They sell contracts to large customers but they buy contracts from existing or the new independent generators. As a result, a second 'contract market' has been created, given that all non franchise customers can negotiate a contract for the purchase of electricity with any willing supplier.

Given the structure of the industry, as well as the coexistence of the tree markets the issues that were chosen to be investigated are:

- The study and modelling of the interactions between the electricity pool and the contract market.
- The development of scenarios about the competitive position of the distribution companies, given the imminent opening of the market to competition.
- The exploration of generators' policies in biding their plants to the Pool and their potential to manipulate the Pool Market.
- The study of the impact of a large number of gas and coal take-or-pay fuel

contracts on pool and contract prices.

- The exploration of the generator's policies in offering and pricing electricity contracts.
- How the total benefits in the system are allocated between parties.
- What is the effect of an abrupt change in circumstances (eg end of fossil fuel levy).

7.5.1 Model Structure

The second step (STEP 2) in the OO/DEVS approach, is to study the system that is being modelled and identify the main entities (objects) in the system. Given the issues under consideration the modelling team of DC suggested the following model objects:

- The Electricity Generation side
- The Distribution Companies side
- The Electricity Consumers
- The Pool Market
- The Contract Market between the generation and distribution sides
- The Contract Market between the distribution and the customers

This initial model cut provided the first level of the system decomposition diagram. In decomposing the latter model components, the modelling team of DC made the following model design choices:

- The **Electricity Generation Side** was decomposed into four Generation companies National Power, PowerGen, Nuclear Electric and a fourth company representing the independent generators (IPPs) within the industry.
- The **Distribution Side** was decomposed further into four 'types' of distribution company. It was decided that each type will represent (i) a different electricity purchasing approach, and (ii) a different customer targeting approach. Therefore, each company represents a 'generic' distribution company.
- The **Customer Side** was broken down to three types of customers: (i) the below 100 kW market, which corresponds to the domestic market, (ii) the market that corresponds to the range of 100 kW to 1 MW, and covers retailing and small

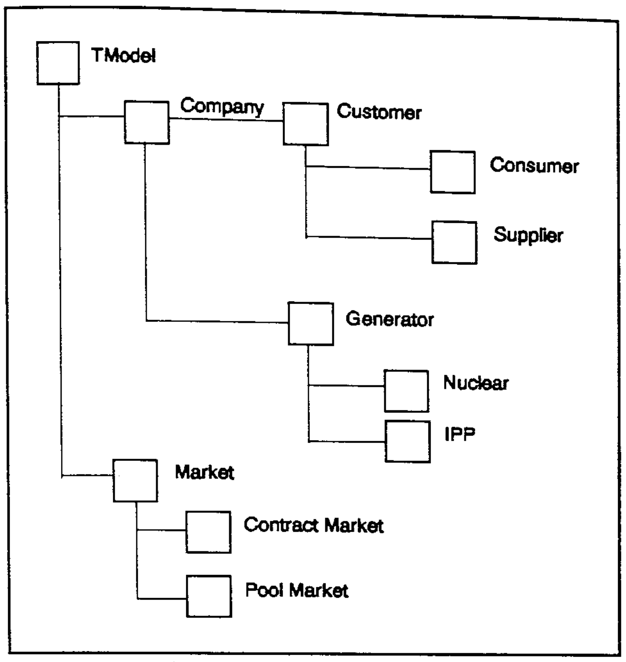


Figure 2: The Class Hierarchy Diagram

industry, (iii) the market over 1 MW which corresponds to the big industrial customers. A further subdivision, captive vs competitive market customer (or franchise vs non-franchise), was introduced in each of the above categories. The objective of the subdivision was to capture the fact that the market is gradually opening up to competition.

The next step (STEP 3), was to identify the required functionality of the different entities, and therefore set the system boundary. The functionality that was identified by the modelling team is based on the facts that the generators own plant, bid their plants to the pool, and offer electricity contracts to the buyers of electricity (distribution companies). The electricity pool receives the bids, produces plant schedule, and determines electricity prices. The distribution companies, buy contracts from the generators, buy electricity from the pool and offer contracts and tariff prices to the customers. Whereas, the customers (consumers of electricity) buy electricity, either through contracts from any of distribution companies, or by paying the tariff price to their local distribution company. The latter discussion constitutes effectively STEP 5.

Figure 1, depicts the decomposition diagram, that the DC modelling team arrived at, for the model as a whole at STEP 6. As we have pointed out earlier on in this thesis, a link from an upper level (for example Generators) to a lower level (for example National Power) can be interpreted as 'National Power is part of the Generators'. This structure allows a change of focus of the level of detail, in browsing the model, by'zooming' in or out.

STEP 6 was probably the most important process, regarding the model in hand, as the behaviour of the model objects was modelled in very elaborate decision rules, including

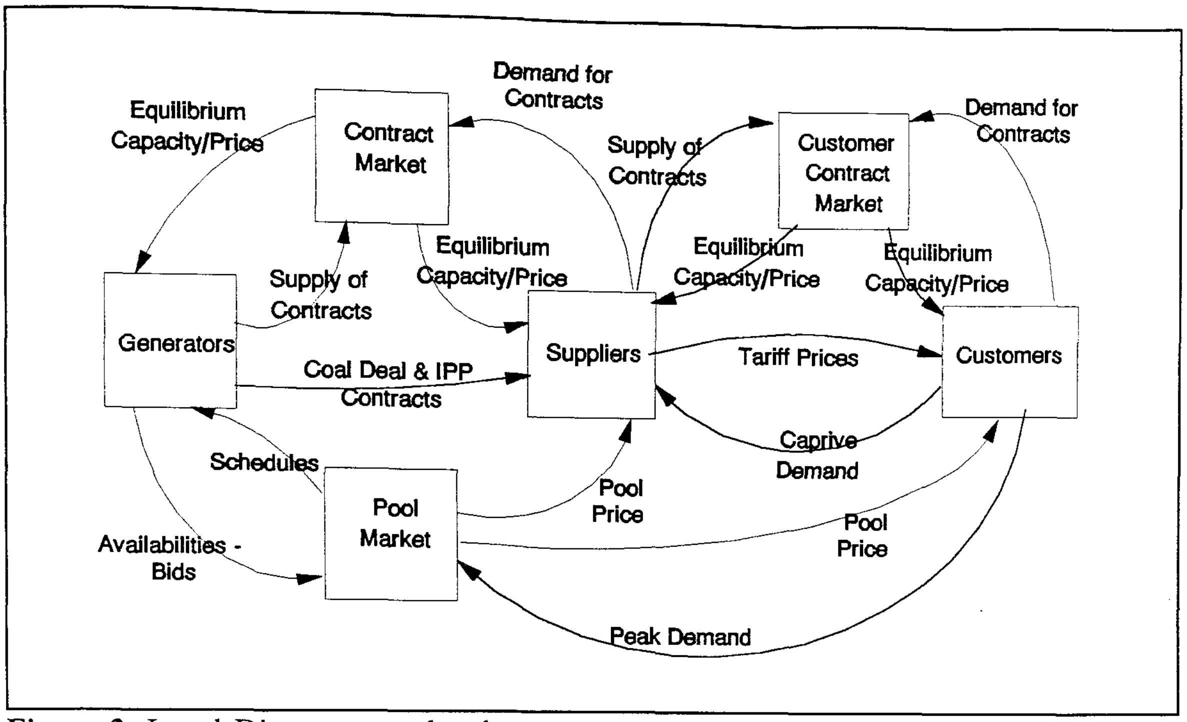


Figure 3: Level Diagram: top level

a significant amount of data, as well as information based on the mental models of the team. This step is discussed in detail in the next section.

Finally, making a number of design choices in STEP 8, a number of object classes was generated, encapsulating the required functionality. In creating these classes, common functionality and characteristics can be captured by general classes, from which more specific classes can be derived. A possible way of doing this is depicted in the object hierarchy of figure 2. All classes are subclasses of the basic entity class *TModel*, that contains essential simulation capability. A class *Company*, was suggested to capture the common characteristics of all companies, within the model. It was decided that these characteristics should include the maintenance of profit-and-loss and balance sheet data. Generation and distribution companies are specialisations of the class *Company* derived by adding functionality to this class. It was suggested that generators would be defined as instances of a class *Generator*, or its subclasses (IPPs and Nuclear were modelled as separate classes to reflect certain differences in their biding and contracting strategy). The distribution companies were defined as instances of the class *Supplier*. Finally, the pool and the contract market were designed to be subclasses of a class *Market*, which models the function of balancing demand and supply and thus producing the price for a product.

Two instances of the class contract market were used in the model, in order to reflect the existence of the contract market between Generators and Distribution Companies, as well as the existence of a contract market between the Distribution Companies and the Customers.

Following the specification of the system decomposition and the object hierarchy diagrams, the modelling team went back to STEP 4 to refine the influences between the different entities. This can be done by dissecting the system decomposition hierarchy at different levels and showing the influences using the level diagram.

Figure 3, for example, shows the interactions at the highest level. At this level **Generators** is the aggregation of all generators and **Suppliers** is the aggregation of all major electricity buyers (distribution companies). **Customers** represents the aggregate of the three types of customers, as they have been defined previously. The **Pool Market** balances demand and supply for electricity and determines electricity prices, while the **Contract Market** balances demand and supply for different types of contracts and determines contract prices. In addition, the **Customer Contract Market** balances supply and demand for contracts between the Customers and the Suppliers.

As can be observed, the Generators influence the Pool Market by bidding their plants at specific prices, and the Contract Market by supplying contracts. The Pool Market schedules the plants and produces schedules and SMP prices, which feed back to the Generators as well as to the Suppliers and Customers. The Suppliers, influence the Contract Market through their demand for contracts, and the Customer Contract Market through a supply of contracts. They also influence the Customers with their tariff prices. The Customers, finally, influence the Customer Contract Market through their demand for contracts, and the suppliers through their demand for actual electricity.

7.5.2 Encapsulated Decision Rules - Entity Behaviour

As we have pointed out earlier, STEP 6 proved of particular importance, as a lot of discussions and effort was devoted in modelling the decision rules of the model entities in great degree of detail. The step, is also important in producing the computerized OO/DEVS model, that can be meaningfully simulated.

OO/DEVS is very flexible when it comes to specifying behaviour (decision rules), which is modelled through the methods of the subclasses of TModel. As we have pointed out in Chapters 4 to 6, the user may contruct equations, logical rules, time-related events or even external algorithms for carrying out complex calculations.

It should be mentioned, that at this stage an 'Object Specification Form' was used, for the specification of each object's public and private behaviour, instances, variables, collaboration with other objects in the system and message specification. Such type of form is quite common to many object oriented design methods, and was felt that the modified version of it, adapted for OO/DEVS, helped considerably in developing the model objects. Examples of the Object Specification form, will be given further on in this chapter.

In what follows, we present a discussion of the decision rules and variables, that the modelling team of DC has suggested, regarding the main entities, of the model in hand:

(iii) **The Electricity Contract & the Contract Markets**

As we have pointed out at the beginning of this chapter, one of the steps in the OO/DEVS model conceptualization phase, is the identification of the main objects in the system. Apart from the model entities that we have already discussed the next most interesting object, from a modelling point of view, was the object **electricity contract**, which is traded in the contract market.

In reality the contract market is very complex, with a large number of distinct products. The basic type of contract is a two-way contract that requires (i) the generator to pay the customer the difference between the pool price and a reference price, the strike price, whenever the pool price exceeds the strike price, or (ii) the customer to pay the generator the difference between the strike price and the pool price whenever the latter is lower. The overall effect is that both parties are provided with a fixed price equal to the strike price, for electricity purchased under the contract. A variation of a two-way contract is the one-way type, which requires the generator to pay the customer the difference between the pool price and an agreed strike price for an agreed number of units whenever the pool price exceeds the strike price. This type of contract effectively caps the customer's electricity cost [see also James Capel &Co (1990)].

An electricity supply contract is usually based on a given amount of capacity (MW) for which the buyer often pays a fixed fee, the option fee. There are also minimum and maximum take constraints on the number of hours the contract can be exercised. In this respect we can identify base load and peak load contracts. The former have a low strike price and a high minimum take, whereas the latter have high strike price and low minimum take. Contracts can also be profiled in such a way that the contracted capacity varies throughout the contract duration. These contracts can offer customised type of cover.

Most contracts offer cover against variations of the System Marginal Price (SMP), but demand for uplift or capacity component cover exists and contracts may be offered that provide cover for the Pool Output Price (POP) or any of its components.

The duration of contracts may vary. On the one side of the spectrum, we have the contracts signed by the distribution companies with independent power producers (IPPs), which are in general long term (10-15 years), as well as the Coal Deal where until 1997/98 the distribution companies have to buy 30mt p.a. of the output of British Coal fired plants [Smith New Court (Dec 1992)]. On the other hand, we have the existence of short-term traded contracts (electricity futures arrangements or EFAs) with a duration of a few weeks. The vast majority of the contracts signed so far have durations longer than 1 year and EFAs have played so far only a marginal role. [see also Barclays de Zoete Wedd (1992), Smith New Court (Aug 1993)].

Finally, most contracts link the strike price to various escalators, mainly fuel prices and RPI. The following table summarises the important dimensions that characterise contracts:

CAPACITY	No of MW, min/max take, base/medium/peak load, profiling
PAYMENTS	Option fee, strike price, one-way, two way
LOAD	Base-medium-peak, take-or-pay, profiling
DURATION	Long term (eg IPP contracts),coal deal, short term
REFERENCE PRICE	SMP, Capacity Component, Uplift, combination of them
INDEXATION	Fuel prices, RPI

Table 1: The main dimensions of electricity supply contracts

Due to the complexity of the contract market, simplifications were considered to be necessary while care was required to maintain the main features of the market. Different contracts were grouped in a small number of contact types. It was decided, initially, in addition to the long-term contracts with independents and the coal deal, base, medium and peak load annual contracts to be considered according to the take constraints. Generators would decide how many contracts (no of MW) they would be prepared to sell, at different prices, for each type of contract. This in effect, is the supply curve for this financial product. Similarly, distribution companies and non-franchise customers would decide how much they are prepared to buy at different prices (demand curve).

From a modelling perspective, class Curve was created as the superclass of two subclasses: SypplyCurve and DemandCurve. Class curve models the commonalities between the two types of curve (eg. aggregation of a curve), while the two subclasses model the specific characteristics of demand and supply. As a result a **Contract Market** was modelled as the object where supply and demand curves can aggregated as they are submitted, demand and supply are balanced, and the equilibrium price is eventually

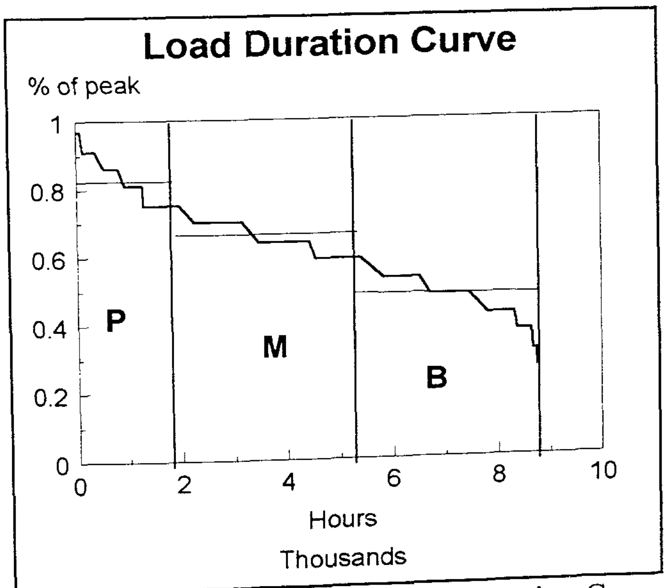


Figure 4. Dissection of the Load Duration Curve

calculated. It should be stressed, that the real contracts market operates as an auction, and therefore the above process is consistent with the way that market operates and clears.

As we have already discussed we have modelled contracts as annual distinguished in terms of base, medium and peak load. The distinction in terms of load, is based on the number of hours assigned to each load, as a percentage of the total hours in a year. It was decided to assign 40% of the total hours to Base load, 40% to Middle, and 20% to peak. This split yields 3514 hours in the Base slot, 3396 hours in the Middle slot, and 1825 hours in the Peak. By superimposing this split on the Load Duration Curve, we can also find the plant load factor needed to produce the energy corresponding to each type of load (see figure 4 and table 2).

	Peak Load	Middle Load	Base Load
Hours	1 - 1825	1825 - 5222	5222 - 8736
% of Total	20%	40%	40%
Load Factor	82%	65%	47%

Table 2: The split of the Load Duration Curve

The above view of contract modelling, even though was initially considered quite abstract, was overall regarded attractive, because it proved a very versatile tool in modelling a wide range of supply curves (corresponding to actual plant stacking), as well as a wide range of demand curves (expressing demand preferences). Such curve representations allowed the modelling team to debate the different levels of risk aversion on the part of distribution companies by shifting the demand curve to the left or their right. Similarly, a squeeze of the contract market by the generators was modelled by moving the supply curve for contracts upwards. Other types of oligopolistic behaviour were also discussed. Regarding the 'Customer Contract Market', the distribution companies supply contracts (which reflect the mix of their own contract market and pool purchases), while the Customers (the Consumers of electricity) submit demand curves which reflect their willingness to contract for electricity. In addition to the above, the modelling view of a curve which can be represented generically as an object, that can be aggregated or find its equilibrium with another curve, proved very concise during the actual OO/DEVS model building.

Overall, contracts were modelled as:

(i) Short term contracts (annual) which are represented as supply curves for base, medium and peak loads. The price axis of the supply curve reflects the bid prices of the plants plus a contract premium. Each curve is composed by stacking up the plants in terms of price while the capacity axis contains the cumulative plant capacity. Plants are distinguished into the three fore-mentioned types of load in terms of their utilization. For instance, plants that have availability of 60% or more compose the 'base load' curve, plants that have 20% or more availability compose the 'medium load' curve, while the remaining plants compose the 'peak load' curve.

(ii) Coal Deal contracts that have a duration of 4 years and are represented as a one point supply curve, where the quantity (MW) represents the percentage of the National Power and PowerGen output allocated to the Coal Deal, whereas the price is predefined for each of the four years. The Suppliers split the Coal Deal in accordance to their share of the domestic market [see Smith New Court (Dec 1992), pp. 49-51 for details regarding the coal deal].

(iii) The IPP contracts, which have been modelled as a one point supply curve which feeds directly to the suppliers. The quantity included in the IPP contracts corresponds to the output of the Independent generators, and the price is linked to the bid price of the IPP plants plus a fixed cost.

Object Name: Contract Market **Inherits from**: Market	
Public Behaviour: Receive Demand Curves Receive Supply Curves Find the equilibrium between demand and Supply	**Variables:** supply, demand, equilibrium price
Private Behaviour: Aggregate Demand Curves Aggregate Supply Curves	
Instances: (2 instances) Contract Market (between Generators and Suppliers), Customer Contract Market (between Suppliers and Customers)	
Collaborates with the Object: Consumer, Supplier, Generator, Supply Curve, Demand Curve	

Table 3: Object Specification Form (partial) for object Contract Market

(ii) The Customers

As we have mentioned earlier on, the modelling team of DC has suggested the split of the customers into three groups, i.e. 'Domestic', 'Commercial' and 'Industrial', with a further sub-division for each of the groups into a 'competitive' and a 'captive' part. An initial allocation between captive and competitive is defined at the beginning of the simulation run, with subsequent changes year on year to reflect the development of the market. It was felt, that this feature, would provide an interesting dynamic element in the model. Customer demand is initialized to 50,000 MW (peak demand) for the first year of the simulation, and increases thereafter with a rate of 1.1% per annum. In addition, customers have a base, medium and peak demand based on the split presented in Table 2.

Each customer has a decision rule that formulates next year's price expectation, based on current year's price plus some expected change. Based on this price expectation a Customer formulates a demand curve, for each of the fore-mentioned type of load, expressing its price preferences using the tariff price as a bench mark. The shape of the customer demand curve, denotes an inelastic demand, given that the customers will always buy electricity at any price below the tariff price.

The most important variable for each customer category, was considered to be the function of the outturn price vs expected price. This function was set a measure of customer value in the system, and initially was set as a the fraction of outturn price over expected.

Object Name: Consumer **Inherits from**: Company\Customer	
Public Behaviour: Produce Captive Demand Produce Competitive Demand Bid for contracts Receive contracts Receive PIP	**Variables:** demand, contract cover, demand Growth Rate, captive Demand, competitive Demand, expected Captive Price, actual Captive Price, degree Of Satisfaction, captive Percentage, duos, pip
Private Behaviour: Produce Demand preferences for contracts Aggregate Supply Curves	
Instances: (3 instances) Domestic, Commercial, Industrial	
Collaborates with the Object: Generator, Contract Market, Demand Curve	

Table 4: Object Specification Form (partial) for object Consumer

(iii) **The Suppliers**

In the beginning of each financial year the Suppliers (distribution companies) have to set tariffs and offer contracts to customers, having an estimate of what the pool price will be. Given the experienced pool price volatility, the risks of over or under-charging are very high. The main role of electricity supply contracts has been to reduce and if possible eliminate this risk. Distribution companies would prefer to be fully covered for their forecast demand, if the risk premium involved is not very high. The risk premium is measured as the difference between the cost of buying electricity through a contract and the cost of buying from the pool (net contract cost).

Cost stability is a key factor in achieving a number of other objectives such as increasing customer satisfaction, broadening the customer base, avoiding conflicts with the regulator and pleasing the City. But cost stabilisation is not the only objective in determining the level of cover and the composition of the contract portfolio. The cost of the cover is also an important consideration, despite the fact that regulation allows distribution companies to pass this cost on to franchise customers. Electricity companies are competing with other energy companies and with each other, thus cheap energy supply will in the long run be a definite competitive advantage. In addition gross inefficiencies will attract the attention of the regulator, since'economic purchasing' is part of the licence requirements. Distribution companies have also used supply contracts to influence developments in the generation market. By signing long term contracts with independent power producers they tried to reduce the oligopolistic power of National Power and PowerGen and they managed to establish a sizeable new competitive force in this market. As we discussed earlier on, the IPP contracts have been represented explicitly in the model.

A Supplier participates both in the pool and the contract market. In the pool the Supplier buys the electricity it needs for its committed demand. In the contract market the Supplier purchases contracts to reduce the variability of its electricity purchase costs. We model the Suppliers' preference for contracts through a demand curve for contracts. The initial decision rule is that if the net contract cost is zero for a particular contract, a Supplier is prepared to buy enough to satisfy its expected demand. For higher risk premiums, they are prepared to reduce the level of cover and take some pool risk. Required electricity purchases change each year as a result of success or otherwise in the 'Competitive Market'. Suppliers will formulate a view of their expected success in the Competitive Market and use this, plus their captive market commitments to determine their purchasing targets from the Generators. This will in turn depend on their expectations of prices and available volumes in the Pool and Contract Markets.

As we have mentioned earlier on in this chapter, the modellers made the design choice to define different types of suppliers in terms (a) of their purchasing behaviour (i.e. pool risk aversion in buying contracts, IPP contract purchasing) and (b) of their selling behaviour (i.e. by offering different 'product ranges', eg some only offering to the captive market, others also offering to the 'Competitive Market', on different bases such as 'Pool + Margin', 'Fixed Price', 'Contract + Margin' etc). It should be pointed out that one of the instances of the class Supplier, represents the 'Direct Sales' companies set up by the Generators willing to compete in the Customers Contract Market [see Smith New Court (Dec 1992) p.29]. As a result 'Direct Sales' do not receive any coal deal or IPP contracts, and do not target the domestic customers. The objective of introducing these types of supplier was to allow different commercial strategies to be compared. It should be pointed out this was considered by DC, as one of the key aspects of model development as the final model would show how different approaches fare over the years in terms of market share and profitability.

Object Name: Supplier **Inherits from**: Customer	
Public Behaviour: Demand for Contracts Supply of Contracts Receive Captive Demand Receive Contracts	**Variables:** Existing Contracts, Yearly Contracts, Captive Demand, Competitive Demand, Expected Competitive Demand, Revenues, Costs, Profit Margins for tariffs and contracts, Duos, Tuos, VAT, Levy
Private Behaviour: Formulate Competitive Customer size expectations Formulate demand preferences for contracts Formulate contract supply curves	
Instances: (4 instances) Types A, B and C, and Direct Sales	
Collaborates with the Object: Supply Curve, Demand Curve, Contract Market, Pool Market, Consumer	

Table 5: Object Specification Form (partial) for object Supplier

(iv) The Electricity Pool Market

The electricity pool balances demand and supply for electricity and calculates electricity prices. Generators bid for their plant capacity on a daily basis and based on forecast

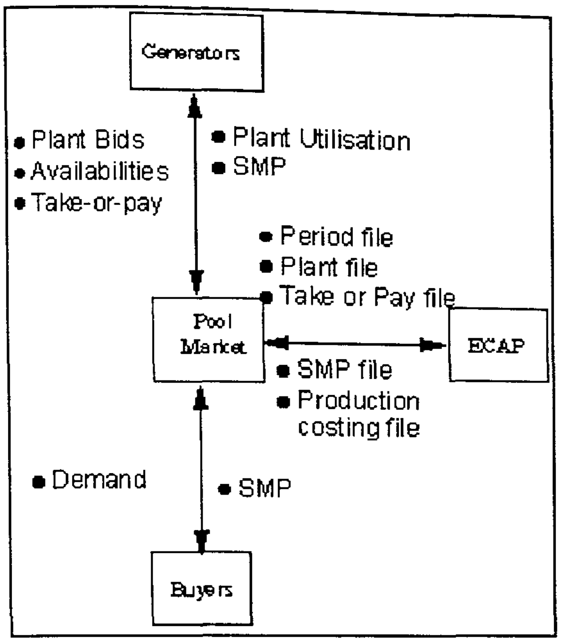


Figure 5: The interface of the OO/DEVS Electricity Markets Model to ECAP

demand a day-ahead schedule is calculated. In this model, since we are interested in medium to longer term interactions, an aggregate view of the pool is taken. The pool price calculation is annual, but accurate enough since it uses the ECAP (Vlahos 1989) production costing algorithm.

It should be noted that, given a set of plants, their bid prices, availabilities and take-or-pay constraints, as well as a demand profile (load duration curve), the ECAP algorithm will generate the optimal production schedule putting the plants in a merit order and produce the System Marginal Prices corresponding to the a number of seasons for any year. Such a representation of the Pool Market is necessary, as it provides an accurate model of the plant economics in the system, which is the basis to explore realistically the strategic behaviour of the system entities (figure 5 provides an overview of the interface with ECAP and the OO/DEVS model).

Object Name: Pool Market **Inherits from**: Market	
Public Behaviour: Receive Plant Bids Receive peak demand Produce SMP Produce Schedules	**Variables:** smp, lolp, uplifts, schedules plants, availability patterns, take or pay patterns, peak demand
Private Behaviour: none	
Instances: (1 instance) Pool Market	
Collaborates with the Object: Consumer, Supplier, Generator, Plant	

Table 6: Object Specification Form (partial) for object Pool Market

(iv) The Generators

It has been said and it is true that without the income from contracts both PowerGen and National Power would now be bankrupt. The levels of pool prices that have prevailed in the first two years of pool operation, are hardly adequate to cover costs (see figure 6). Hence, it would seem perfectly rational for them to be very keen to offer new contracts to distribution companies and to extend existing ones.

Instead they pursued a dual track strategy. Firstly, they entered the supply business very aggressively, taking market share from distribution companies (we have modelled this fact through a supplier called 'Direct Sales' which targets only the competitive market, and has no IPP or Coal Deal contracts). Secondly, they used the threat of pool price manipulation to force contract prices at levels much higher than pool prices. Towards the end of 1991, the coefficient of variation (ratio of standard deviation to the average) of pool prices went up to about 60% (!), double the level of the previous 12 months [OFFER (1991)].

This strategy of the generators appears to be double edged given that distribution companies are their largest customer, buying the electricity from the generators in the first place. In the supply business distribution companies have a competitive advantage in that

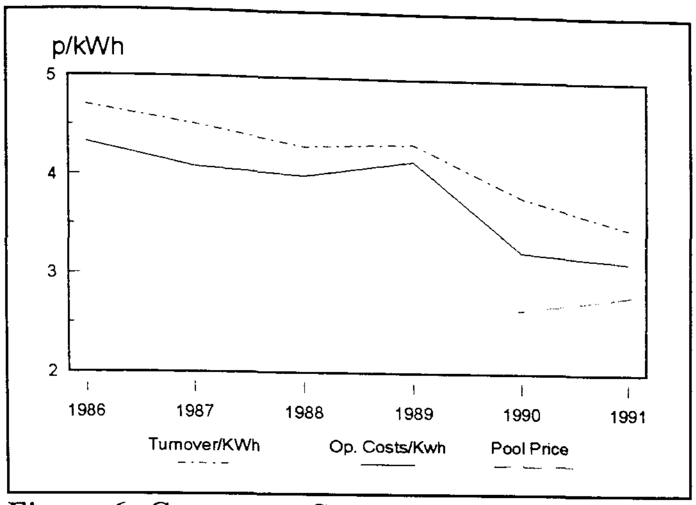


Figure 6: Generators Costs vs Pool Prices

they already have the sales infrastructure that the generation companies lack and a long established relationship with their customers. However, the generators in trying to gain market share have to offer low prices. The overall effect will probably be that prices decrease and margins erode as generators and distributors fight for the same customers.

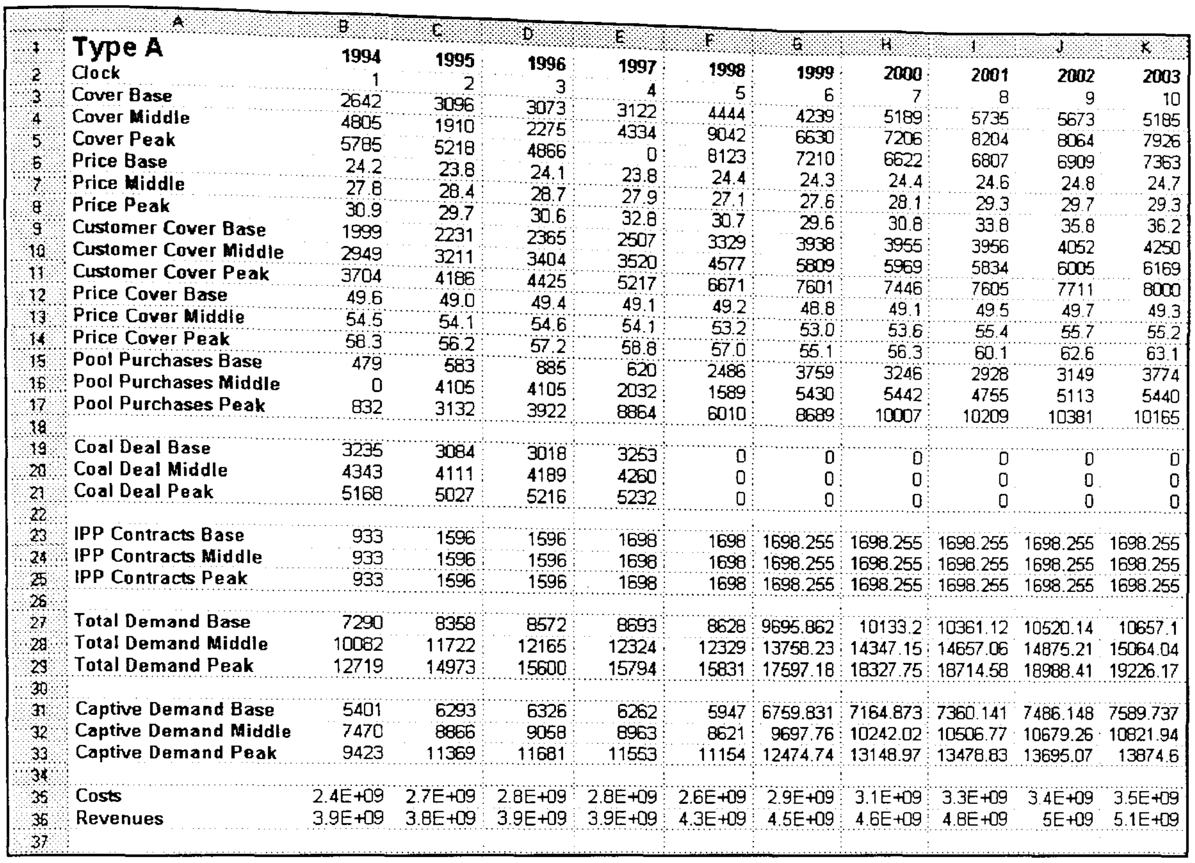
The generators have two distinct ways to apply their strategy. The first one is through bidding their plants to the pool. In addition, generators supply the contract market with contracts of the types that we described earlier on. They have many ways to influence both markets. They can affect the pool market by employing bidding tactics, such as making plant unavailable to the pool or varying the bid prices. They can also decide to offer more or less contracts to more or less attractive prices. All these possibilities need to be investigated. But as a starting point, our decision rule assumes cost reflecting bidding (cost + margin), and a contract supply curve that presumes willingness to contract most of their capacity. This reflects their publicly declared intentions.

In terms of modelling, each plant in the system, is represented as an instance of the class *Plant*. This class encapsulates specific plant characteristics, like its name, owner company, capacity, availability, last utilization, type of fuel, economic life, as well as starting and ending production date. As a result, each Generator owns a set of such objects. It should be noted, that prior to bidding its plants to the pool, each generator groups them in base medium and peak plants, using as a bench mark last years plant utilization (the split of 40% for base load plant, 40% for middle load plant, and 20% for peak load plant is used, as has already been presented earlier in this chapter). Based on this grouping, the generators bid their plant to the pool adding a different mark up for each type of load.

Finally, investment and disinvestment have been included, as the generators in our model bring new plant into production while they retire old plant capacity. This is achieved externally (i.e. there are no actual investment or retirement rules in the model) through the fact that each plant has a starting and ending production date. Plant investments and retirements reflect the declared intentions of the generators [for example see Smith New Court (Dec. 1992) & OFFER (1993)].

Object Name: Generator **Inherits from**: Company	
Public Behaviour: Plant Bids Supply Contracts	**Variables:** plants File, supply, plants, sales Contracts, bids, capacity, utilizations, smps, mark ups, coal Deal, non Market Contracts
Private Behaviour: Formulate Plant Bids Formulate supply contract preferences	
Instances: (4 instances) National Power and PowerGen, 2 more instances of two subclasses: IPP and Nuclear	
Collaborates with the Object: Supplier, Supply Curve, Plant, Contract Market, Pool Market	

Table 7: Object Specification Form (partial) for object Generator



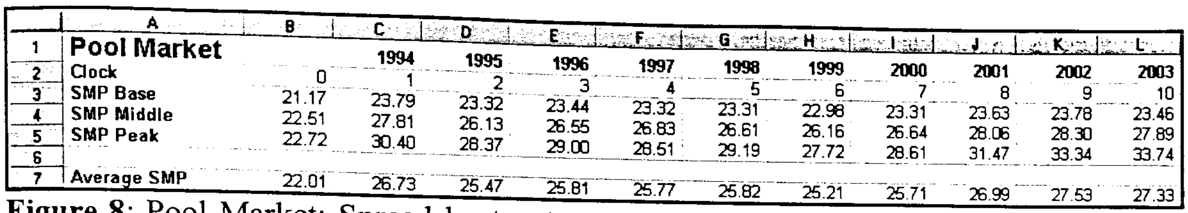
	A	B	C	D	E	F	G	H	I	J	K
1	**Type A**	**1994**	**1995**	**1996**	**1997**	**1998**	**1999**	**2000**	**2001**	**2002**	**2003**
2	Clock	1	2	3	4	5	6	7	8	9	10
3	Cover Base	2642	3096	3073	3122	4444	4239	5189	5735	5673	5185
4	Cover Middle	4805	1910	2275	4334	9042	6630	7206	8204	8064	7926
5	Cover Peak	5785	5218	4866	0	8123	7210	6622	6807	6909	7363
6	Price Base	24.2	23.8	24.1	23.8	24.4	24.3	24.4	24.6	24.8	24.7
7	Price Middle	27.8	28.4	28.7	27.9	27.1	27.6	28.1	29.3	29.7	29.3
8	Price Peak	30.9	29.7	30.6	32.8	30.7	29.6	30.8	33.8	35.8	36.2
9	Customer Cover Base	1999	2231	2365	2507	3329	3938	3955	3956	4052	4250
10	Customer Cover Middle	2949	3211	3404	3520	4577	5809	5969	5834	6005	6169
11	Customer Cover Peak	3704	4186	4425	5217	6671	7601	7446	7605	7711	8000
12	Price Cover Base	49.6	49.0	49.4	49.1	49.2	48.8	49.1	49.5	49.7	49.3
13	Price Cover Middle	54.5	54.1	54.6	54.1	53.2	53.0	53.6	55.4	55.7	55.2
14	Price Cover Peak	58.3	56.2	57.2	58.8	57.0	55.1	56.3	60.1	62.6	63.1
15	Pool Purchases Base	479	583	885	620	2486	3759	3246	2928	3149	3774
16	Pool Purchases Middle	0	4105	4105	2032	1589	5430	5442	4755	5113	5440
17	Pool Purchases Peak	832	3132	3922	8864	6010	8689	10007	10209	10381	10165
18											
19	Coal Deal Base	3235	3084	3018	3253	0	0	0	0	0	0
20	Coal Deal Middle	4343	4111	4189	4260	0	0	0	0	0	0
21	Coal Deal Peak	5168	5027	5216	5232	0	0	0	0	0	0
22											
23	IPP Contracts Base	933	1596	1596	1698	1698	1698.255	1698.255	1698.255	1698.255	1698.255
24	IPP Contracts Middle	933	1596	1596	1698	1698	1698.255	1698.255	1698.255	1698.255	1698.255
25	IPP Contracts Peak	933	1596	1596	1698	1698	1698.255	1698.255	1698.255	1698.255	1698.255
26											
27	Total Demand Base	7290	8358	8572	8693	8628	9695.862	10133.2	10361.12	10520.14	10657.1
28	Total Demand Middle	10082	11722	12165	12324	12329	13758.23	14347.15	14657.06	14875.21	15064.04
29	Total Demand Peak	12719	14973	15600	15794	15831	17597.18	18327.75	18714.58	18988.41	19226.17
30											
31	Captive Demand Base	5401	6293	6326	6262	5947	6759.831	7164.873	7360.141	7486.148	7589.737
32	Captive Demand Middle	7470	8866	9058	8963	8621	9697.76	10242.02	10506.77	10679.26	10821.94
33	Captive Demand Peak	9423	11369	11681	11553	11154	12474.74	13148.97	13478.83	13695.07	13874.6
34											
35	Costs	2.4E+09	2.7E+09	2.8E+09	2.8E+09	2.6E+09	2.9E+09	3.1E+09	3.3E+09	3.4E+09	3.5E+09
36	Revenues	3.9E+09	3.8E+09	3.9E+09	3.9E+09	4.3E+09	4.5E+09	4.6E+09	4.8E+09	5E+09	5.1E+09
37											

Figure 7: Supplier Type A; Spreadsheet output

7.6 Running the Electricity Markets Model

As soon as the Electricity Markets Model was built and tested, the modelling team identified the main variables of interest, within each of the model entities. These variables were monitored within a spreadsheet environment (Quarto Pro for Windows) through the spreadsheet output facilities, build within OO/DEVS. Naturally, variable monitoring provided a second line of model testing and debugging.

It should be pointed out, that within the spreadsheet each model entity was represented in a different sheet. In that respect, fourteen different sheets were used to output variables from the fourteen basic model entities (i.e. four generators, four suppliers, three consumers, the contract market, the customers contract market and the pool market). Each sheet includes a column with the variable names (as model output), and subsequent rows with the corresponding values produced during the simulation run. The sheer amount of



	A	B	C	D	E	F	G	H	I	J	K	L
1	**Pool Market**		1994	1995	1996	1997	1998	1999	2000	2001	2002	2003
2	Clock	0	1	2	3	4	5	6	7	8	9	10
3	SMP Base	21.17	23.79	23.32	23.44	23.32	23.31	22.98	23.31	23.63	23.78	23.46
4	SMP Middle	22.51	27.81	26.13	26.55	26.83	26.61	26.16	26.64	28.06	28.30	27.89
5	SMP Peak	22.72	30.40	28.37	29.00	28.51	29.19	27.72	28.61	31.47	33.34	33.74
6												
7	Average SMP	22.01	26.73	25.47	25.81	25.77	25.82	25.21	25.71	26.99	27.53	27.33

Figure 8: Pool Market; Spreadsheet output

information outputted to the spreadsheet work-sheets (which in no way represents the whole information included in the model), presents a good measure of how well information can be structured and represented within OO/DEVS.

Figure 7, depicts the model output for one of the suppliers (Type A), represented in the model. As can be observed, the variables of interest regarding the instances of class Supplier, include the amount of capacity bought from the contract market (for the fore-mentioned three types of load), the amount of capacity bought from the Pool Market, and their corresponding prices, as well as the amount of capacity bought in Pool Deal and IPP contracts. The demand that the supplier has to meet, as well as cost and revenue figures are also included.

The information presented in Figure 7, can be seen as a way of verifying the actual model behaviour through its decision rules. For example, it can be observed that the Coal Deal ceases to exist after 1997, as well as the fact that the annual values corresponding to capacity bought from the Coal Deal and IPP contracts vary from year to year. The latter variation, reflects the plant investment and disinvestment built into the model. It is also interesting to observe the movements of the amounts of capacity bought between the Contract Market and the Pool Market. Indeed, if the contract market prices are compared to the POP (Pool Output Price) values depicted in Figure 8, then it can be seen how the Supplier makes capacity purchasing decisions, given the level of the pool prices (note that every supplier has a degree of pool risk aversion, and therefore always contracts capacity).

Having identified the variables of interest in the model, the modelling team suggested a number of control variables, within each model entity, which would be interesting to experiment with. Based on this set of variables, a number of scenarios was set up, and the behaviour of the model was explored. In what follows, we present one of these scenarios with the objective to demonstrate further the functionality/behaviour of the model, as well as to give a flavour of how the model is used to produce useful insights.

One of the objects of particular interest was the class Supplier and its instances. As we have discussed earlier on, the different types of supplier model different electricity purchasing and selling strategies. In the scenario discussed herein, we concentrate on the purchasing policies of the Suppliers. These policies are based on the capacity purchased from the Pool Market, the Contract Market and the IPP and Coal Deal contracts. As a result, each instance of Supplier can control the capacity bought from the four above sources. The scenario presented here is based on the following assumptions:

- All suppliers (except 'Direct Sales') have the same Captive market share, and all suppliers target the same percentage of the Competitive market (i.e. expected size of their market of 30%).
- 'Direct Sales' has no Captive market share.
- All suppliers buy a percentage of the Coal Deal contracts based on their share of the Captive market (i.e. 'Direct Sales' have no Coal Deal contracts).
- All suppliers (except the 'Direct Sales') share equally the IPP contracts (i.e. 1/3 of the total capacity in IPP contracts, each).
- 'Direct Sales' have no Coal Deal or IPP contracts
- All suppliers have a contracts demand curve, which is represented as a line the slope of which represents the pool risk aversion of the specific supplier. Each supplier will buy all its demand for electricity in contracts if the price will be the same as POP. Otherwise, the supplier will buy less, given the degree of its risk aversion. The aggregate demand curve for contracts, as it is submitted in the contract market is depicted in Figure 9.
- Supplier A has a slope of 1.1 (that means that they will not pay more than 10% over the POP for capacity, as a risk premium).
- Supplier B has a slope of 1.3
- Supplier C and the 'Direct Sales' have a slope of 1.5 (they are prepared to pay as much as 50% over POP as risk premium).

Figure 10 depicts the purchasing mix of the four types of supplier, as it can be clearly

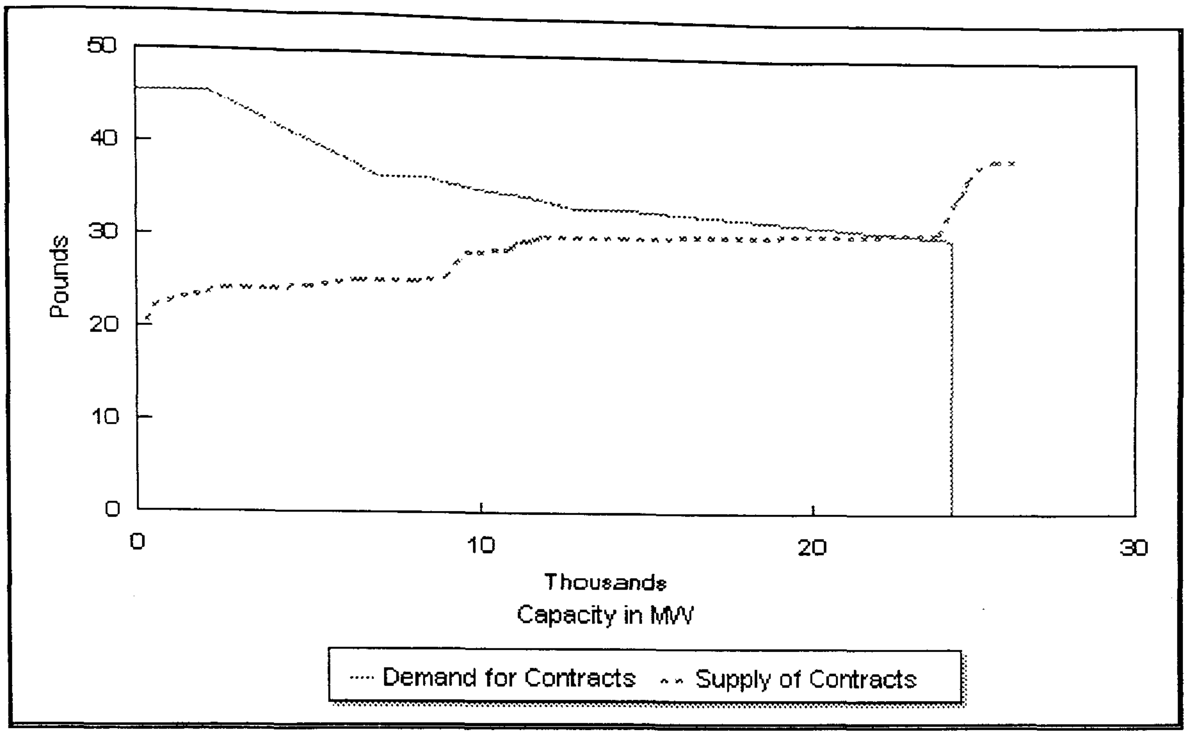


Figure 9: Demand vs Supply Curves (for Peak Load Contracts)

seen the less risk averse (Type A) buys more capacity from the pool, while the least risk averse (Type C and the 'Direct Sales') make almost no pool purchases. The most interesting result is however depicted in figure 11, which shows the relative market shares of the four Suppliers. As can be observed, Types A B and C fare in a similar way up to the end of the coal deal. The differences in their market shares is a result of their differences in pool risk aversion. As a matter of fact, before the termination of the coal deal, the pool risk averse supplier (Type C) is increasing its market share, in contrast to the risk taking one. Nevertheless, after the end of the coal deal the positions of the suppliers are reversed, and the least pool risk averse supplier gains a considerable amount of market share. In addition, the 'Direct Sales', being in a market share gaining position before the end of the coal deal, find itself loosing market share rapidly .

In observing the above behaviour two comments can be made:

(a) The existence of the coal deal, taking a quite large portion of the demand of the suppliers, works as a straightjacket regarding market behaviour. As a consequence, the coal deal results into less contracts offered to the Contract Market, at less attractive prices. Nevertheless, after the end of the coal deal the Contract Market gains in significance as

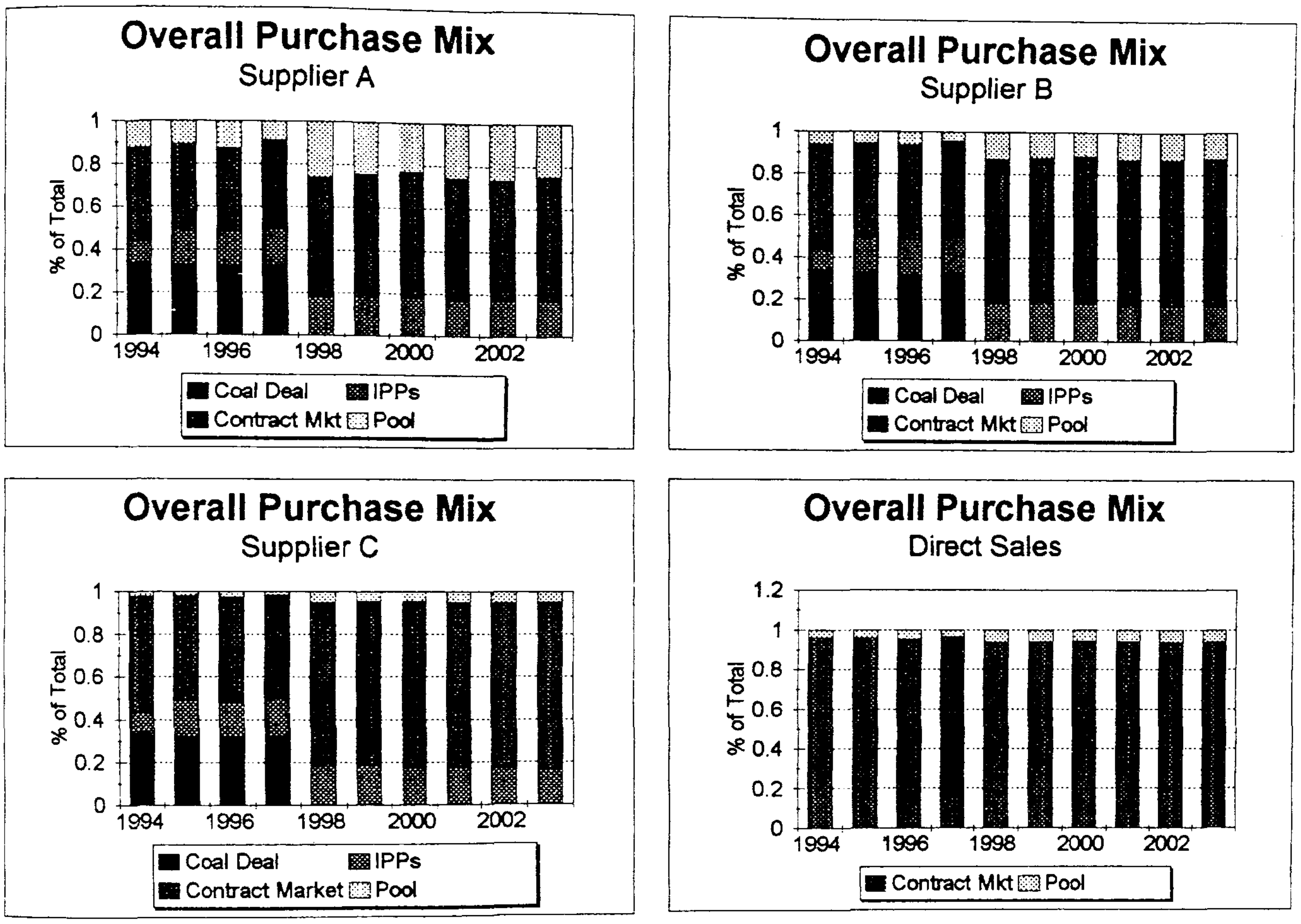


Figure 10: Purchase Mix for the four types of Supplier

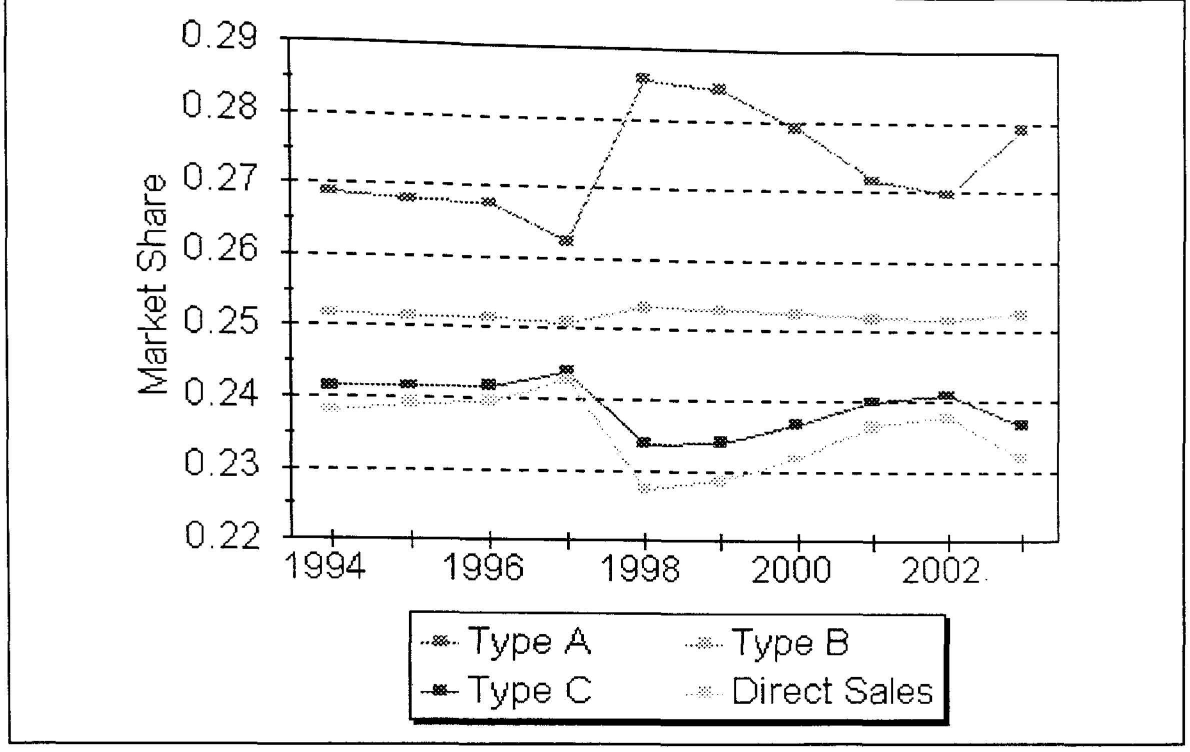


Figure 11: Relative Market Shares for the four types of Supplier

a large proportion of electricity is treated in that market (see Figure 10). Additionally, as an overall bigger market is in operation, the less risk averse players improve their competitive position as they can offer cheaper prices and thus gain market share.

(b) The result depicted in Figure 11, regarding the fact that the 'Direct Sales' loose market share rapidly after the end of the coal deal, strengthens the argument that a systemic view should be taken in analysing the privatised electricity industry. The 'Direct Sales' do not behave any differently, in terms of their purchasing policies, before or after the end of the coal deal. However, the key to the interpretation of this finding is that the coal deal is priced at prices higher than the average POP, as a result after the end of the deal, Suppliers A B and C find themselves buying cheaper electricity and gaining comparatively more market share.

Finally, we should stress that in the above scenario no explicit market manipulation strategies have been modelled, for any of the model entities. The above behaviour is the result of the structure of the market and the very basic behaviour of the market participants. In that respect, we find particularly attractive the fact that even under a very simple scenario useful insights can be gained, regarding the long term evolution of the industry.

7.7 Discussion

As we have pointed out in Chapters 2 & 3, the development of OO/DEVS has been motivated on the basis on what we summarised as: structure, focus, time-representation and reusability. Having developed a model within a business environment we can now discuss how the modelling team of DC has viewed the above four issues, from a practitioner's perspective. The views presented herein, correspond to a written evaluation of the OO/DEVS framework and environment, that the DC modelling team provided at the end of the modelling exercise.

Structure & Focus

In terms of *structure* the concept of the object was considered particularly attractive as: "*...building up a model from 'Objects' mirrors the natural world and allows actual characteristics to be modelled*". In addition, as it has been stated: "*The concept of an 'Object' is very wide ranging, covering both physical entities (eg. Generators) and more abstract constructs (eg. Curve), yet all can be handled within the same framework*". The idea of an object as a data container and manipulator, was viewed as a powerful modelling device due to the fact that "*objects can be created to simplify the handling and combination of large amounts of data, essentially in matrix form, extending the scope of the models*". As we discussed in the previous section, the modelling team appreciated particularly the structured way of modelling and handling information within OO/DEVS, as soon as the model output was presented in a spreadsheet form.

Inheritance (i.e. generalization relationships) was regarded as a useful tool as: "*New objects can be rapidly created using others as templates, while still retaining all the characteristics of the original*". It has to be stressed, that inheritance proved valuable, at the point that it was felt that the model should be extended, in order to capture some particular characteristics of the model entities. This was the case, with the 'plant biding decision rule' of Nuclear and the Independent Generators. While the two entities had all the characteristics of the class EGenerator, it was felt that their actual (in the real system) behaviour required a slightly different way of modelling. This was achieved by creating two subclasses of the class EGenerator and overloading one of its methods (method bid).

The provision of aggregation relationships was also judged as a useful modelling feature regarding model focus as: "*Objects of the same type can be aggregated together (eg to simulate mergers)*" and "*Models can be built and viewed at different levels of detail/dissagregation in particular areas, while still retaining full compatibility* [with the OO/DEVS model as a whole]".

In addition, the representation of association relationships within OO/DEVS was considered as a feature providing considerable flexibility in designing and building models as: "*Links can be set up between any components in the model, allowing particular interactions to be explored*".

Overall, the use of the decomposition diagram, the level diagram, as well as the object specification forms, were judged as practical tools for conveying information about the system structure. The level diagrams in particular, proved a useful aid in discussing the ways that the model entities influence each other.

Finally, the potential to link OO/DEVS with other models and spreadsheets was viewed as a particularly powerful feature, as in this instance allowed ECAP to be integrated within the model. The integration of ECAP allowed a very important aspect of the system to be modelled precisely, using a very elaborate model, whose development preceded OO/DEVS itself. In a similar fashion, the ability to link OO/DEVS to spreadsheets, facilitated model development, debugging and presentation by giving access to a wide range of tools provided within spreadsheet environments. Overall, OO/DEVS was viewed not only as a modelling and simulation platform, but as a possible integrator of existing corporate models and information.

Time-Representation

It was overall felt, that as the modellers took a year by year view in modelling the Electricity markets, that the discrete event view provided by OO/DEVS was useful in modelling the interactions within the industry, as well as features like the Coal Deal and the introduction and retirement of new plants into the system. It should be pointed out, that the latter examples are the very instances which are difficult to model naturally within SD, and it is the exact case where one would use discrete delays in *ithink*. Overall, it was commented that: "*The use of messages to trigger methods in receiving objects, allows control over the timing of particular events. It should also make it easier to incorporate processes occurring part-way through a simulation cycle*".

Reusability

It should be said that the model components (objects) of the 'Electricity Markets Model' were designed with the primary objective that will be reused. In that respect objects like Curve, Plant, Contract Market and Pool Market were designed and built so that they capture the characteristics of the industry in a generic fashion. The expectation of the modelling team is that the objects of this model will constitute the onset of a OO/DEVS electricity industry model base.

Overall, OO/DEVS has proved in practice effective, in a number of areas that could not be supported by SD modelling (and software like *iThink*), these areas include:

- The natural representation of the model components, which proved invaluable in designing the model and communicating its characteristics to the management team.
- The decomposition and level diagrams (with their GUI version) proved particularly useful tools in viewing the model from different perspectives.
- The ability to reformulate the model fast, changing either decision rules, message passing between model entities or even the overall structure of the model during the model conceptualization phase.
- The integration of 'hard' and 'soft' decision rules in the same object, or within different objects in the same model.
- The concepts of inheritance and aggregation proved particularly useful in moving

from the prototype of the model in more elaborate versions, where previously developed model components could be readily used, while different model views (aggregates) could be easily explored.

- The ability to link the model to a spreadsheet was considered particularly useful for model debugging purposes, as the results of the model could be readily used for verification through the use of existent financial models.

Finally, a number of drawbacks of the framework was pointed out by the DC modelling team. It was commented that OO/DEVS "*Still requires familiarity with Smalltalk, which is most readily achieved by users with a programming background. This is particularly apparent in specifying methods and decision rules*". It was also suggested that the use of some Smalltalk jargon (eg. terms like class, instance, etc) could be problematic for the unfamiliar user. In addition to the above, the need for greater auditability and the ability to track through processes and establish why particular results occur, was identified. Overall, the considerable size of the model and the bulk of information contained in it, suggested the need for a set of tools for enhanced model browsing and debugging.

It should be pointed out that the above drawbacks are mainly related to the current state of the development of the framework. In relation to the first point in particular (the need for Smalltalk programming in specifying the decision rules) it should be mentioned that the ability of specifying decision rules within a spreadsheet, (see section 5.4) partially alleviates the problem. Nevertheless, our experience showed that when large amounts of information are to be handled within the decision rules, Smalltalk represents a better environment in doing so. The above drawbacks and comments, point out to a number of future research and development directions, that we will discuss in the last chapter of this thesis.

7.8 Summary

In this chapter, we presented the development of the Electricity Markets Model, using the OO/DEVS framework within a business environment. We have initially focused on a set of eight steps, that during the model building with the DC modelling team, proved a structured and concise way for model development. We have then presented the background, scope and structure of the Electricity markets model, by summarising the characteristics of the pool and contract markets for buying and selling electricity in the newly privatized UK electricity market, as well as by presenting how these characteristics were modelled using the OO/DEVS framework as a modelling platform.

The model presented here has been based on the main entities of the system and the way they interact. In this model we have incorporated fairly elaborate decision rules, with the aim to encapsulate the main objectives of the players within the industry. In addition, to these decision rules, we have incorporated the ECAP [Vlahos (1989)] decision costing algorithm within the pool market entity in order to produce a realistic System Marginal Price on which contract costing is reflected both by the buyers and the generators of electricity. Even though, the model itself contains a significant amount of quantitative data, and information contained within the mental models of the DC modelling team, our experience showed that the OO/DEVS modelling paradigm coupled by a user friendly GUI (and linked to a spreadsheet environment), have provided a platform to model this information in a very structured and natural way, integrating at the same time the hard and soft modelling perspectives.

In section 7.6 of this chapter, we have presented one of the scenarios developed for model experimentation and focused on the nature of model output, as well as some interesting results regarding the nature of the electricity industry. The DC modelling team has developed a number of additional scenarios, the presentation of which is beyond the scope of this chapter. Our future expectation is that the generality and scope of the model will facilitate the development of a number of future scenarios, with minor modifications of the entities' instance variables and decision rules.

Finally, in the last section of this chapter we presented the views of the modelling team based on their modelling experience with OO/DEVS. It should be pointed out, that the experience gained by using OO/DEVS within a business environment confirmed our views presented in the final discussion in Chapter 6, regarding the research issues of structure, focus, time-representation and reusability. Overall, modelling within OO/DEVS showed that the concepts supported by the framework address the research questions set in Chapters 2 and 3.

Moreover, the development of a model with DC for over a period of four months has proved that OO/DEVS can be used as a decision support tool that can support effectively the development of natural, modular and reusable models, while it can accommodate 'hard' and 'soft' decision rules in a unified way. The fact that the DC management team started from a position of scepticism, and turned into advocates of the approach underlines this point. Nevertheless, a number of research and development issues remain open. These are the issues that we will address in the next chapter.

Chapter 8

Conclusions & Further Research Directions

Contents:

8.1 Thesis Summary

At the introduction of this thesis, we discussed the increased need for industry simulation, and the increased popularity of the approach for scenario development, management debate and learning, as well as strategy formulation. Having highlighted the modelling requirements for effective industry simulation, we set as the task of this research work the development of a modelling and simulation methodology, that can meet the challenges of industry modelling in the 90's and beyond.

Having identified System Dynamics as the principal methodology, used to date, for industry modelling and simulation, we presented (see Chapter 2) the main concepts on which System Dynamics is based by decomposing it into three distinct parts: (i) model building , (ii) core technology, and (iii) experimentation/learning. Using these three areas, as well as a classification scheme as a vehicle of comparison between industry and manufacturing simulation, we identified a number of weaknesses in relation to the second of the above parts, i.e. the System Dynamics core technology. More specifically, at the end of Chapter 2, we classified the issues related to our critique as:

- ▸ **Natural Model Representation**
- ▸ **Structure/Focus**
- ▸ **Reusability**
- ▸ **Time-Representation**.

These four groups of issues provided the basis for our research agenda set at the end of Chapter 2 and the introduction of Chapter 3. More specifically the research questions that we attempted to address in this thesis, evolved around:

- ▸ The ability and value of entity based modelling regarding industry simulation.
- ▸ The need for model modularity, reusability and extensibility.
- ▸ The ability to map a complex system through a number of relationships over and above the association one which is provided by System Dynamics (i.e. aggregation and generalization relationships).

- The question of whether or not the naturalness of discrete event time representation can be merged with the ability to 'view the system from above' provided by System Dynamics and its continuous time representation.

Having set our research agenda, in Chapter 3 we investigated the research question of entity modelling by exploring the views of software engineering, manufacturing simulation and cognitive psychology. The result of our survey was the discussion of the concepts related to Object Oriented Analysis, Design and Programming, as well as the expressive power that object representation adds to modelling. In attempting to partially address the question of relationship modelling, we discussed the concept of generalization relationship modelling, provided by Object Orientation and its possible value within the industry simulation problem domain.

In the second part of chapter 3, we investigated the question of discrete event modelling, and focused on DEVS as a discrete event formalism which can address in a concise way discrete event modelling while at the same time provide the ability to model aggregation/disaggregation relationships. In should be pointed out that the choice of DEVS is consistent with the research objective, set at the very beginning of this thesis, of providing a technology which is a super-set of System Dynamics.

At the end of Chapter 3, it became evident that the combined concepts supported by Object Orientation and DEVS would render the theoretical basis to address sufficiently the questions of our research agenda. Nevertheless, having felt that the theoretical ideas did not provide on their own a substantial answer to our research questions, we embarked on the development of a combined OO/DEVS framework and its software implementation. As a result, in Chapter 4 we presented the current DEVS implementations, found in the literature, as well as a new Smalltalk DEVS implementation. The Smalltalk implementation of DEVS, brought into light a number of research issues, regarding a pure object oriented DEVS implementation. In order to address these issues, we embarked on the development of the OO/DEVS framework, with the research objective to merge OO and DEVS. At the end of Chapter 4, we presented a small model building example within OO/DEVS, and developed a practical implementation of the modelling properties provided by OO/DEVS, thus demonstrating in practice a solution to the four groups of issues set out in our critique of System Dynamics in relation to industry simulation.

The actual software implementation underlined a new group of research issues related to graphical model specification. We attempted to address these issues in Chapter 5, where we discuss the development of a number of diagrams for model specification, as well as their implementation in a OO/DEVS GUI. The nature of the GUI, as well as a full demonstration of model building within OO/DEVS is attempted at the end Chapter 5, through the use of the 'Beer Game'.

Modelling the 'Beer Game' in OO/DEVS gave us a good indication on how well the OO/DEVS technology addresses the research questions evolved around the issues of entity representation, structure/focus, reusability and time-representation. Following this experience we embarked on a more realistic practical investigation by modelling the investment behaviour in the newly privatized UK Electricity Industry, and comparing the OO/DEVS model with a previously built System Dynamics one. Even though the two models were built purely from an academic point of view and in an academic environment, the comparison suggested that the attainment of a more functionally explicit structure adds value to the application of industry simulation.

Nevertheless, the practical issue of whether or not the functionality provided by OO/DEVS adds value to a model built for business purposes remained an open question. This question was addressed in Chapter 7, where we presented the development of a OO/DEVS model within a business environment, over a period of a year. A model of the purchasing and selling policies within UK Electricity Industry, was developed jointly with one of the UK Electricity Distribution Companies. The structure of the model as well as sample results are presented in some detail. Chapter 7 closes by presenting the evaluation of OO/DEVS by the company's management/modelling team. Their comments provided an interesting yardstick in evaluating the functionality of OO/DEVS from a user's point of view, as well as a practicioner's perspective on how well the framework and its implementation address the research questions set at the beginning of this thesis. From an initial stand point of scepticism, the team of this company, became advocates of the approach, developing their 'own' model and presenting it themselves to senior management.

8.2 Thesis Research Contribution

An account of the main contributions of this thesis to the establishment of the importance of state-of-the-art modelling concepts in industry simulation, is given below:

Methodological Contributions

- ▸ A review of System Dynamics in terms of its modelling view, simulation software, application areas and methodological developments, were carried out.
- ▸ A critique of the System Dynamics core technology, through a new classification scheme of simulation models, and a comparative study between industry and manufacturing simulation methodology, developments and research directions.
- ▸ A state-of-the-art review of Object Oriented modelling and the Discrete Event System Specification, which aimed at placing the System Dynamics modelling view under the perspective of the broader computer modelling issues.
- ▸ An innovative OO/DEVS modelling framework and Graphical User Interface, as well as their fully-functional Smalltalk software implementation. OO/DEVS supports the modelling capabilities identified as a result of the SD critique, by combining the expressive powers of Object Orientation and DEVS in a uniform modelling paradigm.
- ▸ Extensive model building and experimentation were carried out aimed at:
    - (i) Investigating and testing the modelling properties of OO/DEVS.
    - (ii) Establishing and validating the answers to the questions set in our research agenda, through a number of practical examples.

Overall, our research hypothesis was that the framework suggested and developed in this research work, based on a synthesis of Object Orientation and the Discrete Event System Specification, can address at a sufficient level the questions resulted from our System Dynamics critique (in Chapter 2), and facilitate the natural representation of organisations and their behaviour in a given market place. Our modelling experiments, ranging from simple industry models (i.e. the 'Beer Game') to a large-scale model of the Electricity Industry developed in a business environment, have verified the above hypothesis.

Modelling Contribution to the Electricity Industry

- The case studies presented in this thesis demonstrated how the developed approach to industry simulation, can be used to address the multi-dimensional policy issues that the UK Electricity Industry faces under the recent privatization.

- We have demonstrated how the integration of 'hard' and 'soft' decision rules with a range of detail and strategic views, can be used to model the industry in an effective way.

- In the Electricity Markets Model, in particular, we have modelled at a significant level of detail, the influences between the UK electricity pool and contract markets, as well as the resulting purchasing and selling behaviour of the industry players, with quite insightful results. In that respect, we have demonstrated that OO/DEVS has proved a effective platform for building decision support models, and that this research work can question the intuition of decision makers within the electricity industry.

8.3 Further Research Directions

As we have pointed out, this research work has focused primarily on the methodological area of industry modelling and simulation. The UK Electricity Industry has provided us with a number of case studies, which have been used as a vehicle to investigate and demonstrate the modelling characteristics of OO/DEVS. In what follows, we attempt to identify a number of further research issues opened up by this thesis, in relation to the OO/DEVS framework, as well as the electricity industry as a principal application area of the methodology.

Further Enhancements of the Methodological Framework

(a) The exploration of the compatibility between the System Dynamics way of thinking and modelling and the OO/DEVS world view, represents a very interesting research issue. This issue could be explored by adding to OO/DEVS the ability to model using standard System Dynamics - like equations. This underlines the view of OO/DEVS as a super set of the System Dynamics core technology. In that way, one could aim at a technology that can initially model a system in a System Dynamics level, but can also provide the ability to structure the model components further, in a more natural way. This is technically feasible, as DEVS is a super set of continuous type representations (see Chapter 3 section 10).

(b) The practical issue of providing alternative ways for decision rule specification, points to a number of research directions. Such a modelling ability, could enrich the standard modelling capabilities of the OO/DEVS implementation (i.e. Smalltalk code and spreadsheet decision rule modelling). The questions evolving around the development of natural-language like modelling constructs, present a very rich research agenda. Possible areas of research, regarding the above research direction, include the development of rule based specifications of the models using PROLOG logic rules, the incorporation of expert system shells for decision rule specification, or even the use of neural networks for decision rule modelling.

(c) The exploration of the methodological issues related to model-base building capabilities within OO/DEVS, with the practical aim of creating a model base of reusable model components. The current implementation of OO/DEVS allows storage and retrieval of models through a Smalltalk object filer. Even though, this is sufficient for the current use of the framework, the possibility of providing relational or even object oriented database facilities should be investigated. The capability of storing objects, during a simulation run, into a temporal database is one more interesting research issue.

(d) The development of a set of software tools for OO/DEVS model browsing and debugging. This is a challenging task that has to draw from research work into the design of high level debuggers for computer languages. The existence of such tools will speed up significantly model building, and facilitate the development of robust models.

Further Research in the Electricity Industry Application Area

(a) This research work aims at opening up new modelling possibilities, within the scope of the electricity industry, that were impossible under the 'old' optimisation view, or limited under the system Dynamics framework. The electricity markets model, constitutes an initial contribution towards this direction. The target of further work in this direction should be the construction of a set of reusable model components along with a set of models, that can assist the investigation of a series of issues that resulted by the recent privatization. We view this set of models as building blocks that can be reused within the industry for future analysis.

(b) Concerning policy issues in the industry, an interesting research area is the exploration of the structural characteristics of the post-privatization industry and the assessment of how this structure can influences competition, as well as the investigation of the results of various gaming tactics by the industry players. We believe, that the results of such modelling exercises could prove very useful yardsticks not only to the industry players, but to the industry regulator as well. For example, the possible Monopolies and Mergers Commission split of the generators, the new Pool trading rules, the 'optional' pool membership are some of the issues being considered, and could be easily modelled.

Microworld Design & Training Simulators

As we have pointed out in our review of System Dynamics, one of the most exciting areas of research within the discipline, has been the use of models as the basis for the development and use of 'microworlds', for management training and learning. This is an evolving research area where OO/DEVS can contribute, by providing 'microworlds' where the user can modify the structure of the model (i.e. by exploiting the ability to readily add/remove industry players and aggregate/disaggregate model entities) in addition to the various control variables. The technical and cognitive questions related to 'packaging' OO/DEVS models and tailoring their user interface for learning purposes, represent a stimulating research area which can benefit from similar research in System Dynamics.

Applications with Different Level of Modelling Detail

OO/DEVS has been devised and implemented having focused on the industry simulation problem domain. Even though the main case studies in this thesis are drawn from the electricity industry, the framework could be easily used to model a multiplicity of industries. In addition, the generality of the concepts supported by the framework and its implementation, should support the modelling of problems with focus at different levels of detail. The most readily available area seems to be organizational modelling, where the different organizational entities and their interactions are modelled and simulated. Another interesting area that OO/DEVS should be tested, is manufacturing simulation. Again the interactions and modelling of machines-agents in a factory floor, seem to be an area, the modelling of which OO/DEVS could facilitate.

List of References

References

ADA Reference Manual, U.S. Department of Defence, July 1980

Adelsberger, H H (1987); "Prolog as a simulation language", Proceedings of the 1987 Winter Simulation Conference.

Allen P M (1988); "Dynamics Models of Evolving Systems", System Dynamics Review, Vol 4, No 1-2

Altair, F B (1988); "Object-Oriented Database Systems", Proceedings of the 7th ACM SIGACT-SIGMOD-SIGART

Ansoff H I & D Slevin (1968); "An Appreciation of Industrial Dynamics", Management Science, Vol 14, No 7

Aracil Jaxier & Miguel Toro (1989); "Generic Qualitative Behaviour of Elementary System Dynamic Structures", the Proceedings of the 1989 System Dynamics International Conference, pp 239-345

Balmer D W (1987); "Modelling Styles and their Support in the CASM Environment", Proceedings of the 1987 Winter Simulation Conference

Balmer D W & R J Paul (1986); "CASM the Right Environment for Simulation", Journal of the Operational Research Society, Vol 37, pp 443-452

Balsi Osman (1990); "Guidelines for Successful Simulation Studies", Proceedings of the 1990 Winter Simulation Conference

Barclays de Zoette Wedd (1992); Electricity Research: National Power/ PowerGen

Basnet, C., S Karacal & T Beaumariage (1990); "Experiences in Developing an Object Oriented Modelling Environment for Manufacturing Systems", Proceedings of the 1990 Winter Simulation Conference

Beck, K & W Cunningham (1989); "A laboratory for teaching Object-Oriented Thinking", OOOPSLA 1989 Conference Proceedings, Vol 24, No 10

Birrel, N D & M A Ould (1985); A Practical Handbook for Software Development, Cambridge University Press

Blaha, M.R., W.J Premerlani & J.E. Rumbaugh (1988); "Relational Database Design Using an Object-Oriented Methodology", Communications of the ACM, Vol 31, No 4

Blanning, R.W. (1987); "An Object-Orientated Paradigm for Organizational Behaviour", Proceedings of Decision Support Systems

Booch, F G (1981); "Describing Software Design in ADA", SIGPLAN Notices, September 1981

Booch, F G (1986); "Object Orientated Development", IEEE Transactions in Software Engineering, February 1986 V.SE-12, No. 2 pp 211-221

Bunn, D.W., & Larsen, E.R. (May 1992a); "Sensitivity of Reserve Margin to Factors Influencing Investment Behaviour in the Electricity Market of England & Wales", Energy Policy, pp 490-502

Bunn, D.W., & Larsen, E.R. (1992b); "Modelling the Effects of Regulatory Scenarios on Investment Behaviour in the British Electricity Market", in the Proc. 23rd Pittsburgh Symposium on Modelling and Simulation, pp 1965-1972

Burns, J.R., & J. Darrell Morgeson (1988); "An Object-Oriented World View for Intelligent, Discrete, Next-Event Simulation", Management Science Vol. 34, No. 12

Camara A, P Antunes, M D Pinheiro & M J Fonseca de Seixas (1987); "Linguistic Dynamic Simulation - A New Approach", Simulation Vol 49, No 5

Camara A, P Antunes, F Ferreire & M J Fonseca de Seixas (1990); "Exploring "ideas": a Multidimensional Dynamic Simulation Approach", The proceedings of the 1990 System Dynamics International Conference, pp 181-190

Cassandras, C G & S G Stickland (1989); "Sample Path Properties of Timed Discrete Event Systems", Proceedings of the IEEE, Vol 77, No 1, January 1989

Coad, Peter & Edward Yourdon (1989); Object Oriented Analysis, Englewood Cliffs, NJ: Yourdon Press/Prentice-Hall

Concepcion, A.I., & Zeigler, B.P.(1988); "DEVS Formalism: A Framework for Hierarchical Model Development", IEEE Transactions on Software Engineering, Vol 14, No 2, February 1988

Cox, B (1986); Object Oriented Programming: An Evolutionary Approach, Addison-Wesley

Cox, B (1990); "There Is a Silver Bullet", BYTE October 1990

Coyle, R G (1977); Management System Dynamics, J Willey & Sons

Coyle, R G (1981); "The Dynamics of the Third World War", Journal of Operational Research Society, Vol 32, pp 755-765

Dahl O, B Myrhaug & K Nygaard (1984); Simula67 Common Base Language, Norwegian Computing Centre

Davidsen Pal, John D. Sterman & George P. Richardson (1990); "A Petroleum Life Cycle Model for the United States with Endogenous Technology", System Dynamics Review, Vol 6, No 1, pp 66-93

De Geus Arie (1988); "Planning as Learning", Harvard Business Review, Vol 66, No 2

Derrick, E J (1988); "Conceptual Frameworks for Discrete Event Simulation Modelling", M.S. Thesis, Department of Computer Science, Virginia Tech, Birginia

Digitalk Inc., Smalltalk/V for Windows (1991); Tutorial and Programming Handbook, Los Angeles, Digitalk Inc.

Duke R D (1981); "A Paradigm for Game Design", In C.S. Greenblat & R D Duke, Principles and Practices of Gaming Simulation, Sage

Edwards J M & Henderson-Sellers B (1993); "A Graphical Notation for Object-Oriented Analysis and Design", Journal of Object Oriented Programming, Vol 5, No 9, pp 53-73

Electricity Council (September 1985); Report on the Generation Security Standards

Financial Times (8th of June 1992); Eggar urges deal for coal Power.

Fishman, G S (1973); Concepts and Methods in Discrete Event Digital Simulation, John Willey & Sons, New York

Ford, Andrew & Bull, Michael (1989); "Using System Dynamics for Conservation Policy Analysis in the Pacific Northwest", Vol 5, No 1, pp 1-16

Ford, A & Yabroff, I W (1979); Technical documentation of the electric utility policy and planning analysis model, LA-7773-MS, Internal Report US Department of Energy.

Forrester, J W (1961); Industrial Dynamics, MIT Press.

Forrester, J W (1968); "Market Growth as Influenced by Capital Investment", MIR, Winter 1968.

Forrester, J W (1971); World Dynamics, Cambridge, MA, MIT Press

Forrester, J W (1968); Principles of Systems, Cambridge, MA, Productivity Press

Forrester, J W (1978); "Lessons from System Dynamics Modelling", System Dynamics Review, Vol 3, No 2

Forrester, J W (1988); "The next frontier: Understanding social systems", Working Paper System Dynamics Group, MIT, D-3963 SUNY.

Forrester, J W (1992); "Policies, Decisions and Information Sources for Modelling", in

Modelling for Learning, A Special Issue of The European Journal of Operational Research, Vol 59, No 1, pp 42-63

Forrester, J W, Graham, A , Senge, P , & Sterman J (1983); "An Integrated Approach to the Economic Long Wave", Working Paper, System Dynamics Group, MIT, D-3447-1

Forrester, J W, N J Mass, C J Ryan (1976); "The System Dynamics National Model: Understanding Socio-Economic Behaviour & Policy Alternatives", Technological Forecasting and Social Change, Vol 9, July 1976

Foschiani, S (1989); "Development and Use of System Dynamics Models as Tools for Strategic Planning of Flexible Assembly Systems", Proceedings of the 1989 System Dynamics Conference, pp 89-96

Fox M S & Smith S F (1984); "ISIS - A Knowledge Based System for Factory Scheduling, Expert Systems", Vol 1, No 1, July 1984

Frangini M (1991); "Visual-Based Applications gain wide acceptance", Computing Canada, Vol 17, No 3

Gibson, Elizabeth (1990); "Objects - Born and Bred", BYTE October 1990

Glynn, P W (1989); "A GSMP Formalism for Discrete Event Systems", Proceedings of the IEEE, Vol 77, No 1, January 1989

Goldberg, A & D Robson (1983); Smalltalk-80: The Language and its Implementation, Reading, MA: Addison-Wesley

Graham I (1993); "Migration Using SOMA: A Semantically Rich Method of Object Oriented Analysis", Journal of Object Oriented Programming, Vol 5, No 9, pp 31-43

Graham & P Senge (1990); "Computer-based Case Studies and Learning Laboratory Projects", System Dynamics Review, Vol 6, No 1, pp 100-105

Greenberger, M., M A Crenson & B L Crissey (1976); Models in the policy process, New York: Russel Sage Foundation

Hackman J R & Morris C G (1975); "Group Tasks, Group Interaction Process, and Group Performance Effectiveness, A Review and Proposed Integration". In L. Berkowitz, ed., Advances in Experimental Social Psychology, Vol 8, Academic Press

Henriksen, J O (1988); "One System, Several Perspectives, Many Models", Proceedings of the 1988 Winter Simulation Conference

Highland, H J (1977); "A Taxonomy Approach to Simulation Model Documentation", Proceedings of the 1977 Winter Simulation Conference

Holmes, Andrew (June 1990); Electricity in Europe: Power and Profit, Financial Times Business Information

Holmes, Andrew (1992); Privatizing British Electricity, Financial Times Management Reports

Homer J B (1992); "A System Dynamics model of the National Cocaine Prevalence", System Dynamics Review, Vol 9, No 1, pp 45-78

Hooper, J W & Reilly, K D (Feb 1982); "An Algorithmic Analysis of Simulation Strategies", International Journal of Computer and Information Sciences, Vol 112

James Capel & Co (February 1990); Reshaping the Electricity Supply industry in England & Wales

Jensen K (1987); "Coloured Petri Nets", in Petri Nets: Central Models and their Properties, ed G Rosenberg, Springler, pp 248-199

Jiuqiang Han, Sun Guoji & Wu Biao (1991); "System Dynamics Simulation Language - DYNAMOC", Proceedings of the 1991 System Dynamics Conference, pp 264-271

Kavrakoglou, I (1985); "Multicriteria Decisions in Power System Planning", in M Zeleny (ed) Multicriteria Decision Making, JAI Press, Connecticut, pp. 198-209

Kettenis, D.L. (1992); "COSMOS: A Simulation Language for Continuous, Discrete and Combined Models", Simulation Vol 58, No 1, pp.32-41

Kim, T G (1990); "The Role of Polymorphism in Class Evolution in the DEVS-Scheme Environment", Proceedings of the 1990 Winter Simulation Conference

Kim T.G. & Park S. B. (1992); "The DEVS Formalism: Hierarchical Modular Systems Specification in C++"; Proceedings of the 1992 European Simulation Multiconference

Kiviat, P J (1969); "Digital Computer Simulation: Computer Programming Languages", RM-5883-PR, The Rand Corporation, Santa Monica, California

Klainhaus, A M (1986); "BAMBOO - A Behavioural Analysis Expert System for System Dynamics Models", Proceedings of the 1986 System Dynamics International Conference

Klainhaus, A M (1989); "Knowledge-Based Modelling", in the Proceedings of the 1989 System Dynamics International Conference

Knapp, Verna (1987); "The Smalltalk Simulation Environment, Part II", Proceedings of the 1987 Winter Simulation Conference

Kreutzer, W (1986); System Simulation: Programming Styles and Languages, Addison-

Wesley, Reading, Massachusetts

Kyratzoglou I M (1988); "Computer Aided Petri Net Design for Decision-Making Organizations", Proceedings of the 1988 Winter Simulation Conference

Labov, W (1973); "The Boundaries of Words and their Meanings", in New Ways of Analysing Variation in English, ed C J N Bailey & R W Shuy, Vol 1, Washington DC: Georgetown University Press

Larsen, E R, Morecroft, J D W , Murphy J (1991); "Helping Management Teams to Model", 1991 International System Dynamics Conference, pp 223-240

Law A M (1987); "Simulation of Manufacturing Systems", Proceedings of the 1987 Winder Simulation Conference

Linvy, M (1987); "DELab - A Simulation Laboratory", Proceedings of the 1987 Winter Simulation Conference

Lomow, G & D Beazner (1990); "A Tutorial Introduction to Object Oriented Simulation & Sim++", Proceedings of the 1990 Winder Simulation Conference

Mandler J M (1984); Stories, Scripts, and Scenes: Aspects of Schema Theory, Lawrence Erlbaum Associates Publishers

Mashayeski, J B (1992); "Transition in the New York State Solid Waste System: A Dynamic Analysis", System Dynamics Review Vol 9, No 1, pp 223-240

Mathewson S C (1989); "Simulation Support Environments", in Computer Modelling for Discrete Simulation (Edited by M Pidd), John Willey & Sons Ltd

McIntyre, S C & Higgins, L F (1988); "Object Oriented Systems Analysis and Design: Methodology and Applications", Journal of Management Information Systems, Vol 5, PT 1, Summer 1988

McIntyre, S C & Higgins, L F (1989); "Embedding Stakeholder Analysis in Object Oriented Organizational Modelling", Proceedings of the 21st Hawaii International Conference in Systems Science, Jan 1989

Meadows D H (1972); The Limits to Growth, New York: Universe Books

Merten, P., Loffler, R., and Wiedmann, K.P. (1987); "Portfolio Simulation: a Tool to Support Strategic Management", System Dynamics Review Vol 3, No 2, pp 81-101

Meyer, B. (1988); Object Oriented Software Construction, Prentice Hall

MicroWorld Creator^{TM}; Microworlds Inc., 347 Broadway Street, Cambridge MA 02139.

Moore, J C & A B Whinston (1986); "A Model of Decision-Making with Sequential Information-Acquisition (Part 1)", Decision Support Systems, Vol 2, No 4

Morecroft, J D W (1982); "A Critical Review of Diagramming Tools for Conceptualising Feedback System Models", Dynamica, Vol 8, Summer 1982

Morecroft, D W (1984); "Strategy Support Models", Strategic Management Journal, Vol 5, pp 215-229

Morecroft, J D W (1988); "System Dynamics and Microworlds for Policy-Makers", European Journal of Operational Research, Vol 35

Morecroft J D W, D C Lane, P S Viita (1991); "Modelling Growth Strategy in a Biotechnology Startup Firm", System Dynamics Review, Vol 7, No 2, pp 93-116

Morecroft, J D W & van der Heijden K. (1992); "Modelling the Oil Producers: Capturing Oil Industry Knowledge in a Behavioural Simulation Model", in Modelling for Learning, A Special Issue of The European Journal of Operational Research, Vol 59, No 1, pp 102-122

Mosekilde E and E R Larsen (1988); "Deterministic Chaos in the Beer Production-Distribution Model", System Dynamics Review, Vol 4, Nos 1-2, pp 131-147

Mourant R R (1992); "An Interface for Hierarchical Modelling in Object Oriented Simulation", Computers and Industrial Engineering, Vol 23, Nos 1-4, pp. 233-235

Nail, Roger (1992); "A System Dynamics Model for Rational Energy Policy Planning", System Dynamics Review, Vol 8, No 1, pp 1-21

Nance R E (1981); "The Time and State Relationships in Simulation Modelling", Communications of the Association for Computer Machinery 24, pp 173-179

The National Grid Company (March 1992); Seven Year Statement For the Years 1992/3 to 1998/9.

Ninios, P, Vlahos, K & Bunn D.W. (1993); "Industry Simulation: Systems Thinking with an Object Oriented/DEVS Technology", forthcoming: European Journal of Operational Research

O'Keefe, R. (1986); "Simulation and Expert Systems - A Taxonomy and Some Examples", Simulation Vol 46, No 1, pp 10-16

OFFER (December 1991); Report on Pool Price Inquiry, Office of Electricity Regulation

Ören, T I & Zeigler, B P (1979); "Concepts for Advanced Simulation Methodologies", Simulation, Volume 32

Ozdemirel, N E et al. (1988); "A Preliminary Group Technology Classification Scheme for Manufacturing Simulation Models", Proceedings of the 1988 Winter Simulation Conference

Parnas, D L (1972); "On the Criteria To Be Used in Decomposing Systems into Modules", Communications of the ACM, December 1972, Vol 15, No 12

Paul, R (1989); "Visual Simulation: Seeing is Believing?", in the Impacts of Resent Computer Advances on Operational Research, Elsevier Science Publishing Co, Inc.

Paul, R (1991); "Recent Developments in Simulation Modelling", Journal of Operational Research Society; Vol 42, No 3

Paul, R (1991); "The Three-Phase Discrete Event Modelling Approach", CASM Report, Working Paper, London School of Economics

Paribas Capital Markets Group (1990); The U.K. Electricity Privatization

Peterson, S (1992); "Software for Model-Building and Simulation: An Ilustration of Design Philosophy", in Modelling for Learning, A Special Issue of the European Journal of Operational Research, Vol 59, No 1, pp 197-203

Pidd, M (1988); Computer Simulation in Management Science, J Willey & Sons

Pracht, W.E. (1990); "Model Visualization: Graphical Support for DSS Problem Structuring and Knowledge Organisation", Decision Support Systems, 6, pp.13-27

Professional Dynamo Plus (1986); Reference Manual, Pugh-Roberts Associates, Inc.

Radzicki, M J (1990); "Methodologia oeconomiae et systematis dynamics", System Dynamics Review , Vol 6, No 2, pp 123-147

Rahn, R J (1985); "Aggregation in System Dynamics", System Dynamics Review, Vol 1, No 1, pp 111-122

Resmussen D R & E Mosekilde (1988); "Bifurcation and Chaos in a Generic Management Model", European Journal of Operational Research, Vol 35, No 1, April 1988

Richardson G P, Vennix J A M, Andersen D F, Rohrbaugh J, Wallace W A (1989); "Eliciting Group Knowledge for Model-Building", Computer-Based Management of Complex Systems, editors P Milling & E Zahn, Springer-Verlag, Berlin

Richmond, B., Peterson, S. & Charyk, C. (1990), *ithink^{TM} Documentation*, High Performance Systems, Hanover, NH

Roberts, E B (October 1974); "A Simple Model of R&D Project Dynamics", R&D Management, No 1

Roberts, S, & Heim, J (1988); "A Perspective on Object Oriented Simulation", Proceedings of the 1988 Winter Simulation Conference

Rosenblit, J., Hu J., Kim, T.G., Zeigler, B. (1990); "Knowledge Based Design and Simulation Environment (KBDSE): Foundation Concepts and Implementation", Journal of Operational Research Society, Vol 41, No. 6, pp.475-489

Sanders W H & J F Meyer (1989); "Reduced Base Model Construction Methods for Stochastic Activity Networks", Proceedings of the 3rd International Workshop on Perti Nets and Performance Models

Sebastian H J (1990); "Knowledge Based Discrete Control Problems: A Decision Support Approach", Decision Support Systems, Vol 6, No 4

Senge, P (1990); The Fifth Discipline: The art and Practice of the Learning Organization, Doubleday/Currency

Senge, P & J D Sterman (1992); "Systems Thinking and Organizational Learning: Acting Locally and Thinking Globally in the Organization of the future", in Modelling for Learning, A Special Issue of The European Journal of Operational Research, Vol 59, No 1, pp 137-150

Seymour J (1991); "Graphical User Interfaces give PC Users a Winning Hand", Today's Office, Vol. 26, No. 1

Shtub, A (January 1992); "Evaluation of Two Schedule Control Techniques for the Development and Implementation of New Technologies: a Simulation Study", R&D Management, Vol 22, No 1

Simon, H (1969); The Sciences of the Artificial, MIT Press

Smith E E & D Medin (1981); Categories and Concepts, Harvard University Press, Cambridge, Massachusetts 1981

Smith New Court (August 1992); The 12 RECs (Alistair Buchanan), Electricity Research Report

Smith New Court (December 1992); National Power and PowerGen (Alistair Buchanan), Electricity Research Report

Sterman, J D (December 1987); "Testing Behavioural Models by Direct Experiment", Management Science, Vol 33, No 12

Sterman, J D (1985); "A Behavioural Model of the Economic Long Wave", Journal of Economic Behaviour and Organization, Vol 6

Sterman, J D (1989); "Modelling Managerial Behaviour: Misconceptions of feedback in a Dynamic Decision Making Environment", Management Science, Vol 35, No 3

Stone, C M & D Hentchel (1990); Database Wars Revisited, BYTE October 1990

Stroustrup, B (1986); The C++ Programming Language, Reading, MA: Addison-Wesley

Thomasma, T & Ulgen O.M.(1988); "Hierarchical, Modular Simulation Modelling in Icon-based Simulation Program Generators for Manufacturing", Proceedings of the 1988 Winter Simulation Conference

Toro M & J Aracil (1988); "Qualitative Analysis of System Dynamics Ecological Models", System Dynamics Review, Vol 4, No 1-2, pp 56-80

Toyoda, Y & S Mawatari (1991); "Mathematical Formulation of System Principles in System Dynamics", in the Proceedings of the 1991 System Dynamics International Conference, pp 608-617

Tu, Yi-Ming & N Shiao (1992); "Integration of System Dynamics and Rule Based Reasoning Mechanism", the Proceedings of the 1992 System Dynamics International Conference, pp 715-724

Ulgen, O., T Thomasma & Y Mao (1989); "Object Oriented Toolkits for Simulation Program Generators", Proceedings of the 1989 Winter Simulation Conference

Van Hee, K. M., L J Somers & M Voorhoeve (1991); "A Modelling Environment for Decision Support Systems", Decision Support Systems Vol 7, pp 241-251

Vennix, J A M, J W Gubbels, D Post, H J Poppen (1990); "A Structured Approach to Knowledge Elicitation in Conceptual Model Building", System Dynamics Review, Vol 6, No 2, pp 194-208

Vlahos, K (1991); Capacity Planning in the Electricity Supply Industry, PhD Thesis, London Business School.

Vujosevic, R (1990); "Object Oriented Visual Interactive Simulation", Proceedings of the 1990 Winter Simulation Conference

Walter, John & Carol Lopitato (1992); "An Artificial Intelligence Approach to Socio-Economic Simulation: Application to the Economic Integration of Europe", Proceedings of the 1992 System Dynamics International Conference

Wegner, Peter (1989); Learning the Language, BYTE, March 1989

Wirfs-Brock, R & Wilkerson, B (1989); "Object Oriented Design: A Responsibility Driven Approach", OOPSLA '89 Proceedings

Widmeyer, E R (1988); "Logic Modelling with Partially Ordered Preferences", Decision Support Systems, Vol 4, No 1

Wolstenholme, E F (1990); System Enquiry, Wiley & Sons

Wolstenholme, E F (1982); "System Dynamics in Perspective", Journal of Operational Research Society, Vol 33

Wolstenholme, E F & Al-Alusi A (1987); "System Dynamics and Heuristic Optimisation in Defence Analysis", System Dynamics Review, Vol 3

Yourdon, Edward (1990); "Auld Lang Syne: Is it time for you to ring out the old and ring in the new?", BYTE, October 1990

Zeigler, B.P. (1976); Theory of Modelling and Simulation, Willey, New York

Zeigler, B.P. (1980); "Concepts and Software for Advanced Simulation Methodologies", in Simulation with Discrete Models: A state-of-the-Art View, editors: T I Oren, C M Shub, P F Roth, IEEE

Zeigler, B.P. (1984); Multifaceted Modelling and Discrete Event Simulation, Academic Press

Zeigler, B.P. (1987); "Hierarchical, Modular Discrete-Event Modelling in an Object-Oriented Environment", Simulation Vol 49, No 5, pp 219-230

Zeigler, B.P. (1989); "DEVS Representation of Dynamical Systems: Event-Based Intelligent Control", Proceedings of the IEEE, Vol. 77, No. 1

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

Zhang W & Mourant R (1990); "Simulation and Knowledge Based Objects", Proceedings of the 12th Annual Conference on Computers & Industrial Engineering, Vol 19, Nos 1-4, pp 97-101