Inheritance in Systems Comprising Reactive Components – A Behaviour Perspective

A.S. El-Hassan

Submitted for the Degree of Doctor of Philosophy from the University of Surrey

Department of Computing, School of Electronic Engineering, Information Technology & Mathematics, University of Surrey, Guildford, Surrey GU2 5XH, UK

September 2000

© A.S. El-Hassan 2000
ABSTRACT

The aggregation hierarchy is one of the most significant data abstraction mechanisms that emerged as a result to semantic extensions to traditional systems analysis and design methods. The way inheritance works in this hierarchy is studied in this thesis. Special emphasis is placed on the behaviour of objects which are related via an aggregation hierarchy.

A framework is introduced for capturing the behaviour of a system from the respective behaviour(s) of its components. This framework is based on a 3-level behaviour modelling hierarchy. One of the most significant contributions of this framework is the ability to apply inter-object interactions when building a behaviour model of a system. These interactions are significant in that they can yield totally distinct models of the systems functionality.

Some of the notions that are supported by the behaviour modelling framework include unreachable and transient states, transition chains (cascades) and concurrency. The framework also enables the creation of behaviour model (semantic) hierarchies, wherein certain facets of the systems behaviour or functionality can be hidden (abstracted out) in a gradual fashion that suits the requirements of the problem domain. This creates what is effectively, distinct views of the behaviour or functionality of the system.

The notions and concepts that are introduced here are verified and presented in a comprehensive case study that shows what can be achieved using these ideas. Suggestions are also made for future work which can help overcome some of the limitations introduced throughout this research.
Acknowledgements

I would like to thank all of the people who helped me throughout my period of study at the University of Surrey. In particular I wish to thank my supervisor, David Bowers, for his guidance, patience and encouragement, throughout what was, on many occasions, a steep hill to climb. Specifically, I must thank David for shifting his schedule to suit mine on several occasions in the past 2 years, for example, 18:30 meetings.

Thanks also to Mike Shields for all the help and advice he provided in the thesis and in the revisions to the thesis that took place after the exam, I am very grateful for all your help.

Many thanks to Khurshid Ahmed who has encouraged me throughout my studies, and for providing some good ideas regarding the shape of the thesis.

I also wish to thank the following people from the Department of Computing, who have helped me in the past few years: Brent Work, Anne Balmforth, David Pitt, Terry Hinton, Heather Fulford, Gary Dear and Helen Tye, thank you all very much.

It is not easy doing a PhD and a Full-Time job at the same time. I therefore wish to thank Mike Nicholls from Pulse Software for providing the flexitime to enable me to attend PhD meetings with my supervisor and to prepare my thesis for submission.

Finally, many thanks to my friends and family, who have been very patient, understanding and supportive, thank you all very much.
## CONTENTS

Chapter 1  Introduction  
1.1 Introduction  
1.2 Scope and Objectives  
  1.2.1 Scope  
  1.2.2 Objectives  
1.3 How The Thesis is Organised  
  1.3.1 Chapter Two: Literature Review & Problem Identification  
  1.3.2 Chapter Three: Underpinning Technology  
  1.3.3 Chapter Four: Proposed Approach  
  1.3.4 Chapter Five: Case Study  
  1.3.5 Chapter Six: Conclusion and Further Work  
  1.3.6 Appendices  
  1.3.7 Thesis Bibliography  

Chapter 2  Literature Review & Problem Identification  
2.1 Introduction  
2.2 The Transition To Object Systems - The Role of Semantic Data Models  
2.3 Object Oriented Design Techniques - Overview  
2.4 Object Based Design Methods – A Closer View  
  2.4.1 Multiple Options for Modellers  
  2.4.2 Features Common to All Methods  
  2.4.3 Weaknesses in Early OO Attempts  
  2.4.4 Weaknesses in OO Methods  
  2.4.5 Weaknesses Relating to Behaviour Specification  
2.5 Problem Identification  
  2.5.1 The Effect of Abstraction Hierarchies  
  2.5.2 Inheritance of Behaviour  
  2.5.3 The Role of Composite Objects  
2.6 Summary
Chapter 3  Underpinning Technology  

3.1 Introduction  
3.2 More on Problem Identification  
  3.2.1 The Role of Encapsulation  
  3.2.2 Composite Objects  
  3.2.3 Example  
3.3 Describing Behaviour  
  3.3.1 Visual Formalism  
  3.3.2 Hierarchic Notation & Statespace Reduction  
  3.3.3 Inter-Component Communications  
  3.3.4 Concurrency and Multiple Components  
  3.3.5 Hierarchical Coloured Petri Nets (HCP Nets)  
3.4 Evaluation  
  3.4.1 Deficiencies  
  3.4.2 UML Revisited  
3.5 Proposed Approach - Preview  

Chapter 4  Proposed Approach  

4.1 Introduction  
4.2 Chapter Map  
4.3 Definition of Terms and Notation  
4.4 Proposed Approach - Summary  
4.5 Background  
4.6 Proposed Approach - Three Levels of Abstraction  
  4.6.1 The Pi Level of Abstraction - Π  
  4.6.2 The Gamma Level of Abstraction - Γ  
  4.6.3 The Phi Level of Abstraction - Φ  
  4.6.4 Behaviour Modelling Steps  
4.7 Example - Building Society Shares  
  4.7.1 System Specification - Π  
  4.7.2 The Γ Level of Abstraction  
  4.7.3 Φ Level of Abstraction  
4.8 Variations and Exceptions  
  4.8.1 Theme Variations  
  4.8.2 Limitations When Building Complex Γ Using a Binary Approach - Why Two Γ Models  
4.9 Summary
# Chapter 5 Case Study

5.1 Introduction 100

5.2 Notation and System Description 101

5.2.1 The Structure of our System 102
5.2.2 The Objects in the System - Components 102
5.2.3 System Interaction Constraints 108

5.3 Building The Combined Behaviour Model (\(\Gamma\)) of the System 111

5.3.1 Generating Behaviour Model as a Combination of ‘Binary’ Behaviour Models From Component Groups - \(\Gamma_a\) 112
5.3.2 Modelling Systems Behaviour Using Alternative Binary Models 126
5.3.3 Abstracting Behaviour Model – The \(\Phi\) Behaviour Model 134
5.3.4 Abstracted \(\Gamma\) (\(\Gamma''\)) 137
5.3.5 Comparison of \(\Phi(\Gamma)\) and \(\Gamma''\) 144
5.3.6 State Elimination in Abstracted/Combined Superstates -- Where the Approach Went Wrong 145

5.4 Synopsis 147

# Chapter 6 Conclusions & Future Directions 149

6.1 Introduction and Chapter Map 150

6.2 Contributions & Exceptions 150

6.2.1 Advantage Over Other Approaches 152
6.2.2 Exceptions 153
6.2.3 Other Restrictions 154

6.3 Applicability 156

6.3.1 Distributed Objects Systems 156
6.3.2 Object Oriented Programming Languages 157
6.3.3 Other Benefits 159

6.4 Further Work 160

Appendix A Steps for Generating \(\Gamma\) from \(\Pi\) 160
Appendix B Why \(\Pi\) to \(\Gamma\) Transformation is 2 Stages 160
Appendix C Notation 160
Appendix D Concepts, Definitions & Restrictions 160
Appendix E Rules For \(\Pi\) to \(\Gamma\) Transformation 160
Appendix F Proof of Theorem C1 in Chapter 4 160
Appendix G Figures & Tables for Chapter 5 160

Bibliography
CHAPTER 1

INTRODUCTION
1.1 Introduction

All information systems are based on models, which are used by system developers and users to understand the way systems work, and to formulate new ideas and explore potential solutions. According to [Howard 99], “all computerised systems in use today are based on someone’s models of reality, they are representations of sets of rules as to how a system is perceived as operating. Such representations are nearly always formulated in terms of models which are incomplete for one reason or another”.

From the time when information systems were a series of switches that were toggled on and off on the front panels of large and bulky computers, there have been constant and inexorable efforts to improve the ease of use and efficiency of information systems, including their underlying data models. This search has been likened to a search for a “silver bullet” solution [Hallahan 99].

In this thesis we shall look at system modelling methods with special emphasis on modelling complex systems that relate to real world scenarios where a system is made up of multiple other objects, each of which having distinct functionality.

1.2 Scope and Objectives

1.2.1 Scope

The past 25 years have witnessed the emergence of modelling approaches that include the Data Flow Diagrams, Entity Relationship models, State Transition Diagrams and Object-Oriented (OO) models. These approaches were attempts to improve on prevailing techniques in order to help the information systems designers cope with ever increasing demands for faster systems that can handle
larger amounts of more complex data at a fraction of the cost both materially and in terms of effort.

More and more modelling approaches appeared on the premise that they could provide more scope to capture accurately the semantics (meaning) of the real world domain that relates to the system. In this thesis, we focus on one of the modelling paradigms that emerged in the past decade, namely, the object-oriented (OO) approach to designing information systems.

The origins of OO started in 1977 when [Smith & Smith 77] proposed significant semantic enhancements to the capabilities of prevailing models. The term Object emerged from the shift in focus in the new system design method from the traditional functionality of the system to the constituent entities (objects) of that system. The functionality of the system (or what it does) began to be distributed amongst its objects or (what it is made up of). The advantages and enhancements that OO methodologies offered over traditional systems are beyond the scope of this section of the thesis.

Semantic extensions to traditional systems proposed that systems should reflect more accurately their real world counterparts. Things were organised in hierarchical fashion, they tended to be made up of other things or components, or specialised versions of others, and they could also be grouped together if they, for example, shared a common purpose.

In addition, these extensions proposed that the functionality of an object in a system should be encapsulated within the definition of the object. Further, these extensions suggested that objects pass (inherit) structure and behaviour across the object hierarchies, which later became known as abstraction hierarchies.
1.2.2 Objectives

The objective of this thesis is to study the concepts brought about by semantic approaches to systems analysis and design. In particular the thesis aims to address the following notions:

- Identify an interesting problem area – Inheritance of Behaviour in Aggregation Abstractions.
- Show the difficulties that designers face when tackling this problem.
- Identify specific areas where lack of coverage leads to insufficient problem understanding and, hence, inadequate solutions.
- It may be necessary to restrict the target domain of the research in order to achieve a certain level of understanding about the problem within reasonable time limits.
- Introduce a framework for solving the problem, or the aspect of the problem that is the focus of the research.
- Verify any new notions, assumptions or theories.
- Demonstrate workability with examples.
- Suggest future directions.

Aggregation and generalisation will have a major role to play in any future systems design methodologies. Abstraction is a powerful tool for creating new objects from existing ones, thus enabling planners and designers to concentrate the decision making part of the organisation on what matters most.

This thesis argues that aggregation is probably the most significant feature of OO. The reason for this is simple: No system is ever made up of a single object, aggregation inherently exists as a feature of every system.

Another significant contribution of semantic and OO methodologies that this thesis will look at is the notion of inheritance. The ability to pass data (and functionality) up and down data abstraction hierarchies is a major contributor to systems that are much more reflective of the real world scenarios they relate to.
This phenomenon results in more accurate systems, less bulky, and overall, a lot easier to maintain, migrate and support.

**Behaviour** plays a major role in systems analysis through the notion of **encapsulation**. This means bringing data to life, instead of an outsider manipulating data by calling functions that act on it. One creates objects that take on lives of their own, setting each other’s properties, calling each other’s methods, sending messages to each other and generally interacting in ways that generally reflect the way things interact in real life.

Together with behaviour encapsulation, inheritance can lead to many possibilities in the field of systems design. For example, systems interfaces can delegate required functionality to the particular component in the system that is best suited to the requirement.

Although all the points above are advantages that can improve the power of new systems, there are nonetheless a few issues that need addressing: What is the best technique for capturing the idea of aggregated objects that are made up of other ones? What is behaviour and how to represent it? Inheritance, as a concept, can be quite difficult to understand and represent within a model of a system; how is it possible to represent this in an analysis model?

What this thesis does not introduce is yet another methodology for the design of information models or systems. The ideas that are introduced here are presented as an add-on or semantic enhancement to enrich further current established methodologies, whether these are OMT [Rumbaugh et al 91], Booch [Booch 91], UML, or a combination of old and new such as is the case with object-relational databases [Hallahan 99].
1.3 How The Thesis is Organised

This thesis focuses on inheritance and encapsulation and the role they play in providing functionality or behaviour from one level of abstraction in OO systems to another. The remainder of this thesis is organised as follows:

- Chapter 2: Literature Review and Problem Identification
- Chapter 3: Underpinning Technology
- Chapter 4: Proposed Approach
- Chapter 5: Case Study
- Chapter 6: Conclusion and Further Work
- Appendix: Steps and Heuristics, Special Considerations
- Bibliography

The contents of each of these parts is briefly described in the following section.

1.3.1 Chapter Two: Literature Review & Problem Identification

In this chapter, we present some of the models and methodologies that featured prominently in the run up to the advent of OO systems. These include: Semantic Data Models, OMT, Petri Nets and others. In the literature review chapter, we look at the main features of each methodology or approach, in addition to the contribution each one makes, from the perspective of behaviour modelling and systems that are made up of multiple complex components.

1.3.2 Chapter Three: Underpinning Technology

This chapter looks at some dynamic modelling facets that will be utilised in this thesis, and checks how dynamic modelling technologies provide these facets. These technologies or methods include Petri Nets, State Charts, and State
Transition Diagrams. The purpose here is to establish a set of technical ideas to be utilised for addressing the main focus of this thesis, i.e. behaviour representation. The principal useful constructs contributed by each method are noted, and any gaps in these methods are highlighted. Those useful constructs are then built upon in the proposed approach chapter to reach a set of ideas which will better address the issue at hand.

Chapters two and three serve to establish the need for further research in one or two area of reactive systems design. This is done not by suggesting a completely new object design technique, but through the semantic extension and enhancement to some of the ideas found in the literature on the subject of OO systems design.

1.3.3 Chapter Four: Proposed Approach

Having identified in the literature review chapter the salient features of OO that merit further research, this chapter introduces the proposed solution. As mentioned earlier, this thesis does not introduce a complete new methodology for designing information systems, many theses are needed for such a task. This chapter focuses on the extra semantic input that we propose to add to current methodologies in order to enhance their accuracy and correctness in representing what is a complex and intricate real world.

This semantic enhancement is introduced within the context of hierarchies of abstractions that are the trademark of OO methodologies. Our contribution is to the aggregation hierarchy in particular, wherein data as well as behaviour are inherited across the abstraction hierarchies. We introduce some new semantics in the presentation of object behaviour; further, we introduce a framework for capturing the inheritance of behaviour from one level in an aggregation hierarchy to the other. Some visual formal verification of the proposed solutions is introduced in this chapter.
1.3.4 Chapter Five: Case Study

This chapter demonstrates the workability of the proposed methodology. Whereas most of the ideas in this thesis are presented within the framework of specially simplified examples, this chapter presents a more realistic and complete demonstration. An illustrated step by step guide to the methodology is provided here, where each step shows the difficulties that need to be dealt with in the derivation of a good and semantically rich behaviour model.

In effect this section serves as a dual purpose demonstration. Firstly, it shows that the approach proposed in the previous chapter can in fact be made to work. Secondly, it exposes some of the gaps in this proposed approach, hence acts as an introduction to the following chapter on further work.

1.3.5 Chapter Six: Conclusion and Further Work

No methodology for designing systems relating to the real world can ever be 100% complete or foolproof. The only perfect representation of real world intricacies is the real world itself. Continuous enhancements to the most popular systems design approaches serve as an illustration of this fact and the approach proposed in this thesis is no exception.

In this chapter, we look at some of the problem areas of the material in the preceding chapters. Further we look at the proposed approach itself. Given that the general area of application of the proposed solution is in the aggregation hierarchies, and given that there are at least three other abstraction mechanisms in OO methodologies, there is obviously a lot more work that needs to be carried out.

The extra work that is proposed in this chapter can be categorised into three main types. First, there is work that is needed to address the gaps in the proposed approach to inheritance of behaviour in aggregation. Secondly, there are other areas of abstraction, such as association and generalisation. Finally, there are
other areas relating to the methodology, such as a Visual Basic program which acts as an example for demonstrating the applicability of the proposed methodology to current information systems design technologies databases and programming languages.

1.3.6 Appendix: Method Step by Step, Definitions, Theorem Proof, Diagrams for Chapter 5

This part of the thesis lists the steps that are proposed for designing a behaviour model of a complex reactive system. Appendices also show other information relating to specific sections, such as exceptional conditions, as will be seen in chapters 4 and 5.

1.3.7 Thesis Bibliography

This section presents various reading material on the background of this thesis, covering a contrast of areas from OO methods to formal methods, Petri Nets, Statecharts, Active Component Modelling/Passive Component Modelling
CHAPTER 2

LITERATURE REVIEW & PROBLEM IDENTIFICATION
2.1 Introduction

This thesis addresses behaviour modelling in OO systems, focusing on the concepts of aggregation and inheritance. The problem of modelling behaviour and inheritance is highlighted in the context of weaknesses in object analysis and design techniques.

The OO methodologies have their roots in semantic models which captured a great deal of detail but were lacking in certain areas such as describing the behaviour of systems and expressing inheritance.

2.2 The Transition To Object Systems - The Role of Semantic Data Models

In the mid-70’s, researchers in systems design attempted to simplify the design and use of systems by providing modelling constructs that support the user’s view of data [Chen 76], [Schmidt & Swenson 75], [Smith & Smith 77]. Two ideas formed the main thrust of these projects:

- Data independence, which implies freeing the user from all consequences of physical implementations, and

- Greater semantic expressiveness. [Smith & Smith 77] introduced to data modelling abstractions that were primarily used by psychologists and Artificial Intelligence researchers. These were aggregation and generalisation. In addition to generalisation and aggregation abstractions, many semantic data models support classification and association [Brodie 84], [Hammer 81]. The aggregation abstraction
underpins the subject of much discussion in the following chapters of this thesis.

There are many more contributions made in the field of systems analysis and design by semantic data models; however, this work is focused on abstraction hierarchies.

2.3 Object Oriented Design Techniques - Overview

In this section we look at popular OO design and analysis techniques. Our purpose is to highlight some inadequacies inherent in these techniques, and thus to define our area of research.

Object-Oriented (OO) techniques for software design [Loomis 87] have been presented as the answer to several problems including [Bouzeghoub 97]:

- accurate representation of real world entities,

- code-re-use,

- rapid development

- scope to cater for complexities such as those associated with parallel and distributed systems.

According to [Halbert 87], these design techniques can improve software development by enhancing maintainability, extensibility, and reusability. However, mastering OO fundamental concepts - types, hierarchies and inheritance - is essential to realising the benefits of this design and programming style.

The concepts of reuse [Booch 91], [Booch 97], [McGregor 92], inheritance [Lalonde 85], encapsulation [Snyder 86] and class generalisation [Kradie 86], [Bouzeghoub 97] have a great deal to offer when dealing with the complexities normally associated with the representation of real world systems in information models.
These concepts allow the creation of a high level class with generic skeleton functionality which can in turn be used repeatedly by several subclasses at lower levels of abstraction. Inheritance lets the designer specify important relationships between types, and supports the concept of re-use, which is applied to structure as well as behaviour. [Eriksson 98], [Coleman 94], [Rumbaugh et al 91], [Booch 91], [Coad 90], [Shlaer 88] discuss the advantages of OO in some detail.

New methods are certain to emerge both as a result of further research, or the alliances and associations between current methods; many of these will inevitably be overlooked in this chapter. In the following sections, we shall look only at the general concepts rather than the details of the OO design methods; the reader is directed to the respective *white papers* and manuals of each method for a full description of the features and facets they offer.

### 2.4 Object Based Design Methods – A Closer View

The chapter is concerned with some of the OO methods in general. However, the guiding methods for us in this section will include Booch's Object-Oriented Design with Applications **OODA** [Booch 91]. Object Oriented Modelling and Design **OMT** [Rumbaugh 91], Coad and Yourdon’s Object Oriented Analysis **OOA** [Coad 90], and the **Fusion Method** [Coleman 94]. In addition, we shall also look briefly at the **UML** method for systems design.

UML became more and more popular as the research undertaken for this thesis was drawing to a close. Hence it is treated here less extensively than other methods. However, the contributions of the UML method [Booch 97] to the specific area of behaviour modelling and inheritance will be looked at in more detail in chapter three.
2.4.1 Multiple Options for Modellers

The large numbers of methodologies that are available to systems designers make it difficult to choose the right technique for the problem of designing complex systems, [Khoshafian 95] touched upon this issue:

"... to say that there are too many object-oriented analysis and design methodologies is an understatement. Studying, summarising, and comparing all the current methodologies is a daunting task; choosing a single method and adopting it for the purpose of constructing a new system for an organisation is equally difficult".

It is, therefore, probably true that a more appropriate approach would be to apply selectively aspects of each method that best match the organisations' requirements and as such, to tailor-make an appropriate object-oriented system.

2.4.2 Features Common to All Methods

A quick browse through the methods mentioned above (within their respective white papers/books; a detailed survey of OO methods is beyond the scope of this thesis) reveals a few common features and contributions that constitute common ground on which they all agree. These features include:

- Strong Role for the Object Model
- Class Hierarchies and Abstraction
- Static Model & Behaviour Model
- Behaviour and Polymorphism
- Encapsulation - Hidden (built-in) Behaviour
• Inheritance - Structure and Behaviour

• Objects Have Attributes, Identity and are Persistent

Our work acknowledges all of the above concepts as significant OO building blocks. However, what we are concerned with is a particular set of weaknesses and deficiencies in these OO techniques, those are the ones relating to behaviour specification and inheritance.

2.4.3 Weaknesses in Early OO Attempts

With all the innovations that OO methods bring, there are inevitably a few areas where they may be considered to be lacking. In this section we consider some of these areas which merit further work.

• General Inadequacies

According to [Halbert 87] “…using these (OO) concepts correctly can be a hard skill to acquire, and guidelines are required to aid OO software designers”. [Banerjee et al 87] highlights at least two problems in OO systems design; “…the lack of consensus about the object oriented model; different OO systems support different notions of objects. The other is that most existing OO systems are programming language systems. As such their data models completely ignore many important database issues, such as deletion of persistent objects, dynamic changes to the database schema, and predicate-based query capabilities…”

With regards to general OO technology, [Kim 90]; [Zdonik 90], [Maier 90]; [Bloom 89], [Maier 91] have exposed a few inadequacies, these include:

• Lack of formal foundation
• General disagreement in interpreting OO concepts
• Lack of a declarative query language
• Use of a navigational as opposed to a declarative interface
• Inheritance conflicts in class hierarchies, etc.

Some of these inadequacies cannot be solved by formulating a new model; for instance, the lack of a consensus on OO concepts requires the agreement of standards committees before the development of common standards. Other inadequacies such as lack of a standard declarative query language, multi-lingual support, etc., are the subject of much current research and widely accepted solutions may not feature for some years to come.

The lack of a reasoned approach to resolve inheritance conflicts in class hierarchies is one of the fundamental issues of OO. Here we can find property name conflicts between classes and their sub/super-classes, which are usually solved by the property name in the subclass takes precedence over property names in the superclass with the same name.

• **Dynamic Aspects are Difficult to Model**

With respect to modern OO methods, problems with the technique become even more complex as focus shifts from static to dynamic properties of systems. This facet of systems is something which constitutes one of the main contributions of the almost all methods above, [Ling 93] state that “the specification of systems behaviour in general constitutes a difficult issue”.

When considering dynamic aspects of OO systems in particular, the crux of the problem appears to be the difficulty of describing reactive behaviour in ways that are clear and tangible, and yet at the same time are formal and rigorous. Such solutions need to preserve and capitalise on concepts such as encapsulation and reactivity [Wegner 92].

On this [Tagg & Liew 93] noted “... current methodologies are nearly all weak in this area (dynamic modelling). There is a tendency to suppose that one can put in
a few Methods for each Object Class and the job is done. The problem seems to be that in the rush to encapsulate, inter-object dynamics are forgotten. There are plenty of Object Life Histories or State Transitions, but only for one Object Class at a time. Hence the weakest subarea is that of modelling concurrency and synchronisation where multiple objects are involved. The most useful behaviour model diagrams seem to be Object Communications Diagram (Booch “Object Diagram”), State Transition Diagrams (for each important, active Object Class), and Event Trace Diagrams (for each key scenario, with multiple Objects)

Furthermore, in many OO systems, it is found that dynamics are scattered - since operations must belong to specific objects, they are necessarily at a low level of detail and they rarely correspond to the tasks that the users perform in their work. This makes it difficult to get an overview of the system dynamics because:

- The sequence of operations execution will not be visible, or if made visible, they are usually unwieldy because the operations, in the first place, are only low level.

- Higher level tasks involving operations from several objects will not be visualised and can be established only by detailed reading of information from several parts of the model.

There are some OO methods which have put focus on dynamics, namely use cases [Jacobson 92] and scenarios [Kenneth 92]. These put the attention initially on dynamic rather than static aspects, investigating system behaviour before classes are identified. However, the view of dynamics is rather partial and high level, focusing on events, states, and transitions, not on aggregate dynamic concepts such as processes [Hoydalsvik 93]; there seems to be no provision made for the concept of unexpected behaviour which we will discuss later in this thesis.

Understanding the structure and internal relationships of large class libraries, frameworks or applications is essential for fulfilling the OO promise of code reuse. Furthermore, discerning global and local patterns of class interactions is
critical for maintenance and further development of software applications. Although OO techniques let software designers work at higher levels of abstraction than traditional techniques, the task of generating and maintaining large systems remains a difficult one. According to [Pauw 93], “this is caused by:

- The dichotomy between the code structure as hierarchies of classes and the execution structure as networks of objects.

- The atomisation of functionality - small chunks of functionality dispersed across multiple classes.

- The sheer number of classes, and the complexity of relationships in applications and frameworks.

While procedural language tools are often inappropriate for Object-Oriented programs (they are at an inappropriate level of abstraction), tools that are directed at Object-Oriented Software have focused primarily on static code structure. The disadvantage of these is that they only provide views of the code at disjoint points in time”.

It is believed that tools are required that focus on the dynamic aspects [Booch 97], [Bouzeghoub 97], of object-oriented systems in order to aid the understanding of concepts of code reuse for the debugging and tuning of applications. Moreover visual tools are considered the most effective for this task. The reason for this is that steady streams of text usually overwhelm users. Scientific visualisation and program visualisation have demonstrated repeatedly that the most effective way to present large volumes of data to users is in a continuous visual fashion [Kimelman 91], [Nielson 90], [Upson 89].
2.4.4 Weaknesses in OO Methods

The particular aspect of the methods of concern in thesis is the behaviour of a system based on the respective behaviour(s) of its components. In this section, we look briefly at how contemporary methods tackle this particular problem.

It seems that all methods in the previous section share some aspects of OO: several types of models, static, semi-static, dynamic and even code models. However, [Bouzeghoub 97] notes that some methods seem to be aimed at programming language systems rather than a tool for analysis and design of systems. OMT, for example has several model types, static, dynamic and functional, is rich in concepts and seems to cater well for many of the ideas first seen in the Semantic Data Modelling area. This makes OMT appropriate for OO applications development. Fusion, like OMT has three types of models, these progress well from specification to coding, it also has a more comprehensive design process than other methods.

There seems to be little focus in any of the methods above on the combined concepts of aggregation – behaviour – inheritance. This thesis focuses on these three aspects due to the important role they play at the heart of every system, there are no systems that are not made up of other components (aggregation). All components interact with the real world, no component sits idle and at the same time carries out functionality and services (behaviour). Finally, the components of the system pass their attributes and functionality to the system itself, and this effect can be repeated at several levels of abstraction (inheritance).

While all methods acknowledge the importance of these three concepts separately, there is little attention to the argument that these (three) concepts imply much significance when combined together. A particular aspect that we find prominent is the restricted application of inheritance hierarchies to generalisation as opposed to aggregation. In many OO methods manuals generalisation hierarchies are synonymous with inheritance hierarchies in many of these methods. The
exception to this is found in OMT, which provides a reasonable insight into the concept of aggregated components as a unit.

Almost all methods seem focused on the coding aspect of systems design without having sufficiently covered semantic aspects, which are of prime importance. These concepts include, for example, association. This important abstraction mechanism is poorly covered in all methods and arguably warrants further attention.

Finally, there is the problem of the concepts themselves: For example, not all authors in these methods agree on some of the concepts which are, basic as far as the process of constructing OO systems is concerned. When talking about aggregation, for instance, not all methods agree with OOA [Coad & Yourdon 90] that an example of aggregation is a relationship between an organisation and its clerks, [Rumbaugh et al 91] state that a company is NOT an aggregation of its employees. [Fowler 97] summed up this scenario very well “…when the gurus can’t agree, what do we do?”.

2.4.5 Weaknesses Relating to Behaviour Specification

In this section, we evaluate the coverage that inheritance of behaviour in complex systems receives in the various OO methods discussed above. What we are searching for is a method for representing aggregate systems behaviour through the behaviour(s) of such systems- components. Any such method needs to shed some light on concepts such as aggregated systems, their complexity and their construction methods from active components.

The idea of the how to construct, represent and implement new (aggregate) objects from current ones is lacking in OOD [Booch 91]. This is despite the fact that the mere concepts of aggregation, complex objects and abstraction are reasonably treated in the method. OOD has, however, made significant contributions to the idea of utilising an OO programming language in the implementation process.
OMT [Rumbaugh et al 91] provided an insight into the issues considered in this thesis regarding the significance of the complex object and the importance of aggregation in static and dynamic areas of systems design. It conforms to the fairly standard model of layers of details, each called a model (static, dynamic and functional). The idea that the state of the system is a combined state that includes one state from each of the components of that system is interesting and will be looked at further in this thesis.

The problem with OMT, however, is that it does not follow through with the encapsulation of behaviour ideas. The system’s behaviour model remains a collection of separate state models each in its own corner of the page. Whatever interaction there is between the components is too abstract and not combined to yield a single behaviour model.

The subject of behaviour modelling is insufficiently covered in OOA/OOD [Coad & Yourdon 91], even though the method supports new semantic concepts such as inheritance, operations and classes. The method, therefore, adds little contribution to the research in this thesis.

When it comes to notions such as object identity, complex objects and dynamic attributes, OOA [Shlaer & Mellor 88], probably by virtue of its age, is much closer to the classical systems than to the OO paradigm. However, there is no mapping between static aspects of objects and the operations they can perform and, like OOA/OOD, there is little provision for behaviour encapsulation. There are, however, many good ideas that have come from OOA: noteworthy is the notion of object lifecycles and the utilisation of state diagrams for the specification of functional aspects of activities that take place inside the boundaries of a single state.

With respect to object behaviour, Fusion method [Coleman 94] is also weak. A peculiar aspect of Fusion as a method is that the dynamic aspects of classes are considered only as a follow-up to the functional design of a system. This implies that object/class functionality is invisible until that stage, whereas in other methods (OMT, for example) the methods an object offers are encapsulated within its definition from the outset.
UML [Booch 97] has emerged during this research as one of the most popular systems design languages. Most of the ideas in UML have existed in some form or another as part of OMT, OOSE [Jacobson 92], and OOD. It is widely held that there is a lot of promise in the UML approach; it is a comprehensive tool that covers more aspects of systems analysis, design, implementation and support than other approaches.

Our interest here is the method by which the behaviour of components in an aggregate can be gathered together to yield a single behaviour of a single entity, the system. While UML offers many insights that relate to this problem, the solution remains conspicuous by its absence, just as it did in OMT and OOD before the emergence of UML. One prominent of UML feature is the concept of Use Cases. A use case is an interaction of the whole system with the environment. In tune with the OO requirement that behaviour is encapsulated in classes, a use case generates partial behaviour(s) from various scenarios and assigns those to respective classes in the system. The behaviour of the system is then described by aggregating the partial behaviours from all use cases and all classes.

The work in this thesis can be contribute here, this is by helping to understand the concept of behaviour aggregation, this work allows designers to investigate completeness and consistency by identifying system scenarios which can cause unexpected or disallowed behaviour.

2.5 Problem Identification

Having looked at some of the background work that lead to the new OO methods emerging, we are now in a position to look in more detail at the problem this thesis proposes to tackle. This problem will be introduced within the context of Abstraction Hierarchies, Inheritance of Behaviour and Composite Objects.
2.5.1 The Effect of Abstraction Hierarchies

As a prelude to identifying the problem that we are addressing, this section discusses the role of abstraction hierarchies within the process of behaviour modelling. We start with the association abstraction.

**Association**

The notion of association in most object oriented design approaches describes an object-object link and is distinct from the notion of an association as introduced by Brodie and Silva in their ACM/PCM paper of 84 [Brodie & Silva 84]. The latter’s notion of association is that of a group of objects having an ‘IS-MEMBER-OF’ relationship to a set.

Association, in our opinion, constitutes a special case of the aggregation hierarchy wherein all objects at the lower-level side of the hierarchy are of the same type. Unfortunately, this view of association is hardly considered in any of the techniques we have reviewed.

A popular implementation of the association abstraction today is probably that used in Microsoft’s development tools, like Visual Basic, Visual C++ and Visual J++ [Microsoft 99]. Here, the notion of object *collections* plays a major role in grouping objects of the same type into a single, popular notion that is used by almost every programmer/developer. This is consistent with the view presented above, which emphasises the similarity, but distinguishes between association and aggregation data abstractions.

**Aggregation**

According to [Smith & Smith 77] and [Brodie De Silva 84] the definition of aggregation is a form of abstraction in which a relationship between component objects is considered as a higher-level composite object. This is referred to as the “is-part-of” relationship.
In some OO design techniques [Rumbaugh et al 91], [Booch 91], there is a tendency to weaken this definition. For example a relationship between an entity and its attributes is sometimes wrongly considered as conforming to this definition. Here the reader will often find examples like a STUDENT is an aggregate of NAME, ADDRESS, etc. Looking back at the definition of aggregation above, we find that it strictly specifies that all sides in such a relationship have to be objects, we also mentioned above that objects have identity. It is difficult to see how ADDRESS or NAME can be considered as objects taking part in an aggregation hierarchy. We believe that such items are better considered as object properties. Aggregation will be the subject for further discussions in the following chapters.

2.5.2 Inheritance of Behaviour

The behaviour of objects and classes is a definition of the operations (sometimes referred to as services) that these objects in their classes are capable of exhibiting. This is usually captured diagrammatically using methods like State Transition Diagrams (STD’s) which show for each object:

- A set of all possible states that the object goes through in its lifecycle. These include an initial state as well as, possibly, a final state.

- The set of all permissible state changes that take place between states

- A set of events/messages that cause the state changes

- Variation of STD’s may show information associated with states, for example, actions. In addition there are activities which are generally associated with state changes.
One of the main contributions of the object-oriented paradigm is inheritance. This applies to notions of both structure and behaviour. However, as we have seen in the previous section, inheritance is narrowly viewed in terms of generalisation hierarchies, this is to the extent that such class hierarchies are often referred to as inheritance hierarchies.

While we do not dispute the significance of inheritance within generalisation hierarchies, we suggest that there is a role for inheritance to play within other class hierarchies; aggregation in particular. This role manifests itself clearly in composite objects that are made up of components that interact. This concept is explained further in the next section.

### 2.5.3 The Role of Composite Objects

A composite object has a complex internal structure defined in terms of other (more elementary, but possibly complex) objects. An aggregation relationship [Smith & Smith 77], [Brodie & Silva 82] exists between the class of the composite object and the classes of each of its composing objects. However, while classes have a well-established semantics and may be described in object-oriented programming languages, there is no standard method for representing the semantics resulting from the aggregation relationship between classes [Rubin 92], [Monarchi 92].

We discuss composite objects and the role of aggregation relationships, and focus on the dynamic aspects of such class hierarchies, as these constitute an even more difficult problem to represent within the framework of object-oriented analysis and design.

The notation that will be used for studying aggregate objects has been applied by [Smith & Smith 77] [Brodie & Silva 82] and used by [Rumbaugh 91], [Coleman 94] and [Booch 97]. This is normally (but not exclusively) represented as shown in Figure 2.1.
In the following sections we will be using this notation in conjunction with state-based behaviour description techniques to study the mechanisms by which the behaviour of low-level components of an aggregation can affect that of the higher-level aggregate. In other words, the question we will be addressing is: Given the behaviour description of the components of an aggregate, is it possible to describe the behaviour of the aggregate itself?

What we have tried to achieve here is to quickly skim through enough methodologies to provide an indication of the lack of coverage with respect to the specialised area that we are interested in.

In the next chapter, we will look, in more detail, at some of the technologies that formed the basis for our approach to solving the behaviour inheritance problem. This will be followed by the proposed solution to this problem. Subsequent chapters of the thesis will include a case study that will demonstrate the workability of the proposed approach.

2.6 Summary

In this chapter we have looked at some of the contributions made by popular semantic data modelling techniques as well as modern OO design methodologies. We have also identified a salient area that merits further research, this area is the representation of behaviour and its inheritance in complex systems. A thorough discussion of even a small percentage of methods on the subject of OO design can easily exceed the size of a single thesis. Our focus has been turned towards aspects of the methods that are directly related to the contribution that this thesis...
is attempting to make. In the following chapter, we shall look, in some detail, at how to represent the behaviour of systems, and at some of the contributions made in this area by methods within the fields of Automata Theory, State Diagrams and Petri Nets. These methods will form a basis for the underpinning technology applied by this thesis.
CHAPTER 3

UNDERPINNING TECHNOLOGY
3.1 Introduction

In the previous chapter of this thesis, we looked briefly at semantic data modelling techniques [Falkenberg 96] and OO design methodologies [Kent 98], [Fowler 97]. We saw that although these methods have much to contribute within the domain of systems design, there seem to be some areas that are insufficiently covered. This thesis focuses on one of those areas: inheritance of behaviour within aggregation hierarchies.

This chapter studies some of the methods that can be utilised for specifying dynamic aspects of systems [Deutsch 91], [Bowers 93]. The objective here is to exploit some of the ideas in these methods by extending them to build a basis for the proposed approach.

Many of the OO design methods discussed in the previous chapter agree with the significance of state based behaviour specification techniques. We will focus on methods that utilise the notion of state as it offers semantic richness in terms of specification details; a state not only shows what is taking place now within the system, but also what can happen next, and, possibly, how we got here.

The background work of this chapter stems from fields that include Automata [Hopcroft 79], State Diagrams [Harel 88a], Finite State Machines [Gill 81], various forms of Petri Nets [Brachman 79], Communicating Sequential Processes [Hoare 78], The Calculus of Communicating Systems [Milner 80], Sequence Diagrams [Zave 85], and Temporal Logic [Pnueli 86]. The specific areas we will be focusing on in the chapter include State Transition Diagrams, State Diagram Matrices, StateCharts and Petri Nets.
3.2 More on Problem Identification

We looked briefly at the problem identification in the previous chapter. This section describes, in slightly more detail, the aims and objectives of this research. We start with a look at the role of encapsulation.

3.2.1 The Role of Encapsulation

Encapsulation is a fundamental principle of OO design. It offers the possibility of levelling and abstracting system specifications. The basic concept is that any collection of interacting objects may be regarded at some level of abstraction as a single unit which encapsulates within its description the descriptions of all the corresponding components. This notion can aide recursive synthesis [Bowers 93a] which is based on the hierarchical composition of high-level objects from lower-level components. The fundamental principles of this approach are as follows:

- Every object has both structure and behaviour

- The externally observable behaviour of an object represents an interface for the object, which shows the services the object is capable of providing. Aggregate objects (at an appropriate level of abstraction) should be treated as a unit (system) and have a single externally observable behaviour

- Components of an aggregate interact with each other as well as with the problem domain
3.2.2 Composite Objects

From the previous discussion of OO design methodologies we can see a requirement for a method to model the dynamic aspects of such systems, which have a reactive nature in addition to being complex systems comprising other (more elementary) objects.

A composite object has a complex internal structure defined in terms of other (more elementary, but possibly complex) objects. An aggregation relationship [Smith & Smith 77]; [Brodie & Silva 84] exists between the class of the composite object and the classes of each of its component objects. However, while classes have a well-established semantics and may be described in object-oriented programming languages, there is no standard method for representing the semantics resulting from the aggregation relationship between classes [Rubin 92]; [Monarchi 92].

We discuss composite objects and the role of aggregation, and focus on the dynamic aspects of such class hierarchies, as these constitute a difficult problem to represent within the framework of object-oriented analysis and design.

The reactive nature of composite objects implies that they have an interface which handles messages from their environment. This reaction can involve the sending of several more messages to objects within the hierarchy, or the carrying out of an activity and its termination. In situations where the reactions to the input event yields a second message to another object and then a third message to a third object and so on, we say that there is a chain of reactions to the input event. Chains of reactions constitute a fundamental part of the contributions of this thesis and will feature in the following chapter.

In the following sections, we will be using state-based behaviour description techniques to study the mechanisms by which the behaviour of low-level components of an aggregation can affect that of the higher-level aggregate.
The question that we are faced with at this stage is as follows: Given the behaviour description of the components of an aggregate, is it possible to describe the behaviour of the aggregate itself? Figure 3.1 illustrates the question: Given the behaviour(s) of components A1 and A2, what is the behaviour of aggregate A?

![Diagram](image)

**Figure 3.1 The Problem Illustrated**

### 3.2.3 Example

To illustrate the main concept we utilise a simple example of a generic object VEHICLE that is composed of two objects: SWITCH and ENGINE. Figure 3.2 shows the behaviour description of both.
Figure 3.2 Behaviour Description of 2 Objects

The notation used in the graphs here has been applied by [Rumbaugh et al 91] and is based on the use of labels for transitions between states (which are shown as ovals), the directed arcs representing state transitions. Each transition is labelled by the name of the event/stimuli that caused it [Harel 87].

The problem of determining the behaviour of the VEHICLE from the respective behaviours of its two components can be solved in a straight-forward fashion by generating a state transition diagram which will have 12 (4 x 3) states. The question is: Are these states all essential as far as the behaviour specification of VEHICLE is concerned? Our discussion below attempts to answer this question.

Semantic Issue which will tackled by this thesis

In addition to the main issue of capturing the behaviour of the aggregate object from that of its lower-level components, there are a few semantic issues which need to be addressed, both as part of a technique focusing on aggregation hierarchies and as part of any modelling technique in this field. These include the following:

- Conditions on transitions (in their varying forms)
• Internal stimuli and transitions that trigger other transitions (the effect of exit/entry actions)
• Broadcast events that affect multiple states in concurrent components
• Constraints which reduce the state space resulting from the Cartesian product of the states of the components
• State grouping and diagram levelling as a method to capture depth and concurrency in an economical way
• Multiplicity of paths in the state space at the externally observable level
• State equivalence in relation to simple events and aggregate events

3.3 Describing Behaviour

According to [Champeaux 93], “behaviour description is a notoriously difficult problem. Physics borrows from mathematics the notion of differential equations to describe changing entities, fluids, gases etc. This trick is unavailable to us. The behaviour of objects in our domain of interest practically never satisfies differential equations; even a simple device like a piston engine is beyond the formalisms of differential equations”.

There is a requirement within systems design for a method and a notation that has a natural correspondence with the problem at hand. The problem we are interested in is to represent (using state based notation) the formalisms of behaviour inheritance across aggregation hierarchies. We are searching for a technique that has the following features: Based on Visual Formalism, Supports States Based Behaviour, Supports Multiple Components, Supports Hierarchic Modelling and can be utilised for complexity reduction. The collective list of methods that provide these features includes State Transition Diagrams, Statecharts, State Diagram Matrix and Petri Nets.
3.3.1 Visual Formalism

Methods that have visual and formal features have an advantage over others in the field of systems design, "...because they are to be generated, comprehended, and communicated by humans; and formal because they are to be manipulated, maintained, and analysed by computers...." [Harel 88]. All of the methods that follow in this chapter fit well within this category. State Transition Diagrams (STD) are the most basic and show state space, events and actions that result from state changes. A typical STD appears as illustrated in Figure 3.3

![Figure 3.3](image)

However, STD’s are too simple and flat, and cannot be used to solve the problem of aggregate objects behaviour specification for the following reasons [Harel 88a]

- State transition diagrams do not provide means of expressing conditional transitions
- State diagrams are uneconomical when it comes to transitions. For example if an event causes multiple transitions from several states, it needs to be attached to all of the states separately which results in a complex multitude of arrows.
- State diagrams are unfeasible when it comes to representing state spaces across a time plane; the number of states grows exponentially as the system grows linearly.
- State diagrams are inherently sequential in nature and do not cater for concurrency, a very significant aspect of aggregate behaviour.
**Statecharts**, developed by David Harel, [Harel 87] use a visual notation to model concurrently behaving objects in terms of the states and events. Figure 3.4 shows a typical statechart which shows hierarchy and concurrency. We shall revisit statecharts when discussing hierarchies below.

![Figure 3.4 Hierarchy and Concurrency in Statecharts](image)

Another visual modelling technique is **State Diagram Matrix** (SDM), which is a hierarchical formalism that can be used to capture the behaviour of reactive and multi-component systems. The main construct of SDM is a three-dimensional graphical specification technique to capture the behaviour of each component state machine. SDM will be discussed in more detail within the context of concurrency.

**Petri nets** [Petri 62] are used as a tool for the study of systems that allows a system to be modelled in a formal and visual way. They have been developed particularly to model systems with components that behave concurrently. These will discussed further within the section on concurrency and state reduction below.
A typical petri net looks as shown in figure 3.5, and is composed of the following parts:

- A set of Places (states)
- A set of transitions
- An input function
- An output function

![Figure 3.5 A Typical Petri Net](image)

### 3.3.2 Hierarchic Notation & Statespace Reduction

The second feature of the behaviour modelling technique this thesis addresses is hierarchic representation of dynamic features of objects and systems. This feature is not supported by state transition diagrams, since they offer no natural notion of depth or hierarchy.

**Statecharts** offer better expressiveness in this area, they support both depth and orthogonality. **Depth** is represented through the inner aspects of the blobs (states) themselves (see Figure 3.6). The model in Figure 3.6b is equivalent to that in 3.6a. Because states A and B do not overlap and are completely inside D, this implies that the latter is the exclusive or X-OR of the former. i.e., being in D is equivalent to being in either A or B but not both, see [Harel 87].

The small default arrows depend on their encompassing blobs. In Figure 3.6a, state A is the default **Entry State** of the three in the system, this fact is represented in 3.6b with the
top default arrow. The bottom arrow, however, states that B is default among A and B if we are already in D, and therefore the h arrow is not continued beyond the boundaries of D.

Statecharts express orthogonality using the AND decomposition which is captured using partitioning, another feature of higraphs which is based on the use of one Cartesian product. Figure 3.7 shows an arbitrary superstate A as composed of two orthogonal substates X and Y. These are related through an AND relationship, i.e., being in A is equivalent to being in both X and Y at the same time, hence the two default arrows that point to X and Y.
States X and Y can be decomposed into their substate models resulting in two orthogonal state diagrams as shown in Figure 3.8.

![Figure 3.8 Substates Further Decomposed into Sub-Models](image)

This representation reduces by some considerable magnitude the size of the potential state model which could result from modelling X and Y in one model. Such a model would include 12 states with some 15 transitions between them as shown in Figure 3.9.

![Figure 3.9 Non-Orthogonal Statespace](image)

**State Diagram Matrix SDM** offer some promising ideas when it comes to reducing the complexity of state models. SDM applies a form of **state grouping** in which states with similar behaviour within a domain in the matrix are grouped together to form a macro state-transition description. Figure 3.10 shows an example. The only issue here is that...
the criteria to use in the grouping of states across the domains of the SDM are not described in a precise or clear way which can make the whole process a little ambiguous with a lot of guesswork.

Figure 3.10 State Grouping
Based on Similar Behaviour

Petri Nets in their original format do not offer much in terms of dealing with hierarchic behaviour models [English 93] and state reduction of models. However, Petri Net extensions, particularly Hierarchical Coloured Petri Nets [Huber 91] offer a wealth of ideas that can be utilised for the purpose of reducing the number of states in systems behaviour models in a hierarchic fashion. This particular aspect of petri nets is discussed in a separate section below.

3.3.3 Inter-Component Communications

The sections above imply a departure within this research from the very basic notion of state transition diagrams towards techniques which are much more adept at dealing with semantic constructs and complex and hierarchic features of the systems we are interested in.
The communications that take place between objects within a system are most important because they produce what we refer to as the *behaviour signature* of the system, which is the externally observable behaviour of the system. Within *statecharts*, communications (synchronisation) between components $X$ and $Y$ of a system take place as follows:

- **Simultaneous Transitions.** These take place in both regions of the orthogonal diagram as a result of the occurrence of an event that can be responded to by both. Event $d$ for instance causes a transition from state $X_3$ to state $X_4$ and from state $Y_3$ to $Y_2$ at the same time, i.e.,

  $$(X_3, Y_3) \xrightarrow{d} (X_4, Y_2)$$

- **Merging and Splitting Transitions.** These are transitions from/to the orthogonal box to/from states outside the box respectively.

- **Conditions.** Conditions on transitions are Boolean expressions which prevent the transition from firing in response to stimuli until these expression are TRUE. Sometimes these expression can take the form, say $(Y$ in $Y_2)$ for the example above. This implies that a transition with such a condition will only take place if the superstate $Y$ is in $Y_2$, i.e.,

  $$(X_1, Y_2) \xrightarrow{C_1(Y \text{ in } Y_2)} (X_4, Y_2)$$

- **Output Events** (sometimes referred to as actions). These can be attached to the transitions in one machine and can affect another. This notion brings statecharts one step closer to *Mealy* machines [Hopcroft 79]. For example, Figure 3.11 shows a three component orthogonal state $A$. If $A$ is in the combined state $X_1.Y_2.Z_1$ and event $e_1$ arrives, then the resulting state will be
$X_2. Y_1. Z_2$ as $e_1$ causes a transition in $Z$ and this causes one in $X$ and the latter causes one in $Y$. This is called a chain reaction of length 3.

State Diagram Matrices do not have anything particularly significant to offer here. Communications between Petri nets representing the behaviour of multiple components of one system that share the same pool of resources take place in several ways including most notably, communication protocols and shared events. Figure 3.12 shows an example of two communicating processes.
3.3.4 Concurrency and Multiple Components

Murata et al argue that “statecharts is insufficient for explicitly specifying the interaction of concurrent state transitions, nor can it describe state abstractions in multi-layer state machines” [Murata et al 93].

Hence they propose **State Diagram Matrix SDM**, which they argue can be used to capture the behaviour of reactive and multi-component systems. The main construct of SDM is a three-dimensional graphical specification technique to capture the behaviour of each component state machine CM (consisting of a maximum of three of the sequential machines SM’s that make up the overall system PM),
\[ PM = (SM_1, SM_2, ..., SM_n) + R \]  

PM: is a plural machine representing the system  
State of \( PM \) at any time is a combined state, one state from each \( SM_i \).  
\( SM_i \): one of a set of sequential machines that behave concurrently  
\( S_I \): Set of states of \( SM_i \)  
\( R \): transition control rules, which are constraints imposed among the state  
transitions of \( SM_1, SM_2, ..., SM_n \) so that \( PM \) totally satisfies its  
behavioural requirements.

To model the behaviour of a \( PM \) system with \( n \) sequential machines an \( n \)-dimensional  
space is required. The alternative is to divide the \( SM \) set into subsets \( CM \)'s each with at  
most three \( SM \)'s, hence,

\[
PM = (CM_1, CM_2, CM_3, ..., CM_m) + R' \text{ is a } \\
\text{three-some grouping of the } SM \text{'s where }  
CM_1 = \{(SM_1, SM_2, SM_3) + r_1\}  
CM_2 = \{(SM_4, SM_5, SM_6) + r_2\}  
\text{...}  
CM_m = \{(SM_{m-2}, SM_{m-1}, SM_m) + r_m\}  
R = R' + \{r_1 + r_2 + .. + r_m\} 
\]

i.e.,  
\( r_i \) constraints on a single \( CM_i \)  
\( R' \) constraints between \( CM \)'s  
\( R \) Total

Each \( CM \) is modelled as a three-dimensional state space where the states of the  
component are represented as black dots and the transitions as arcs between the dots.
Conventional Description for SDM Behavioural Specifications

In an event-status matrix, there is a vertical axis that defines the $SM$ states and a horizontal axis that defines input $I$. In the state-transition diagram (For instance see Figure 3.13) which is equivalent to the event-status matrix, $S$ and $f$ are described as circles and arcs respectively. The contents of $I$ and $g$ are related to each state transition of $SM$, and are described near the arc corresponding to the state transition.

![Figure 3.13 SDM in 3-Dimension](image-url)
These description methods are simple and suitable for describing SM specifications when there are relatively few states in $S$. However, as the size of $S$ gets larger, readability and clarity of the specifications becomes lower. This is because of the combined complexity of state transitions.

Concurrency in Petri nets is represented by two or more transitions awaiting the same stimulus. When that stimulus occurs, if all the other conditions are satisfied for both (or all) the enabled transitions, then either one of these can fire, thus resulting in a non-determinant scenario. For example, in figure 3.14, P1 is an input place for two transitions, $t_1$ and $t_2$, both of which are fully enabled. When P1 is filled, only one of $t_1$ and $t_2$ can fire, because either will remove the enabling token from P1, thus disabling the other transitions.

\[\text{Figure 3.14 Concurrency in Petri Nets}\]

3.3.5 Hierarchical Coloured Petri Nets (HCP Nets)

Since Petri’s doctoral thesis was written in 1962, there have been several developments of Petri nets which addressed particular problems, such as support for modelling of time-critical systems [Ghezzi et al 91], [Berthomieu 91]. Other variations of the original Petri Net theme include:

- **Place-Transition Nets.** These introduced multi-token notation [Jantzen 79].
- **Elementary Nets.** Attempted to focus on some of the technical aspects of petri nets [Rozenberg 87], [Thiagarajan 87].
• **Timed & Stochastic Nets.** In these models, time variables are attached to tokens and transitions so the net can simulate real-time applications [Hillion 89]. Stochastic nets attach probabilities to arcs and transitions [Dutheillet 91], [Marinescu 88]

• **High-Level Nets.** Based on token-type checking mechanisms. Most known of such models are Predicate Transition Nets [Genrich 81], [Genrich 87], and Coloured Petri Nets [Jensen 86], [Jensen 91].

• **Extended High Level Nets.** These deal with representation of concurrent real-time systems [Camurri et al 92], [Papelis 92].

[English 93] surveys the specific contributions of these techniques. One particular extension to petri nets, Hierarchical Coloured Petri Nets (HCP Nets) [Huber 91] introduced several good ideas that can be utilised for behaviour modelling and, specifically, state reduction as will be seen in chapter 4 and chapter 5.

HCP Nets are a technique for creating one large net from several small coloured nets, referred to as pages. This is achieved using five hierarchical constructs:

• Substitution of transitions
• Substitution of places
• Invocation of transitions
• Fusion of places
• Fusion of transactions

Each small net is held in a page and the set of pages which make up the complete model are related using the five constructs above. A typical page hierarchy would look like that shown in Figure 3.15.
Each node in the hierarchy represents a page, and each arc represents a hierarchical relationship between two pages. While square-cornered boxes represent pages which can be used as subpages in one relationship but may be superpages in another. Round-cornered boxes in the hierarchy represent superpages which are not used as subpages. Each arc is labelled with the name of the compound node from the superpage of the relationship.

An initial state for the execution of an HCP-net is specified as a set of starting pages, which are referred to as prime pages and denoted by an M-tag. In the final model, only pages which are superpages should be defined as prime pages. However, during the process of building a model, subpages may be defined as prime pages. This is in order to limit the set of pages used in a simulation.
3.3.5.1 Substitution Transitions

These allow the user to replace a transition (and its surrounding arcs) by a more complex net, thus giving a more detailed description of the activity than that provided by the single transition. This idea is similar in many respects to that of Yourdon Diagrams [Yourdon 82] and to the module concept of modern programming languages.

Figure 3.16 below illustrates the idea with an example of an assembly line consisting of machines and buffers. The machines are identical; hence they are modelled only once. The same applies to the buffers. The page in the left part of the Figure is a model that shows each machine and each buffer. The complete model consists of three machines and two buffers. The details of the behaviour of the machines and the buffers are given in the two adjacent pages in the right part of the figure.

The result is a hierarchical CP-net where five substitution transitions at page Assembly Line Page1 level are related to two subpages Machine Page2 and Buffer Page3. The inscription next to each HS tag (HS = Hierarchy + Substitution) defines the mapping between each of the substitution transitions and the corresponding subpage. The inscriptions show the name and number of the page to use for substitution as well as how each of the places around the compound transition is assigned to one of the border nodes of the subpage.
Places in the superpage work in the same way as actual parameters of procedures in programming languages, they are referred to as **socket places** of the substitution transition. Places in the corresponding subpages work in the same way as formal parameters in procedure definition, and are called **port places**.

The latter are identified in the subpage diagram by a B-tag (for border), with an inscription that defines the mode of the place (In for input, Out for output, I/O for both, and G for any). The process of assigning actual parameters (socket places) to their corresponding formal parameters (port places) is known as **port assignment**.
3.3.5.2 Substitution Places

A substitution place hides a more detailed subnet at the upper-level of abstraction. It has an interface which consists of transitions and behaves in a way which resembles an abstract data type. The way to model this is to have two-levels of a model one with the substitution place in the same role as a standard place, and one with the details of the substitution place shown. The interface indicates the lower-level model where a subpage that represents this substitution place is found. Figure 3.17 shows a substitution place Place-1 with its subpage SP #1.

Figure 3.17 Substitution Places

Transitions surrounding subplaces are known as socket transitions. It can be seen from Figures 3.16 and 3.17 that substitution places work in the opposite way that substitution transitions operate: the two are in fact mirrors of each other.
3.3.5.3 Invocation Transitions

Invocation transitions are not substituted by their corresponding subpage as in substitution transitions. Rather, every time they occur, there occurrence triggers the creation of a new instance of the subpage. The instances of the subpage persist concurrently with the other page instances in the model until an exit condition is reached.

Tokens are passed between the invocation transition and the subpage instance when an invocation page instance is created or terminated. This occurs is similar to the passing of parameters between a subroutine call and its execution.

3.3.5.4 Fusion Sets

This concept allows the modeller to conceptually fold a set of nodes into one single node without needing to represent the space as a single object graphically. The folding creates a fusion set containing an arbitrary number of places/transitions and these nodes are called fusion set members. The places within a fusion set are concurrent, so that when a token is removed/added from/to one place in a fusion set, it is also removed/added from/to all the other places in the set. All places in the set must therefore have the same colour set and initial marking.

The places in a fusion set can be drawn from a single page or from several pages in which case the fusion set would is referred to as a global fusion set. Places from the fusion set are tagged with an FG (Fusion + Global) tag and an inscription that shows the name of the set.

In the case of single-page fusion the set is called a page fusion set, as in the example of Figure 3.18. Here there is a page fusion set containing two members B1 and B2 with their FP-tags (for Fusion Page) and the inscription showing the name of the fusion set to which they belong.
In the case where the page of FB has more than one instance, there are two possibilities:

- To merge all instances of all fusion set members into a single conceptual mode,

or

- Only merge node instances which appear in the same page instance (merge into a node for each instance).

Both possibilities are considered useful as they give the user more flexibility; Figure 3.19 shows the case where the example presented above in Figure 3.18 has two page instances.
3.4 Evaluation

In order to achieve our objective, which is a method for describing the behaviour of complex systems, we can exploit some of the features that we have discussed above. These features, added together, should provide a powerful notation that can handle a large proportion of the semantics of complex systems.

The main constructs of STD’s that can be beneficial here are:

- Graphical formalism give more involvement for modellers and users
- They support State Based (Support for State Events, etc) behaviour modelling

Statecharts take these concepts much further to add:

- Support for orthogonal behaviour of components
- State grouping constructs
- Composite events/transitions (merging/splitting transitions)
- Concurrent Transitions
- Merging and splitting Transitions
- Multiple Entry States in Multi-component systems
- Conditions (between components)
- Output Events
- Combined State

SDM's contribution here is primarily the recursive decomposition of the system into groups of three parallel machines and the state grouping ideas.

The main constructs of Petri nets theory include:

- Support for concurrency
- Hierarchical extensions

### 3.4.1 Deficiencies

This section looks at the constructs that these specification techniques lack in the context of the specification of aggregate systems behaviour.

**Statecharts**

The various concepts introduced in statecharts formalism are helpful. However, they are in many ways semantically incomplete and in need of further clarification. For example, there are several ways in which orthogonal components in a system can affect each others' behaviour in addition to those mentioned by Harel. In particular, the notion of entry/exit actions that are associated with transitions is essential to this idea. Indeed
entry/exit actions are included (in a rather limited fashion) in OMT, which is based on statecharts. Here, OMT does not clarify their effect sufficiently.

Furthermore, the concept of state-grouping, which is fundamental to statecharts, needs clarification. The criteria used to group a set of two or more states are not laid out clearly and have to be deduced from a set of examples.

Finally, there are several semantic details on triggers, conditions, merging and splitting transitions, and transition paths that require further investigation. This is in order to add a set of comprehensive guidelines on semantic extensions to statecharts without which the whole technique can be put to little use in behaviour modelling of complex systems.

**State Diagram Matrix**

State diagram matrices are based on the use of a three-dimensional state space for the modelling of complex systems behaviour, which is a variation on a familiar theme, i.e., graphical formalism. This has many advantages over traditional flat state diagram techniques, as it caters for more complex real world scenarios, e.g., several concurrent components.

It is a rather more complex and skilful task to model in a three-dimensional space (having divided the system into sets of three orthogonal state machines) in a recursive way until all the system’s components are included. This method is so complex that it has to be further simplified by projecting the three-dimensional state space to a three-dimensional matrix. Matrix variables are then used to further simplify further as the size of the matrix becomes too large in proportion to the square of number of each axes states.

A method based on statecharts but with a more comprehensive notation for building statechart hierarchies would be a more acceptable alternative that is appealing to database
users many of whom are unfamiliar with multi-dimensional formalisms as well as designers.

**Petri Nets**

[Peterson 81] noted that even though Petri nets can model finite state machine systems, the state machine model has an advantage over Petri net models in that it is easier to understand. Construction of composite machines using Petri net theory, known as parallel net composition, relies on the requirement that to combine Petri net Pa with Petri net Pb, then the output alphabet of Pa has to match the input alphabet of Pb, or vice versa.

Peterson claims that this representation has an advantage in that it allows the two nets to behave concurrently. However, the concurrency advantage is not unique to the parallel net composition; it is equally supported by the more comprehensible finite state machine theory where concurrency is represented as a cross-product of two or more machines.

The author acknowledges the inherent complexity and explosive nature of cross product machine theory. However, it is the purpose of this research to look at that issue and to apply hierarchical constructs (some of which are developed to support Petri net theory itself) and other simplification techniques to solve this complexity problem.

### 3.4.2 UML Revisited

UML has made many contributions to the field of behaviour modelling. In particular, the author sees the following points as stepping stones in the path of behaviour recognition by object design methodologies like UML [Eriksson 98]:

- **Inter-Object Interactions**

These are represented as messages between the objects that form a system. There are several categories of messages (simple, synchronous and asynchronous).
• States & State Transitions

The UML contribution here conforms to the contributions in the previous work of the respective UML authors. UML uses a similar notation and idea to what we use in the next few chapters.

• Guard Conditions

These are rather too simple in UML, and in our opinion do not conform to a good mapping to the real world. The idea will be extended further in the chapter outlining our proposed approach.

• Substates

The notion of substate (and its superstate counterpart) is a very significant contribution. This has been recognised by [Harel 87], [Harel 88a], [Harel 88b] some ten years prior to the emergence of UML. We shall make extensive use of the substate/superstate notions in this thesis. In UML, this is discussed within the context of two (or more) state diagrams sending messages. This, inter-model co-operation will constitute an integral part of the contribution of this thesis.

• Concurrency

Like the notion of substates, concurrency was identified many years before UML. It is nonetheless an integral aspect of any systems design technique that aims to describe accurately the real world. Again, this idea falls within the domain of this thesis, and will be discussed in great detail.
• **Synchronisation**

As part of the subject of Real Time Systems Design, UML introduces the idea of coordinating concurrent threads of control within OO systems. Although this notion is limited in its applicability to shared resources and objects as well as the prevention of deadlocks and similar system issues, it is nonetheless a significant addition to the work of Booch and Rumbaugh prior to UML. Synchronisation of activities and events across different parts of large systems is also one of the issues that will be dealt with in this thesis.

### 3.5 Proposed Approach - Preview

The general approach for the expression of aggregate objects' behaviour proposed here is based on the notion of encapsulation. This offers the possibility of levelling and abstracting of system specifications [Bowers 93]; basically what is suggested is that the behavioural details of the components of an aggregate are irrelevant at higher levels of abstraction.

The specification of the individual components' behaviour is necessary to generate a specification of the behaviour of the higher-level aggregate object; this in turn becomes a representative encapsulation of the behaviours for all of the components. Once this task is carried out, and unless the user/designer is specifically concerned with individual components, there ought to be no requirement to keep track of the individual behaviour(s) of the components.

This is in effect one of the main advantages emphasised by object-oriented methods, and therefore should be applied correctly. Hence at high levels, what the users need to see is a reasonable number of complex objects which carry collectively the system's functionality and contain (encapsulate) within them the details of the other more detailed aspects of the system.
In the example of VEHICLE above, each transition at the VEHICLE level is representative of one or more (concurrent) transitions in each component. The total number of states and transitions is too large for a modelling technique to handle easily. However, it is not necessarily true that every possible combination state from the respective components is significant as far as the behaviour of the overall aggregate object is concerned. What is required here is a method of reducing the state space of the aggregate to a form that has a minimal, yet sufficient statespace.

Our next objective is to further reduce this. We have seen some useful hints and ideas within HCP Nets above that we can utilise here. We propose to carry this task out using state grouping techniques which collect together in a superstate any two (or more) states that exhibit between them identical externally observable behaviour.

Such a modelling notation should greatly reduce the complexity of systems and enable analysts to use several hierarchies of behaviour models. These hierarchies can be referenced by designers or users according to the level of detail that is required.

Accordingly, in the case of the VEHICLE example can be specified as a diagram that contains 5 states as opposed to 12 (Figure 3.20), these states are the significant ones (as far as the externally observable behaviour of VEHICLE is concerned) and incorporate within them the rest of the state space.

This implies that a state FAILED for instance is in fact a composite state of two product states, say \(\{[\text{IDLE, ON-1}] + [\text{IDLE, ON-2}]\}\), each of which is a combination of two states from the components level.
In order for this method to function correctly, the set of criteria for state grouping mechanisms has to be studied and finalised. The techniques we have seen in this chapter, particularly statecharts, are promising notion and represents good starting point as far as this problem is concerned. In addition there are several semantic aspects concerning the concurrently behaving components that require clarification. These are discussed in the following chapters.

In the next chapter, we will delve further into the problem domain and introduce a proposed solution to this problem. This will be followed by a case study that will demonstrate the workability of the approach.
CHAPTER 4

PROPOSED APPROACH
4.1 Introduction

The previous chapter of this thesis discussed the contributions made by some of the popular modelling approaches representing intricate scenarios from the real world. By considering those methods, we have identified an important area. This area is the representation of dynamic aspects of complex systems, which comprise reactive components and are organised in abstraction hierarchies. In addition, we looked at some approaches which have been developed to tackle this issue.

It is important to emphasise that the proposal made in this thesis is not intended as a basis for a new object or systems design methodology. Rather, the proposed approach, and any contributions, made here are intended as an enhancement to existing approaches; for example, the approach might be used to check the consistency of a UML solution.

4.2 Chapter Map

This chapter introduces an approach for representing inheritance of dynamic aspects of complex systems which are made up of multiple components. The chapter is organised as follows: Section 4.3 lists term definitions for the most used concepts. Section 4.4 is a summary of the proposed approach. Section 4.5 discusses the background to the chapter. In section 4.6 the three levels of behaviour specification that form the basis of this approach are introduced. The chapter demonstrates the workability of the approach by means of a simple example in section 4.7, which is followed by a look at specific issues such as variations and failure conditions which can affect the approach in 4.8. Section 4.9 is a summary and, finally the chapter discusses plausibility of the approach in section 4.10. Some of the term definitions and concepts for this chapter are located in appendix D
4.3 Definition of Terms and Notation

In the following sections of this chapter, some of the following terms will be referred to as we attempt to explain the details of the proposed approach.

**Object Behaviour**: The functionality which the object provides. This is manifested as methods that the object performs when a message is passed to it. Object Behaviour is described using states, events and transitions.

**State**: The stage of functionality the object is in at the current moment in time. In other words, what the object is doing now. In the sections below we shall encounter the following types of states:

- **Initial State**: The first state of the object when it is created/instantiated. This is denoted as follows:

  ![Initial State Diagram](image)

- **Final State**: Explicitly acknowledged by many OO design methods, in this research it is defined implicitly. There is no reason why a behaviour model for an object does not have many final states. No special notation is used for this type of state.

- **Transient State**: This is basically a state within which the object remains for near 0 time, hence no activity can take place here. This is presented as follows:

  ![Transient State Diagram](image)
**Unreachable State:** A state which exists in the system specification, but is impossible to reach (from a defined initial state) given all possible iterations of events in that same system specification.

**Superstate:** When building a behaviour model, B, by combining two behaviour models, B₀ and B₁, all states in B will be of the form “B₀Sₓ.B₁Sᵧ”, where B₀Sₓ is state Sₓ from the behaviour model B₀ and state B₁Sᵧ is state Sᵧ from the behaviour model B₁. In many cases, where the source of the states is obvious (for example when there are only two objects in the system) then “B₀Sₓ.B₁Sᵧ” is shown as “Sₓ,Sᵧ”. For example,

```
Object₁Sₓ. Object₂Sᵧ  Equivalent to  Sₓ,Sᵧ
```

**Transition:** What the object does in response to suitable event.

**Event/Stimulus:** Anything that takes place in the object’s environment

**Condition:** An optional Boolean expression that has to be satisfied for a state transition to take place in response to an event

**Action:** An optional event that occurs on the entry to a state, the exit of a state, or during the transition from one state to another.
In Figure 4.1 below, the object changes from state $S_0$ to state $S_1$ in response to event/stimulus $e_i$ and fires event $e_g$ (as an action) if the condition $C_1$ is satisfied.

![State Transition Diagram]

**Figure 4.1 State Transition Showing Cause Event, Condition and Action**

**Activity:** The functionality the object is executing within a state.

**Aggregate System of Objects:** A group of objects that collectively make up a system.

**Constituent Object:** An object that is part of an aggregate system of objects, sometimes referred to as Component.

**Behaviour Aggregation:** The process of combining two (or more) behaviour models from the aggregate objects to create a behaviour model for the system of objects.

**Transition Chains:** Occurs when the action (for example $e_g$ from figure 4.1) that is associated with a state transition is capable of causing a transition somewhere in the system. Figure 4.2 shows the effect of the transition from figure 4.1 if $e_g$ went on to cause a transition elsewhere in the system.
Note that the triggered transition CANNOT be in the same object as the cause action. The "t" in te1 indicates that this transition is caused by another transition (caused by ei) as part of a chain, sometimes modelled as "t(ei)"

![Diagram](image.png)

**Figure 4.2 A Transition Caused by Another Transition as Part of Chain**

### 4.4 Proposed Approach – Summary

The approach in this thesis is based on 3 levels of abstraction, each level showing a different amount of behaviour details. The first level is referred to as the Pi behaviour model (II). Here the behaviour(s) of the objects that make up the system are described individually. These behaviour models can be added up or aggregated at this level to produce a single behaviour model, which is based on the state product from the individual behaviour models. At this level of abstraction, no semantic interactions (conditions, chains concurrency) are shown, this is due to the autonomous nature of the constituent object behaviour models at this level.

The second level of abstraction is the Gamma behaviour model (Γ). Here, all the interactions that can take place between the constituent objects are added to the behaviour model, resulting in unreachable states, transient states, and other types of semantic effects.

The final level of abstraction is the Phi behaviour model (Φ). This is an abstraction of the Γ model, where the users of the system are expected to contribute some semantic requirement that, for example, the behaviour model
need not show states where the Bank Account is IN CREDIT and is IDLE. See
the example in section 4.7.

The work of this thesis does not have much to do with how the \( \Pi \) level is
generated, as this is merely the description of the system as provided by the
problem domain. The \( \Gamma \) level is a systematic transformation that is well-defined
and deterministic, meaning that there is only one \( \Gamma \) behaviour model for each
system. The \( \Phi \) level is provided by this research as a means of hiding various
aspects of the behaviour model of the system in order to give emphasis to other
aspects of the behaviour model. The \( \Phi \) level is an abstraction and there are
several of those for each system, depending on the user requirements. What this
thesis will propose is that it is possible to check consistency of a given \( \Phi \)
behaviour model in relation to its \( \Gamma \) counterpart, but there is no one correct \( \Phi \)
behaviour model for any system.

4.5 Background

The encapsulation of dynamic properties of objects in systems has been the focus
of several object oriented (OO) methods. Dynamic properties have been
emphasised through use cases [Jacobson 92] and scenarios [Kenneth 92], [Booch
97], [Bouzeghoub 97]. These methods focused on dynamics from the outset of
OO design rather than on the static aspects, which form the starting point of data
centred approaches. The main idea in these methods was to investigate system
behaviour before classes are identified. The main drawback of this approach is
that the view of dynamics is rather low level, i.e., focusing on events, states, and
transitions, as opposed to abstraction techniques such as aggregation and dynamic
concepts such as processes [Hoydalsvik 93].

This thesis suggests that the behaviour of individual components in a large
complex system is necessary to generate a specification of the behaviour of the
higher-level aggregation, which results from these components. Once this task is
carried out (and unless the user/designer is specifically concerned with individual components), there ought to be no requirement to keep track of the individual behaviour(s) of the components. As an analogy, a pilot in a (simplified) aeroplane need know only if the plane is functioning safely, and if an engine fails, need know only that the engine failed, not that the second or third blade on the propeller has been damaged.

One of the main advantages propagated by proponents of hierarchical object-oriented methods [Giua 95], [Harel 88a, 88b], [Brave 93], [McGregor 93] is to do with levels of abstraction. Here, the user/modeller needs to see a reasonable number of complex objects which collectively incorporate the system’s functionality and contain (encapsulate) within their definition the details of the other more detailed aspects of the system (temporal functionality for example). For further details of more objects, the user can move down the object hierarchy as required.

For the purpose of representing temporal aspects of systems of objects, the notion of state [Davis 93], [Harel 87, 88a, 88b], [Brave 91] [Rumbaugh et al 91] seems a suitable candidate tool. This is due in part to the state’s suitability as an answer to the question “What can take place in the object’s lifecycle next”.

In the following sections, we present the approach using a graphical notation. Graphical and visual formalism notations [Davis 88] [Edwards 93] have an advantage over other formalisms. Visual methods appeal to a broader audience, including not only audience with a mathematical background, but also others with backgrounds that cover the general area of systems design, system administration and support within the general area of business systems and business process design techniques.
4.6 Proposed Approach – Three Levels of Abstraction

The behaviour of a single object is not an aspect that can be described simply. Utilising the notion of states, the behaviour of a single object may be described as the re-distribution of all the functionality of the object over the set of states that this object takes throughout its lifecycle. This is because states can tell us what can happen next as well as what activity is taking place now.

This implies that a good description of object behaviour would include a set of states with a subset of the object functionality available at each state. For a complex object system made up of several objects, this can become difficult, especially if the objects in such a system interact.

According to [Rumbaugh et al 91]: “the behaviour of the system (of objects) is based on the state of the system. The state of the system is a combination of states; one from each component that make up the system”. While this may be true for simple, non-interacting systems, the issue is really more complex than that. Basically, what we are dealing with is a collection of objects that form an aggregation hierarchy to create a higher-level entity: the system of objects, which is an object in itself. Furthermore the nature of any such system of objects, whether related to a real world scenario or to the inner modules of a computer, is that the components that together form this system actually react (in response to stimuli) and interact (communicate responses) within the system [Harel 87, 88a, 88b].

The intricate nature of such systems raises the need for several levels of abstraction where a certain proportion of the detail can be added or hidden gradually as the user or modeller traverses from one level of abstraction to the other.

As an initial step in the specification of systems behaviour it is suggested that there might be three levels at which this complex information can be described.
Each of these levels, referred to as "levels of abstraction", offers a simpler view of the other. These levels of abstraction are defined as follows:

- The Pi level ($\Pi$) of abstraction describes the behaviour(s) of all the components of the aggregate object. Here the behaviour models of the systems components are shown individually.

- The Gamma level ($\Gamma$) combines the behaviour models from the $\Pi$ level to generate a single behaviour model relating to the aggregate object. The individual behaviour models of the components are combined with the semantic interactions added-in.

- Finally, the Phi level ($\Phi$) is an abstraction of the $\Gamma$ level wherein irrelevant details can be hidden.

The notation used in this thesis is described in some detail in Appendix C.

4.6.1 The Pi Level of Abstraction - $\Pi$

At this level of abstraction, the dynamic aspects of the system are described as seen in many methods [Rumbaugh et al 91], [Booch 91], using the notion of a separate behaviour model for each component that plays a role in the build up of the system. For example, assume our system is made up of several components or modules $M_1, M_2, M_3$, as shown in figure 4.3,

![Diagram of an aggregate system with 3 components](image)

Figure 4.3 An Aggregate System with 3 Components
then the behaviour at this level may be described using a series of independent state transition diagrams as shown in Figure 4.4.

![Figure 4.4 Behaviour of Aggregate System in terms of Component(s) Behaviour Models](image)

The behaviour model above implies that at any stage within the lifecycle of the system, the state of the system is a complex (aggregate) state, which is made up of one state from each component. Hence, a possible state would be \((M_1S_3, M_2S_1, M_3S_2)\) and so on, see figure 4.5. This is consistent with the notation in [Rumbaugh et al 91].

The advantage in this approach is that it offers a straightforward representation of the autonomy of each component within the boundaries of the system. Secondly, this approach almost always yields behaviour models which are moderate in their level of complexity; each of the components shows its state space independently, thus avoiding the complexity of the combinatorial explosion of states as will be seen in other approaches.

The main drawback of this approach is that (real world) components interact and communicate with each other in response to external stimuli, thus providing the overall functionality that is normally associated with the whole system; this approach fails to exhibit such interactions. This means it is left to the
modeller/designer to cater for, say, constraints that result in the systems lifecycle from these interactions. Such a process can be quite difficult due to the complexity of the interactions between components.

![Diagram](image)

**Figure 4.5 The StateSpace of the II Model**
*(No Transitions are Shown)*

### 4.6.2 The Gamma Level of Abstraction - Γ

The Γ abstraction is a single behaviour model, which takes into account as many interactions as possible between aggregated components. This is a rigorous amalgamation of the models of relevant components; the model is complete, minimal and accurate and has the distinct advantage of the ability to incorporate all types of interactions that take place between components as described below.

Since the Γ behaviour model is a transformation of the II behaviour model in which all the interactions between components are taken on-board, it follows that
given a set of components and interactions (II Level), there can be only one single 
\( \Gamma \) model. We consider how the various forms of interactions between components 
are modelled.

### 4.6.2.1 Broadcast Events

One of the main advantages that the Object paradigm offered over other OO 
methods is the approximation to the real world systems it represents. In the real 
world, many components of a system can be affected by a single stimulus. For 
example, a single press of a button in a factory allows power through to several 
machines which all run simultaneously. [Harel 87, 88a] identified the importance 
of this kind of event (broadcast event) in the modelling of dynamic aspects of 
systems.

Figure 4.6 shows a broadcast event affecting two of the three components of a 
system, with the third component unaffected. This figure shows the model as it 
would appear in the \( \Pi \) behaviour model. Each component has its own behaviour 
model, with one shared event between machine \( M_1 \) and machine \( M_2 \); the shared 
event is \( e_1 \). Each state shown is prefixed with the machine name it relates to.

![Figure 4.6 A Broadcast Event (e₁) Affecting Multiple Components](image)

The way this can be shown using the \( \Gamma \) model is shown in Figure 4.8 below. 
However, first we will show an example in figure 4.7 of what happens when the 
effect of broadcast events is not applied correctly. For the purpose of this exercise 
we will ignore the effect of Component number 3 as it is completely independent
as far as Broadcast events are concerned. Furthermore we hide other events on the system.

![Diagram of broadcast event affecting multiple components]

*Figure 4.7 A Broadcast Event (e₁) Affecting Multiple Components – Effect of Broadcast Event Not Shown (inaccurate)*

Next we propose a more accurate method for modelling this system, this takes on-board the effect of broadcast events (for our immediate purpose, we ignore other events).

![Diagram of broadcast event affecting multiple components showing effect]

*Figure 4.8 A Broadcast Event (e₁) Affecting Multiple Components – Showing the Effect of Broadcast Events (more accurate)*

The model represented in Figure 4.8 above is in accordance with the theory we propose for the construction of the Γ model; the main aspect of Γ we demonstrate here is the ability to represent broadcast events accurately and in a manner which more closely represents the real world. The Π model, as shown in figure 4.7, hides/ignores such semantic aspects, and is thus incomplete.
The explanation for the model in figure 4.8 is that as the broadcast event is transmitted to the system of objects, ALL objects that can respond to this event should do so. This implies that objects which are in a state which permits an object to respond, (sometimes referred to as acceptable state for an event), in actual fact, do respond.

The Π model, as represented in Figure 4.7 also carries the hazard of nondeterminancy. This is manifested by the following question: Which of the two transitions occurs as a response to event $e_i$? Is it

\[ M_i;S_{i1} \xrightarrow{e_i} M_i;S_{i2} \]

or is it

\[ M_i;S_{i1} \xrightarrow{e_i} M_i;S_{i1} \]

The answer to this question is, in fact, neither; the two components change state simultaneously. This feature of the Π model illustrates an important feature of OO systems referred to as Concurrency in Behaviour, which is almost impossible to represent naturally in the Π level.

### 4.6.2.2 Transition Chains

In the following sections, when referring to state changes in a behaviour model, one or more components respond to a stimulus by changing state. This scenario is described as a state responding to a stimulus.

The second aspect of systems design considered significant is the notion that an activity can occur in an object, not only as a result of a direct external stimulus, but also as a result of another activity in associated objects. Several object modelling techniques offer methods for dealing with object responses to events;
however, these responses tend to be over-simplified in relation to the real world scenarios that are being modelled.

For example, how should a model represent a scenario where an object responds to an event from beyond its boundaries, and then carries on to produce further events affecting, in turn, other objects in the aggregation of the overall system? The notion of actions [Rumbaugh et al 91], [Booch 91], [Booch 97] which are associated with state transitions is of potential benefit here.

We argue that such a situation as the one above can be catered for; the solution is demonstrated in figure 4.9. The external stimulus \( (e_1) \) is responded to by the only object which is in a suitable state to accept this stimulus, machine \( (M_1) \) in state \( S_1 \). This results in a response which includes as a transition action, a further stimulus, \( (e_7) \). \( (e_7) \) happens to be acceptable by component \( (M_2) \) in state \( (S_1) \); thus triggering a second state transition within the aggregate system, this time \( e_7/e_2 \). Event \( (e_2) \) is not acceptable by any other component in their current states, so the chain terminates.

The scenario above constitutes one of the simplest ways in which a chain of transitions can take place in complex reactive systems. There is, however, no reason why the second component may not have an action associated with its transition which is acceptable by the current state of component \( (M_3) \) or even component \( (M_1) \) itself. Figure 4.9 illustrates the scenario above in both the simple \( \Pi \) level (Figure 4.9a) and in the semantically enhanced \( \Gamma \) level (4.7b).

![Figure 4.9a A Single External Stimuli Generates a Response that Generates Further Stimuli](image)
A noteworthy consequence of this situation is the concept of a transient state. This is a state that lies in the path of a series of transitions that form a chain; this state is not reachable by any other transition outside the chain. For example, if the combination state \((M_1S_3, M_2S_1)\) were not reachable by any further transitions, then it would fall into this category. The significance of such states is that they are essential for an accurate behaviour model, yet such states should not carry any significant functionality, as their lifespan is always negligible.

### 4.6.2.3 Constrained Transitions

The third mode of communication between components of an aggregate system in the \(\Gamma\) model is manifested through a special form of conditional transitions [Rumbaugh et al 91]. There is a widely accepted notion of conditions that affect the time at which a transition may take place from a state, this can be based on factors like time and temperature for example. In most modelling methods, this is captured quite sufficiently, for example, in OMT, this is manifested by adding a "\(/\)" character to the label of the transition and following that with the condition, for example, event \((e_1)\) can cause a transition only if \((t)\) –time-is larger than, say
This is represented as \((e_1/t > 1600)\). As mentioned in section 4.3, we use the notation \(e_1(c_1)/\) to represent conditions.

The OMT approach is sufficient for scenarios based solely on conditions as external properties. The conditions that affect transitions can, however, originate from other objects within the aggregated system.

For example (see Figure 4.10), if a component \((M_1)\) is in a state \((S_1)\), and if stimulus \((e_1)\) were an acceptable one in this state, but a pre-condition for any transition from \((S_1)\) required component \((M_2)\) to be in state \((S_2)\), then \((M_1)\) should not be granted a change from \((S_1)\) on every occurrence of \((e_1)\). This is manifested as shown in Figure 4.10a (II level) and demonstrates the difficulty of determining when \((M_1)\) will or will not make a change.

![Diagram](Figure 4.10a State of component \((M_2)\) is a pre-condition on a transition in component \((M_1)\))

The \(\Gamma\) model caters more naturally to capturing such complexity, see figure 4.10b.
4.6.2.4 Other Interactions

The communication modes described above have been identified from relating the object behaviour models to their real-world counterparts, which they (object models) claim to model a great deal more accurately and naturally than other techniques. There are most certainly many other modes of communications that can play a role between the components of an aggregate system. These modes may or may not have respective real-world counterparts. However, it is a point of this research to emphasise that any method or approach for representing the real world accurately must incorporate as many of these communication modes as possible.

4.6.3 The Phi Level of Abstraction - Φ

The previous sections discussed two levels of abstraction which can be applied systematically to result in accurate, well-defined models of the real world in its intricate nature. The application of such techniques can dramatically simplify models. This means that, with these techniques, several levels of behaviour
specification can be built, each is equivalent yet offering differing levels of complexity of the previous one.

Having said that, there are situations when "simpler" is not simple enough, and it is required to press on even further in the pursuit of the simplest behaviour models possible. This is where abstraction comes into the formula. \( \Phi \) is an abstraction, not a transformation. It is therefore distinct from \( \Pi \) and \( \Gamma \).

This implies that \( \Phi \) level in our proposed modelling technique is not the result of applying a mapping to the preceding \( \Gamma \). In fact we propose that, while the \( \Pi \) to \( \Gamma \) transformation is a mapping, \( \Phi \) is essentially a selection of *possibilities* all of which are more or less correct *views of the same thing*. As such, this research does not offer a comprehensive set of rules for generating a \( \Phi \) model; what this thesis introduces, rather, is a set of conditions to be satisfied for testing the consistency of a \( \Phi \) against a \( \Gamma \) or \( \Pi \) model. These can be summarised as follows:

- A \( \Phi \) model must not introduce new functionality not present in the underlying model
- \( \Phi \) models are abstractions of previous models, hence, they may hide functionality, but not *essential* functionality, such as the first half of a chain of transitions

\( \Phi \) models themselves may be further abstracted into \( \Phi \) - \( \Phi \) hierarchies. Because the model at this level is an abstraction of previous models with fewer details showing, the role of the user of the system being modelled is of utmost significance and should be utilised further here. The user(s) of the system can make significant decisions on what behaviour is *less significant* from a particular perspective, and hence may be hidden or abstracted out. Different users will almost always require different perspectives; therefore resulting in a multitude of \( \Phi \) models. We will show some of the possibilities of \( \Phi \) when discussing the case study in chapter 5.
4.6.4 Behaviour Modelling Steps

This section provides a discussion of the steps that are applied when constructing a hierarchical abstraction to capture the dynamic aspects of systems functionality. It is important to define the concepts and to specify the restrictions and assumptions that we impose on the behaviour models this research deals with in order for these ideas to apply correctly.

4.6.4.1 \( \Pi \) to \( \Gamma \) Transformation

The \( \Gamma \) transformation is made up of 2 distinct operators \( \Gamma a \) and \( \Gamma b \) (see appendix b):

- \( \Gamma a \), aggregates two behaviour models and removes only states excluded specifically by the system requirements, for example, (see the example in section 4.7), states “Client: Single” and “Account: Overdrawn” cannot co-exist.

- \( \Gamma b \) covers state reduction and encapsulation, including the removal of unreachable states such as those that are dependent on transient states.

The sets of rules that are used to generate the \( \Gamma a \) abstraction are based on an exhaustive search, listing all the basic scenarios that take place in a \( \Pi \) level and showing the way they are represented in \( \Gamma a \). The rules will apply to a dual component system. The steps to take to convert a \( \Pi \) model to a \( \Gamma a \) model are shown in appendix A. However, composite model components change states as a result of different types of stimuli. These changes will include single component change, simultaneous change and chains of transitions. The components are \( (M_1) \) with two states: \( (M_1S') \) and \( (M_1S'') \), and \( (M_2) \) with two states: \( (M_2S') \) and \( (M_2S'') \). The complete set of rules are shown in appendix E.
4.6.4.2 Π to Γ Transformation in More Complex Systems

The rules in appendix E show how to construct Γa for a given Π scenario. The modelling process can be more difficult when the system includes three or more components.

For addressing the issue of complex systems with three or more components (see figure 4.11), we suggest the following procedure:

First, generate binary Γa models for all the components. Γa is a function that combines the behaviour models for 2 components. States and transitions that are potentially disallowed, for example states that rely for reachability on transient states, are not deleted at this stage. The process (of building Γa models for 2 components at a time) is repeated until all components are in some Γa. Each Γa includes all the behaviour facets that exist between its argument components.

The second stage is to build a Γb behaviour model by removing the states or transitions that become redundant, these include states that rely for reachability on transient ones.

```
SYSTEM
M_1 ▼ ▼ ▼ M_2 M_3
```

*Figure 4.11 A Three-Component Aggregate System*

Since Γa is both commutative and associative, as we shall see later in this chapter, it suffices to define Γa on pairs of components; this can then be extended to more than two components.
For example, we can define
\[ \Gamma a(M_1, M_2, M_3, M_4) = \Gamma a(\Gamma a(M_1, M_2), \Gamma a(M_3, M_4)) \]
or
\[ \Gamma a(M_1, M_2, M_3, M_4) = \Gamma a(\Gamma a(M_1, M_2), M_3), M_4) \]
and so on, the result will always be the same. This is stated by the following two theorems:

**C1 (\( \Gamma a \) is Associative):**

\[ \Gamma a (\Gamma a (M_1, M_2), M_3) \]

is equivalent to

\[ \Gamma a (M_1, \Gamma a (M_2, M_3)) \]

**C2 (\( \Gamma a \) is Commutative):**

\[ \Gamma a (M_1, M_2) \]

is equivalent to \( \Gamma a (M_2, M_1) \)

Where

- \( M_i \): Machine (Component) \( i \)
- \( M_1 \) Equivalent to \( M_2 \): Implies that machines \( M_1 \) and \( M_2 \) are identical except for the state labels. For example, the state \((X,Y)\) is in \( \Gamma a(M_1,M_2) \) if and only if state \((Y,X)\) is in \( \Gamma a(M_2,M_1) \)

These theorems are sufficient for dealing with complex systems containing more than two components. We will prove these theorems based on the set of rules for carrying out the \( \Pi \) to \( \Gamma a \) transformation in appendix E. The proof of theorem C1 is shown in Appendix F.

**4.6.4.3 Proof of Theorem C2**

A \( \Gamma a \) combined state of the form \((M_1S', M_2S'')\) represents the product constituent states \( S' \) from machine \( M_1 \) and \( S'' \) from machine \( M_2 \). The future behaviour
implied by this combined state is independent of the label ordering, so that \((M_1S', M_2S'')\) is effectively identical to \((M_2S'', M_1S')\). This may be generalised to the rest of the statespace of \(\Gamma\). Furthermore, because the process of generating \(\Gamma\) excludes none of the behaviours of the two concerned machines \(M_2\) and \(M_1\), this implies the commutative aspect of \(\Gamma\) is correct.

In the previous section, we have seen that the states of \(\Gamma(a(M_1,M_2))\) may be paired off with those of \(\Gamma(a(M_2,M_1))\). By checking the rules in appendix E, it follows that a transition of the form

\[
\begin{array}{c}
X,Y \\
\text{e/a} \\
X',Y'
\end{array}
\]

exists in \(\Gamma(a(M_1,M_2))\) if and only if a transition of the form

\[
\begin{array}{c}
Y,X \\
\text{e/a} \\
Y',X'
\end{array}
\]

exists in \(\Gamma(a(M_1,M_2))\). Therefore the two \(\Gamma\)’s are structurally identical.

### 4.7 Example – Building Society Shares

In this section we demonstrate the use of the \(\Pi\), \(\Gamma\) and \(\Phi\) abstraction hierarchies for modelling behaviour aspects of a real world system that holds information relating to a persons account held at a building society. The building society is about to embark on a transformation into a bank, which implies that many customers will be offered shares as well as cash incentives. The decisions on what to offer is based on various aspects of the customers’ persons as well as of the type of account they hold with the society and the status of such accounts.
For the purpose of this example we will make the rules as simple as possible with sufficient complexity to require as many of the concepts introduced in the previous sections of this chapter as possible.

### 4.7.1 System Specification - II

The system has two component objects: (Account Holder) and (Account). (Account Holder) can be (Single), (Aged 18+), (Aged 18-) or (Married) etc. Figure 4.12 shows the state model of (Account Holder)

![Figure 4.12 State Model for (Account Holder)](image)

The behaviour of (Account) is shown in Figure 4.13 and implies that account can be (Closed), (Clear), (Credit) or (Debit).

![Figure 4.13 State Model for (Account)](image)

“tD” implies chain transitions (transition caused by transition)
The behaviour models in Figures 4.12 and 4.13 show some of the relationships that exist between our two objects. These are:

A - Event (M) causes a simultaneous transition in both (Account) and (Account Holder).

B - Event (D) in (Account Holder) causes a transition (tD), for example between the states (Single.18+) and (Dead). This transition, in turn, acts as an event on (Account) object thus triggering the transition between states (Credit) and (Closed) here, hence an example of a trigger effect.

C - It is at this stage that other relationships or constraints on the system should be introduced; hence we introduce a third relationship here as follows:

The regulations of the building society disallows unmarried members who are aged less than 18 years from going into state (Debit) on their accounts. Furthermore these regulations imply no activities may be taken on an account once the owner client is deceased.

4.7.2 The Γa Level of Abstraction

This section shows the Γa model that can be derived from the two models in section 4.7.1. This shows every possible combination of states from the two models above, taking into consideration that some state combinations are disallowed (see relationship (c) above). The result is shown in Figure 4.14.
This model shows that the two combination states (Dead-Credit) and (Dead-Debit) are transient states, (labelled by a *); that is, the system does not pause in any of them long enough to carry an activity such as that associated with normal states. The reason is the action that causes an entry to such states also generates an action which is acceptable by these states, hence the second transition labelled (t(D)) in the model above.

In fact we go further to suggest that a more accurate representation, Γb can be shown by connecting all the states that transit to a “transient” state, (Sₖ) directly to the state succeeding (Sₖ) in the transition chain. The result is shown in Figure 4.15.
4.7.3 Φ Level of Abstraction

The next stage of modelling is the Φ abstraction. This, as explained in previous sections of this chapter, is an abstraction that is based on input from business. For the sake of this example, we might assume a semantic requirement for highlighting which accounts are good candidates for a transfer into a “Savings” account and which need to remain (Current).

Savings accounts, for the purpose of this example are accounts which are infrequently used by their owners with the exception of funds deposits. Current accounts on the other hand are frequently accessed for funds withdrawal and deposit.

Semantics dictate a categorisation of the behaviour model states according to activity. The rules as related to the state model imply the following:
- No activity – Close account
  States (Dead-Closed)

- Low Activity – Good candidate for a “Savings Account”
  (Single 18- Debit) and (Married 18- Credit)

- High Activity – Only suitable for “Current Account”
  (Single 18- Credit) and (Married 18+ Credit) and (Married 18+ Debit)

- Moderate Activity – Further Monitoring
  All other states

The resulting model is shown in Figure 4.16, superstates are marked by a double border.

![Figure 4.16 The $\Phi$ Model based on $\Gamma$ of Figure 4.15](image)

This model is derived from the $\Gamma$ model in Figure 4.15 as follows.

- Group all the states of one activity category into one superstate
- All the transitions to and from a state will now apply to the superstate that contains it
• Any state grouping/abstraction that generates non-deterministic behaviour is disallowed. An example is shown in Figure 4.17

![Figure 4.17 Non-Determinism](image)

Figure 4.17 Non-Determinism

• State groupings that generate behaviour which cannot be reproduced in $\Gamma$ are disallowed

4.8 Variations and Exceptions

4.8.1 Theme Variations

In the above sections we have proposed that for modelling the behaviour of a system of several components, say $(C_1, C_2, C_3, C_4)$, the steps are:

• Aggregate behaviour models for $C_1$ and $C_2$ into $\Gamma_L$ and
• Aggregate behaviour models for $C_3$ and $C_4$ into $\Gamma_R$ and
• Generate the full $\Gamma$ by aggregating $\Gamma_L$ and $\Gamma_R$
• Finally, $\Phi$ may be abstracted from $\Gamma$. 
We now consider the effect of applying a $\Phi$ abstraction to one of the components in the $\Pi$ level, before generating the first of the $\Gamma$ models. For illustration of the ideas, we shall use a system of 2 components $C_1$ and $C_2$. The behaviour models we will generate are:

1. The $\Pi$ model for $C_1$ and $C_2$,
2. $\Gamma(C_1, C_2)$,
3. An abstraction $\Phi'$ of $C_2$ as $\Phi'(C_2)$,
4. $\Gamma' (\Phi', C_2)$

We will end this section with a comparison of $\Gamma$ from step 2 with $\Gamma'$ from step 4. We anticipate that $\Gamma'$ from step 4 will constitute an abstraction, $\Phi$, of $\Gamma$ in step 2.

To start with, figure 4.18 shows the $\Pi$ model for components $C_1$ and $C_2$.

![Figure 4.18 The $\Pi$ Model for $C_1$ and $C_2$](image)

The transition $t_{e_2}$ between $S_2$ and $S_0$ in $C_2$ implies that this transition (a chain transition) will take place as a result of any transition in $C_1$ caused by $e_2$. The $\Gamma$ model for this $\Pi$ model consists of 9 states and is shown in figure 4.19.
The third step is to generate the abstraction of the behaviour model for \( C_2 \), we will call this abstraction \( \Phi' \). This is based on grouping states \( S_0 \) and \( S_1 \) and is shown in figure 4.20.

The fourth step is to generate a \( \Gamma \) model of \( C_1 \) and \( \Phi' \). This will be referred to as \( \Gamma' \) and is shown in figure 4.21.
In conclusion, and by comparing the $\Gamma$ model (figure 4.19) with the $\Gamma'$ (figure 4.21) model, it can be seen that the latter is an abstraction of the first, albeit a strong abstraction that hides one third of the whole behaviour model.

4.8.2 Limitations When Building Complex $\Gamma$ Using a Binary Approach – Why Two $\Gamma$ Models

In this section we consider the failure conditions and conditions required for success of the methodology. There may be a real world scenario for which the proposed approach is erroneous. This scenario is manifested as follows. (see Figure 4.22 and Appendix B),

Figure 4.21 An Aggregation, $\Gamma'$ of $\Phi'$
Figure 4.22 Failure Conditions

- If we abstract only one of the two $\Gamma$ models to result in $\Phi_1(\Gamma_1)$ and leave $\Gamma_2(C_3,C_4)$ as is,
- If $\Gamma_2(C_3,C_4)$ were dependent (for a subset of its behaviour) on an aspect of the behaviour of $\Gamma_1(C_1,C_2)$, say a transition $X$ in $\Gamma_2(C_3,C_4)$ caused by a transition $Y$ in $\Gamma_1$ and
- If transition $Y$ were hidden in the transformation from $\Gamma_1$ to $\Phi_1$, then

These conditions would imply, in at least one of the hierarchies of the systems behaviour model ($\Pi$, $\Gamma$ or $\Phi$), that the subset of the behaviour of the system represented as $\Gamma_2(C_3,C_4)$ will not hold. This is because it may contain transitions that are caused by events which are not available (hidden/abstracted in $\Phi_1(\Gamma_1)$) at that level of model.

There are a few more conditions /scenarios which can result in an incorrect behaviour model of the system. These are to do with the extent of the $\Pi$ to $\Gamma$ transformation when dealing with binary numbers of components.

In the previous section, we mentioned that, for systems with a large number of components, the method for building a $\Gamma$ model is to build sub models ($\Gamma''$)
comprising two components. These are then treated as component behaviour models and are rebuilt into a new (Γ) etc.

When building a Γ behaviour model for a system of 3 or more component objects, this is done using the idea of building a series sub-Γ models (Γ", Γ"", etc.). In this case, it is quite conceivable that an aspect of the behaviour of the system will be lost in process of generating some of these sub-Γ models. This takes place when a state, say S₀S₉, is removed in the build-up of, say Γ", due to that state being unreachable. It is quite conceivable that there exists a transition or condition within the other parts of the system, Γ"", or Γ""" that can render this state S₀S₉ reachable.

This seemingly puts the C1 theorem into questionable state. An illustration of this problem and how to tackle it is provided in the case study chapter that follows this one.

A possible solution for this problem is to divide the process of Γ generation into two stages as mentioned above. In stage 1 (Γ¹), the level includes all states that result from the product of the states in Π, minus all states that are redundant by rules which prevent state co-existence. In the second stage (Γ²), which is only applicable once all the components are within the Γ level, here all other types of state elimination (for example, non-reachable) can be removed.

An alternative approach would be to gather all components into a single Γ, and not to carry out a binary behaviour model construction. The single Γ includes all the conditions and inter-component interactions in one place. Since all aspects of the system are onboard, it is possible to remove unreachable states as well as ones that are disallowed by state co-existence rules. The disadvantage here is that is difficult to deal with the explosion in the number of states generated by the first step (the product of states from all components).
This difficulty can be tackled by representing the resulting state model using *state transition tables* which show the same information as in a state transition diagram in tabular format. This approach will be illustrated in the following chapter. These tables offer an advantage over graphical models in that they are easier to traverse *systematically*. Making easier to deal with them from the point of view of programming the process using a case tool that can automate this function.

In conclusion, and in relation to the associative feature of the work, we can say that there are exclusions to this particular feature of the method. This is particular to the real world scenarios where the behaviour in one set of components is more dependent on the other components in the system than in *mainstream* scenarios we have seen above. Nonetheless with the proposed solution to these exceptional conditions, it is possible (albeit more difficult) to achieve the purpose of building a reduced behaviour model, whichever components we start with and regardless of how complex the components interactions are.

### 4.9 Summary

In this chapter, we have introduced a method for representing dynamic features of objects which form an aggregation hierarchy. This method is not intended as a complete methodology to analyse, design, prototype and code complex OO systems; rather it is offered as a way of enhancing the semantic expressiveness of popular methodologies like [Rumbaugh et al 91], [Booch 91], [Rumbaugh 95], [Yourden 82], [Meyer 88], [Shlaer 88], [Booch 97] and [Coleman 94].

In the following chapter, we will demonstrate this approach through a case study that is specifically designed to reveal the salient features of complex systems where the method in this chapter can play a role. The main objective is still to achieve systems that closer, more correctly and more accurately represent the real world scenarios they relate to.
The case study will be a good platform for showing the features as well as the problems facing the designer of a behaviour model for a complex reactive system. On specific idea that was mentioned in this chapter but never completed is related to one of the conditions that cause this methodology to result in inaccurate, in fact, incorrect behaviour specifications, this will be addressed and in the case study chapter.
CHAPTER 5

CASE STUDY
5.1 Introduction

The previous chapters of this thesis have put emphasis on systems which are both complex in nature on the one hand, and have structure as well as behaviour on the other. In chapters 2 and 3, we looked at the main problem that will be tackled by this research, i.e. how to represent the dynamic aspects of systems that are made up of multiple objects or components, especially as these components are reactive in nature. Chapter 4 went on to describe the proposed enhancements to some of those methodologies. Furthermore, chapter 4 identified additional complexities and semantic features in real world scenarios that need to be addressed. Those semantic features are, in our opinion, fundamental to the success of any systems analysis and design methodology.

This chapter acts, primarily, as a demonstration of the workability of the methodology that was proposed in the previous chapter. A real-world example is proposed in the form of a case study. The chapter is organised as follows:

Section 5.2 looks at the general notation used throughout this chapter. The notation relates to the description of the behaviour of the system components in terms of states, events, and semantic conditions that govern the aspects of the component interactions, such as when can a component respond to a certain event and when must it not be allowed to do so. This section also list the components and the functions that take place between (and within) those components. This covers individual details of each component in terms of states and functionality, in addition to the different events that take place and how the system reacts to them.

Section 5.3 starts the process of constructing the behavioural model of the system (from the behaviour models of the components) in a hierarchical fashion. Here we build the behaviour model of the system in stages as described in the previous chapter. The latter parts of this section will focus on exception handling, special conditions and limitations of the case study and the methodology, although many
exception scenarios will be dealt with as and when they arise within the build-up of the case study.

Section 5.4 is a summary of the chapter and an introduction to the final chapter of this thesis.

It is important to bear in mind that, in all sections of this chapter (and indeed the thesis), the focal point of this research is the behavioural aspects of the system. Therefore, little attention is given to the static aspects that have no direct bearing on the dynamic or temporal behaviour and functionality of the system.

5.2 Notation and System Description

The system in this case study is an aggregate [Smith & Smith 77], [Rumbaugh et al 91] system, i.e. it is made up of a group of constituent objects which are related by means of being part-of a new object – the system. The behavioural description of each component is made up of the following: States that the component goes through, events that each state accepts, transitions with which each state responds to those accepted events, activities within states, and actions along transitions between states. For abbreviation purposes it is stated in this thesis that the state responds to an event. The author acknowledges that it is the object that responds to the event and changes from the state and so on.

As described in chapter 3, there are many approaches to representing this kind of information about systems, some formal, some textual, some visual. The method of choice for showing the behaviour (functionality) aspects of our Pensions and Investments system is the visual one; there are two versions of this method: the state transition diagram based one, and the state transition table based one as will be shown below.
5.2.1 The Structure of our System

This chapter uses an example which is derived from the pensions and investment domain. The description of the structure of the system in relation to its components is shown in Figure 5.1. This shows the system is made up of four components (Member, Scheme, Contributions and Returns), this also clearly shows the hierarchical nature of the aggregate system, wherein the system is at a different plane to its components. The reader is referred back to examine chapters 2 and 3 for further explanation of the notions of encapsulation and abstraction and hierarchical description of object behaviour.

![Figure 5.1 Aggregate System with 4 Components](image)

The behaviour of the system is a derived from the behaviour(s) of its components, which are as described in the sections that follow.

5.2.2 The Objects in the System - Components

This section discusses the constituent objects of the system, with the main emphasis being on the dynamic aspects of these components (states, transitions, concurrency, chains etc.). We begin with Member.
5.2.2.1 Member

Member can go through the states: “Employed Full-Time” (which is the initial state), “Employed Part-Time”, and “Unemployed”. The possible events and transitions from each of those states are exposed in the behaviour model of Member. The state diagram based behaviour model of Member is shown in Figure 5.2; this shows the states of Member as nodes labelled with the first letter of the object name “M” and the abbreviation of the state “Member Employed Full Time” is therefore shown as “MEF”, etc.

![Figure 5.2 Behaviour of Member](image)

The diagram shows transitions as arcs; each labelled with the event that causes it (we use arbitrary event labels as the meaning of the event is implied by the source and destination states). Table 5.1 shows the equivalent of this model represented as a state transition table. It is worth noting that in state transition table notation, the first column of the table lists the source states of the behaviour model, and the top row of the table shows the destination states.

The initial state is identified as the first cell in the source states collection, this is always the same as the first cell in the destination states collection. Initial states are marked in the state transition diagram notation with an unlabelled transition with no source state. Source states are mapped to destination states using the list of possible events at the intersection points/cells. For example, if state $S_0$ on the left is mapped to state $S_1$ in the first row via the number 4, then this implies that event $e_4$ causes state $S_0$ to change to state $S_1$. 
State transition tables also cater for actions which can be shown with the event that identifies the transition, say 4/3 to indicate a transition caused by event $e_4$ and resulting in action 3.

The notion of final state is also permitted in this modelling approach as an implicit concept. Any state which has no transitions from it is, by definition, a final state. Hence an object can have only one initial state but several final states.

<table>
<thead>
<tr>
<th></th>
<th>MEF</th>
<th>MEP</th>
<th>MUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEF</td>
<td></td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>MEP</td>
<td>1</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>MUE</td>
<td>3</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.1 Behaviour of Member

5.2.2.2 Scheme

The behaviour of Scheme is based on the states: In-Profit, Varying and In-Loss, Scheme In-Profit is the initial state. Figure 5.3 shows the behaviour model for Scheme as a state transition diagram, Table 5.2 shows the equivalent state transition table notation.
It can be seen at this stage that the two objects Member and Scheme share some common events (Event 4). This implies that the two objects can respond to this event in a concurrent fashion (see the previous chapter for a look at concurrency in aggregation). Furthermore, it can be seen that the same event (Event 4) can cause more than one transition within the same component (Scheme in this case). More interactions and event sharing will feature in this case study as the rest of the components are introduced.

<table>
<thead>
<tr>
<th></th>
<th>SIP</th>
<th>SIL</th>
<th>SV</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIP</td>
<td>4</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>SIL</td>
<td>6</td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>SV</td>
<td>8</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.2 Behaviour of Scheme

5.2.2.3 Contributions

The states of Contributions are as follows.

Contributions Pledged: (i.e. the Member object has made some written commitment to start contributing by Direct Debit or other method).

Contributions Credited: (When the amount due has been cleared from, say the Member’s bank account to the Investment Company’s bank account), and Contributions Withdrawn: (If, for example, Member is in financial difficulty – Unemployed for example – he/she can withdraw their contributions from the Scheme.)
Figure 5.4 and Table 5.3 show the state diagram model and the state transition table model of the Contributions component respectively.

![State Diagram](image)

**Figure 5.4 Behaviour of Contributions**

<table>
<thead>
<tr>
<th></th>
<th>CP</th>
<th>CW</th>
<th>CC</th>
</tr>
</thead>
<tbody>
<tr>
<td>CP</td>
<td></td>
<td>t_e4</td>
<td>10</td>
</tr>
<tr>
<td>CW</td>
<td>12</td>
<td></td>
<td>t_e1</td>
</tr>
<tr>
<td>CC</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 5.3 Behaviour of Contributions**

It can be seen from the two figures above that Contributions Credited is a final state of this object, hence if withdrawals are required, then they would need to be applied prior to this state. Further, this component reveals a new kind of event, labelled as “t_e”. This event implies that the current transition is in fact caused by a stimulus from one of the components within the system, and not (as is frequently the case) from an outside, real-world, stimulus or event. See the section on Chains of Transitions in the previous chapter for further details of this phenomenon.

Within the context of our Contributions object, t_e implies that state CW (Contributions Withdrawn) automatically changes to CC (Contributions Credited)
as soon as other objects in the system respond to event or external (real world) stimulus \( e_1 \). One particular component that responds to stimulus \( e_1 \) is the Member object, which changes from MEP to MEF as a result.

This process is a spontaneous one and the delay between the two transitions (the one caused by event \( e_1 \) in Member and the Contributions change to CC) is negligible, hence the notion of chains of transitions. Put simply, this particular interaction between Member and Contributions means that as soon as Member's status changes from Part-Time to Full-Time Employed, his/her contributions are immediately confirmed to the investment company's account and, hence, cannot be withdrawn thereafter.

There are a few more interactions between the components of our system, and these are discussed in some detail at the end of this section.

### 5.2.2.4 Returns

The final component in our system, and by far the most basic one is the Returns object. This represents the income or profits made by the investment scheme and paid to the Member's account. The possible states for Returns are as follows.

- **Returns Pledged**: Wherein the investment company writes to the Member to advise of the availability of a certain sum at his/her disposal, and
- **Returns Credited**: In response to a request from the Member, the investment company credits the funds to his/her account. The behaviour model of Returns is shown in Figure 5.5 (state diagram) and Table 5.4 (state transition table).

![Figure 5.5 Behaviour of Returns](image)

It can be seen that state Returns Credited is a final state of this object.
Table 5.4 Behaviour of Returns

Table 5.5 summarises all the possible states in the system:

<table>
<thead>
<tr>
<th>Object</th>
<th>States</th>
<th>Abbreviation in Case Study</th>
</tr>
</thead>
<tbody>
<tr>
<td>Member</td>
<td>Employed Full-Time</td>
<td>MEF**</td>
</tr>
<tr>
<td></td>
<td>Employed Part-Time</td>
<td>MEP</td>
</tr>
<tr>
<td></td>
<td>UnEmployed</td>
<td>MUE</td>
</tr>
<tr>
<td>Scheme</td>
<td>In-Profit</td>
<td>SIP**</td>
</tr>
<tr>
<td></td>
<td>In-Loss</td>
<td>SIL</td>
</tr>
<tr>
<td></td>
<td>Varying/Fluctuating</td>
<td>SV</td>
</tr>
<tr>
<td>Contributions</td>
<td>Pledged</td>
<td>CP**</td>
</tr>
<tr>
<td></td>
<td>Withdrawn</td>
<td>CW</td>
</tr>
<tr>
<td></td>
<td>Credited</td>
<td>CC</td>
</tr>
<tr>
<td>Returns</td>
<td>Pledged</td>
<td>RP**</td>
</tr>
<tr>
<td></td>
<td>Credited</td>
<td>RC</td>
</tr>
</tbody>
</table>

Table 5.5 All System States (** Initial State)

As the names of all the objects in the system start with distinct letters, it is safe to assume for the purpose of this chapter that, in the name of the state, the first letter is an indication of the owner object.

5.2.3 System Interaction Constraints

This section discusses the rules and principles that govern the inter-component behaviour in the system above. These rules pertain to concepts such as chains of transitions, conditional transitions, concurrency in behaviour, etc. Therefore, they are roughly divided into these categories.
5.2.3.1 State Exclusion Constraints

These rules govern which states from component X can co-exist with states from component Y, etc.

- Member state MEP and Contributions state CC cannot co-exist, independently or as part of another superstate.
- Scheme state SIP and Returns state RC cannot co-exist, independently or as part of another compound state.
- Member state MEF and Returns state RP cannot co-exist, independently or as part of another superstate.
- Contributions state CW and Returns state RC cannot co-exist, independently or as part of another superstate.

5.2.3.2 Transition Chains

These rules dictate which transitions are carried further at the destination state because another component received the action that was generated by the end of the first transition.

- Any Transition caused by event 4, can cause a response in Contributions.
- Any Transition caused by event 1, can cause a response in Contributions.

5.2.3.3 Concurrency Constraints

These constraints relate to the ability of more than one object in the system to behave and respond to external stimuli independently of other objects.

- Event 4 is applicable to both Member and Scheme.
- Event 10 is applicable to both Contributions and Returns.
5.2.3.4 Event Exclusion Constraints

These rules make it impossible for some states to accept certain events.

- Compound state MEP,CW (or any superstate of it) cannot accept event 3
- Event 7 not applicable to SIP if part of a Compound state containing Member state MEP
- Event 12 not applicable to Member state MUE, or any Compound state it is part of
- Event 10 not applicable to Member state MUE, or any Compound state it is part of
- Event 10 not applicable to Scheme state SV, or any Compound state it is part of

5.2.3.5 Other Constraints

<table>
<thead>
<tr>
<th>Interaction Rule</th>
<th>&quot;R&quot; Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event 4 is applicable to both Member and Scheme</td>
<td>R1</td>
</tr>
<tr>
<td>Event 10 is applicable to both Contributions and Returns</td>
<td>R2</td>
</tr>
<tr>
<td>Any Transition caused by event 4, can cause a response in Contributions</td>
<td>R3</td>
</tr>
<tr>
<td>Any Transition caused by event 1, can cause a response in Contributions</td>
<td>R4</td>
</tr>
<tr>
<td>Event 12 not applicable to Member state MUE, or any superstate it is part of</td>
<td>R5</td>
</tr>
<tr>
<td>Event 10 not applicable to Member state MUE, or any superstate it is part of</td>
<td>R6</td>
</tr>
<tr>
<td>Event 10 not applicable to Scheme state SV, or any superstate it is part of</td>
<td>R7</td>
</tr>
<tr>
<td>SuperState MEP,CW (or any superstate of it) cannot accept event 3</td>
<td>R8</td>
</tr>
<tr>
<td>MEP and CC cannot co-exist, independently or as part of another superstate</td>
<td>R9</td>
</tr>
<tr>
<td>SIP and RC cannot co-exist, independently or as part of another superstate</td>
<td>R10</td>
</tr>
<tr>
<td>MEF and RP cannot co-exist, independently or as part of another superstate</td>
<td>R11</td>
</tr>
<tr>
<td>CW and RC cannot co-exist, independently or as part of another superstate</td>
<td>R12</td>
</tr>
<tr>
<td>Event 7 not applicable to SIP if part of a superstate containing MEP</td>
<td>R13</td>
</tr>
<tr>
<td>Combined [MEF,SIP,CP,RP] is immune to all rules – Initial State</td>
<td>R14</td>
</tr>
</tbody>
</table>

Table 5.6 Interaction Constraints
A miscellaneous set of interaction constraints that affect the system’s behaviour.

- Compound state [MEF,SIP,CP,RP] is immune to all rules - Initial State

Table 5.6 summarises these rules and provides each with an “R” number, which will be used in the subsequent sections of this chapter.

5.3 Building The Combined Behaviour Model (I') of the System

This section covers the application of the proposed methodology (see previous chapter) for the generation of a single behaviour model for the system components combined. The basis for combining the behaviours of our system’s components is that they interact together to yield the overall functionality of the system, hence it makes sense to have the ability to look at the behaviour of the system as an abstract object on its own, as a whole unit.

There are several courses that can be followed to generate the combined behaviour model (the I' model). These are listed below. In the following sections, it is likely that component behaviour will be used interchangeably with component; the context of this whole chapter (the whole thesis in fact) is the behaviour of objects:

- Combine all four component models to get one Cartesian product
- Combine the two components (chosen randomly), then add a third component, and finally add the last component behaviour
- Combine components in pairs (binary) and then combine the results (binary again) to obtain the final full Cartesian product

The option taken in this section will be to try as many combinations as possible both to illustrate the methodology and also to support the workability argument.
that this thesis presents. Our first step is to generate the (binary) behaviour models of the components above.

5.3.1 Generating Behaviour Model as a Combination of ‘Binary’ Behaviour Models From Component Groups - Ta

In this section we will build the behaviour models of Member and Scheme on the one hand, and Scheme and Returns on the other. We start with Member and Scheme. Prior to this however, a word about combining transitions and states from two behaviour models.

While the combinations of the states from two behaviour models is as simple as generating the product space of the states from the two models, the case with transitions is slightly more difficult. When combining transitions, they by and large tend to conform to one of the following possibilities demonstrated with arbitrary examples:

a) Single Component Change

When the states of object A are to be combined with those of object B, the first possibility is that object A changes from state $AS_1$ to $AS_2$ via a transition caused by event $e_1$, and object B remains in state $BS_1$ and does not respond. This implies that the combination state $[AS_1, BS_1]$ will change to $[AS_1, BS_1]$ via event $e_1$. This is illustrated in Figure 5.6.

![Figure 5.6 Behaviour Pattern (single component response)](image)
b) Two Components Change Concurrently

The second possibility is that object A changes from state \( AS_1 \) to \( AS_2 \) via a transition caused by event \( e_x \), and object B in state \( BS_1 \) responds to event \( e_y \) and changes to state \( BS_2 \). This implies that the combination state \([AS_1, BS_1]\) will change to \([AS_2, BS_2]\) via event \( e_x \). This is illustrated in figure 5.7. This is the basic idea in the notion of concurrency.

\[\begin{align*}
&AS_1 \xrightarrow{e_x} AS_2 \\
&BS_1 \xrightarrow{e_y} BS_2
\end{align*}\]

**Figure 5.7 Behaviour Pattern (multiple component response)**

c) Two Components Change Independently

The third possibility is that object A in state \( AS_1 \) changes to \( AS_2 \) via a transition caused by event \( e_x \), and object B in state \( BS_1 \) responds to a distinct event \( e_y \) that causes a transition to state \( BS_2 \). This implies that \([AS_1, BS_1]\) will change to \([AS_2, BS_2]\) via event \( e_x \) and to \([AS_1, BS_2]\) via event \( e_y \) as shown in figure 5.8.

\[\begin{align*}
&AS_1 \xrightarrow{e_x} AS_2 \\
&BS_1 \xrightarrow{e_y} BS_2
\end{align*}\]

**Figure 5.8 Behaviour Pattern (multiple component response)**
**d) Chains of Transitions**

The final option is that a transition in the first component causes a transition in the second. This is the idea in the notion of Chains of transitions. The previous chapter discusses these two concepts in more detail. Figure 5.9 illustrates the idea in Chains of transitions.

![Diagram of Chains of Transitions]

**Figure 5.9 Behaviour Pattern (Chains)**

In all the patterns above, it can be seen that the basic idea is that wherever a state is located in terms of joining states from other components in the system, its transitions always follow. It is important to add that all the patterns above show what happens to transitions when combining states and not what states result when combining two components. The answer to the states question is the Cartesian product of states (see chapter 4 for further information of this subject).

**5.3.1.1 Behaviour Model for the ‘Binary’ Combination of Member and Scheme - Γα**

In this section we will generate half the behaviour model of our system, that of the two components Member and Scheme.
• **Step 1. List States - Γα**

This first step in this procedure is to list all the possible combined states for Member and Scheme combined, this is a Cartesian product, or a pairing of each state from Member with each state from Scheme. The list of states is therefore:

MEF,SIP**; MEF,SIL; MEF,SV;
MEP,SIP; MEF,SIL; MEF,SV;
MEP,SIP; MEF,SIL; MEF,SV

** indicates initial superstate (product state).

• **Step 2. Incorporate Transitions - Γα**

Each combined state can, generally, change via two transitions, one from each of the components that the combined state relates to. This can be done using a state transition diagram or a state transition table notation, for the purpose of this exercise we will opt for the state transition table option.

<table>
<thead>
<tr>
<th></th>
<th>MEF,SIP</th>
<th>MEF,SIL</th>
<th>MEF,SV</th>
<th>MEP,SIP</th>
<th>MEP,SIL</th>
<th>MEP,SV</th>
<th>MUE,SIP</th>
<th>MUE,SIL</th>
<th>MUE,SV</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEF,SIP</td>
<td>4?</td>
<td>7</td>
<td>2</td>
<td>4?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEF,SIL</td>
<td>6</td>
<td>9</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEF,SV</td>
<td>8</td>
<td>4?</td>
<td></td>
<td>2</td>
<td>4?</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEP,SIP</td>
<td>1</td>
<td>4?</td>
<td>7</td>
<td>4?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEP,SIL</td>
<td>1</td>
<td>6</td>
<td>9</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEP,SV</td>
<td>1</td>
<td>8</td>
<td>4?</td>
<td>4?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MUE,SIP</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MUE,SIL</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MUE,SV</td>
<td>3</td>
<td>5</td>
<td>8</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.7 Behaviour Model for Member.Scheme
(interaction rules not applied)
The reason for this is that, in complex behaviour models, it can be difficult to cope graphically with a substantial number of states and transitions, especially as the latter start to cross the boundaries of states in order to reach from the source state to the destination state.

For this step, the state transition table lists the source states in the first column and the destination states in the first row. Note that, as we are not sure which states are going to be reached and which ones are going to end up unreachable, it is important to include the entire state space of the sub-system we are dealing with. The initial behaviour model is shown in Table 5.7.

**Notes:**

a) In this table, the states on the left are the source states, the states in the top row are the destination states (identical set), and the numbers on the path are the transition labels or the event names/numbers that caused the state change. For example, state [MUE,SIL] changes to state [MEF,SIL] via event 3 as shown in the last but one row of Table 5.7. This is consistent with the first behaviour pattern above, where only one component of the two responds to certain events.

b) "4?" Implies that there are concurrent transitions caused by event 4. Those will be combined to show the final destinations superstate, for example [MEF,SIP] should change to [MUE,SIL] as a combined/concurrent response to event 4. However, this combination of transitions will not be applied until an exhaustive search and application of the interaction rules has been employed. The reason behind this delay in combining transitions is that there could be a rule that voids one of the two concurrent transitions, hence it would result in an incomplete, in fact, incorrect real-world representation. Further discussion of this phenomenon will be introduced in a later section.
• Step 3. Apply Interaction Rules (no chains or concurrency) - $\Gamma_a$

The next step is to search through the rules (Table 5.6) and apply them to the behaviour model. We will exclude any rules to do with Chains or Concurrency at this stage, see the description of $\Gamma_a$ in chapter 4. Interaction rule “R13” dictates that superstate [MEP,SIP] should not accept event 7 to change to superstate [MEP,SIV]. Apart from this rule all other behaviour aspects of Member.Scheme are potentially sound. Further, we will mark the potential final destination of all concurrent transitions (of the form $X?$) with square brackets “[X]”, this will help to recognise this category of transitions in the following steps. Table 5.8 shows the new behaviour model for Member.Scheme in its latest form.

<table>
<thead>
<tr>
<th></th>
<th>MEF,SIP</th>
<th>MEF,SIL</th>
<th>MEF,SV</th>
<th>MEP,SIP</th>
<th>MEP,SIL</th>
<th>MEP,SV</th>
<th>MUE,SIP</th>
<th>MUE,SIL</th>
<th>MUE,SV</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEF,SIL</td>
<td>6</td>
<td>9</td>
<td>2</td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEP,SIP</td>
<td>1</td>
<td></td>
<td>4?</td>
<td>7</td>
<td></td>
<td>4?</td>
<td>[4]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEP,SIL</td>
<td>1</td>
<td></td>
<td>6</td>
<td>9</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEP,SV</td>
<td></td>
<td>1</td>
<td>8</td>
<td>4?</td>
<td></td>
<td></td>
<td>[4]</td>
<td>4?</td>
<td></td>
</tr>
<tr>
<td>MUE,SIP</td>
<td>3</td>
<td></td>
<td>5</td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>MUE,SIL</td>
<td>3</td>
<td></td>
<td>5</td>
<td></td>
<td></td>
<td>6</td>
<td></td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>MUE,SV</td>
<td>3</td>
<td></td>
<td>5</td>
<td></td>
<td></td>
<td>8</td>
<td></td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.8 Behaviour Model for Member.Scheme
(some interactions applied)
5.3.1.2 Behaviour Model for the 'Binary' Combination of Contributions and Returns - \( \Gamma_a \)

In this section we will construct the behaviour model for Contributions and Returns, in a similar approach to the previous behaviour model.

- **Step 1. List States - \( \Gamma_a \)**

The possible states that a system comprising Contributions and Returns will go through in its lifecycle includes \([CP,RP**];[CW,RP]; [CC,RP];[CP,RC]; [CW,RC]; [CC,RC]** \( \ast \) indicates initial superstate.

- **Step 2. Add Transitions - \( \Gamma_a \)**

Table 5.9 shows the initial behaviour model with only the states and their transitions included.

<table>
<thead>
<tr>
<th></th>
<th>CP,RP</th>
<th>CW,RP</th>
<th>CC,RP</th>
<th>CP,RC</th>
<th>CW,RC</th>
<th>CC,RC</th>
</tr>
</thead>
<tbody>
<tr>
<td>CP,RP</td>
<td></td>
<td>Te4*</td>
<td>1?</td>
<td>1?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CW,RP</td>
<td>12</td>
<td></td>
<td>Te1*</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CC,RP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>CP,RC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Te4*</td>
</tr>
<tr>
<td>CW,RC</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Te1*</td>
</tr>
<tr>
<td>CC,RC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 5.9 Behaviour Model for Contributions,Returns**

*(interaction rules not applied)*

**Notes:**

a) Once again we can see that there are some superstates which seem to have two destination superstates reachable via the same event number. This is a case of concurrency where two objects react to the same event at the same time. For
the moment, this will be marked by the "X?" which indicates that this transition is likely to change its destination state as it gets combined with another identical transition on the same row. The combined transitions destination state is reachable via an "[X]".

b) Furthermore, there is a new notation shown in the table which relates to transition chains and marked with a "*". This is consistent with the behaviour pattern illustrated previously which shows how chains of transitions work. In this case $Te_4*$ in the first row for instance, indicates that superstate CP,RP will change to superstate CW,RP as a response to any transition in the system caused by event $e_4$. The interesting aspect of this phenomenon will become apparent as the behaviour model for the entire system is built, where the notion of transient states (see Chapter 4) is introduced. This takes place when a state at the second half of a transition chain is reached only via the first half of that transition chain.

- Step 3. Apply Interaction Rules - $\Gamma_b$

At this stage, we can see that Rules "R12" implies that the superstate CW,RC is not permitted. This state is therefore removed from Table 5.9; Table 5.10 illustrates this idea. Notice we are not replacing the "X?" or the "*" transitions until the full behaviour model for the entire system is built.

<table>
<thead>
<tr>
<th></th>
<th>CP,RP</th>
<th>CW,RP</th>
<th>CC,RP</th>
<th>CP,RC</th>
<th>CC,RC</th>
</tr>
</thead>
<tbody>
<tr>
<td>CP,RP</td>
<td>T$e_4*$</td>
<td>10?</td>
<td>10?</td>
<td>[10]</td>
<td></td>
</tr>
<tr>
<td>CW,RP</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CC,RP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>CP,RC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>CC,RC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.10 Behaviour Model for Contributions, Returns
(some interaction rules applied)
5.3.1.3 Combine ‘Binary’ Models – Complete System - Γb

In this section we will use the result of the binary behaviour models above to generate the behaviour model for our Pensions and Investment System. The main idea here is that, in a similar fashion to the binary models above, we will treat those models as though they were two new components that form an entirely new system.

• Step 1. List States

The possible states in this behaviour model are the result of the product of the two state spaces in our two binary models above, this implies we have \((9 \times 5) = 45\) states. These are shown in Table 5.11.

<table>
<thead>
<tr>
<th>MEF, SIP, CP, RP**</th>
<th>MEF, SIP, CW, RP</th>
<th>MEF, SIP, CC, RP</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEF, SIP, CP, RC</td>
<td>MEF, SIP, CC, RC</td>
<td>MEF, SIP, CP, RP</td>
</tr>
<tr>
<td>MEF, SIL, CW, RP</td>
<td>MEF, SIL, CC, RP</td>
<td>MEF, SIP, CP, RC</td>
</tr>
<tr>
<td>MEF, SIL, CC, RC</td>
<td>MEF, SV, CP, RP</td>
<td>MEF, SV, CW, RP</td>
</tr>
<tr>
<td>MEF, SV, CC, RP</td>
<td>MEF, SV, CP, RC</td>
<td>MEF, SV, CC, RC</td>
</tr>
<tr>
<td>MEP, SIP, CP, RP</td>
<td>MEP, SIP, CW, RP</td>
<td>MEP, SIP, CC, RP</td>
</tr>
<tr>
<td>MEP, SIP, CP, RC</td>
<td>MEP, SIP, CC, RC</td>
<td>MEP, SIP, CP, RP</td>
</tr>
<tr>
<td>MEP, SIL, CW, RP</td>
<td>MEP, SIL, CC, RP</td>
<td>MEP, SIL, CP, RC</td>
</tr>
<tr>
<td>MEP, SIL, CC, RC</td>
<td>MEP, SV, CP, RP</td>
<td>MEP, SV, CW, RP</td>
</tr>
<tr>
<td>MEP, SV, CC, RP</td>
<td>MEP, SV, CP, RC</td>
<td>MEP, SV, CC, RC</td>
</tr>
<tr>
<td>MUE, SIP, CP, RP</td>
<td>MUE, SIP, CW, RP</td>
<td>MUE, SIP, CC, RP</td>
</tr>
<tr>
<td>MUE, SIP, CP, RC</td>
<td>MUE, SIP, CC, RC</td>
<td>MUE, SIL, CP, RP</td>
</tr>
<tr>
<td>MUE, SIL, CW, RP</td>
<td>MUE, SIL, CC, RP</td>
<td>MUE, SIL, CP, RC</td>
</tr>
<tr>
<td>MUE, SIL, CC, RC</td>
<td>MUE, SV, CP, RP</td>
<td>MUE, SV, CW, RP</td>
</tr>
<tr>
<td>MUE, SV, CC, RP</td>
<td>MUE, SV, CP, RC</td>
<td>MUE, SV, CC, RC</td>
</tr>
</tbody>
</table>

Table 5.11 State Space for System Behaviour – no interaction rules

(*** indicates initial state)
- **Step 2. Apply Interaction Rules - \( \Gamma_b \)**

<table>
<thead>
<tr>
<th><strong>State</strong></th>
<th><strong>Rule</strong></th>
<th><strong>State</strong></th>
<th><strong>Rule</strong></th>
<th><strong>State</strong></th>
<th><strong>Rule</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>MEF,SIP,CP,RP**</td>
<td></td>
<td>MEF,SIP,CW,RP</td>
<td></td>
<td>MEF,SIP,CC,RP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;R11&quot;</td>
<td></td>
<td>&quot;R11&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEF,SIP,CP,RC</td>
<td>MEF,SIP,CC,RC</td>
<td>MEF,SIL,CP,RP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;R10&quot;</td>
<td>&quot;R10&quot;</td>
<td>&quot;R11&quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEF,SIL,CW,RP</td>
<td>MEF,SIL,CC,RP</td>
<td>MEF,SIL,CP,RC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;R11&quot;</td>
<td>&quot;R11&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEF,SIL,CC,RC</td>
<td>MEF,SV,CP,RP</td>
<td>MEF,SV,CW,RP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;R11&quot;</td>
<td>&quot;R11&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEF,SV,CC,RP</td>
<td>MEF,SV,CP,RC</td>
<td>MEF,SV,CC,RC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;R11&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEP,SIP,CP,RP</td>
<td>MEP,SIP,CW,RP</td>
<td>MEP,SIP,CC,RP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;R9&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEP,SIP,CP,RC</td>
<td>MEP,SIP,CC,RC</td>
<td>MEP,SIP,CC,RP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;R10&quot;</td>
<td>&quot;R9&quot;</td>
<td>&quot;R9&quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEP,SIL,CW,RP</td>
<td>MEP,SIL,CP,RP</td>
<td>MEP,SIL,CP,RC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;R9&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEP,SIL,CC,RC</td>
<td>MEP,SV,CP,RP</td>
<td>MEP,SV,CW,RP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;R9&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEP,SV,CC,RP</td>
<td>MEP,SV,CP,RC</td>
<td>MEP,SV,CC,RC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;R9&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MUE,SIP,CP,RP</td>
<td>MUE,SIP,CW,RP</td>
<td>MUE,SIP,CC,RP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;R10&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MUE,SIP,CP,RC</td>
<td>MUE,SIP,CC,RC</td>
<td>MUE,SIP,CC,RP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;R10&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MUE,SIL,CW,RP</td>
<td>MUE,SIL,CC,RP</td>
<td>MUE,SIL,CP,RC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MUE,SIL,CC,RC</td>
<td>MUE,SV,CP,RP</td>
<td>MUE,SV,CW,RP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MUE,SV,CC,RP</td>
<td>MUE,SV,CP,RC</td>
<td>MUE,SV,CC,RC</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 5.12 State Space for System Behaviour with State Elimination Rules (** indicates initial state)**
The interaction rules are likely to be a lot more effective at this level than at the level of ‘Binary’ behaviour models. This is because all the possible superstate permutations are now on board. At this stage, the interaction rules we are interested in are the State Elimination Rules. Table 5.12 shows the state space from Table 5.11 with each state marked “in the row below it” with the interaction rule that eliminates it.

Table 5.13 shows the final state space for our system. From this we can see that the application of interaction rules has reduced the complexity of the system by around 50% from 45 states to the current 26. Further state elimination is also possible through the application of transition combinations and the notion of transient states and unreachable states.

<table>
<thead>
<tr>
<th>MEF,SIP,CP,RP**</th>
<th>MEF,SIL,CP,RC</th>
<th>MEF,SIL,CC,RC</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEF,SV,CP,RC</td>
<td>MEF,SV,CC,RC</td>
<td>MEP,SIP,CP,RP</td>
</tr>
<tr>
<td>MEF,SIP,CW,RP</td>
<td>MEF,SIL,CP,RP</td>
<td>MEP,SIL,CW,RP</td>
</tr>
<tr>
<td>MEP,SIL,CP,RC</td>
<td>MEP,SV,CP,RP</td>
<td>MEP,SV,CW,RP</td>
</tr>
<tr>
<td>MEP,SV,CP,RC</td>
<td>MUE,SIP,CP,RP</td>
<td>MUE,SIP,CW,RP</td>
</tr>
<tr>
<td>MUE,SIP,CC,RP</td>
<td>MUE,SIL,CP,RP</td>
<td>MUE,SIL,CW,RP</td>
</tr>
<tr>
<td>MUE,SIL,CC,RC</td>
<td>MUE,SIL,CP,RC</td>
<td>MUE,SIL,CC,RC</td>
</tr>
<tr>
<td>MUE,SV,CP,RP</td>
<td>MUE,SV,CW,RP</td>
<td>MUE,SV,CC,RP</td>
</tr>
<tr>
<td>MUE,SV,CP,RC</td>
<td>MUE,SV,CC,RC</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.13 State Space for System Behaviour with State Elimination Rules Applied (** indicates initial state)

- Step 3. Add Transitions - Γb

Table 5.14 (in appendix G) shows the behaviour model of the entire system. This is the highest level of complexity the behaviour model will reach. Over the next few steps we shall start reducing this complexity through the application of further interaction rules, particularly those to do with transition chains and concurrency.
• Step 4. Apply More of the Interaction Rules - Γb

The Γ behaviour model built so far is Γa, see chapter 4. This does not cater for removing states and transitions when dealing with transition chains and concurrent transitions. The process of eliminating redundant states via chains and concurrency is part of the process of building Γb.

This section revisits the set of interaction rules in a search for a situation in the behaviour model, in its latest form, where any of the 14 rules is now applicable. A close examination of the states and transitions in Table 5.14 reveals that rules “R5”, “R6” and “R7” are now applicable to some superstates, therefore some transitions have to be disallowed. Table 5.15 (appendix G) shows the resulting behaviour model, with every deleted transition replaced with the interaction rule that deletes it. Finally Table 5.16 (appendix G) shows the result of removing those transitions.

• Step 5. Apply Concurrency Interactions - Γb

In this step, we revisit our behaviour model in Table 5.16 and apply another subset of the interaction rules. This time it is concurrent transitions that are to be applied. In the previous steps, we have marked the potentially concurrent transitions with the “X?” notation and marked the sum of every two “X?” transitions on the same row with an “[X]” notation, again on the same row.

In light of the previous step, where a few transitions have been removed, it is now safe to search for concurrent transitions in order to combine them. It is precisely because some transitions (including concurrent ones) are subject to cancellation by various interaction rules that we left this step so late in this modelling process.

It is risky to combine concurrent transitions “X?” and replace them with a single supertransition “[X]” too early. This is because if any constituent transition were to be removed by an interaction rule, this might cause ambiguity. This is due to
the fact that a supertransition replaces the two (or more) concurrent ones, rendering the rule which deletes either or both of them of little use.

The reason for this is that the deleting rule may not find the illegal transitions in their usual place, as they have been removed and replaced with a new one “[X]” to a different target superstate; a clear case of “moving the target”. This is precisely why both chains of transitions and concurrent transitions can only be marked clearly in the early stages of behaviour modelling and then applied or combined after other interaction rules such as state elimination and transition elimination have taken course.

Table 5.17 (appendix G) shows the resulting behaviour model with all concurrent transitions combined into supertransitions. It can already be seen that the complexity of our system is far smaller than it would have been without component interactions taking place.

- **Step 6. Apply Transition Chains Type Interactions - ᵇ**

In this step, transition chains in our behaviour model are dealt with. The main feature of transition chains is when a state $S_1$ is reached a via transition caused by event $e_n$, and when this transition itself causes another transition from state $S_1$ to, say $S_2$. The net effect of this phenomenon is that, if state $S_1$ is not reachable via any more transitions, then it ($S_1$) does not have any opportunity to start any activity, apart from the spontaneous change to $S_2$, hence state $S_1$ is called transient. See chapter 4 for further details of this subject.

Table 5.18 (appendix G) shows our systems behaviour model with this type of interaction applied. Transitions that are part of a chain are identified with the “X>>” notation. For example, the superstate MEF,SIP,CP,RP changes to MUE,SIP,CP,RP. This then changes as a result of the original transition to MUE,SIP,CW,RP. In fact, then, the response that state MEF,SIP,CP,RP gives to
event $e_4$ is a change to state MUE,SIP,CP,RP, followed, spontaneously with another change to MUE,SIP,CW,RP.

As discussed in chapter 4, the significance of transient states has far reaching implications. These kinds of states are essential in the system, they cannot be removed like, say, unreachable states, as they constitute the *stepping stones* of transition chains. However, the most significant aspect of these states is that if there are other states in the systems behaviour model, which are reachable exclusively from transient states then those other states are, in fact, redundant.

The reason for this is simple: Assume state $S_x$ is only reachable via event $e_y$ from transient state $S_{t0}$. Assume, further, that $S_{t0}$ is on the path of a chain of transition to state $S_{t1}$. State $S_x$ Relies for its existence on the event $e_y$ from $S_{t0}$. However, this event will never have a chance to be responded to from $S_{t0}$, hence $S_x$ is rendered unreachable.

- **Step 7. Look for Unreachable States - $\Gamma_b$**

In this section, we will search our behaviour model in Table 5.18 for unreachable states. The obvious solution for this is to pick states in the top row of the table which have no transition leading to them in their entire column. However, the problem is slightly more complex than it seems.

A state that has a transition - in its column - leading to it, is still not guaranteed reachability. There are several reasons for this. One reason was explained above in the section on transient state. Another reason is that some parts of, or a set of, states from the whole behaviour model are unreachable in their entirety. This type of unreachability to a subset of the behaviour model is easily detected in visual behaviour models, where all is required is to find the area of the state diagram which is *severed* from the part of the behaviour model around the initial state. In state transition tables this is detected as follows. The initial state is the most reachable state in the model, this is given a reachability factor of 0. Next, all states directly reachable from the initial state are also reachable and are given a
reachability factor of 1. Next, all states reachable from states with a reachability factor of \( n \) are also reachable and have a reachability factor of \( n + 1 \).

All remaining states are unreachable, and the maximum \( n + 1 \) achieved in the behaviour model is referred to as the models \textit{depth}. This signifies the maximum number of transitions or events required from the initial state to reach the last reachable state (or the one farthest from the initial state or the one with the maximum reachability factor).

This process, the result of which is shown in Table 5.19 (appendix G) reveals that the following states are unreachable. \text{MUE,SIP,CC,RP}; \text{MUE,SIL,CC,RP}; and \text{MUE,SV,CC,RP}. The next step is to remove these three states, and any other states that are exclusively reachable from them.

The final states list contains 23 states as demonstrated in figures 5.10 and 5.11 (both in appendix G) which show the final \( \Gamma \) behaviour model with and without the unreachable state and then with those states removed.

\textbf{5.3.2 Modelling Systems Behaviour Using Alternative Binary Models}

In this section, we construct a behaviour model of the system in section 5.3.1 using an alternative combination of components. Member is aggregated with Contributions on the one hand, while Scheme is aggregated with Returns on the other. The two results are finally aggregated to achieve the \( \Gamma \) model. In theory there ought to be no differences in the final result (\( \Gamma \)) whichever approach is used for generating the binary models. We start with the behaviour model for Member and Contributions.
5.3.2.1 Behaviour Model for ‘Binary’ Combination of Member and Contributions

In this section we will generate half the behaviour model of our system, that of the two components Member and Contributions.

- Step 1. List States - Πa

The first step in this procedure is to list all the possible combined states for Member and Contributions combined.

The list of states is as follows:

MEF,CP**; MEP,CP; MUE,CP;
MEF,CW; MEP,CW; MUE,CW;
MEF,CC; MEP,CC; MUE,CC

** indicates initial superstate. The meaning of the abbreviated state names is described above.

- Step 2. Add Transitions - Γa

The state transition table lists the source states in the first column and the destination states in the first row. Once again, it is important to include the entire state space of the sub-system we are dealing with.

The initial behaviour model is shown in Table 5.20.
Table 5.20 Behaviour Model for Member, Contributions
(interaction rules not applied)

- Step 3. Apply Interaction Rules (no Chains or Concurrency) - \( \Gamma_b \)

Table 5.21 Behaviour Model for Member, Contributions
(some interactions applied)
In this step we apply interaction rules to the behaviour model. We will exclude any rules to do with Chains or Concurrency at this stage. The effect of this is as follows:

Rule “R9” dictates that superstate [MEP,CC] should not exist.
Rule “R6” implies event 10 not applicable to superstate [MUE,CP].
Rule “R5” implies event 12 not applicable to superstate [MUE,CW].

The result is shown in Table 5.21.

5.3.2.2 Behaviour Model for the ‘Binary’ Combination of Scheme and Returns - Γb

In this section we will construct the behaviour model for Scheme and Returns, in a similar approach to the previous behaviour model.

• Step 1. List States - Γa

The possible states that a system composed of Scheme and Returns will go through in its lifecycle includes:

SIP,RP**; SIL,RP; SV,RP;
SIP,RC; SIL,RC; SV,RC
** indicates initial superstate.

• Step 2. Add Transitions - Γa

Table 5.22 shows the initial behaviour model with only the states and their transitions included.
Table 5.22 Behaviour Model for Scheme,Returns
(interaction rules not applied)

Notes:

It can be seen from Table 5.22 that there are few, if any, interactions between Scheme and Returns. This is manifested through a lack of shared events (concurrency) or transition chains.

- Step 3. Apply Interaction Rules - Γb

Table 5.23 Behaviour Model for Scheme,Returns
(some interaction rules applied)

From the set of interaction rules above, we can see the following:
Rule “R7” implies superstate [SV,RP] does not respond to event 10.
Rule “R10” dictates that superstate [SIP,RC] should not exist.
The resulting behaviour model is shown in Table 5.23
5.3.2.3 Combine ‘Binary’ Models – Complete $\Gamma$α System

The next step is to use the result of the binary behaviour models above to generate
the behaviour model for the System.

- Step 1. List States - $\Gamma$α

The possible states in this behaviour model are the result of the product of the two
state spaces in our two binary models above.

<table>
<thead>
<tr>
<th>MEF,CP,SIP,RP**</th>
<th>MEF,CP,SIL,RP</th>
<th>MEF,CP,SV,RP</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEF,CP,SIL,RC</td>
<td>MEF,CP,SV,RC</td>
<td>MEF,CP,SIP,RP</td>
</tr>
<tr>
<td>MEP,CP,SIL,RP</td>
<td>MEP,CP,SV,RP</td>
<td>MEP,CP,SIL,RC</td>
</tr>
<tr>
<td>MEP,CP,SV,RC</td>
<td>MUE,CP,SIP,RP</td>
<td>MUE,CP,SIL,RP</td>
</tr>
<tr>
<td>MUE,CP,SV,RP</td>
<td>MUE,CP,SIL,RC</td>
<td>MUE,CP,SV,RC</td>
</tr>
<tr>
<td>MEF,CW,SIP,RP</td>
<td>MEF,CW,SIL,RP</td>
<td>MEF,CW,SV,RP</td>
</tr>
<tr>
<td>MEF,CW,SIL,RC</td>
<td>MEF,CW,SV,RC</td>
<td>MEF,CW,SIP,RP</td>
</tr>
<tr>
<td>MEP,CW,SIL,RP</td>
<td>MEP,CW,SV,RP</td>
<td>MEP,CW,SIL,RC</td>
</tr>
<tr>
<td>MEP,CW,SV,RC</td>
<td>MUE,CW,SIP,RP</td>
<td>MUE,CW,SIL,RP</td>
</tr>
<tr>
<td>MUE,CW,SV,RP</td>
<td>MUE,CW,SIL,RC</td>
<td>MUE,CW,SV,RC</td>
</tr>
<tr>
<td>MEF,CC,SIP,RP</td>
<td>MEF,CC,SIL,RP</td>
<td>MEF,CC,SV,RP</td>
</tr>
<tr>
<td>MEF,CC,SIL,RC</td>
<td>MEF,CC,SV,RC</td>
<td>MUE,CC,SIP,RP</td>
</tr>
<tr>
<td>MUE,CC,SIL,RP</td>
<td>MUE,CC,SV,RP</td>
<td>MUE,CC,SIL,RC</td>
</tr>
<tr>
<td>MUE,CC,SV,RC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.24 State Space for System Behaviour – no interaction rules

(*** indicates initial state)

This implies we have (8 X 5) = 40 states. These are shown in Table 5.24.
Step 2. Apply Interaction Rules - $\Gamma_b$

The interaction rules are likely to be a lot more effective at this level than at the level of ‘Binary’ behaviour models, this is simply because all the possible superstate permutations are now at available. Table 5.25 shows the state space from Table 5.24 with each state marked “in the row below it” with the interaction rule that eliminates it.

<table>
<thead>
<tr>
<th>MEF,CP,SIP,RP**</th>
<th>MEF,CP,SIL,RP</th>
<th>MEF,CP,SV,RP</th>
<th>R11</th>
<th>R11</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEF,CP,SIL,RC</td>
<td>MEF,CP,SV,RP</td>
<td>MEF,CP,SIP,RP</td>
<td>R11</td>
<td></td>
</tr>
<tr>
<td>MEP,CP,SIL,RP</td>
<td>MEP,CP,SV,RP</td>
<td>MEP,CP,SIL,RC</td>
<td>R12</td>
<td></td>
</tr>
<tr>
<td>MEP,CP,SV,RC</td>
<td>MUE,CP,SIP,RP</td>
<td>MUE,CP,SIL,RP</td>
<td>R12</td>
<td></td>
</tr>
<tr>
<td>MUE,CP,SV,RP</td>
<td>MUE,CP,SIL,RC</td>
<td>MUE,CP,SV,RC</td>
<td>R12</td>
<td></td>
</tr>
<tr>
<td>MEF,CW,SIP,RP</td>
<td>MEF,CW,SIL,RP</td>
<td>MEF,CW,SV,RP</td>
<td>R12</td>
<td></td>
</tr>
<tr>
<td>R11</td>
<td>R11</td>
<td>R11</td>
<td>R11</td>
<td></td>
</tr>
<tr>
<td>MEF,CW,SIL,RC</td>
<td>MEF,CW,SV,RP</td>
<td>MEF,CW,SIP,RP</td>
<td>R12</td>
<td></td>
</tr>
<tr>
<td>R12</td>
<td>R12</td>
<td>R12</td>
<td>R12</td>
<td></td>
</tr>
<tr>
<td>MEP,CW,SIL,RP</td>
<td>MEP,CW,SV,RP</td>
<td>MEP,CW,SIL,RC</td>
<td>R12</td>
<td></td>
</tr>
<tr>
<td>MEP,CW,SV,RC</td>
<td>MUE,CW,SIP,RP</td>
<td>MUE,CW,SIL,RP</td>
<td>R12</td>
<td></td>
</tr>
<tr>
<td>R12</td>
<td>R12</td>
<td>R12</td>
<td>R12</td>
<td></td>
</tr>
<tr>
<td>MUE,CW,SV,RP</td>
<td>MUE,CW,SIL,RC</td>
<td>MUE,CW,SV,RC</td>
<td>R12</td>
<td></td>
</tr>
<tr>
<td>R12</td>
<td>R12</td>
<td>R12</td>
<td>R12</td>
<td></td>
</tr>
<tr>
<td>MEF,CC,SIP,RP</td>
<td>MEF,CC,SIL,RP</td>
<td>MEF,CC,SV,RP</td>
<td>R12</td>
<td></td>
</tr>
<tr>
<td>R11</td>
<td>R11</td>
<td>R11</td>
<td>R11</td>
<td></td>
</tr>
<tr>
<td>MEF,CC,SIL,RC</td>
<td>MEF,CC,SV,RP</td>
<td>MUE,CC,SIP,RP</td>
<td>R12</td>
<td></td>
</tr>
<tr>
<td>MUE,CC,SIL,RP</td>
<td>MUE,CC,SV,RP</td>
<td>MUE,CC,SIL,RC</td>
<td>R12</td>
<td></td>
</tr>
<tr>
<td>MUE,CC,SV,RC</td>
<td>MUE,CC,SV,RP</td>
<td>MUE,CC,SIL,RC</td>
<td>R12</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.25 State Space for System Behaviour with State Elimination Rules (** indicates initial state)
Table 5.26 shows the final state space for our system. From this we can see that the application of interaction rules has significantly reduced the complexity of the system from 45 states to the current 26. Further state elimination is also possible through the application of transition combinations and the notion of transient states and unreachable states.

<table>
<thead>
<tr>
<th>MEF,CP,SIP,RP**</th>
<th>MUE,CC,SV,RC</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEF,CP,SIL,RC</td>
<td>MEF,CP,SV,RC</td>
</tr>
<tr>
<td>MEF,CP,SIL,RP</td>
<td>MEF,CP,SV,RP</td>
</tr>
<tr>
<td>MUE,CP,SV,RC</td>
<td>MUE,CP,SIP,RP</td>
</tr>
<tr>
<td>MUE,CP,SV,RP</td>
<td>MUE,CP,SIL,RC</td>
</tr>
<tr>
<td>MUE,CP,SIL,RP</td>
<td>MUE,CP,SV,RC</td>
</tr>
<tr>
<td>MUE,CP,SIL,RC</td>
<td>MUE,CP,SV,RP</td>
</tr>
<tr>
<td>MUE,CP,SIL,RP</td>
<td>MUE,CP,SV,RC</td>
</tr>
</tbody>
</table>

Table 5.26 State Space for System Behaviour with State Elimination Rules Applied (** indicates initial state)

The state of the systems behaviour model so far is consistent with the previous model generated using the [Member,Scheme][Contribution,Returns] combination, at least in the number of states.

The remaining steps of generating the full F model should confirm the equivalence of the two approaches. However, this will not be carried out in this chapter; rather, we will build a model using other combination options in the following section.
5.3.3 Abstracting Behaviour Model – The Φ Behaviour Model

The behaviour models in the previous section are final representations of the system’s functionality as an aggregation of the respective functionality of its component objects. There exists a further level of behaviour modelling wherein the focus is given to a certain facet of the behaviour model of the system. This level is known as the Φ level. Here the behaviour model (Γ) is abstracted based on some semantic requirement. The significance of this level in the behaviour model is that it provides a new perspective on the functionality of the system. For example, Φ enables decision makers to focus, from a semantic point of view on the aspects of the system that are directly related to the decision making process.

The starting point for the Φ in this section is the Γ in Table 5.19 and Figure 5.11 (Figures 5.11 and 5.12 are in appendix G). The behaviour model is simple enough at this stage of development to be described using the state diagram notation. The semantic basis for building the Φ level is a need to gather as many of the superstates that contain within them, as a substate, the state SIL,RC. The semantic significance of this is that it will expose the functionality of the system where the investment company is losing most of its funds. SIL,RC implies the investment Scheme is In Loss and that the Client has already been promised some kind of a Return, hence Returns Credited.

- **Step 1. First Set of Groups**

In this step, the two areas of the diagram containing the substate SIL,RC are grouped, or abstracted, together. The first group contains the superstates [MEP,SIL, CP,RC], [MEF,SIL, CP,RC], and [MUE,SIL, CP,RC]. The second group contains the two superstates [MEF,SIL, CC,RC] and [MUE,SIL,CC,RC]. When grouping multiple states, the main idea is to collect together these states, the transitions of each state now belong to the overall rectangle or box.
representing the abstract state. This is very similar to the state groups we built when designing the \( \Gamma \) level. Figure 5.12 shows the result.

The main feature of the result in Figure 5.12, which constitutes the first stage of the build up of a \( \Phi \) behaviour model level, shows that the first abstract state grouping (containing \([\text{MEP},\text{SIL}, \text{CP}, \text{RC}], [\text{MEF},\text{SIL}, \text{CP}, \text{RC}], \) and \([\text{MUE},\text{SIL}, \text{CP}, \text{RC}]\)) causes a non-determinancy scenario. Non-determinancy occurs when some event \( e_x \) on a state \( S_x \) (abstract or real) causes two (or more) transitions. See chapter 3 for further details of this phenomenon.

The reason behind this is that the constituent states accept event \( e_9 \) and change to three distinct states (lying outside the group). These are \([\text{MEF},\text{SV}, \text{CP}, \text{RC}], [\text{MEP},\text{SV}, \text{CP}, \text{RC}]\) and \([\text{MUE},\text{SV}, \text{CP}, \text{RC}]\). However, when the group is created, the distinct destination states are left out, as they contain no instances of the SIL, RC substate. Hence the group collects the sources of the transitions caused by event \( e_9 \) but does not collect the destinations, causing a *fanning* out of the transitions caused by event \( e_9 \) from the new group of states to the destination states, a clear case of non-determinancy.

A mirror situation occurs with the second group of states, i.e., \([\text{MEF},\text{SIL}, \text{CC}, \text{RC}]\) \([\text{MUE},\text{SIL}, \text{CC}, \text{RC}]\) have a non-determinancy situation with event \( e_9 \) leading from the state group to the two states \([\text{MUE},\text{SV}, \text{CC}, \text{RC}]\) and \([\text{MEF},\text{SV}, \text{CC}, \text{RC}]\). This scenario (non-determinancy) is unacceptable at any level of the behaviour model hierarchy, hence the next step will be to eliminate it.

Another feature of these two state groupings is that all transitions that existed between the constituents of the group are still valid, and are manifested as a loop transition to the new group. These are transitions 1, 2, 3, 4, 5 in the first group and transition 3 in the second.
• Step 2. Eliminate Non-Determinancy

In this step, the problem of non-determinancy is addressed. The obvious solution to this problem is to collect into a new group of states, all those states that are reached via the fanning transitions caused by event \( e_g \) above. Hence there are two new groups in the systems behaviour model, each containing the destination states reachable via event \( e_g \) from the first step above, see Figure 5.12 in appendix G.

This behaviour model can be abstracted further by collecting the four new groups of states into one single super-group. However, it is important to avoid going too far with the abstraction of the behaviour as this may inevitably hide significant behaviour.

The main criterion in judging the correctness of the \( \Phi \) model are as follows:

a) There must not be new behaviour introduced which does not exist in the \( \Gamma \) model

b) The \( \Phi \) model can hide or abstract behaviour but may not actively disallow otherwise existent behaviour

c) The \( \Phi \) model, like its \( \Gamma \) model origin, must be deterministic

One important feature of \( \Phi \) that is not true for the \( \Gamma \) model is that for any system, there can be only a single correct \( \Gamma \) behaviour model. This is an actual, non-abstract behaviour model that is the aggregation of the respective behaviour(s) of the components of the system. In the case of the \( \Phi \) model, we are dealing with abstractions of the \( \Gamma \) model; we saw in this section two attempts at this abstraction. In another example, it may be suitable to go much further in the abstraction direction and have several layers of those, hence enabling the systems
analysts to focus at the level that suits their needs most. So for every \( \Gamma \) there can be several \( \Phi \)'s.

5.3.4 Abstracted \( \Gamma \) (\( \Gamma' \)) For
(Member,[Returns,Scheme,Contributions])

Tables 5.29-5.32 are in appendix G, also figures 5.14 and 5.15 are in appendix G. Tables: 5.36-5.41 are in appendix G, also figures: 5.18 and 5.19 are in appendix G.

The abstract behaviour model in the previous section shows a much simpler model where focus can be made only on those states that contain a specific facet of behaviour (SIL,RC) and monitor the relation between all states containing this facet and the rest of the behaviour model. The major difficulty in achieving this level in the behaviour model is that it has to be reached via the laborious path where a very complex behaviour model was constructed initially, then the reduction was applied consequently to this complex model.

In this section, we will attempt to build a similar behaviour model to the \( \Phi \) in the previous section using a different approach. The approach is to abstract the states containing the (SIL,RC) pattern from an earlier stage in the behaviour model hierarchy. The obvious feature of this approach is that Scheme (the object that SIL relates to) and Returns (the object that RC relates to) must be aggregated together from the outset. Other components are then added to those gradually. The behaviour model will be built as follows:

a) Build \( \Gamma \) of Contributions and Returns

b) Add a third component, preferably one with fewer interactions with both Contributions and Returns, hence Scheme

c) Abstract the above, by combining all states containing SIL,RC

d) Add behaviour of the final component, Member
This should further clarify whether the $\Gamma$ of two components, one of which is already abstracted, results in a $\Phi$ of the $\Gamma$ of those components, i.e.

Is $\Gamma' [X, \Phi(Y)]$ equivalent to one of the $\Phi$'s of the $\Gamma (X,Y)$?

- **Step1. Aggregate Contributions and Returns - $\Gamma a$**

Table 5.10 above showed this to be as follows

<table>
<thead>
<tr>
<th></th>
<th>CP,RP</th>
<th>CW,RP</th>
<th>CC,RP</th>
<th>CP,RC</th>
<th>CC,RC</th>
</tr>
</thead>
<tbody>
<tr>
<td>CP,RP</td>
<td></td>
<td>Te4*</td>
<td>10?</td>
<td>10?</td>
<td>[10]</td>
</tr>
<tr>
<td>CW,RP</td>
<td></td>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CC,RP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>CP,RC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>CC,RC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.27 Behaviour Model for Contributions, Returns
(some interaction rules applied)

**Step 2. Add Scheme - $\Gamma a$**

Scheme was chosen over member as it has fewer interactions with Contributions and Returns. From section 5.2.2.2, the behaviour of Scheme is as follows:

<table>
<thead>
<tr>
<th></th>
<th>SIP</th>
<th>SIL</th>
<th>SV</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIP</td>
<td>4</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>SIL</td>
<td>6</td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>SV</td>
<td>8</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.28 Behaviour of Scheme
In this section we will aggregate the two behaviours above to generate the behaviour model for Scheme.(Contributions,Returns). Table 5.29 shows the initial result of the aggregation of those two tables, with none of the interaction rules applied. Table 5.30 shows the behaviour model in Table 5.29 incorporating more interaction rules. The state elimination rules are applied, Table 5.30 shows each transition marked with the rule that voids it. In the case where an entire row/column is marked with a rule number, this implies the state itself is eliminated by the shown rule.

- **Step 3. Add Interaction Rules - \( \Gamma_b \)**

In Table 5.31, rules relating to state and transition elimination are applied. Further states to be merged/abstracted are also identified, and these are also shown visually in Figure 5.14 where states to be abstracted/grouped are also marked with a " * ". These are all superstates containing occurrences of the \( \text{SIL,RC} \) pattern and include the superstates \( \text{SIL,CP,RC} \) and \( \text{SIL,CC,RC} \).

**Step 4. Generate Abstract State**

States containing the \( \text{SIL,RC} \) pattern are combined to generate the new abstract state \( \text{CP,CC,SIL,RC} \), the result is shown in Table 5.32 and Figure 5.15. The diagram in Figure 5.16 shows that a potential non-determinancy situation would occur if only one abstract state \( \text{SIL,CP,CC,RC} \) was created. This is as follows:

![Diagram of Potential Non-Determinancy](image)

**Figure 5.16 Potential Non-Determinacy**

The solution to this problem is to combine the two superstates \([\text{SV,CP,RC}]\) and \([\text{SV,CC,RC}]\) at the destination end of the two non-determinant transitions caused
by event $e_0$ from the abstract state SIL,CP,CC,RC, hence the abstract state [SV,CP,CC,RC].

- Step 5. Add the Behaviour of Member – States Only

Member’s behaviour model is shown in Figure 5.17

![Figure 5.17 Behaviour of Member](image)

<table>
<thead>
<tr>
<th>$\text{MEF,SIP,CP,RP}^{**}$</th>
<th>$\text{MEF,SIP,CW,RP}$</th>
<th>$\text{MEF,SIP,CC,RP}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{MEF,SIL,CP,RP}$</td>
<td>$\text{MEF,SIL,CW,RP}$</td>
<td>$\text{MEF,SIL,CC,RP}$</td>
</tr>
<tr>
<td>$\text{MEF,SIL,CP,CC,RC}$</td>
<td>$\text{MEF,SV,CP,RP}$</td>
<td>$\text{MEF,SV,CW,RP}$</td>
</tr>
<tr>
<td>$\text{MEF,SV,CC,RP}$</td>
<td>$\text{MEF,SV,CP,CC,RC}$</td>
<td>$\text{MEP,SIP,CP,RP}$</td>
</tr>
<tr>
<td>$\text{MEP,SIP,CP,RP}$</td>
<td>$\text{MEP,SIP,CW,RP}$</td>
<td>$\text{MEP,SIP,CC,RP}$</td>
</tr>
<tr>
<td>$\text{MEP,SIL,CP,RC}$</td>
<td>$\text{MEP,SIL,CP,CC,RC}$</td>
<td>$\text{MEP,SV,CP,RP}$</td>
</tr>
<tr>
<td>$\text{MEP,SIL,CP,CC,RC}$</td>
<td>$\text{MUE,SIP,CP,RP}$</td>
<td>$\text{MUE,SIP,CW,RP}$</td>
</tr>
<tr>
<td>$\text{MUE,SIP,CC,RP}$</td>
<td>$\text{MUE,SIL,CP,RP}$</td>
<td>$\text{MUE,SIL,CW,RP}$</td>
</tr>
<tr>
<td>$\text{MUE,SIL,CC,RP}$</td>
<td>$\text{MUE,SIL,CP,CC,RC}$</td>
<td>$\text{MUE,SV,CP,RP}$</td>
</tr>
<tr>
<td>$\text{MUE,SV,CP,CC,RC}$</td>
<td>$\text{MUE,SV,CC,RP}$</td>
<td>$\text{MUE,SV,CP,CC,RC}$</td>
</tr>
</tbody>
</table>

**Table 5.33 States in Abstracted $\Gamma$, $\Gamma'$**

(* Mark Initial State*)

This behaviour model is combined/aggregated with the behaviour model in Table 5.32 in this section to generate the $\Phi$ model which is expected to be one of several possible abstractions of the $\Gamma$ models generated in sections 5.3.2 and 5.3.3. Table
5.33 shows the list of possible states from the initial aggregation of the two, referred to henceforth as $\Gamma^\prime$.

- **Step 6. Add the Behaviour of Member – State Elimination Rules**

The next step is to add interaction rules relating to state existence. Table 5.34 shows the list of states from Table 5.33, each marked (in the same cell) with the interaction rule that eliminates it.

<table>
<thead>
<tr>
<th>MEF,SIP,CP,RP**</th>
<th>MEF,SIP,CW,RP - R11</th>
<th>MEF,SIP,CC,RP - R11</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEF,SIL,CP,RP - R11</td>
<td>MEF,SIL,CW,RP - R11</td>
<td>MEF,SIL,CC,RP - R11</td>
</tr>
<tr>
<td>MEF,SIL,CP,CC,RC</td>
<td>MEF,SV,CP,RP - R11</td>
<td>MEF,SV,CW,RP - R11</td>
</tr>
<tr>
<td>MEF,SV,CC,RP - R11</td>
<td>MEF,SV,CP,CC,RC</td>
<td>MEF,SV,CP,CP,RP</td>
</tr>
<tr>
<td>MEP,SIP,CW,RP</td>
<td>MEP,SIP,CC,RP - R9</td>
<td>MEP,SIP,CP,RP</td>
</tr>
<tr>
<td>MEP,SV,CP,RP</td>
<td>MEP,SV,CP,CC,RP - R9</td>
<td>MEP,SV,CP,CP,RP</td>
</tr>
<tr>
<td>MEP,SV,CP,CC,RC - R9</td>
<td>MUE,SIP,CP,RP</td>
<td>MUE,SIP,CW,RP</td>
</tr>
<tr>
<td>MUE,SIP,CC,RP</td>
<td>MUE,SIP,CP,RP</td>
<td>MUE,SIP,CC,RP</td>
</tr>
<tr>
<td>MUE,SIL,CP,RP</td>
<td>MUE,SIL,CP,CC,RC</td>
<td>MUE,SIL,CP,RP</td>
</tr>
<tr>
<td>MUE,SIL,CP,RP</td>
<td>MUE,SIL,CW,RP</td>
<td>MUE,SIL,CW,RP</td>
</tr>
<tr>
<td>MUE,SIL,CP,CC,RC</td>
<td>MUE,SV,CP,RP</td>
<td>MUE,SV,CP,CC,RC</td>
</tr>
<tr>
<td>MUE,SV,CC,RP</td>
<td>MUE,SV,CP,CC,RC</td>
<td>MUE,SV,CP,CC,RC</td>
</tr>
</tbody>
</table>

Table 5.34 States in $\Gamma^\prime$ – Elimination Rules Shown

(** Marks Initial State)

The final set of permissible $\Gamma^\prime$ states is shown in Table 5.35

<table>
<thead>
<tr>
<th>MEF,SIP,CP,RP**</th>
<th>MEF,SIL,CP,CC,RC</th>
<th>MEF,SV,CP,CC,RC</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEP,SIP,CP,RP</td>
<td>MEP,SIP,CW,RP</td>
<td>MEP,SIP,CP,RP</td>
</tr>
<tr>
<td>MEP,SIL,CP,RP</td>
<td>MEP,SIP,CW,RP</td>
<td>MEP,SIP,CP,RP</td>
</tr>
<tr>
<td>MEP,SIL,CW,RP</td>
<td>MUE,SIP,CP,RP</td>
<td>MUE,SIP,CC,RP</td>
</tr>
<tr>
<td>MUE,SIP,CP,RP</td>
<td>MUE,SIP,CP,RP</td>
<td>MUE,SV,CP,RP</td>
</tr>
<tr>
<td>MUE,SIL,CP,RP</td>
<td>MUE,SIL,CP,CC,RC</td>
<td>MUE,SV,CP,RP</td>
</tr>
<tr>
<td>MUE,SIL,CP,RP</td>
<td>MUE,SV,CP,RP</td>
<td>MUE,SV,CP,CC,RC</td>
</tr>
<tr>
<td>MUE,SV,CC,RP</td>
<td>MUE,SV,CP,RP</td>
<td>MUE,SV,CC,RP</td>
</tr>
</tbody>
</table>

Table 5.35 States in $\Gamma^\prime$ – Elimination Rules Applied

(** Marks Initial State)
• **Step 7. Add the Behaviour to \( \Gamma' \) – Transitions**

The transitions that occur in Figure 5.15 and Figure 5.17 constitute the basis for generating the next level of \( \Gamma' \). To this end, the two figures are treated as though they relate to two actual components. The transitions from those two components are therefore aggregated in a similar fashion to what we have seen in the previous sections of this chapter. The result is shown in Table 5.36.

• **Step 8. Apply Interaction Rules to \( \Gamma' \) – Transition Elimination (\( \Gamma_b \))**

Table 5.37 shows the interaction rules that eliminate a subset of the transitions in \( \Gamma' \). Each transition is replaced with the number from the rule that eliminates it, the result of which is shown in Table 5.39.

• **Step 9. Apply More Interactions – Concurrency (\( \Gamma_b \))**

In this step, the interactions of the concurrent category are applied. Those are the ones relating to a single event causing multiple reactions in multiple components. They include responses to event \( e_4 \) for example from state \([\text{MEF, SIL, CP, CC, RC}]\). As a reminder, the basic rule for aggregating two transitions caused by the same event is as follows. If superstate \([X,Y,Z]\) changes - via transition \( t_x \) - to superstates \([X,Y1,Z]\) and to \([X1,Y,Z]\) via the same transition, then these two occurrences - of \( t_x \) - can be combined to let \([X,Y,Z]\) change to \([X1,Y1,Z]\). However, if for any reason, one of the two transitions fails to take place, then the other ones in the group will fire as normal. That is to say if, for example, Component Y fails to change from Y to Y1, then the superstate \([X,Y,Z]\) changes to superstate \([X1,Y,Z]\) via the event/transition \( t_x \).
• **Step 10. Apply More Interactions – Chains (Fb)**

The other major mode of interaction in our system is chains of transitions. It can be seen from Table 5.39 that superstate \([\text{MEF,SIP,CP,RP}]\) changes to superstate \([\text{MUE,SIP,CP,RP}]\) via event \(e_4\). It can also be seen in the same table that \([\text{MUE,SIP,CP,RP}]\) further changes via event \(t_{e_4}\) (the transition caused by the transition caused by event \(e_4\)) to state \([\text{MUE,SIP,CW,RP}]\). This implies that, in the real world, which this behaviour model relates to, the combined state \([\text{MEF,SIP,CP,RP}]\) would not only change to superstate \([\text{MUE,SIP,CW,RP}]\) after event \(e_4\) takes place, rather it carries on to reach superstate \([\text{MUE,SIP,CW,RP}]\). This is reflected in Table 5.40, where the initial event to \([\text{MUE,SIP,CP,RP}]\) is marked with the “>>” notation to indicate a second part to this transition also labelled “4”.

• **Step 11. Check for Reachability (Fb)**

The final step in building the \(\Gamma^*\) behaviour model is to check that all new superstates are actually reachable. This is done in a systematic way similar to that shown in section 5.3.1.3 (step 7) above. The result shown in Table 5.41 indicates that the following states are unreachable (These states are marked in Table 5.41 with a “+”):

a) \([\text{MUE,SIP,CC,RP}], [\text{MUE,SV,CC,RP}]\) and \([\text{MUE,SIL,CC,RP}]\).

b) \([\text{MEF,SIL,CP,CC,RC}], [\text{MEF,SV,CP,CC,RC}], [\text{MUE,SV,CP,CC,RC}], [\text{MUE,SIL,CP,CC,RC}]\).

Despite of the fact that those superstates are reachable within their respective group (a and b), it is the whole group of superstates that is impossible to reach from the initial state which is the most reachable state in the system.

The information in Table 5.41 is shown in a much clearer way using the visual notation in Figure 5.18. It is only possible at this stage to visualise the behaviour.
model due to its early complexity. This gives the advantage of being able to carry out fast analysis to verify aspects of the behaviour model such as non-reachability of states.

5.3.5 Comparison of $\Phi(\Gamma)$ and $\Gamma'$

The $\Phi$ behaviour model resulting in section 5.3.3 and the abstracted $\Gamma$ ($\Gamma''$) resulting in section 5.3.4 need to be compatible in terms of the behaviour they display in order that the methodology described previously to be failsafe. Although the whole notion of the $\Phi$ model is a methodology that is best described as a non-systematic reduction in the complexity of the behaviour models reached at the $\Gamma$ level – see chapter 3 –, there are nevertheless some semantic similarities that can be brought to surface. These include.

a) All superstates that exist in $\Gamma''$ should be in $\Phi$, or in a group of states in $\Phi$ (in case abstraction hides those superstates)

b) All superstates in $\Phi$ and in $\Gamma''$ should be identifiable in $\Gamma$

c) The initial superstate should be the same in $\Gamma$, $\Gamma''$ and $\Phi$

Because the $\Gamma''$ is a $\Phi$ that was applied at a premature level in the behaviour model hierarchy, it is right to anticipate that it will be a stronger abstraction of the final $\Gamma$. The reason for this is that the $\Phi$ in the $\Gamma''$ is applied to abstract behaviour prior to the inclusion of all components of the system. When the new component is added (Member in our example in section 5.3.4) the abstraction will be manifested for every state in the behaviour specification of this latest component. It is, therefore, a foregone conclusion that the $\Gamma''$ constitutes a $\Phi$ of the original $\Gamma$.

Looking at the $\Gamma''$ and $\Phi$ in section 5.3.4 and 5.3.3 respectively, we can see that the $\Gamma''$ is not compatible with the $\Phi$, also, by comparing $\Gamma''$ with the $\Gamma$ in Figure
5.11. This dissimilarity is manifested in many respects including the following points.

a) In $\Gamma'$ (Figure 5.18), the superstate [MEP,SIL,CP,CC,RC] is deleted by Rule 9 (see step 6 in section 5.3.4). This implies the two superstates [MEP,SIL,CP,RC] and [MEP,SIL,CC,RC] are eliminated. This, however, is not the case in $\Gamma$ (Figure 5.11 in appendix G), which clearly shows that the superstate [MEP,SIL,CP,RC] does indeed exist.

b) The same superstate [MEP,SIL,CP,RC] also exists in the $\Phi$ (Figure 5.13 in appendix G).

These two examples are sufficient to cast doubt on the methodology used for building the $\Gamma'$. However, there does not seem to exist any inconsistency of behaviour between $\Gamma$ and $\Phi$. In the following section we will rebuild the $\Gamma'$ behaviour model using a slightly modified technique when applying interaction rules.

5.3.6 State Elimination in Abstracted/Combined Superstates – Where the Approach Went Wrong

We have seen in the previous section that the $\Gamma'$ and the $\Phi$ are not compatible. In this section we will attempt to trace the cause of this and suggest ways for dealing with it.

If we take a second look at the application of the state elimination procedures in step 6 of section 5.3.4 above, we can see the following. In particular, we need to look at the effect of Rule 9 which deletes the two superstates [MEP,SIL,CP,CC,RC] and [MEP,SIL,CP,CC,RC]. It is clear that these two states are abstract (not real) system superstates. This is because each is made up as follows:
[MEP,SIL,CP,CC,RC] is an abstraction or grouping that combines the two real superstates [MEP,SIL,CP,RC] and [MEP,SIL,CC,RC]. Similarly, the grouping [MEP,SV,CP,CC,RC] is an abstraction that combines the two (real) superstates [MEP,SV,CP,RC] and [MEP,SV,CC,RC].

In any behaviour model of any system, a real system state (like those found in the $\Gamma$ level) should contain only one state from each component. Thus, for our system, four components imply four states.

The question that arises here is this: **When an interaction rule eliminates a superstate that exists in a group or abstraction, do we delete the whole superstate or just the offending part?** We believe that such a rule should be applied to the whole superstate in the case of real ($\Gamma$) states, and that an abstracted superstate should be split (un-abstracted) to its constituent real states and then the offending part of those should be deleted. This is to guarantee that only redundant states are deleted, and that their deletion does not sweep with it essential behaviour.

The cost of ignoring this phenomenon is that too many states will mysteriously vanish in the build-up of the $\Gamma'$. Further, the final result, as demonstrated at the end of the previous section, will feature a non-equivalence of behaviour between the $\Gamma'$ and the $\Phi$ that results from applying the same abstraction (as in the $\Gamma'$) to the $\Gamma$ level of the systems behaviour model. This cost is obviously unacceptable.

Based on the answer to the question above, the states to delete in step 6 of section 5.3.4 are therefore [MEP,SIL,CC,RC] and [MEP,SV,CC,RC], just the offending/illegal part of the abstraction. The rest of the steps are identical to steps from 7 in section 5.3.4, the final result is therefore as shown in Figure 5.19 in appendix G. This still shows the following superstates as unreachable (which is consistent with both $\Gamma$ in section 5.3.1 and $\Phi$ in section 5.3.3): [MUE,SIP,CC,RP], [MUE,SV,CC,RP] and [MUE,SIL,CC,RP].
5.4 Synopsis

This chapter showed a few options for designing a behaviour model of a system comprising multiple components. It has also shown the inherent complexity that any behaviour modelling technique needs to address. There are no doubt a few more areas where further experimental work needs to be carried out, we will look at some of those in the final chapter of this thesis.

One of the important points that this chapter served to forward is the effect of the interactions that take place between the components in the system. This further supports the suggestion (see chapter 4) that the behaviour of the system is directly related to the effect of these inter-component interactions.

It has been shown that without the interactions that take place between the components in the system, the behaviour or functionality of the system as a unit would be severely limited to solitary, and semantically trivial, responses to external stimuli. These inter-component reactions are what gives the notions of chains of transitions and concurrent activities their significance [Shlaer 88b].

With concurrency [Harel 87], [Harel 88a], [Harel 88b] in particular, this view corresponds well with that of [Liberty 98] on the significance of concurrency in today’s complex systems running on sophisticated operating systems that can handle concurrency and multithreading. In a simple situation, one user interacts with a system, which does one thing at a time. A user pushes a button and the system reacts, the user waits for that reaction before starting a new command. In a modern system, you no longer have the luxury of assuming that only one reaction will happen as a result of the event, although this is achieved by giving the illusion of multitasking. In many cases it is implemented as a multitasking activity.

It is our assumption in this thesis that for a perfect representation of a real world scenario, a system on a computer must be able to emulate as many of the features of the real world side as possible. This applies, in no small measure, to the two
notions of chains and concurrency. Although it is, technically, a lot simpler to implement chains, concurrency enabled computer platforms are not far away. This is becoming apparent with the advent of multithreading operating systems running on multiprocessor hardware.

The final chapter of this thesis looks at possible improvements to the approach proposed in this and the preceding chapters and identifies areas where more research needs to be undertaken. Furthermore, implementation and application issues are also discussed.
CHAPTER SIX

CONCLUSIONS AND FUTURE DIRECTIONS
6.1 Introduction and Chapter Map

This chapter is made up of a further three sections. Section 6.2 revisits the main points and aspects of this thesis. This pinpoints the main contribution of the research, and the major restrictions that were incorporated in the notation and the approach in order for it to work correctly. Section 6.3 addresses Applicability. This discusses a few real world areas where the notion of complex reactive systems plays a role and where the work presented here may be of some benefit. Finally, Section 6.4 considers possible Further Work. This constitutes both a look back at what was not covered by the thesis, and a look forward towards what new areas are relevant and merit some research.

This thesis has tackled a specific area within the domain of OO systems design, namely the inheritance of behaviour within the aggregation abstraction hierarchy. It began with a look at the work of some mainstream approaches to designing OO systems in general, e.g., OMT, OOAD and UML. This was followed by a look at methods for specifying aspects of reactive systems, State Diagrams, Petri Nets, Statecharts, State Diagram Matrices. This has helped identify inheritance, behaviour and aggregation as important concepts that are insufficiently covered.

6.2 Contributions & Exceptions

The importance of aggregation has been highlighted for two main reasons: Firstly, aggregation exists in every system that is made up of components, in fact the terms “made up” and “aggregation” are intertwined. Secondly, even though aggregation was one of the reasons behind the work of Smith & Smith [Smith & Smith 77] on abstraction, it is generalisation which has received the best share of research and attention since.
This research has achieved the following objectives within the subject of OO systems analysis and design:

- Identified an interesting problem area that is insufficiently covered in literature on the subject – Inheritance of Behaviour in Aggregation.

- Showed the difficulties that designers are faced with when tackling this problem – Inheritance is difficult to represent, with the added behaviour factor, it is a complex issue.

- Restricted the circumstances that to target in order to achieve, within reasonable limits, a feasible solution – Determinancy, Single Events and Actions, Single Inheritance.

- Introduced a framework for solving the problem – Π, Γ and Φ behaviour models.

- Verified any new notions or theories – Proofs of theorems

- Demonstrated workability with a real-world example – Case Study

- Suggested future directions – Start by relaxing restrictions for example

The problem of representing inheritance (which is, at least, difficult when dealing with the static aspects of systems within the data abstraction hierarchies) has been shown to be a difficult task when the focus of the analysis and design process shifts to dynamic aspects of systems.

The process presented in this thesis for modelling complex systems consists of a set of steps and heuristics (see Appendix A and Chapter 4) for building the behaviour model of a system given, as a starting point, the respective behaviour(s) of the component objects that make up this system.
As a semantic extension to the behaviour model of the system (the $\Gamma$ model), the idea of system behaviour abstraction has been introduced. This (the $\Phi$ model) takes the behaviour of the system, as a unit, and with the input from the users of the system, starts a process wherein certain facets of the behaviour model can be abstracted or hidden. This results in a subset “view” of the functionality of the system that suits the particular requirements of the user. The thesis has also shown that this abstraction process can be applied at the level of components ($\Pi$), prior to the state combination process, to abstract/hide a subset of the behaviour of a particular component, and produce an abstracted $\Gamma$.

6.2.1 Advantage Over Other Approaches

This thesis has identified, and suggested methods to deal with, specific areas where there is lack of coverage in popular literature, e.g., OMT, OOAD and UML. This lack of coverage leads to insufficient problem understanding, which, in turn, leads to inadequate solutions to the problem of representing real world scenarios. These include, for example, where components in an aggregate interact in ways that cause chains of transitions, concurrency and synchronisation.

For the specific problem of behaviour inheritance within aggregation systems, this thesis has introduced a framework for capturing the behaviour of a system ($\Gamma$ model) based on the behaviour(s) of its components ($\Pi$ model). This framework also deals with the problem of inheritance of behaviour from the components to the system that these components constitute.

This framework takes the (state-based) behaviour models of the components of the aggregate object/system and derives, by state combination, the behaviour of the system. The state combination is a systematic process that not only takes into consideration the particular component’s behaviour model, but also incorporates the inter-component relationships and integrates those relations into the resulting behaviour model of the system.
This is carried out in a fashion that preserves all dynamic aspects of the components, such as states, transitions and actions. Furthermore, the hidden dynamic aspects of the system that exist between the components, but are not explicitly visible within the behaviour model of each component, are identified and dealt with (avoided or eliminated) as part of the process of constructing the behaviour of the system from the behaviour(s) of its components. These hidden dynamic aspects between components include, for example, the set of unreachable compound states, and chains of transitions, etc.

The thesis demonstrated that the framework can be made to work. This was done with the use of arbitrary examples as well as real world scenarios which show that the ideas here are, at least, semantically viable. While arbitrary examples within the thesis serve the purpose of demonstrating the notions and ideas as and when they are presented, the case study is a lot more than that. The case study collects in one place, all the ideas and notations and presents them with the use of a real world example. This example is simple enough to fit within the space of a chapter in a thesis, yet sophisticated enough to show the intricacies of inter-object messaging, chains of transition, concurrency and unreachable states.

6.2.2 Exceptions

From the case study in chapter 5, it can be seen that the associative property of the $\Gamma$ transformation is not as straightforward as had been anticipated. On applying this operation to a system of 3 or more components, we encountered a number of problems that implied that $\Gamma$ as a single operator was not associative.

To cope with this situation, we suggested that $\Gamma$ is in fact 2 distinct operators:

- $\Gamma_a$, covering only the exclusion of composite states excluded specifically by the system requirements.. e.g., states “Client: Single” and “Account:Overdrawn” cannot co-exist.
• Γb, covering state reduction and encapsulation. State reduction includes the removal of unreachable states for example, the ones that are dependent on transient states.

By encapsulation, we mean that the internal state of the composite machine formed by Γb can not influence further any distinct (external) machine, other than by messages (events) which cross the encapsulation boundary.

Given this separation of Γ into Γa and Γb, it is clear that Γa is associative, and that the operator may be applied recursively, in the manner discussed in Chapter 4 and Appendices D and E. The state reduction phase of Γb, however, is not necessarily associative when applied to subsets of the components of an object in the absence of encapsulation.

Thus, Γ is well-defined as a function of N arguments, but only Γa can be applied recursively.

### 6.2.3 Other Restrictions

The approach is also subject to other restrictions. These are mainly to do with the intricate nature of reactive systems. It would probably be a difficult task to handle these types of system and their behavioural aspects without some limitations on what to consider and what to exclude.

These limitations include:

• All Behaviour models must be deterministic. Within the behaviour model of a component, an event (stimulus) that can cause a transition from a state, can only cause one transition from that state, so that when that event does take place, the objects or component knows exactly which way to transit.
• While aggregation may imply multiple inheritance through the notion of one aggregate object is made up of multiple components, the work here specifically excludes multiple inheritance in aggregation hierarchies wherein a component can be part of two (or more) systems.

• Aggregation is the focus of the thesis, other abstractions are acknowledged but not dealt with.

• All behaviour models have one event at a time properties. While concurrency of transitions (responses to events) is accepted within the system, concurrency of events or actions is explicitly excluded.

The exclusion from the outset of all non-deterministic transitions was very important. The reason for this is that these kinds of scenarios can be difficult to handle in simple, single component, systems where the solution involves conflict resolution mechanisms.

Further, we have focused in this thesis on the concept of aggregation, and chose to ignore the (at least) three other modes or hierarchies of abstraction. This is because aggregation is the most common one with the least attention, and the fact that some abstractions (e.g., association) are still not clearly defined and used.

The other aspect of the field, which is conspicuous by its absence is the ability of aggregated systems to accept aggregated events and then distribute the right event to the right component wherein, concurrent responses and parallel streams of control can take place. While this is acknowledged by the author, it is considered a case of “running before walking”. This thesis demonstrates the enormous difficulties in dealing with the representation of the facets of behaviour in aggregated systems, it is essential that we develop a thorough understanding of what can take place in these systems in their simplest forms prior to tackling more complex ones.
6.3 Applicability

There are numerous areas of application for anything with the term Object in it. With a wide range of tools based around the concept, this is not surprising. For example, the work of [Harel 87, 88a, 88b] and [Brave 91] is applicable in areas as wide ranging as agriculture and defence. The notion of superstate or state of the aggregate is becoming widely used within the field of aviation design as well. Here, aircraft designers are interested in giving the pilot only the information that he/she needs to know that the craft is functioning properly. The pilot, for example is not concerned how the inner components of the craft respond to him/her increasing the throttle. Rather, he/she is only concerned that the aggregate system as a unit will reach the desired state (superstate) which could be ascending or descending away from danger.

In this section, we will look at two particular areas which are related directly to the subject of information systems design, these are distributed systems and OO programming languages.

6.3.1 Distributed Objects Systems

Within the area of distributed objects technology, data and business logic are encapsulated within the boundaries of the object, thus allowing these objects to be situated in a multitude of locations within the boundaries of the distributed system. The notion of distributed systems as a methodology for designing solutions of the client/server variety is beyond the scope of this thesis. We will look, however, at the main contributions of the technology that are directly related to the subject of this thesis.

According to [Orfali et al 96], objects and components within a distributed framework are not enough to build robust and flexible applications. These objects/components need to be packaged together as suites. Within the area of Client/Server systems development, it is these suites that permit the building of complex and tailored systems, i.e., from an off the shelf set of tools.
Distributed objects have been likened to independent software components that can exist under varying networks, operating systems or platforms. There is no doubt that the keyword in the distributed objects paradigm is Systems, or how objects work together across machines and networks to create client/server solutions.

From this, it can be seen that the notion of aggregation is at the heart of this emerging technology. Systems which are made up of pre-built components need to be checked, tested and verified to see whether the desired functionality has been achieved by these components. The reason for this is simple: A list of well thought, robust and tested components, when put together, do not necessarily yield a successful system. It is the state of the system as a unified whole, that matters after the assembly (which can be carried out across remote and internet based networks) has taken place. Therefore, the state of the system as a unit needs to be addressed and made accessible at all times within the lifecycle of this system.

6.3.2 Object Oriented Programming Languages

Microsoft's Visual Basic [Microsoft 99] has emerged over the past 10 years as arguably the most popular and widely spread development tool for systems which vary in complexity from simple static GUI's to vast multi-tier internet and intranet based enterprise solutions. In this section, we illustrate a scenario where the notion of states and dynamic behaviour can actually be put to use within the context of developing Visual Basic solutions.

In an article published by Fawcettes Visual Basic Programmers Journal (VBPJ), [Malluf 95] presents a good demonstration of the workability of the state machine theory for data access within the capabilities of Visual Basic.
The main idea in this is to break up long operations into smaller parts that can be executed under specific states to provide specific parts of the overall required functionality. This is proposed as a method within the design of database frontends in order to enhance the robustness of the application. This, it is proposed, is particularly useful in applications that handle multi-user contention, validation and record movement.

Each activity in a state is viewed as a separate thread of control. As far as the state of the system is concerned, there is one superstate, and many threads of control (activities in the substates). These threads can be run in a single time segment or incrementally over several time segments which can occur as several visits to the states that contain these activities.

A rule-set is pre-defined for the application through a state transition diagram, this controls which transition is the next to execute after a state has completed its activity, hence a move to a new state where a new activity can begin is instigated. For example, there is an initial state “Database Closed”. From this state, the only permitted transition is one to the “Database Open” through an “Open” message.

In terms of benefit to the users, the application works as follows: The application can offer the user the function to, say, start a report generation process from a subset of the tables. Before this process terminates the user can also start adding data to another subset of tables, which can trigger an additional validation process, but before the report process has completed. The timer object, in conjunction with the state transition diagram will trigger the next process following the termination of, either of the current processes. Or start a completely new process based on the user’s input. This frees the programmer from the need to keep a check on Visual Basic generated events like “Got_Focus” and “Lost_Focus” which occur when a control or object gains the compiler’s attention or loses it respectively. This also eliminates the need for third party Windows Messaging control tools which sometimes are the only methods of telling if a process has terminated.
The basic idea here is in fact to use the state transition diagram as a rule base for
the applications timer, which will step through the process list, hence system
states, in a predefined manner (predefined at the level of each thread of control).
This timer object itself is the main problem with this implementation of state
based solution to complex design issues is that Visual Basic. Unlike C++, Visual
Basic does not support real multi-threading [Dietel 94], the final result, therefore,
is a crude implementation of an otherwise innovative approach to designing
complex systems.

6.3.3 Other Benefits

Two of the areas in systems design that consume large proportions of effort from
designers and users alike are Bottlenecks and Redundant Code. With respect to
the first of these, this research has highlighted a method which makes available to
systems designers as well as users a snapshot of the likely dynamic scenarios that
their system is likely to go through. This behaviour model shows every
conceivable functionality of the system in one place (as in the case with the Γ
abstraction level), or a subset of the significant behaviour of the system (in the
case of Φ abstraction level). It can, therefore, be used to pinpoint potential
trouble spots of the system, such as Bottlenecks and Deadlocks. Adjustments can
then be made by eliminating the offending events/activities/sequences before a
single line of code has been written.

The second aspect of systems design (and programming in particular) is
manifested with exceptional cases and scenarios. Here the designers and
programmers have to put in 90% of the design and coding effort to cater for a
subset of scenarios that occur in only 10% of the time in the system life. The
extreme example of this situation implies that these scenarios which are supposed
to cover 10% of the system functionality, in fact never take place. It can be
almost impossible to predict what sequences of events are likely to yield legal
scenarios (within the rules that govern what events and states the system goes
through). However, with the behaviour modelling techniques that are proposed
here, it is quite possible to pinpoint a subset of the systems behaviour which is completely unreachable, as shown in the subject of state reachability in this thesis, and then remove it at an early stage and save on the resources utilised for the design, implementation and support of the project or system.

6.4 Further Work

With every research project, it is inevitable that new ideas and technologies will emerge at all stages of the project. The main ideas for future work are be based on the relaxation of the restrictions that this thesis has imposed, suggestions for future work are presented here.

This research has focused, from the outset, on aggregation and behaviour. However, semantic data models suggested many other ideas and methods for abstracting data including association and specification. While the issue of Specification was beyond the scope of this thesis, the thesis mentioned in the proposed approach chapter that, in our opinion, association is a special case of aggregation. This should, at least in part, imply that some of the ideas here could be applied to the association concept. However, we suggest that a useful area of further research would be to explore this assumption.

More significantly, a totally distinct method for reducing systems behaviour model complexity has been a background issue, but was also left to this section. This idea is the use of **simulation** tools to run through all the possible event scenarios of the systems behaviour model, and mark states as they are reached, by the end of the simulation, which can be as long and sophisticated as required, all non-reachable states would be easily identifiable. While the results of this idea may not be as accurate as, say, the $\Gamma$ model above, they can nonetheless provide a good indication of likely behaviour scenarios. This can then be utilised in building the system by building those aspects of the system with most activity and moving on to less significant parts of the system at later stages.
Another extension that makes an interesting future project will be the idea of a case tool, or an automation program. This application can start with behaviour models of components (which are easily presentable to a computer program using decision table formalism for instance), take all the rules presented here, and build the behaviour model of the perspective system from these components. Such a system could then be extended not only to abstract the behaviour model to create the (Φ level), but also to re-create (Γ) models as and when new components are added to the system and old ones removed.
APPENDIX A

Steps For Generating the \( \Gamma \) Model from the Behaviour Description of an Aggregate System’s Components, \( \Pi \).

- Starting Point

Starting with a set of behaviour specification(s) for each component of the aggregate, independently. For Example, Scheme, Member, Contributions, Returns. This level also shows the set of conditions under which these components can interact or share events and transitions.

1. Aggregate (\( \Gamma_a \)) behaviour descriptions for each two components in the group. For example Scheme-Member, Contributions-Returns as one option, or Scheme-Returns, Contributions-Member as another option. As a guideline, the two objects with the most likelihood of producing combined states which are likely to be prohibited (By State Elimination Rules) should be combined together.

2. Remove prohibited states. These include unreachable states, and semantically prohibited states and dangling states (ones with no transitions to them). Mark Concurrent transitions, Transition Chains and other modes of special interactions. Note: If 2 states, say \( S_{10} \) and \( S_{20} \) (one from each component) concurrently change to a new combined state, say \( S_{11}, S_{21} \) as a result of a shared event \( e \), (sometimes referred to as diagonal transition), this will initially imply that intermediate states (\( S_{11}, S_{20} \) and \( S_{10}, S_{21} \)) are potentially unreachable. If, on adding additional components to \( \Gamma_a \), any diagonal transitions (concurrent ones) are prevented from taking place, then \( \Gamma_a \) will change to one of the intermediate states (which were originally potentially unreachable), with the original combined destination state, \( S_{11}, S_{21} \) becoming the potentially unreachable one. That is why states which are unreachable as a result of chain transitions should not be removed until the \( \Gamma_b \) stage.
3. Still within $\Gamma_a$, repeat Step 2 by treating the new combined states, for example, $S_{11}, S_{21}$ as single component states. The same restrictions apply to the rules of interaction between components, i.e., Do NOT apply the interaction rules such as chains of transitions and concurrency as yet. For dealing with complex systems with many states, use State Transition Tables, which seem to be a good medium for representing the complexity of the system without loss of semantics, they are also capable of coping with the additional semantic factors, such as concurrent transitions. In state transition tables, one solution for system behaviour representation is to have all the states of the overall system (including source states and destination states) on the $y$-axis and a copy of the same states on the $x$-axis. The cells of the table are the transitions/events that cause the change from the source to the destination.

4. If the State Elimination Rules apply to a state in step 3 (or 2) then that state is removed, also any state which is exclusively reachable from that state should be removed. For every state that is removed, ALL attached (TO and FROM) transitions are also removed from the system. This is still within the $\Gamma_a$ stage.

5. Once all components are in the final $\Gamma_a$, we can start to apply interaction rules, this is where $\Gamma_b$ starts. For example, transition chains which can result in transient states. The basic rule here is that no transitions should be made from a transient state, with the exception of the chain transition that caused this state to become transient in the first place (see chapter 4). If a state is reachable from a transient state via a Non-chain transition, then that state is in fact unreachable, and hence needs to be eliminated from the system together with all its dependent states.

6. Step 5 may require repetition depending on the number of interactions. Apply further modes of interaction, such as concurrency (replace each multiple transitions on the same row of the decision table with a combined one and mark it). This step will invariably result in new unreachable states, as well as transient ones.
7. Search for unreachable states. A systematic method for identifying these states is described below.

8. When all interaction rules and modes are onboard, then we are ready to start looking at methods of identifying dangling states. Those are states which seem quite reachable from others in the system, but however, are never reachable through any sequence of events from the initial state in the system.

Identifying Unreachable States

9. The most reachable state is the initial state

10. The second level of reachability applies to all states which are directly (via one event/transition) reachable from the initial state.

11. The third level of reachability applies to all states which are directly (via one event/transition) reachable from the second reachability level states.

12. Repeat until all transitions are checked in.

13. All remaining states are unreachable or dangling.

14. Remove dangling/unreachable states, and re-iteratively, look for dependent states which are exclusively reachable from these, remove and repeat.

15. The reachable state, farthest (in terms of the number of transitions) away from the initial state indicates what is called the depth of the behaviour model.
16. The examples of this thesis have shown that not all systems do in fact have a final state, whereas, all systems DO require an initial state, a final state can be explicitly defined or it can be semantically identified. In this case there is no reason why a system cannot have several final states.
APPENDIX B

Why \( \Pi \) to \( \Gamma \) Transformation is Divided into 2 Stages

The reason the \( \Pi \) to \( \Gamma \) transformation is split into \( \Gamma a \) and \( \Gamma b \) is as follow:

When combining state machines relating to 3 components, \( M_1, M_2 \) and \( M_3 \). When this is done in two steps:

\[
\Gamma (M_1.M_2) \quad \text{referred to as} \quad \Gamma' \\
\Gamma (\Gamma'.M_3) \quad \text{referred to as} \quad \Gamma^* 
\]

In exceptional circumstances, a state or a transition in \( M_3 \), will be dependent on a behaviour facet (state or transition for instance) in \( M_1 \) or \( M_2 \), which may seem redundant in \( \Gamma' \). It is immature to delete (\( \Gamma \) removes redundant behaviour, e.g., unreachable states) any of the (seemingly redundant) states or transition when building \( \Gamma^* \). This is because the redundant behaviour facet may become necessary as the (still excluded) component, \( M_3 \) is added.

One example of behaviour facets which are guaranteed not to be required by the components outside of \( \Gamma^* \) are the set of Unreachable States. If a state is unreachable in \( \Gamma' \), then all combinations of that state (achieved by the cross product with states of \( M_3 \) on the path to \( \Gamma^* \)) are also unreachable, hence removing those states in \( \Gamma' \) saves a great deal of effort.

The main reason for not allowing this application of interaction rules from early stages of the \( \Gamma \) construction has to do with State Reachability. For example, figure B.1 shows three components with some simple interactions and shared events.
**Rule**

$M_1$ cannot change from $S_0$ to $S_1$ if $M_2$ in $S_0$

*Figure B.1 – Three Components System*

When combining $M_1$ and $M_2$, see figure B.2, the combined system will change from the state $S_0,S_0$ to state $S_1,S_1$, thus rendering the states $S_1,S_0$ and $S_0,S_1$ unreachable.

*Figure B.2 $\Gamma(M_1,M_2)$*

However, at least one of these states will be required when the complete $\Gamma(M_1,M_2,M_3)$ is generated, by adding the behaviour of $M_3$ to the system. This is because, as a result of the above rule. In $\Gamma$, the state $S_0,S_0,S_0$ can only change to state $S_0,S_1,S_0$. If $S_0,S_1$ had been eliminated at the previous level (figure B.2) then the resulting behaviour model in $\Gamma(M_1,M_2,M_3)$ would be inaccurate.
This section is an explanation of the notation used in this chapter for modelling dynamic aspects of reactive systems. The notation is based on the idea of state transition diagrams, and the extensions made to those by [Harel 87, 88a, 88b] to cater for additional complexity. Further, special notations have been added to cope with the additional semantic requirements, for example, combined states from multiple components.

- State $S_1$ of Machine $M_1$ (the term Machine and Component are used interchangeably and both refer to an object in the aggregation hierarchy)

- Event ($e_1$) causes a transition from State $S_1$ in $M_1$ to State $S_2$ in the same component.
• Combined State, from two machines, i.e., a compound state, sometimes referred to as superstate

- A transition caused be event \( e_2 \) with constraint \( (C_x) \)

- A transition caused be event \( e_2 \) with output \( (e_3) \)

- A transition caused by the transition caused by event \( e_2 \), rather than \( e_2 \) itself.

- A transition caused by event \( e_2 \), reaches a state \( (S^*) \). On entry to State \( S^* \) (normally a Superstate or combined state from multiple components), this transition triggers an action which causes state \( (S^*) \) to change instantly. Such a scenario results in \( (S^*) \) becoming a transient state in cases where \( S^* \) is not reached by other transitions.
- A chain of transitions caused by event \( (e_2) \) \( S^* \) is in the path of the chain.
APPENDIX D

CONCEPTS, DEFINITIONS AND RESTRICTIONS

There are a few concepts that are used repeatedly. These are explained here. Note that all of these concepts are familiar from the OO design methodologies in chapters 2 and 3, this section re-defines these concepts within the context of this research (the reader is encouraged to refer to the various OO design methods for their version each concept).

State: The stage of functionality the object is in at the current moment in time. In other words, what the object is doing now. In the sections below we shall encounter the following types of states:

Initial State: The first state of the object when it is created/instantiated.

Final State: Although this is explicitly acknowledged by many OO design methods, in this research it is defined implicitly, there is no reason why a behaviour model for an object does not have many final states. If an object can not move out of a state for lack of suitable events, then it (that state) is by its nature, a final one.

Transient State: Refer to the notation section (4) above. This is basically a state with in which the object remains for near 0 time, hence not no activity can take place here.
Unreachable State: A state which exists in the system specification, but is impossible to reach (from a defined initial state) given all possible iterations of events in that same system specification.

Superstate: When building a behaviour model, B, by combining two behaviour models, B₀ and B₁, all states in B will be of the form “B₀Sₓ,B₁Sᵧ”, where B₀Sₓ is state Sₓ from the behaviour model B₀ and state B₁Sᵧ is state Sᵧ from the behaviour model B₁. In many cases, where the source of the states is obvious (for example when there are only two objects in the system) then “B₀Sₓ,B₁Sᵧ” is shown as “Sₓ,Sᵧ”.

Event/Stimulus: Anything that takes place in the object’s environment, this can be from the real world surrounding the system, or from another object within the system. This thesis deals only with single stream events. Two distinct stimuli taking place at the same time is not allowed.

Transition: When an event occurs on an object which is in a state, if this event is accepted by the current state of the object, then the object will respond by moving to another state, this move is a transition.

Action: An (optional) event that lies on the entry to a state, the exit of a state, or just on the transition from one state to another. Actions from one object can be acceptable as events in another object within the system (see Chains of Transitions below). This thesis does not deal with State Entry Actions and State Exit Actions, and all actions are associated with transitions between states.

Activity: The functionality the object is executing within a certain state.

Behaviour: Any possible sequence of the form “State, Event, Transition, Action, State” that describes a state change in an object as a response to some events.
**Insignificant Behaviour (semantically):** The system users can (through business rules for instance) pick (arbitrarily) states, events, transitions and other behaviour concepts, which they decide can be hidden. (this is applicable in the move from the $\Gamma$ to the $\Phi$ model).

**Redundant Behaviour:** The $\Pi$ behaviour model can show some states, transitions and other behaviour facets which will never take place within the available list of stimuli and states, $\Gamma$ will delete all of these types of behaviour.

**Behaviour Equality:** For two behaviour models ($A$ and $B$) relating to an object $O$, to have behaviour equality, then:

- For every behaviour that model $A$ displays (for example, on event $X$, object $A$ moves from state $S_9$ to state $S_{21}$), then model $B$ should do exactly the same as a response to that same event ($X$).

- For every behaviour that model $B$ displays (for example, on event $Y$, object $A$ moves from state $S_4$ to state $S_2$), then model $A$ should do exactly the same as a response to that same event ($Y$).

**Behaviour Inconsistency:** When Behaviour Equality cannot be guaranteed.

**Behaviour Consistency:** For two behaviour models ($A$ and $B$), relating to an object $O$, to have behaviour consistency, then one behaviour model (say $B$) must be an abstraction (refer to the Phi abstraction above) of the other behaviour model ($A$) where:

- $B$ hides all insignificant behaviour
- The remaining behaviour (which $B$ does not hide) is Equal to that in $A$
Autonomous Behaviour Model: When building a $\Gamma$ behaviour model from the $\Pi$ behaviour model of a system of more than 2 components, it is sometimes required that this is done in stages. First, construct a $\Gamma a$ (see chapter 4) for each 2 components in the system. Secondly, combine the results from all $\Gamma a$ models. The only complete behaviour model is the final one that includes all of the $\Gamma a$ models. This $\Gamma a$ is completely autonomous within its boundaries. That is to say there are no events, states or other behaviour facets that depend for their existence on other behaviour facets which are still not shown in this $\Gamma a$.

Behaviour Model Boundary: Is the barrier or border that distinguishes an internal event (one that originates in another object in the system) to from all other events.

Transition Forms in $\Gamma$:
All behaviour in $\Gamma$ models generated from a $\Pi$ with 2 components are of the form:

$$M_1 S', M_2 S' \xrightarrow{e/a} M_1 S'', M_2 S''$$

Where "$M_i$" is machine, "$S'$" is source state, "$S''$" is destination state, "$e$" is the event that caused the transition (shown as directed line) and "$a$" is the -optional- action associated with the transition and takes place on reaching the destination state.

All behaviour in $\Gamma$ models generated from a $\Pi$ with 3 components are of the form:

$$M_1 S', M_2 S', M_3 S' \xrightarrow{e/a} M_1 S'', M_2 S'', M_3 S''$$

Other behaviour relating to more components is analogous.

Assumptions and Restrictions
The restrictions imposed on all systems in this thesis are:

- In $\Gamma$, the compound states $(M_1S', M_2S''')$ on the one hand and $(M_2S'', M_1S')$ on the other both result from the same $\Pi$ scenario; which implies that component $(M_1)$ in state $(S')$ and component $(M_2)$ in state $(S'')$. This implies that ordering of state labels is insignificant.

- All state transitions/events/stimuli in $\Pi$ are deterministic, i.e., no component will ever fail to respond to a stimulus due to multiplicity of possible transitions from current state. For example,

Scenarios like this are not covered by this thesis.

- All stimuli in $\Pi$ conform to the single events stream property. This implies that if $(M_1)$ requires stimulus $(e_1)$ to change to $(S')$ and if $(M_2)$ requires $(e_2)$ to change to $(S'')$, then the two objects cannot behave concurrently on occurrence of a new event. This implies that, at most, one of the two objects will change state on the next stimuli occurrence. This also applies to stimuli from actions on transitions or state entry and state exits. If two machines respond concurrently to the same event, then only one of them is permitted to have an action associated with its transition if that action is an event in the system.

- With the exception of Unreachable States and Redundant Behaviour, all the properties of the $\Pi$ behaviour model are preserved by the $\Gamma$ model.
APPENDIX E

Rules For Π to Γa Transformation

The transformation from Π to Γ is a 2-stage process. We deal with the first stage here, from Π to Γa. Assume Π has two components M₁ and M₂. All transitions that takes place in some Mᵢ, conform to the form:

\[ Mᵢ, S' \xrightarrow{e/a} Mᵢ, S'' \]

where

S’ is source state,
S” is destination,
e is stimuli that caused the transition
a is (optional) action associated with the transition

1 Rules When One Components Responds

R100

If Π has M₁ and M₂ with the following behaviour(s):

\[ M₁, S' \xrightarrow{e/a} M₁, S'' \]

M₂ No Change
There exists another version of this case, where the responding machine is $M_2$.

2 When Two Components Respond Concurrently

In this case only one action ($e / a$) is allowed at a time.

R200

If $\Pi$ has $M_1$ and $M_2$ with the following behaviour(s):

\[
\begin{align*}
M_1, S' & \xrightarrow[e / a]{} M_1, S'' \\
M_2, S' & \xrightarrow[e / \sim]{} M_2, S''
\end{align*}
\]

$\Gamma_a(M_1, M_2)$ contains:

\[
\begin{align*}
M_1, S', M_2, S' & \xrightarrow[e / a]{} M_1, S'', M_2, S' \\
M_1, S', M_2, S' & \xrightarrow[e / \sim]{} M_1, S'', M_2, S'
\end{align*}
\]

There are two versions of this rule, the other one being when the action is associated with $M_2$. 
3 Rules When Two Components Respond Consecutively - Chains

**R300**

If $\Pi$ has $M_1$ and $M_2$ with the following behaviour(s):

\[
\begin{align*}
M_1 S' & \xrightarrow{e/x} M_1 S'' \\
M_2 S' & \xrightarrow{x/a} M_2 S''
\end{align*}
\]

$\Gamma_a (M_1, M_2)$ contains:

\[
\begin{align*}
M_1 S', M_2 S' & \xrightarrow{e/x} M_1 S'', M_2 S' \\
M_1 S', M_2 S'' & \xrightarrow{e/x} M_1 S'', M_2 S''
\end{align*}
\]

An alternative version of this rule has machine $M_2$ at the first half of the chain and $M_1$ at the second.

4 Rules When Two Components Respond Consecutively - Independently

**R400**

If $\Pi$ has $M_1$ and $M_2$ with the following behaviour(s):

\[
\begin{align*}
M_1 S' & \xrightarrow{e/-} M_1 S'' \\
M_2 S' & \xrightarrow{x/a} M_2 S''
\end{align*}
\]
Ga \( (M_1, M_2) \) contains:

\[
\begin{array}{c}
(M_1 S_1, M_2 S_1) \xrightarrow{e'} (M_1 S_2, M_2 S_1) \\
\downarrow x/a \downarrow x/a \\
(M_1 S_1, M_2 S_2) \xrightarrow{e'} (M_1 S_2, M_2 S_2)
\end{array}
\]
APPENDIX F

PROOF OF THEOREM C1

C1 asserts that:

\[ \Gamma a (\Gamma a (M_1, M_2), M_3) \]

is equivalent to

\[ \Gamma a (M_1, \Gamma a (M_2, M_3)) \]

We can pair off the states of \( \Gamma a (\Gamma a (M_1, M_2), M_3) \) and \( \Gamma a (M_1, \Gamma a (M_2, M_3)) \). A state \((M_1S, M_2S, M_3S)\) is paired with \((M_1S, (M_2S, M_3S))\). To complete the proof, we need to show that if a transition such as the one below (referred to as "T*" henceforth)

\[ ((M_1S_1, M_2S_1, M_3S_1)) \xrightarrow{e/a} ((M_1S_2, M_2S_2, M_3S_2)) \]

exists in \( \Gamma a (\Gamma a (M_1, M_2), M_3) \) (henceforth referred to as \( \Gamma a' \)) if and only if the same transition

\[ (M_1S_1, (M_2S_1, M_3S_1)) \xrightarrow{e/a} (M_1S_2, (M_2S_2, M_3S_2)) \]

exists in \( \Gamma a (M_1, \Gamma a (M_2, M_3)) \), referred to as \( \Gamma a'' \).
To show this, there is a need to carry out a case analysis to consider every possible scenario/behaviour possibility that can arise in the system. These scenarios are all based around the question: “Which component(s) respond to the stimuli?”

**Case 1 – Two Components in Π Respond to Stimulus Concurrently and One Component Responds via Chain Transition**

This case is referred to as the Chain Effect. When the stimulus takes place, one of the three component objects responds, and generates an action on that responses transition. This action is consequently picked by a second component. The net effect of this is not very dissimilar to a *cascade* effect. It is this kind of scenario that generates potential transient states.

In this case, T* is generated by a state transition in all component. M₁ and M₂ share a chain transition, and M₃ shares the concurrent transition with M₁. We consider the case in which the triggering action (x) originates in M₁, T* is made up of:

$$
\begin{align*}
M₁S' & \xrightarrow{e/x} M₁S'' \\
M₂S' & \xrightarrow{x/a} M₂S'' \\
M₃S' & \xrightarrow{e/-} M₃S''
\end{align*}
$$
• Build $\Gamma a'$ - $\Gamma a (M_1, M_2, M_3)$

First, we build $\Gamma a (M_1, M_2)$. Rule R300 implies this to be

Next we add $M_3$. Using rule R200 from appendix E, this results in:

• Build $\Gamma a''$ - $\Gamma a (M_1, \Gamma a (M_2, M_3))$

We start by building $\Gamma (M_2, M_3)$. This needs to be combined as two machines responding to two distinct events $M_2$ responds to event $x$ and $M_3$ responding to event $e$. Based on rule 400 in appendix E, $\Gamma a(M_2, M_3)$ is

Adding $M_1$, with rule 300, results in
This is identical to $\Gamma a'$. The rest of the case scenarios are analogous to the one above and can be proved in a similar fashion.
APPENDIX G

TABLES AND FIGURES FOR CHAPTER 5
Figure 5.11
(unreachable states removed)
Figure 5.18

Based on SILRC

The full Q from a (S,R,C)
<table>
<thead>
<tr>
<th></th>
<th>10</th>
<th>4</th>
<th>8</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2.9 Initial (F, C) - No Interactions Shown
<table>
<thead>
<tr>
<th></th>
<th>4</th>
<th></th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>6</td>
<td>4</td>
<td>12</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>12</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>12</td>
<td>12</td>
</tr>
</tbody>
</table>

Note: In order to eliminate non-determinancy, the states

SY'C'p'RC and SY'C'c'RC also had to be combined.
Rule 5.39: Contouring Applied

<table>
<thead>
<tr>
<th>Rule</th>
<th>4</th>
<th>4</th>
<th>4</th>
<th>4</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
</tbody>
</table>

Table 5.39: Contouring Applied
BIBLIOGRAPHY

[Banchillon 88]
Banchillon F. Object-oriented database systems. 7th ACM SIGART-SIGMOD-SIGACT
Austin Texas 1988

[Banerjee et al 87]
Data model issues for object-oriented applications.
ACM Trans. Office Information. Systems. 5(1), Jan. 87

[Bertino & Montesi 92]
Bertino E.; Montesi E. Towards a logical-object oriented programming language for
databases.
Proc. of 3rd Int. Conf. on Extended Database Technology (EDBT),
Vienna, Austria March 92

[Bertino et al 92]
Bertino E.; Negri M.; Pelagatti G. and Sbatella L.
Object-oriented query languages: the notion and the issue.
IEEE Trans. on Knowledge and Data Eng. 4(3) 223-227

[Berthomieu 91]
Berthomieu B., Diaz M. Modeling and Verification of Time-Dependent Systems Using

[Bloom and Zdonik 89]
Bloom T. and Zdonik S. Issues in the design of object-oriented programming languages.
OOPSLA 87 Orlando, FLorida Oct 89

[Booch 91]
Booch G. Object Oriented Design with Applications.
Benjamin Cummings 91

[Booch 97]
Booch G., Rumbaugh J., Jacobson I. The Unified Modeling Language Semantics and
Notation Guide 1, Rational Software Corporation 1997, San Jose CA.

[Borgida et al 89]
Borgida A.; Brachman R.; McGuinness D. and Resnick L. CLASSIC: A structural data
model for objects. Proc. of ACM-SIGMOD Int. Conf. on Management of Data.
Portland OR, May/June 89
[Bouzeghoub 97]
Bouzeghoube M., Gardarin G., Valduriez P.
Object Technology. Software Engineering Series.
Thomson Computer Press 1997

[Bowers 93]
Bowers D.S. Some Principles for the encapsulation of the behaviour of aggregate objects.
In Computing and Control Division, Colloquium on "Recent Progress in Object Technology" Digest No. 1993/238 Dec. 1993

[Brachman 79]

[Brave 91]
Brave Y and Heyman M. Reachability in Discrete Event Systems Modeled as Hierarchical State Machines.
Proc 17th Convention IEEE, Israel May 1991

[Brave 93]

[Brodie 84]

[Brodie & Silva 82]

[Brodie and Ridjanovic 84]
[Brooks 87]
IEEE Comp. April 10-19 (1987)

[Cacace et al 90]
Cacace F.; Ceri S.; Crespi-Reghezii S.; Tanca L. and Zicari R.
Integrating object-oriented data modelling with a rule-based programming paradigm.
Proc. of ACM-SIGMOD Int. Conference On Management of Data,
Atlantic City, NJ, May 1990

[Camurri et al 92]
Camurri A., Franchi P., Vitale M. Extending High-Level Petri Nets for Object-Oriented
Design.
IEEE International Conference on Systems Man and Cybernetics Vol 1 P. 623 1992

[Ceri 92]
Ceri S. A declarative approach to active databases. In Proc of the 8th IEEE Int. Conf. on
Data Eng.,
Phoenix AZ, Feb 92

[Chen 76]
Chen P. The entity-relationship model: Toward a unified view of data. ACM
Transactions on Database Systems 1, 1 1976 pp 9-36

[Cheng 83]
Cheng A & Navathe S. A Methodology for Database Schema Mapping from Extended
Entity-Relationship Models into the Hierarchical Model,
in Entity relationship Approach to Software Engineering (eds Davis G. C. et al.)
Elseveir, North-Holland, New York 1983

[Coad 90]
Coad P. & Yourden E. Object-Oriented Analysis. 2nd Edition

[Codd 70]
Codd E.F. A Relational Model of Data for Large Shared Data Banks.

[Codd 79]
Codd E.F. Extending the Database Relational Model to Capture More Meaning.
ACM Transactions on Database Systems 4, 4 Dec 79, pp397-434

[Coleman 92]
Coleman D, Hayes F and Bear S. Introducing ObjectCharts or How to Use StateCharts
in Object-Oriented Design.
IEEE Transactions on Software Engineering, Vol 18, No 1, 1992
[Coleman 94]
Coleman D., Arnold P., Bodoff S., Dollin C., Gilchrist H., Hayes F., Jeremes P.

[Date 77]
Date CJ. An Introduction to Database Systems.
Addison Wesley 1977

[Davis 88]

[Davis 93]
Davis A.M. Software Requirements: Objects, Functions and States.
Prentice Hall 1993

[DeMarco 78]
Yourden Press 1978

[Desutsch 91]
Object-Oriented software technology. Computer (IEEE Computer Society), 24(9), 112-113

[Deux et al 90]
Deux O. et al The stroty of O2. IEEE Trans. on Knowledge and Data Eng. 2(1), March 90

[Deitel 94]
Deitel HM., Deitel PJ.
C++: How to Program
Prentice-Hall 1994

[Edwards 93]
Edwards JM and Henderson-Sellers B. A Graphical Notation for Object-Oriented Analysis and Design.
JOOP 1993

[Elmasri 89]
The Benjamin/Cummings Publishing Company Inc 1989

[English 93]
English S. Coloured Petri Nets for Object-Oriented Modelling.
A PhD thesis at University of Brighton, England, June 93
[Eriksson 98]

[Eva 89]
Eva M. SSADM - 4: A User's Guide

[Falkenberg 96]
Falkenberg D., Hesse W., Lindgreen P., Nilssen B., Han Oei L., Rolland C., Stemper R.,
Assche Van F., Verrijn-Stuart A., Voss K.
A Framework of Information Systems Concepts
The FRISCO Report 1996

[Farmer 84]
Farmer D.; King R.; and Myers D. A tool for the implementation of databases.
Proceedings of the International Conference on Data Engineering (LA, Calif.)
IEEE New York 19984, pp386-393

[Farmer 85]
Farmer D., King R., and Myers D. The Semantic Database Construct.
IEEE Transactions on Software Engineering 85, SE-11, 7, pp583-591

[Fishman et al 87]
Fishman D. et al.
IRIS: an object-oriented database management system.
ACM Trans. on Office Info. Syst. 5(1) January 87

[Flavin 81]
Flavin M. Fundamental Concepts of Information Modeling
Yourden Press 1981

[Fowler 97]
Fowler M
Survey of Object Oriented Analysis and Design Methods
IEEE, Los Alamitos, CA, USA

[Fowler 97]
Fowler M, Scott K. UML Distilled, Applying the Standard Object Modelling Language

[Ghezzi et al 91]
Ghezzi C., Mandrioli D., Morasca S., Pezze’ M. A Unified High-Level Petri Net
Formalism for Time-Critical Systems.
[Gill 81]
Gill A. Introduction to the Theory of Finite-State Machines.

[Goldberg 84]
Addison-Wesley Reading, MA 1984

[Guttag 77]
Guttag JV. Abstract data types and the development of data structures.
Comm. of ACM, 20, 396-404

[Gray et al 92]
Gray PD.; Kulkarni KG. and Paton NW. Object-oriented databases: A Semantic Data
Model Approach.
Prentice-Hall 92

[Halbert 87]
Halbert D.C., O’Brien, P.D. Using Types and Inheritance in Object-Oriented
Programming.
IEEE Software 1987

[Hallahan 99]
Hallahan S. Small Objects of Desire
PC Week, 16 March 1999, Page 38-39

[Hammer 81]
Hammer M. and McLeod D. Database Description with SDM: A Semantic Database
Model.
MIT in ACM TODS Vol 6 No 3 Sept 1981 351

[Harel 87]
Harel D. Statecharts: a visual formalism for complex systems.
Science of Computer Programming 8 (1987), 231

[Harel 88a]
Harel D., Lachover H., Naamad A., Pnueli A., Politi M., Sherman R., Shtul-Trauring S.
STATEMATE: A working environment for the development of complex reactive
systems.
Proceedings of the 10th IEEE International Conference on Software Engineering.
Singapore April 88

[Harel 88b]
Harel D. On Visual Formalism.
Communication of the ACM 31, 5 (May 1988), 514-530

[Hipperson R.]
Prentice-Hall, Englewood Cliffs NJ. 1992
[Hoare 78]
Hoare CAR. Communicating Sequential Processes.
Communications of the ACM 21, 8, AUG 78, pp 666-677

[Hopcroft 79]
Hopcroft JE and Ullman JD. Introduction to Automata Theory, Languages and
Computation.
Addison-Wesley, Reading Mass 1979

[Howard 99]

[Howe 83]
Howe D. R. Data Analysis for Data Base Design
Edward Arnold (Publishers) Ltd. 1983

[Hoydalsvik 93]
Hoydalsvik G.M., Sindre G. On the Purpose of Object-Oriented Analysis.
OOPSLA '93.

[Huber 91]
In Rozenberg G., (ed.) Advances in Petri Nets 90, LNCS Vol.483,
Springer-Verlag 91, 313-341

[Hughes 91]
Hughes J. G. Object-Oriented Databases
Prentice-Hall Int, UK Ltd. 1991

[Jacobson 92]
Addison-Wesley, 1992.

[Jensen 86]
In Brauer, W., Reisig, W. and Rozenberg G., (eds.).
Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986 Part I,
LNCS Vol. 254,
Springer-Verlag 1987, 248-299

[Jensen 91]
Jensen K. Coloured Petri Nets: A High-Level Language for Systems Design and
Analysis.
In Rozenberg G. (Ed.) Advances in Petri Net 1990.
[Jantzen 79]
Net Theory and Applications, Proceedings of the Advanced Course on General NEt
Theory of Processes and Systems, Hamburg 1979,
LNCS Vol 84, Springer-Verlag, 165-213

[Kappel 89]
Kappel, M. Shrefl. A Behaviour Integrated ER Approach for the Design of Object-
Oriented Databases.
In Proceedings of the 7th International Conference on the ER approach 1989

[Kenneth 92]

[Kent 98]
Kent S., GJ, J
Visualizing Action Contracts in Object-Oriented Modelling

[Kim 87]
Kim W. et al. Object-oriented database support for CAD. MCC Tech. Report, ACA-ST-
293-87, Sept. 87

[Kim 90]
Kim W. An introduction to object-oriented databases. MIT Press 90

[Kimelman 91]
Kimelman D.N., Ngo T.A. The RPS program visualization environment.
The IBM Journal of Research and Development, 35(6), Nov. 1991

[King 84]
King R., McLeod D. A Unified Model and Methodology for Conceptual Database
Design.
In On Conceptual Modelling, Perspectives from Artificial Intelligence, Databases, and
Programming Languages.
Brodie ML., Mylopoulos J., Schmidt JW. (Eds.) Springer-Verlag 1984

[King 86]
King R., McLeod D. The Event Database Specification Model.
Proceedings of the 2nd International Conference on Databases: Improving Usability and
Responsiveness, Jerusalem, Israel, IIPA, 84, pp.299-322

[Kradie 86]
Kradie C. Refinement Through Classes. A development Methodology for Object-oriented
Languages
University of Illinois, Urbana IL 1986
[Lalonde 85]
Lalonde W. and Pugh J. Specialisation, Generalisation, and Inheritance.
Teaching Objectives Beyond Data Structures and Data Types.
SIGPLAN Notices Vol 20 (8) 1985

[Ling 93]
Ling TW. and Teo KP. Towards Resolving Inadequacies in Object-Oriented Data Models.
Info and Software Techno. 35(5) May 93

[Liskov 86]
Liskov B. and Guttag J. Abstraction and Specification in Program Development.
Cambridge, MA: The MIT Press 1986

[Loomis 87]
ECOOP, Paris France 15-17/June 87
Published as Lecture Notes in Computer Science, 276: Springer-Verlag

[Martin 77]
Martin J. Computer Database Organisation
Prentice Hall 1977

[Martin 85]
Martin J. & McClure C. Designing Techniques for Analysts & Programmers.
Prentice Hall 1985

[McGregor 92]
John D McGregor and David A Sykes. Object-Oriented Software Development:
Engineering Software for Reuse Van Nostrand Reinhold 1992

[Maier 91]
Maier D. Comments on the 3rd Generation Database System Manifesto.
Oregon Graduate Inst. 91

[Maier et al 86]
Maier J.; Stein A.; Otis A. and Purdy A.
Development of an OODBMS. OOPSLA 472-482 1986

[Malluf 95]
Malluf I. Apply State-Machine Theory to Data Access.
Visual Basic Programmers Journal, April 1995,

[Manola and Dayal 86]
Manola F. and Dayal U. PDM: An Object-Oriented Data Model.
Proceedings of the International Workshop on OODBMS, 18-25 1986
[McGregor 93]
McGregor JD and Douglas MD. A Note on Inheritance and State Machines.
ACM SIGSOFT, Software Engineering Notes, Vol 18 No 4, OCT 1993

[Meyer 86]
Meyer B. Genericity Versus Inheritance
Proc. OOPSLA, Portland OR 1986

[Meyer 88]
Meyer B. Object-oriented Software Construction.
Prentice Hall, Hemel Hempstead, UK 1988

[Microsoft 99]
Microsoft WebSite at [http://www.microsoft.com/VB/]

[Milner 80]
Milner R. A Calculus of Communicating Systems. LNCS Vol. 92
Springer-Verlag New York 1980

[Murata 89]
Proceedings of the IEEE, vol 77, no 4, April 89

[Murata et al 93]
Murata T. and Kurihara K. State Diagram Matrix for Hierarchical Specification of
Reactive Systems.

[Nielson 90]
Nielson G.M., Shriver B.D., and Rosenblum J.

[Orfali et al 96]
Orfali R., Harkey D. & Edwards J.

[Papelis 92]
Y.E. Papelis, T.L. Casavant.
Specification and Analysis of Parallel/Distributed Software and Systems by Petri Nets
with Transition Enabling Functions.

[Peckham 88]
Joan Peckham and Fred Maryanski. Semantic Data Models.
[Peterson 81]
James L. Peterson The University of Texas as Austin. Petri-Net Theory and the Modeling of Systems Prentice Hall 1981

[Petri 62]

[Pnueli 86]

[Rock - Evans 81]

[Roth et al 88]
Roth M.A., Korth H.F. and Silberchatz A. Extended algebra and calculus for nested relational databases. ACM TODS. 13(4), 389-417

[Rozenberg 87]


[Sakai 83]

[Scheffner 1980]
[Schmid 75]
Schmid HA. and Swenson JR. On the semantics of the relationa data model.
In Proceedings of the ACM Sigmod Conference (San Jose, Calif)
ACM New York pp211-223

[Scholtz et al 93]
Object-Oriented Programming: The Promise and the Reality.

[Shields 87]
Shields MW. An Introduction to Automata Theory, Computer Science Texts. Blackwell
 Scientific Publications 1987

[Shipman 81]
Shipman D.W. The Functional Data Model and the Data Language DAPLEX.
ACM Transactions on Database Systems 6,1 March 81, pp140-173

[Shlaer 88a]
Shlaer S. & Mellor J. Object Oriented Systems Analysis - Modeling the World in Data
 Prentice Hall 1988

[Shlaer 88b]
Shlaer and Mellor S. Object-Oriented Systems Analysis
Eaglewood Cliffs, New Jersey: Yourden Press 1988

[Smith & Smith 77]
Smith JM. and Smith DCP. Database Abstractions: Aggregation and Generalisation.
ACM Transactions on Database Systems 2, 2 March 1977

[Smith & Smith 77]
Smith J. M. & Smith D. C. P. Database Abstractions: Aggregation and Generalisation

[Snyder 86]
Snyder A. Encapsulation and Inheritance in Object-Oriented Programming Languages.
Proc. OOPSLA, Portland, OR. in SIGPLAN NOTICES Vol. 21 (11) 1986

[Stonebraker et al 90]
Stonebraker M.; Rowe L.; Hirohama M.
The implementation of POSTGRES.
IEEE Trans. on Knowledge and Data Eng. 2(1), 125-142

[Stroustrup 91]
Reading, MA: Addison-Wesley Series in Computer Science, 1991
[Su 83]

[Tagg 93]

[Thearle 89]


[Thiagarajan 87]

[Upson 89]

[Wegner 87]
Wegner P. Dimensions of Object-Based Language Design. OOPSLA 87 Preceedings, New York ACM

[Wegner 88]

[Wegner 92]

[Wirfsbrock et al 90]

[Work 91]
Work LB. The Promise of Object-Orientation Computing Sciences Technical Report June 91 University of Surrey
[Yourden 82]
Yourden E. Managing the System’s Lifecycle.
Yourden Press 1982

[Zave 85]
Zave PA. A Distributed Alternative to Finite-state-machine Specifications.
ACM Transactions on Programming Languages Systems. Vol. 7, Part 1, Jan 1985,
pp10-36

[Zdonik and Maier 90]
Zdonik S. and Maier D. Fundamentals of object-oriented databases. Readings in Object-
Oriented Database Systems.
Morgan Kauffmann 90