Formal Object Oriented Development
of Software Systems
using LOTOS.

J. Paul. Gibson
Department of Computing Science,
University of Stirling,
Stirling FK9 4LA.

A thesis submitted in partial fulfilment of the
requirements for the Degree of Doctor of Philosophy
in Computer Science at Stirling University

July 1993
Abstract

Formal methods are necessary in achieving correct software: that is, software that can be proven to fulfill its requirements. Formal specifications are unambiguous and analyzable. Building a formal model improves understanding. The modeling of nondeterminism, and its subsequent removal in formal steps, allows design and implementation decisions to be made when most suitable. Formal models are amenable to mathematical manipulation and reasoning, and facilitate rigorous testing procedures. However, formal methods are not widely used in software development. In most cases, this is because they are not suitably supported with development tools. Further, many software developers do not recognize the need for rigour.

Object oriented techniques are successful in the production of large, complex software systems. The methods are based on simple mathematical models of abstraction and classification. Further, the object oriented approach offers a conceptual consistency across all stages of software development. However, the inherent flexibility of object oriented approaches can lead to an incremental and interactive style of development, a consequence of which may be insufficient rigour. This lack of rigour is exacerbated by the inconsistent and informal semantics for object oriented concepts at all stages of development.

Formal and object oriented methods are complementary in software development: object oriented methods can be used to manage the construction of formal models and formality can add rigour to object oriented software development. This thesis shows how formal object oriented development can proceed from analysis and requirements capture to design and implementation.

A formal object oriented analysis language is defined in terms of a state transition system semantics. This language is said to be customer-oriented: a number of graphical views of object oriented relations in the formal analysis models are presented, and the specifications produced say what is required rather than how the requirements are to be met. A translation to ACT ONE provides an executable model for customer validation. This translation is founded on a precise statement of the relationship between classes and types (and subclassing and subtypes). The structure of the resulting ACT ONE requirements model corresponds to the structure of the problem domain, as communicated by the customer.

The step from analysis to design requires an extension to the requirements model to incorporate semantics for object communication. A process algebra provides a suitable formal model for the specification of communication properties. LOTOS, which combines ACT ONE and a process algebra in one coherent semantic model, provides a means of constructing object oriented design semantics. Design is defined as the process of transforming a customer-oriented requirements model to an implementation-oriented design, whilst maintaining correctness. Correctness preserving transformations (CPTs) are defined for: transferring requirements structure to design structure, manipulating design structure and changing internal communication models.

Design must be targeted towards a particular implementation environment. The thesis examines a number of different environments for the implementation of object oriented LOTOS designs. It illustrates the importance of understanding programming language semantics. We show how Eiffel can be used to implement formal object oriented designs.

A case study which evaluates the formal object oriented models and methods, developed in this thesis, is reported. This identifies re-use at all stages of software development and emphasizes the role of structure: it improves understanding and communication, and makes validation and verification easier and better.

The thesis shows that formal object oriented technology is ready for transfer to industry. These methods should be exploited sooner rather than later: object oriented development can incorporate formal methods without significant cost, and formal methods can utilise the object oriented paradigm to manage complexity. The thesis provides a rationale for formal object oriented development and a set of conceptual tools which makes the development of software systems a true engineering discipline.
Declaration

I hereby declare that this thesis has been composed by myself, that the work reported has not been presented for any university degree before, and that the ideas I do not attribute to others are due to myself.

Paul Gibson
July 1993
Acknowledgements

The completion of this thesis was dependent on many different people, my thanks goes to everyone who encouraged me in this work, even if I forget to mention them by name.

My supervisor, Ken Turner, must be acknowledged for the time and effort which went into my supervision. Ashley McClenaghan must also be thanked for his friendship and support whilst sharing a room with me for almost four years. The computing science department at Stirling University is thanked as a whole, but, in particular, Sam, Graham and Catherine (for their technical support), and Moira and Jane (for their secretarial expertise), and Bob Clark (for ably fulfilling the role of a second supervisor) must be acknowledged.

British Telecom (Research and Development) are acknowledged for their financial contributions, together with the Department of Education for Northern Ireland. BT also offered support in the form of technical advice and personal encouragement: Elspeth Cusack, Steve Rudkin, Jim Lynch, Steve Colwill, Jeremy Wilson, Rob Booth, David Freer and many others are thanked in this respect.

On a personal note, my family are thanked for their support of an eternal student: my mother, father, David and Andrew have given me more help than they could know. My friends: Geoff (who never had a cross word), Flash, Andy, Steve, Dave, Gary, Richard, Ana, Peter, Dominique and Bazza have helped with their sense of humour and understanding.

Finally, I would like to thank my girlfriend, Jane, for putting up with me whilst completing this work: she may not understand what it means, but she, more than anyone, understands what it means to me!

I finish with a sentiment (a lesson from my parents which I have only just fully learned):

When you value something by how much it has cost then you don't really value it at all.
## Contents

1 Introduction
   1.1 Scope: Software Engineering ............................................. 2
   1.2 Context: Structured Development, Formal and Object Oriented Methods ............................................. 2
       1.2.1 Structured Software Development Methods ......................... 3
       1.2.2 Formal Methods .................................................. 4
       1.2.3 The Object Oriented Paradigm ..................................... 5
       1.2.4 Formal Methods and Object Orientation .......................... 5
       1.2.5 LOTOS .......................................................... 6
   1.3 Formulation of an Ideal Development Environment ....................... 7
   1.4 Formal Object Oriented Development (FOOD): Prototyping An Ideal ............................................... 8
       1.4.1 Fulfilling The Ideal Requirements: An Overview .................. 8
       1.4.2 A Step-by-step Construction of the FOOD Environment .......... 10
   1.5 Contributions of the Thesis ............................................. 12
       1.5.1 FOOD: The Philosophy ........................................... 12
       1.5.2 FOOD: The Models ............................................. 13
       1.5.3 FOOD: The Method ............................................. 13

2 Analysis: Modelling Problem Understanding .................................... 14
   2.1 Introducing Formal Object Oriented Analysis (FOOA) ..................... 15
       2.1.1 Introducing Traditional (Functional) Approaches ................. 15
       2.1.2 Object Orientation ............................................. 15
       2.1.3 Formalisation ................................................... 16
       2.1.4 Formalising the Object Oriented Approach ....................... 16
   2.2 Analysis: An Overview ............................................... 17
       2.2.1 Analysis is Problem Domain Understanding ......................... 17
       2.2.2 Traditional Analysis Methods and Models ....................... 18
       2.2.3 Features of Good Analysis ..................................... 20
       2.2.4 Introducing Object Oriented Analysis ........................... 22
       2.2.5 Objects and Classes: The Problems with Terminology ........... 23
   2.3 Object Oriented Analysis: An Informal Approach ......................... 24
       2.3.1 Identifying Objects ............................................. 24
       2.3.2 Identifying Classes ............................................. 26
       2.3.3 Classification Relationships .................................... 27
       2.3.4 Defining Classes of Behaviour ................................ 28
       2.3.5 Explicit Subclassing Relationships ............................ 37
3 An Object Oriented Semantic Framework
3.1 An Overview of the Semantic Framework .......................... 49
3.2 Object-Labelled State Transition System (O-LSTS) Semantics ..... 50
  3.2.1 Definition: an O-LSTS Specification ........................... 51
  3.2.2 O-LSTS Examples ........................................... 55
  3.2.3 State Label Expressions .................................... 59
3.3 An Object Oriented Interpretation of the O-LSTS Model ............... 59
  3.3.1 O-LSTS Classification ...................................... 60
  3.3.2 O-LSTS Interaction: The Executable Semantics ................. 61
  3.3.3 O-LSTS Subclassing (and Subtyping) ........................ 62
  3.3.4 O-LSTS Composition ........................................ 74
  3.3.5 O-LSTS Configuration ...................................... 76
  3.3.6 Structure Diagrams ......................................... 78
3.4 OO ACT ONE: A Formal Object Oriented Analysis Language ............ 78
  3.4.1 Motivation .................................................. 78
  3.4.2 The OO ACT ONE Syntax: Some Examples ..................... 79
  3.4.3 Static Analysis of OO ACT ONE: Syntax and Semantics ....... 92
3.5 An ACT ONE Execution Model for O-LSTS Specifications ............... 93
  3.5.1 The Advantages of Using ACT ONE .......................... 93
  3.5.2 Reviewing the ACT ONE Terminology ......................... 93
  3.5.3 An Overview of the OO ACT ONE — ACT ONE Translation .... 94
  3.5.4 Static Analysis of ACT ONE ................................ 99
  3.5.5 Evaluating Act One Expressions: An Execution Model for OO ACT ONE .... 99
  3.5.6 Event Diagrams ............................................ 100
4 Formal Object Oriented Analysis: The Practical Issues ................. 101
4.1 Subclassing ...................................................... 102
  4.1.1 Categorising Class Hierarchies .............................. 102
  4.1.2 Inclusion Polymorphism and Dynamic Binding ................ 103
CONTENTS

5.3.2 Comparing Object Oriented Design and Object Oriented Analysis .................... 135
5.3.3 Removing Nondeterminism .............................................................. 136
5.3.4 Realising the Abstract Object Oriented Model ....................................... 136
5.3.5 Restructuring The Requirements To Match An Implementation Environment .... 136
5.3.6 Verification and Correctness Preserving Transformations ........................... 138
5.4 Object Oriented Design with LOTOS ....................................................... 139
5.4.1 Design in LOTOS................................................................................. 139
5.4.2 Abstract Data Typing in LOTOS ......................................................... 141
5.4.3 The Process Algebra in LOTOS............................................................. 141
5.4.4 Balancing Processes and Types in Design ............................................. 141
5.4.5 Defining an Object Oriented LOTOS Style of Specification ...................... 142
5.5 FOOA as Input to Formal Object Oriented Design ....................................... 144
5.5.1 Generating Full LOTOS from the Requirements Model .............................. 144
5.5.2 Internal and External Communication .................................................... 147
5.5.3 Defining the Mappings from OOA CT ONE to Full LOTOS ...................... 147
5.5.4 An Object Oriented Interpretation of the Initial LOTOS Designs ............... 148
5.5.5 An Object Oriented Style of LOTOS Specification ................................. 152
5.6 Correctness Preserving Transformations (CPTs): Formalising Design ................. 153
5.6.1 Introduction ....................................................................................... 153
5.6.2 Concepts ......................................................................................... 154
5.6.3 An Overview of CPTs in LOTOS ......................................................... 155
5.6.4 Graphical Views and Tools ................................................................. 157
5.6.5 CPT Driven Design: Some Other Concerns .......................................... 158
5.6.6 Object Oriented LOTOS CPTs and the Resulting Design Trajectory .......... 159
5.7 A Set of Object Oriented Design Decisions as CPTs ....................................... 160
5.7.1 Static Structure Expansion ................................................................. 161
5.7.2 Compositional Re-Structuring For Re-Use .......................................... 165
5.7.3 Re-Structuring for Distributed Control ............................................... 168
5.7.4 Resolving Explicit NonDeterminism ................................................... 172
5.7.5 Removing Parallelism ........................................................................ 174

6 Object Oriented Program Derivation ................................................................ 176
6.1 High-level Object Oriented Design as Input to Implementation ..................... 177
6.1.1 An Overview of Programming Languages and Implementation Concerns ...... 178
6.1.2 Implementation Outside an Object Oriented Framework ......................... 179
6.1.3 Implementation in an Object Oriented Environment: The Advantages .......... 183
6.2 Object Oriented Programming (OOP): The Alternatives ............................... 183
6.2.1 The Roles of Object Oriented Programmers ......................................... 183
6.2.2 Characterisation of OOP Languages .................................................... 185
6.2.3 A Review of OOP Languages ............................................................. 189
6.2.4 Choosing Eiffel .............................................................................. 190
6.3 Translating Design To Implementation: Mapping Semantics .......................... 191
6.3.1 Implementation Languages: The Importance of Semantics .................... 191
6.3.2 Peculiarities of LOTOS Designs .......................................................... 193
6.4 Producing Eiffel from Procedural Object Oriented LOTOS Designs ............... 195
# CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.4.1</td>
<td>Setting Reasonable Bounds</td>
<td>195</td>
</tr>
<tr>
<td>6.4.2</td>
<td>Coding Design Requirements in Eiffel: An Overview</td>
<td>196</td>
</tr>
<tr>
<td>6.4.3</td>
<td>Reference Semantics vs Value Semantics</td>
<td>196</td>
</tr>
<tr>
<td>6.4.4</td>
<td>Coding Object Based Requirements</td>
<td>199</td>
</tr>
<tr>
<td>6.4.5</td>
<td>Coding Object Oriented Properties</td>
<td>203</td>
</tr>
<tr>
<td>6.4.6</td>
<td>Using Eiffel Assertions and Exceptions</td>
<td>205</td>
</tr>
<tr>
<td>6.4.7</td>
<td>Other Aspects</td>
<td>206</td>
</tr>
<tr>
<td>6.5</td>
<td>A Question of Concurrency and Distribution</td>
<td>207</td>
</tr>
<tr>
<td>6.5.1</td>
<td>Concurrency and Objects: Opposing Views</td>
<td>208</td>
</tr>
<tr>
<td>6.5.2</td>
<td>Concurrency: A Problem of Scale</td>
<td>209</td>
</tr>
<tr>
<td>6.5.3</td>
<td>Concurrency and Object Orientation: Resolving Conflicting Requirements</td>
<td>209</td>
</tr>
<tr>
<td>6.5.4</td>
<td>The Future: Formality in Concurrent Compilers?</td>
<td>210</td>
</tr>
<tr>
<td>7</td>
<td>Formal Object Oriented Development: A Case Study</td>
<td>212</td>
</tr>
<tr>
<td>7.1</td>
<td>Introducing the Banking Network Problem</td>
<td>213</td>
</tr>
<tr>
<td>7.1.1</td>
<td>Choosing the Case Study</td>
<td>213</td>
</tr>
<tr>
<td>7.1.2</td>
<td>Limitations of the Case Study</td>
<td>213</td>
</tr>
<tr>
<td>7.1.3</td>
<td>The Scope of the Problem: An Informal Overview of Requirements</td>
<td>214</td>
</tr>
<tr>
<td>7.2</td>
<td>Formal Object Oriented Analysis of the System</td>
<td>216</td>
</tr>
<tr>
<td>7.2.1</td>
<td>What not How</td>
<td>216</td>
</tr>
<tr>
<td>7.2.2</td>
<td>Applying the Skeleton Method to Requirements Capture</td>
<td>216</td>
</tr>
<tr>
<td>7.2.3</td>
<td>A Review of the Analysis and Requirements Capture</td>
<td>234</td>
</tr>
<tr>
<td>7.3</td>
<td>Design: Moving the System from Abstract to Concrete</td>
<td>237</td>
</tr>
<tr>
<td>7.3.1</td>
<td>From Analysis to Design: Choosing the Communication Model</td>
<td>238</td>
</tr>
<tr>
<td>7.3.2</td>
<td>Decomposition of the Banking Network System</td>
<td>239</td>
</tr>
<tr>
<td>7.3.3</td>
<td>Decomposition of the Network Component Process</td>
<td>239</td>
</tr>
<tr>
<td>7.3.4</td>
<td>Restructuring the Network Component Process</td>
<td>240</td>
</tr>
<tr>
<td>7.3.5</td>
<td>Integrating the Transaction Set in the Network</td>
<td>241</td>
</tr>
<tr>
<td>7.3.6</td>
<td>An Explicit Routing Mechanism: Removing Nondeterminism</td>
<td>242</td>
</tr>
<tr>
<td>7.3.7</td>
<td>A Review of the Design Process</td>
<td>244</td>
</tr>
<tr>
<td>7.4</td>
<td>The Eiffel Implementation</td>
<td>245</td>
</tr>
<tr>
<td>7.4.1</td>
<td>The Role of the Final Object Oriented LOTOS Design</td>
<td>245</td>
</tr>
<tr>
<td>7.4.2</td>
<td>Re-Use in the Implementation</td>
<td>246</td>
</tr>
<tr>
<td>7.4.3</td>
<td>Implementing Exceptions</td>
<td>246</td>
</tr>
<tr>
<td>7.4.4</td>
<td>Implementing A User Interface</td>
<td>246</td>
</tr>
<tr>
<td>7.5</td>
<td>A Review of the Case Study</td>
<td>246</td>
</tr>
<tr>
<td>7.5.1</td>
<td>Development Statistics</td>
<td>246</td>
</tr>
<tr>
<td>7.5.2</td>
<td>The Effectiveness of FOOD</td>
<td>247</td>
</tr>
<tr>
<td>7.5.3</td>
<td>Extensions to the Behaviour</td>
<td>247</td>
</tr>
<tr>
<td>7.5.4</td>
<td>The Importance of Structure Throughout Development</td>
<td>248</td>
</tr>
<tr>
<td>8</td>
<td>Conclusions</td>
<td>249</td>
</tr>
<tr>
<td>8.1</td>
<td>Review of Thesis Objectives</td>
<td>249</td>
</tr>
<tr>
<td>8.2</td>
<td>Meeting Objectives: The Contributions of the Thesis</td>
<td>249</td>
</tr>
<tr>
<td>8.3</td>
<td>Future Work</td>
<td>252</td>
</tr>
</tbody>
</table>
# CONTENTS

A Preconditioned Equations: The O-LSTS Model ........................................ 261  
B Static Analysis of OO ACT ONE ............................................................ 264  
\hspace{1em} B.1 Preprocessing: Removing Syntactic Sugar .......................... 264  
\hspace{1em} B.2 Static Semantic Checks of Unsugared OO ACT ONE ............. 266  
C Mapping OO ACT ONE to ACT ONE ....................................................... 270  
\hspace{1em} C.1 Object Based Requirements ................................................. 270  
\hspace{1em} C.2 Example Queue Behaviour .................................................... 274  
\hspace{1em} C.3 Translating Object Oriented Requirements: An Example .......... 275  
D An OO ACT ONE Interpretation of Interaction ...................................... 284  
\hspace{1em} D.1 Interaction ........................................................................ 284  
\hspace{1em} D.2 Data and Control Flow ....................................................... 285  
E Design Issues ..................................................................................... 287  
\hspace{1em} E.1 The ParXStack Process Definition ........................................ 287  
\hspace{1em} E.2 Two Mappings from OO ACT ONE to an Initial Full LOTOS Design ...................................................... 289  
\hspace{1em} \hspace{1em} E.2.1 The MakePar Mapping .................................................... 289  
\hspace{1em} \hspace{1em} E.2.2 The MakeRPC Mapping .................................................. 290
List of Figures

1.1 Thesis Scope .......................................................... 2
1.2 An Overview of the Problem Domain Structure ...................... 3
1.3 Prototyping an *ideal* Software Development Environment .......... 10

2.1 A Hall Residents Class Hierarchy .................................... 28
2.2 Five Object Oriented Relationships: A Simple Car Example .......... 39

3.1 A Resetable Traffic Light as an O-LSTSD ............................ 56
3.2 A Resetable Traffic Light as a Sugared O-LSTSD .................... 57
3.3 A Further Sugaring of the O-LSTSD .................................. 57
3.4 An O-LSTSD Specification of an Integer Counter ..................... 58
3.5 Subtyping: A Simple Example ....................................... 65
3.6 Subtyping is not Subclassing: An Example .......................... 65
3.7 An Extension Example ................................................ 67
3.8 A Specialisation Example .......................................... 68
3.9 Illustrating Contravariance and Covariance .......................... 69
3.10 A Fulfils Example .................................................. 71
3.11 A Transition Reduction Example .................................... 72
3.12 A State Reduction Example ....................................... 72
3.13 A Re-structuring Example ........................................ 73
3.14 An Inclusion Example .............................................. 74
3.15 A Composition Example ............................................ 76
3.16 System Configuration: An Example ................................ 77
3.17 Structure Diagrams: An Example ................................... 78
3.18 Specifying Natural Numbers: A Nat O-LSTS .......................... 80
3.19 Class Hierarchies in O-LSTSDs .................................... 89
3.20 Static Analysis of OO ACT ONE .................................... 93
3.21 A System Event Diagram ........................................... 100

4.1 A Single Inheritance Hierarchy ...................................... 102
4.2 A Multiple Inheritance Hierarchy .................................... 103
4.3 A Structure Diagram of Recursive Behaviour ......................... 110
4.4 Sharing is not an Analysis Issue: An Example ....................... 111
4.5 The Skeleton Analysis Method ...................................... 118

5.1 Restructuring for Re-use: A Design Sequence ....................... 137
<table>
<thead>
<tr>
<th>Figure Number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2</td>
<td>Restructuring for Re-use: A Design Choice</td>
<td>137</td>
</tr>
<tr>
<td>5.3</td>
<td>LOTOS: An Object Oriented Interpretation of Objects and Processes</td>
<td>149</td>
</tr>
<tr>
<td>5.4</td>
<td>LOTOS: An Object Oriented Interpretation of Service Requests</td>
<td>149</td>
</tr>
<tr>
<td>5.5</td>
<td>LOTOS: Representing Communication Models</td>
<td>153</td>
</tr>
<tr>
<td>5.6</td>
<td>A CPT: Illustrating the Concepts</td>
<td>157</td>
</tr>
<tr>
<td>5.7</td>
<td>The Formal Object Oriented Design Trajectory</td>
<td>159</td>
</tr>
<tr>
<td>5.8</td>
<td>Static Expansion ($StExp$) of a ParClass Process</td>
<td>161</td>
</tr>
<tr>
<td>5.9</td>
<td>$StExp$ of a TwinStack Behaviour</td>
<td>164</td>
</tr>
<tr>
<td>5.10</td>
<td>A Composition CPT: Comp</td>
<td>166</td>
</tr>
<tr>
<td>5.11</td>
<td>A Composition Example</td>
<td>167</td>
</tr>
<tr>
<td>5.12</td>
<td>The Distributed Control CPT: $Dist$</td>
<td>168</td>
</tr>
<tr>
<td>5.13</td>
<td>CoinToss: An Example of Nondeterministic Behaviour</td>
<td>173</td>
</tr>
<tr>
<td>6.1</td>
<td>Categorising Control Flow Models</td>
<td>179</td>
</tr>
<tr>
<td>6.2</td>
<td>Characterising Object Oriented Programming Languages</td>
<td>191</td>
</tr>
<tr>
<td>6.3</td>
<td>Composition By Reference: A Form of Sharing</td>
<td>193</td>
</tr>
<tr>
<td>6.4</td>
<td>Sharing Objects: An Implementation Example</td>
<td>194</td>
</tr>
<tr>
<td>6.5</td>
<td>Composition in Eiffel</td>
<td>202</td>
</tr>
<tr>
<td>7.1</td>
<td>Scope of the Case Study</td>
<td>214</td>
</tr>
<tr>
<td>7.2</td>
<td>The Network Class Structure Diagram</td>
<td>219</td>
</tr>
<tr>
<td>7.3</td>
<td>A Network Object Structure Diagram</td>
<td>224</td>
</tr>
<tr>
<td>7.4</td>
<td>The Account Transaction Class Hierarchy</td>
<td>228</td>
</tr>
<tr>
<td>7.5</td>
<td>A Review of the BankingNetwork Components</td>
<td>232</td>
</tr>
<tr>
<td>7.6</td>
<td>The BankingNetwork Class Structure Diagram</td>
<td>233</td>
</tr>
<tr>
<td>7.7</td>
<td>BankingNetwork Design Diagram: Stage 2</td>
<td>240</td>
</tr>
<tr>
<td>7.8</td>
<td>Network Design Diagram: Stage 3</td>
<td>240</td>
</tr>
<tr>
<td>7.9</td>
<td>Network Design Diagram: Stage 4.1</td>
<td>241</td>
</tr>
<tr>
<td>7.10</td>
<td>Network Design Diagram: Stage 4.2</td>
<td>242</td>
</tr>
<tr>
<td>7.11</td>
<td>BankingNetwork Design Diagram: Stage 5.1</td>
<td>242</td>
</tr>
<tr>
<td>7.12</td>
<td>BankingNetwork Design Diagram: Stage 5.2</td>
<td>243</td>
</tr>
<tr>
<td>7.13</td>
<td>BankingNetwork Design Diagram: Stage 6</td>
<td>244</td>
</tr>
<tr>
<td>B.1</td>
<td>Preprocessing of OO ACT ONE Syntactic Sugar</td>
<td>264</td>
</tr>
<tr>
<td>C.1</td>
<td>An Example O-LSTSD</td>
<td>275</td>
</tr>
</tbody>
</table>