

Definitive Programming:
A Paradigm for Exploratory Programming

by
Simon Yun Pui Yung

A thesis submitted for the degree of
Doctor of Philosophy in Computer Science

Department of Computer Science
University of Warwick
Coventry CV4 7AL
U.K.

October 1992

Definitive Programming: A Paradigm for Exploratory Programming

Simon Yun Pui Yung

Supervisor: Dr W M Beynon

Department of Computer Science
University of Warwick
Coventry, U.K.

A thesis submitted for the degree of Doctor of Philosophy

Abstract

Exploratory software development is a method that applies to the development of programs whose requirement is initially unclear. In such a context, it is only through prototyping and experimenting on the prototypes that the requirement can be fully developed. A good exploratory software development method must have a short development cycle. This thesis describes our attempt to fulfil this demand. We address this issue in the programming language level. A novel programming paradigm – *definitive (definition-based) programming* – is developed.

In definitive programming, a state is represented by a set of definitions (a *definitive script*) and a state transition is represented by a redefinition. By means of a definition, a variable is defined either by an explicit value or by a formula in terms of other variables. Unless this variable is redefined, the relationship between the variables within the definition persists.

To apply this state representation principle, we have developed some *definitive notations* in which the underlying algebras used in formulating definitions are domain-specific. We have also developed an agent-oriented specification language by which we can model state transitions over definitive scripts. The modelling principles of definitive programming rest on a solid foundation in observation and experiment that is essential for exploratory software development.

This thesis describes how we may combine definitive notations and the agent-oriented programming concept to produce software tools that are useful in exploratory software development. In this way, definitive programming can be considered as a paradigm for exploratory programming.

Keywords: definitive programming, programming languages, rapid prototyping, state-transition model, modelling and simulation, software development, agents, human-computer interaction.

Table of Contents

Table of Contents.....	i
Table of Listings.....	vii
Table of Figures	viii
Acknowledgements	ix
1 Introduction.....	1
1.1. Programming Language and Software Development.....	1
1.2. Dependency and Observation.....	5
1.3. Motivating Ideas.....	6
1.4. What Has to be Done	11
1.4.1. Brief History of Definitive Programming Research.....	11
1.4.2. The Future	12
1.5. Contribution of This Thesis.....	13
1.6. Outline of the Thesis.....	15
2 Definition-Based State-Transition Models in the Abstract	17
2.1. The Definition-Based State-Transition (DST) Model.....	18
2.2. Comparison of the Concept of State and Transition in Different Programming Paradigms.....	19
2.2.1. Conventional Imperative Programming	21
2.2.2. Functional Programming	23
2.2.3. Object-Oriented Programming (OOP)	24
2.3. Virtues of the DST Model.....	26
2.3.1. Data Dependency, Concurrency and Consistency.....	26
2.3.2. Support for Incomplete Models	28
2.3.3. Possible Transformation	28
2.3.4. Exploratory Software Development.....	30

3 Software Tools Using Dependency	31
3.1. Spreadsheets.....	32
3.1.1. Basic Concepts of a Spreadsheet.....	32
3.1.1.1. Variable names	33
3.1.1.2. Range Functions and Operations.....	35
3.1.1.3. Order of Evaluation.....	35
3.1.1.4. Differences Between a Spreadsheet And a Definitive Script	36
3.1.2. Appraisal of Spreadsheet Programming and Definitive Programming	36
3.1.2.1. What-If.....	37
3.1.2.2. Rapid Prototyping	38
3.1.2.3. Cognitive Dimensions.....	40
3.2. Document Preparation Software.....	42
3.3. Graphics Editors and User Interface Tools	44
3.4. The Make Utility	46
3.5. Theoretical Framework.....	47
4 The Scout Notation.....	48
4.1. The DoNaLD Notation	49
4.2. Motivating the Design of Scout.....	51
4.2.1. Visualisation of State	51
4.2.2. Assumptions of Definitive Notations	53
4.3. The Design of Scout	56
4.3.1. The Window Data Type	57
4.3.2. The Display Data Type	60
4.3.3. Other Data Types and Operators.....	60
4.4. The Implementation of Scout.....	64
5 Definition-Based State-Transition Models in Application.....	68

5.1. The Jugs Problem.....	69
5.2. Modelling a Screen Layout Using Scout	70
5.2.1. Screen Layout Modelling Process.....	70
5.2.2. Special-Purpose Notation for Specific Task	73
5.2.3. Flexibility of Model.....	75
5.2.4. Separation of Control and Presentation.....	77
5.3. Exploratory Screen Layout Development	78
5.3.1. Convenient State Changes	79
5.3.2. Flexible Definition Arrangement	79
5.3.3. Design and Simulation Joined Together.....	81
5.4. Summary.....	87
6 Integrating Definitive Notations.....	88
6.1. Motivation for the Scout Project.....	88
6.2. Scope of the Scout Project	92
6.3. Implementation of Definitive Notations	92
6.3.1. Steps for Implementing a Definitive Notation	93
6.3.2. Run-Time Structure.....	94
6.3.3. The Choice of Graphics Interface	96
6.4. Other Guidelines for the Design and Implementation of Definitive Notations.....	97
6.4.1. Bridging Definition	97
6.4.2. Naming Scheme.....	99
6.4.3. Switching Between Notations	99
6.4.4. Other Considerations	100
6.5. Evaluation of the Scout Project.....	101
6.5.1. Modelling, Understandability and Usability	101
6.5.2. Support for Iterative Design	103
6.5.3. User interface	103

7	Generalising Single-Agent Definitive Systems.....	105
7.1.	Definitive Programming vs Functional Programming.....	107
7.1.1.	Design and Implementation of Admira	107
7.1.1.1.	What Is an Admira Definition?.....	108
7.1.1.2.	Storage of Definitions	108
7.1.1.3.	Evaluation of Expression	109
7.1.2.	Recursive Definition and Circular Definition	109
7.1.3.	State and Interaction	113
7.2.	Agent-like Abstractions and Definitive Scripts	115
7.2.1.	Meta-Definition.....	115
7.2.2.	Semi-evaluation	117
7.2.3.	Inheritance	117
7.3.	Input Management	120
7.3.1.	The Room Example.....	120
7.3.2.	The Vehicle Cruise Control Example.....	122
7.3.3.	Extension to the Scout System	123
7.3.3.1.	Considerations	123
7.3.3.2.	The Extension Plan	126
7.3.4.	Input Handling Techniques	129
7.3.4.1.	Push Button	129
7.3.4.2.	Toggle Switch.....	129
7.3.4.3.	Menu Buttons	130
7.3.4.4.	Radio Buttons	130
7.3.4.5.	Duration-Sensitive Button	131
7.3.4.6.	Clocking	132
7.4.	Summary and Conclusion.....	133
8	Agent-Oriented Definitive Programming.....	135
8.1.	The Railway Station Simulation.....	140

8.2. Terminology in the LSD Notation.....	141
8.3. Transformation from LSD to ADM.....	144
8.4. An Evaluation of the LSD Notation.....	148
8.4.1. Grouping Guarded Commands.....	149
8.4.2. Parallel Action Specification.....	149
8.4.2.1. Limitation of LSD for Specifying a Swapping Agent.....	150
8.4.3. Call-by-Reference Parameter.....	152
8.4.4. Hidden-Text Annotation.....	153
8.5. Translation from ADM to EDEN.....	154
8.5.1. Motivation.....	154
8.5.2. The Translation Scheme.....	154
9 Summary and Conclusion.....	158
References.....	162

Appendices

Appendix A: Technical Document of the Scout System

Appendix B: User Guide to Admira

Appendix C: Program Listing of Admira

Appendix D: The Jugs Example

D.1. EDEN Definitions and Actions

D.2. Scout Display Specification

D.3. Original Display Specification

Appendix E: The Visualisation Example

E.1. The Script

E.2. Sample Output

Appendix F: The Room Example

F.1. The Script

F.2. Sample Output

Appendix G: The Vehicle Cruise Control Simulation Example

- G.1. The LSD Specification of the Simulation**
- G.2. EDEN Implementation of the LSD Specification**
- G.3. Scout Graphical Interface of the Simulation**
- G.4. Sample Output**

Appendix H: The Railway Station Simulation Example

- H.1. The LSD Specification**
- H.2. A Corresponding ADM Program**
- H.3. EDEN Implementation of the ADM Program**
- H.4. Extract of a Textual Simulation**
- H.5. Scout Graphical Interface of the Simulation**
- H.6. A Sample of the Graphical Output**

Table of Listings

Listing 2.1:	Two DoNaLD Specification for Describing the \square Shape	29
Listing 3.1:	An Example of Makefile.....	46
Listing 4.1:	A DoNaLD Description of a Square	54
Listing 4.2:	Another DoNaLD Description of a Square.....	54
Listing 4.3:	A Sample Scout Fragment	55
Listing 5.1:	Definitions for Locations.....	72
Listing 5.2:	Other Scout Definitions.....	72
Listing 5.3:	The Scout Definitions Relating the Pour Menu Option	80
Listing 5.4:	Square-root Guessing Program in EDEN.....	83
Listing 5.5:	Translated Square-root Guessing Program in C.....	84
Listing 6.1:	Program Fragment of the Scout Notation	98
Listing 7.1:	An Admira Script for Calculating Variance	108
Listing 7.2:	A Stack-based Integer Desk Calculator in pLucid.....	114
Listing 7.3:	A Proposed DoNaLD Specification of a Ladder.....	115
Listing 7.4:	A Proposed DoNaLD Specification of Rectangular Objects	118
Listing 7.5:	Specification of a Square.....	119
Listing 8.1:	An LSD Specification of a Station-Master	141
Listing 8.2:	ADM Entity Descriptions of Alarm() and Clock().....	145
Listing 8.3:	ADM Specification of the Station-Master Entity.....	147
Listing 8.4:	A Swapping Agent Illustrating Parallel Actions	150
Listing 8.5:	A Swapping Example (1st Attempt)	150
Listing 8.6:	A Swapping Example (2nd Attempt)	151
Listing 8.7:	A Swapping Example (3rd Attempt).....	151
Listing 8.8:	A Swapping Example (4th Attempt).....	152
Listing 8.9:	A Swapping Agent Using Call-by-Reference Parameters.....	153
Listing 8.10:	EDEN Simulation of a Two-Phase Clock.....	155

Table of Figures

Figure 1.1:	The Exploratory Software Development Cycle.....	4
Figure 1.2:	Programming Paradigms vs Propagation of State Change.....	8
Figure 2.1:	An \square Shape.....	29
Figure 3.1:	Graphical User Interface Mechanism in Our Systems	45
Figure 4.1:	DoNaLD Script of a Room.....	50
Figure 4.2:	A Non-rectangular Region	63
Figure 4.3:	Ways of Putting a String in a Non-rectangular Region.....	63
Figure 4.4:	A Way of Partitioning a Non-rectangular Region	64
Figure 4.5:	The Role of EDEN Actions in the Implementation of a Definitive Notation.....	65
Figure 5.1:	A Sample Jugs Output.....	71
Figure 5.2:	Screen Layout Design	71
Figure 5.3:	Cyclic Overlapping Windows	75
Figure 5.4:	The Trident Way of Software Development	86
Figure 6.1:	An Arrangement of Four Lines.....	89
Figure 6.2:	A Poset Representing the Line Arrangement in Figure 6.1.....	89
Figure 6.3:	An S_4 Graph.....	90
Figure 6.4:	Run-time Structure of a Definitive System.....	95
Figure 6.5:	Run-time Structure of the Scout System	95
Figure 7.1:	A Sample Output of the Room Viewer Example	121
Figure 7.2:	A Sample Output of the Vehicle Cruise Control Simulation	122
Figure 8.1:	Procedures for Animating an LSD Specification	138
Figure 8.2:	Sample Display of the Railway Station Simulation	139

A c k n o w l e d g e m e n t s

I wish to express my gratitude to my friends and colleagues who helped me and encouraged me during these years. In particular, I thank Dr. Steve Russ in Warwick and Dr. Rick Thomas in Leicester University for their useful comments. I must thank my brother, Edward Yung for his love, care and practical help while he was with me in the University.

I am most grateful to my supervisor, Dr. Meurig Beynon, for the continuous help, encouragement and constructive criticism he has given me during the work. Especially, I thank him for allowing me to have a year off for developing my personality and communication skills during the course.

I am so much thankful to the Coventry Chinese Christian Church who supported me during my financial hardship so that I may concentrate on my study. And above all, I have to thank God for providing me both the necessary intelligence for the work.