



Program Construction and Evolution
in a Persistent Integrated
Programming Environment

Alex Miklós Farkas

Thesis submitted for the degree of
Doctor of Philosophy
in
The University of Adelaide
(Faculty of Engineering)

September 1995

Abstract

In orthogonally persistent systems, data is treated independently of its persistence allowing storage management details to be ignored by programmers. The coexistence of all data within a persistent repository provides new opportunities for the persistent environment to participate in a number of software engineering processes, namely, software:

- construction,
- debugging,
- maintenance,
- evolution, and
- distribution.

A novel programming methodology supporting various styles of binding allows the persistent environment to participate in the software construction process. Existing programs and data may be bound to an application at different times during the application's lifetime, namely, at run-time, compile-time and program construction-time. This wide range of binding styles enables the programmer to trade off binding safety and binding flexibility.

A mechanism called Octopus enables the programmer to access a rich source of information about the bindings within arbitrary values. Using this information, existing applications may be interrogated and manipulated in ways in which the programming language alone would not normally permit. However, Octopus ensures that the type system of the programming language remains inviolate, thus providing a type safe mechanism for constructing debugging, maintenance and evolutionary tools. Another feature of Octopus is that it permits portions of an application's object graph to be isolated and transmitted.

Nodules permit a style of template programming that enables data values and executable code to be shared between different Nodule instances. The essence of this mechanism is to allow programs to be constructed and compiled without the requirement that the values used by the program be present. In this manner, individual components may be constructed independently and later assembled to form a complete application. Nodules also provide a

rich source of information about the structure of an application and may be used in conjunction with Octopus to develop and maintain software components.

This thesis describes in detail a persistent programming environment, the Octopus and Nodule mechanisms, and illustrates how each of these mechanisms, when used in conjunction with each other, provide a persistent integrated programming environment that supports a wide variety of software engineering activities otherwise lacking in conventional, or non-persistent systems.

Contents

Chapter 1: Introduction	1
1.1 Persistence	1
1.2 Programming Environments.....	4
1.2.1 Tools.....	4
1.2.2 Integration	5
1.3 Constructing Applications	6
1.3.1 Linking Mechanisms	7
1.3.2 Managing Application Subunits.....	8
1.4 Distribution.....	9
1.5 Research Topics	10
1.5.1 A Novel Programming Technique	10
1.5.2 Octopus	10
1.5.3 Nodules	11
1.5.4 A Persistent Programming Environment	11
1.6 Thesis Structure	12
Chapter 2: Background	13
2.1 Introduction	13
2.2 Binding Mechanisms	13
2.3 Napier88 Overview	15
2.4 Napier88 Infinite Union Types.....	17
2.4.1 The Any Data Type.....	17
2.4.2 The Environment Data Type.....	17
2.5 Types	19
2.5.1 Type Representations	19
2.5.2 Type Environments	21
2.6 Programming in Napier88	21
2.6.1 Napier88 Tools.....	21

2.6.2	An Example Program	22
2.7	Programming with Environments	23
2.8	Programming with Generators	25
Chapter 3: Flexible Binding Strategies		27
3.1	Introduction	27
3.2	An Example	28
3.3	Run-Time Binding	30
3.4	Compile-Time Binding	31
3.5	Construction-Time Binding	32
3.6	The Example Using Run-Time Binding	33
3.7	The Example Using Compile-Time Binding	34
3.8	The Example Using Construction-Time Binding	35
3.9	Discussion	36
3.9.1	Realising the Three Binding Styles	36
3.9.2	Safety Versus Flexibility	37
3.9.3	Program Succinctness	38
3.9.4	A Comparison with Generators	38
3.9.5	Related Work: The Conch System	41
3.10	Representing Program Source Code	43
3.10.1	Kirby's Source Code Model	43
3.10.2	The Aberdeen Source Code Model	46
3.11	Conclusions	47
Chapter 4: Octopus		49
4.1	Introduction	49
4.1.1	Background	49
4.1.2	Octopus	51
4.2	The Abstract Octopus Model	54
4.2.1	Bindings	55
4.3	Viewing	56
4.4	Querying Complex Objects	58
4.5	Evolution of Programs and Data	59

4.6	Distribution of Complex Object Closures	60
4.7	The Octopus Model in Napier88	61
4.7.1	Bindings	63
4.8	Viewing Napier88 Values	64
4.9	Querying Complex Napier88 Objects	66
4.9.1	High Level Query Abstractions	67
4.10	Evolving Napier88 Programs and Data.....	69
4.11	Distribution of Complex Napier88 Object Closures	71
4.12	Evolving Types.....	73
4.12.1	Operations for Evolving Types	77
4.13	Security.....	79
4.14	Related Research	81
4.14.1	Eiffel.....	81
4.14.2	Modulex	82
4.15	Conclusions	83
Chapter 5: Octopus Implementation		86
5.1	Introduction	86
5.2	Napier88 Implementation.....	86
5.2.1	The Data Type Value	86
5.2.2	Type Representations	88
5.2.3	The Persistent Abstract Machine	90
5.2.4	Accessing a Static Environment	91
5.3	Block Retention in PCASE	93
5.4	An Architecture to Support Octopus	94
5.4.1	Block Retention.....	94
5.4.2	Wiring Diagrams	96
5.4.3	Non-Procedural Values	97
5.4.4	Wiring Diagrams for Procedures	98
5.4.5	Implementation Overview.....	99
5.5	Evolving a Parts Database	100
5.5.1	Evolving Type Information.....	103

5.6	Password Protection	103
5.7	Performance of Hybrid	104
5.7.1	Block Retention.....	106
5.7.2	Space Utilisation	107
5.7.3	Speed.....	108
5.7.4	Future Work	108
5.8	Conclusions	110
Chapter 6: Nodules.....		111
6.1	Introduction	111
6.2	The Abstract Model.....	112
6.3	The Nodule Data Type	117
6.4	An Example Application using Nodules	121
6.5	Evolution	126
6.6	Evolving the Stack Example	127
6.6.1	Evolution Using Octopus	127
6.6.2	Evolution Using Nodules	128
6.7	Related Work.....	130
6.7.1	OBJ2 and C++ Classes.....	130
6.7.2	Generators	130
6.7.3	Modula and Ada.....	131
6.7.4	TemplateNapier.....	132
6.8	Future Directions	134
6.8.1	Type Parameters.....	134
6.8.2	Discovering Type Dependencies	134
6.9	Conclusions	135
Chapter 7: Nodules Implementation		137
7.1	Introduction	137
7.2	The Architecture of Nodules and Instances	137
7.3	Locations and Locals	138
7.3.1	Location Wrappers	139
7.3.2	Local Wrappers	140

7.4	An Example	141
7.5	Performance.....	145
7.6	Conclusions	147
Chapter 8: A Persistent Integrated Programming Environment		148
8.1	Introduction	148
8.2	The Aberdeen Programming Environment	148
8.2.1	The User Interface.....	148
8.3	Nodules.....	154
8.4	Representing Procedures	156
8.5	Analysis of Implementation	158
8.5.1	The PS-algol browser.....	158
8.5.2	The Napier88 Browser	161
8.5.3	The Aberdeen Browser	166
8.6	The Impact of Octopus	168
8.6.1	Octopus and the Aberdeen Browser.....	168
8.6.2	Installation Software	169
8.7	Conclusions	170
Chapter 9: Conclusions		172
9.1	Flexible Binding Strategies	172
9.2	Octopus.....	174
9.2.1	Linguistic Reflection.....	174
9.2.2	Browsing, Debugging, Querying, and Evolution.....	175
9.2.3	Distribution	175
9.4	Nodules.....	176
9.5	A Persistent Integrated Programming Environment	177
9.6	Current Status and Future work.....	177
9.6.1	Flexible Binding Strategies.....	177
9.6.2	Octopus	178
9.6.3	Nodules	178
9.6.4	Aberdeen	179
9.7	Conclusions	179

Appendix A: Aberdeen94 User Manual.....	181
A.1 Initialisation	181
A.3 Manipulating Values	182
A.3.1 Window Manipulation	182
A.3.1.1 Moving a Window	182
A.3.1.2 Deleting a Window	183
A.3.1.3 Resizing a Window	183
A.3.2 Using the Browser.....	183
A.3.3 Traversing Scalars.....	183
A.3.4 Traversing Images and Pictures	184
A.3.5 Traversing Structured Values.....	184
A.3.5.1 Traversing structure, abstype and env Values.....	185
A.3.5.2 Traversing proc Values	186
A.3.5.3 Traversing variant Values	187
3.5.4 Traversing Vectors	187
A.3.5.5 Traversing Locations	188
A.3.6 AutoMove Mode	189
A.4 The Interaction Window	189
A.4.1 The Interaction Window Light Buttons	190
A.4.2 The Local Environment	192
coerceToOctopus	192
coerceFromOctopus	192
exportOctopus	192
importOctopus.....	192
ls	192
outputString.....	192
traverse	193
this.....	193
windowManager.....	193
A.5 Tagging Values.....	194
A.6 Re-Traversing a Value.....	196
A.7 The Type Environment Window	196

A.8	The Source Code Manager	200
A.8.1	Folders	201
A.8.2	Storing text	201
A.9	Nodules	201
Appendix B: Miscellaneous Type Declarations		206
B.1	Octopus Types	206
B.2	The Nodule Types	206
B.2	The Callable Compiler	208
References	210