

**The Mesh**  
—A Universally Integrated Design Approach  
For Device Control



**Martin Lumisden Strange**  
MSc University of York, UK  
BSc University of Southampton, UK

**Thesis**  
for the degree of Doctor of Philosophy

The School of Electrical and Electronic Engineering  
Faculty of Engineering, Computer and Mathematical Sciences  
The University of Adelaide

Supervisors: **Dr Matthew Sorell**  
**Assoc Prof Michael Liebelt**

November 2007

---

Copyright © 2007  
All Rights Reserved.  
Martin Lumisden Strange

---

# Dedication

This thesis is dedicated to my grandfather, Albert Maurice Kerby DSO MC, whom I never had the chance to meet. What fascinating discussions about so many things we could have had.



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	The missing, dynamic half of the Internet story . . . . .	1
1.1.1	A universal tool for sharing control? . . . . .	1
1.1.2	Lessons of the Web . . . . .	1
1.1.3	Why the Internet revolution in control-IT is yet to happen . . . . .	2
1.2	The Mesh . . . . .	3
1.2.1	A parallel, dynamic world to the Web . . . . .	3
1.2.2	Some practical examples . . . . .	4
1.2.3	The Long Tail and how our consumer culture is changing . . . . .	4
1.2.4	Current research areas working at the big picture level . . . . .	5
1.2.5	Taking a broader view . . . . .	6
1.3	Motivation and background . . . . .	7
1.3.1	An ever-widening focus . . . . .	7
1.3.2	An unorthodox approach . . . . .	7
1.3.3	Validation in the form of a concept demonstrator . . . . .	7
1.3.4	Answers looking for problems—not problems looking for answers . . . . .	7
1.4	Aim of the thesis . . . . .	9
1.4.1	Conceptual and technical validation of the Mesh . . . . .	9
1.5	Literature review . . . . .	10
1.5.1	Cutting across existing research boundaries . . . . .	10
1.5.2	Using the power of abstraction . . . . .	10
1.5.3	Social and commercial aspects and implications . . . . .	10
1.5.4	User interface design . . . . .	10
1.5.5	The rise of XML-based languages in user interface design . . . . .	11
1.5.6	Other areas of research in large-scale control . . . . .	12
1.5.7	Software engineering techniques . . . . .	13
1.6	Gap statement for this research project . . . . .	14
1.6.1	A universally integrated design approach for device control . . . . .	14
1.7	The challenge of designing the Mesh . . . . .	15
1.7.1	The Web model as starting point . . . . .	15
1.7.2	The critical factor—design of the Meshoil language . . . . .	16
1.8	Open-plan thinking . . . . .	17
1.8.1	Searching for a better design philosophy . . . . .	17
1.8.2	Life is about relationships . . . . .	17
1.8.3	Avoiding SIC design . . . . .	17
1.8.4	Principles of open-plan thinking . . . . .	18
1.9	The concept demonstrator . . . . .	20

# CONTENTS

1.9.1	Created tools . . . . .	20
1.10	Thesis roadmap . . . . .	21
<b>2</b>	<b>Getting started with Meshoil</b>	<b>23</b>
2.1	Language overview . . . . .	23
2.1.1	Design principles . . . . .	23
2.1.2	Modular structure . . . . .	24
2.1.3	Built-in browser functionality through code insertion . . . . .	25
2.2	Example of a simple Meshoil program . . . . .	26
2.2.1	Domestic fan . . . . .	26
2.2.2	Design information encoded in relationships . . . . .	28
2.2.3	Hardware controls . . . . .	28
2.2.4	User interface design . . . . .	29
2.2.5	Making the keys work correctly . . . . .	29
2.2.6	Running the fan user interface . . . . .	30
<b>3</b>	<b>Principles of the Mesh</b>	<b>33</b>
3.1	A house with two fans . . . . .	35
3.1.1	Running this simulation . . . . .	35
3.1.2	Building-brick design flexibility . . . . .	36
3.1.3	Atomic-level networking . . . . .	36
3.1.4	Abstract hardware layer . . . . .	37
3.2	<i>“I don’t want to change anything”</i> . . . . .	38
3.2.1	Standard (1-speed, pan) user interface . . . . .	38
3.2.2	Cost-free user flexibility . . . . .	38
3.3	<i>“I want something simpler to use”</i> . . . . .	41
3.3.1	Simplified (1-speed) user interface . . . . .	41
3.3.2	Universal usability . . . . .	41
3.3.3	The conventional design trade-off . . . . .	43
3.3.4	The software designer’s current headache . . . . .	44
3.3.5	Expanding the user solution space . . . . .	45
3.4	<i>“I want better indication of what the fan is doing”</i> . . . . .	46
3.4.1	Standard (1-speed, pan, LED) user interface . . . . .	46
3.4.2	Futures programming . . . . .	46
3.4.3	Transparent programming environment . . . . .	48
3.5	<i>“I want the fan to do more”</i> . . . . .	50
3.5.1	Advanced (3-speed, pan) user interface . . . . .	50
3.6	<i>“I want to modify an existing feature”</i> . . . . .	54
3.6.1	Convert 3-speed to 5-speed plugin . . . . .	54
3.6.2	Preserving existing parts of a design . . . . .	54
3.6.3	Virtual logic . . . . .	58
3.6.4	Path hijack and release . . . . .	59
3.7	<i>“I want to assemble my own plug-and-play design”</i> . . . . .	60
3.7.1	User-generated software . . . . .	60
3.7.2	User Interfaces For You Meshpoint . . . . .	61
3.7.3	Software as a tradable commodity . . . . .	62
3.7.4	Too much freedom? . . . . .	63
3.7.5	Manufacturer still able to call the tune . . . . .	63

3.8	<i>“I want thermostat control”</i>	64
3.8.1	Thermostat plugin	64
3.8.2	Virtual hardware	64
3.8.3	Not another power cut!?	67
3.8.4	Ubiquitous computing	67
3.8.5	Software-based systems	68
3.8.6	Responsibility optimisation	68
3.9	<i>“I want a sleep mode that times out panning”</i>	70
3.9.1	Panning timeout plugin	70
3.9.2	Method calling	70
3.9.3	Ghost-in-the-machine API	73
3.9.4	Abstract API layer	74
3.9.5	Engineering integration by layering	74
3.10	<i>“I want the fan to operate safely”</i>	75
3.10.1	Safety cutout plugin	75
3.10.2	Team effort in program design	75
3.10.3	Brick nesting provides software scalability	75
3.10.4	Software-team scalability	77
3.10.5	Cut and paste programming	78
3.10.6	Globally-unified hardware control	80
3.10.7	Maximising code reuse	80
3.11	<i>“I want the latest plug-and-play design done for me”</i>	82
3.11.1	Latest design	82
3.11.2	Freedom to restructure a Meshoil program remotely	82
3.11.3	Timing of information transfer across the Mesh	83
3.11.4	Self-updating programs	83
3.11.5	Program morphing	84
3.11.6	Self-generating software	84
3.12	<i>“What use is a fan in winter?”</i>	85
3.12.1	Stopwatch user interface	85
3.12.2	Hardware reuse	86
3.13	<i>“I want to run all these user interfaces”</i>	89
3.13.1	All user interfaces	89
3.13.2	Interface agents	89
3.13.3	Problems faced by conventional interface agent-based systems	90
3.13.4	Universal agent	90
3.13.5	A wiki approach for agent-based systems	91
3.13.6	Improved scrutability	91
3.13.7	Freedom in meeting the user’s needs	92
3.14	<i>“I want integrated control for a second fan...”</i>	93
3.14.1	Remote control	93
3.14.2	Program interaction across the Internet	93
3.14.3	Superprogramming	93
3.14.4	Customising the plugin	94
3.14.5	Linked fan operation	94
3.14.6	Interaction invited—not imposed	95
3.14.7	Superprogramming implementation issues	96

## CONTENTS

3.15	<i>“I just want to control the kitchen fan remotely”</i>	97
3.15.1	Bedroom fan as a hand-held remote controller	97
3.15.2	Point-accurate interoperation	97
3.16	<i>“Give me bi-direction control”</i>	100
3.16.1	Bi-directional remote control	100
3.17	<i>“I want the fans to be clones of each other”</i>	101
3.17.1	Cloned fans	101
3.17.2	Reciprocal mapping of hardware controls	101
3.17.3	Multiple cloning	102
3.18	<i>“I want to control my son’s use of the fan”</i>	103
3.18.1	Restricted fan usability	103
3.18.2	More subtle superprogramming relationships	103
3.19	<i>“I want the fans to take it in turns to pan”</i>	105
3.19.1	Alternate panning	105
3.19.2	Control feedback loop	105
3.19.3	Cascading interactions across the Mesh	106
3.20	<i>“I want the fan to run more efficiently”</i>	107
3.20.1	Increasing executional efficiency of a program	107
3.20.2	Multithreading a program	107
3.20.3	Grid computing	109
3.20.4	Ghost processing	109
3.20.5	Remote program management	111
3.20.6	Complete code outsourcing	112
3.21	<i>“I want software that I can trust”</i>	114
3.21.1	Faulty advanced (3-speed, pan) user interface	114
3.21.2	Code filtering	115
3.21.3	Code filtering as a tool for customising software	115
3.21.4	The problem of unreliable and malicious software	115
3.21.5	Code integrity through accreditation	116
3.21.6	Manual accreditation	117
3.21.7	Automated accreditation	117
3.21.8	Social accreditation	118
<b>4</b>	<b>Universal usability</b>	<b>119</b>
4.1	The state of play in this research area	119
4.1.1	New methodologies needed	119
4.1.2	Flexibility seen as the key	119
4.1.3	Slow progress to date	119
4.1.4	The XML trend	120
4.1.5	Can UIDLs do the job?	121
4.2	Limitations of the UIDL approach	122
4.2.1	Usefulness lost	122
4.2.2	Imposing, not handling, design structure	122
4.2.3	Still dependent on a conventional imperative language	122
4.2.4	The false declarative/imperative divide	123
4.2.5	Offering only half of the solution	123
4.2.6	A mistaken conclusion?	123
4.3	Meshoil as a language for user interface design	125

4.3.1	The Meshoil alternative . . . . .	125
4.3.2	UIML . . . . .	125
4.3.3	XIML . . . . .	125
4.3.4	XForms and XUL . . . . .	126
4.3.5	URC . . . . .	126
<b>5</b>	<b>Case studies</b>	<b>127</b>
5.1	Safety fan . . . . .	128
5.1.1	Overview . . . . .	128
5.1.2	Running this simulation . . . . .	128
5.1.3	Design stage 1—mission statement . . . . .	128
5.1.4	Design stage 2—adding flesh to the bones . . . . .	129
5.1.5	Design stage 3—completing the logic . . . . .	131
5.1.6	Design stage 4—first working version . . . . .	135
5.1.7	Design stage 5—tuning for efficiency . . . . .	140
5.1.8	Multithreading . . . . .	141
5.1.9	Hardware simulation . . . . .	141
5.1.10	Running the safety fan user interface . . . . .	149
5.2	Radio . . . . .	152
5.2.1	Overview . . . . .	152
5.2.2	Hardware controls . . . . .	152
5.2.3	Software . . . . .	152
5.2.4	Running this simulation . . . . .	153
5.2.5	Running the radio user interface . . . . .	162
5.3	Sports car . . . . .	171
5.3.1	Overview . . . . .	171
5.3.2	Hardware controls . . . . .	171
5.3.3	Driving the sports car . . . . .	171
5.3.4	Vehicle dynamics . . . . .	173
5.3.5	Running the sports car simulator . . . . .	176
5.4	Sports car with safety fan and radio . . . . .	182
5.4.1	Running multiple programs simultaneously . . . . .	182
5.5	Calculator . . . . .	184
5.5.1	Overview . . . . .	184
5.5.2	Hardware controls . . . . .	184
5.5.3	Software . . . . .	184
5.5.4	Code reuse by using brick insertion . . . . .	185
5.5.5	Running the calculator user interface . . . . .	199
5.6	3-in-1 calculator . . . . .	202
5.6.1	Overview . . . . .	202
5.6.2	Hardware controls . . . . .	202
5.6.3	User interface . . . . .	202
5.6.4	Method library . . . . .	206
5.6.5	Using the platform’s API to provide method functionality . . . . .	206
5.6.6	Error handling . . . . .	207
5.6.7	Running the 3-in-1 calculator user interface . . . . .	209
5.7	Mobile phone/emergency pager . . . . .	217
5.7.1	Overview . . . . .	217

## CONTENTS

5.7.2	Using Meshoil to control non-Meshable devices . . . . .	217
5.7.3	Running this simulation . . . . .	217
5.7.4	Hardware controls . . . . .	218
5.7.5	Phone features . . . . .	218
5.7.6	Software . . . . .	220
5.7.7	Caller-specific call divert . . . . .	222
5.7.8	Emergency pager features . . . . .	222
5.7.9	Running the mobile phone/emergency pager user interface . . . . .	224
5.7.10	Profiles supported by virtual logic . . . . .	233
5.7.11	Futures programming as the solution for incoming call options . . . . .	235
<b>6</b>	<b>Comparison to other languages</b>	<b>237</b>
6.1	Universal Remote Console—URC . . . . .	238
6.1.1	Overview . . . . .	238
6.1.2	The URC approach to universal usability . . . . .	238
6.1.3	URC’s example of a digital thermometer . . . . .	239
6.1.4	URC a command-driven language . . . . .	240
6.1.5	Limitations of a command-driven approach . . . . .	240
6.1.6	The Meshoil program . . . . .	241
6.1.7	Running the thermometer simulation . . . . .	247
6.1.8	Conclusion for the URC example . . . . .	253
6.2	MindScript—Lego’s robotic control language . . . . .	254
6.2.1	Overview . . . . .	254
6.2.2	Lego robots . . . . .	254
6.2.3	The MindScript program . . . . .	255
6.2.4	Using Meshoil to simulate the robot’s environment . . . . .	257
6.2.5	The equivalent MindScript program in Meshoil . . . . .	258
6.2.6	Running the robot simulation . . . . .	263
6.2.7	Conclusion for the MindScript example . . . . .	266
6.3	Summary of comparisons . . . . .	267
<b>7</b>	<b>Summary</b>	<b>269</b>
7.1	Conclusions . . . . .	269
7.1.1	Open-plan thinking is useful . . . . .	269
7.1.2	Mesh principles are largely unique . . . . .	270
7.1.3	The Mesh addresses issues faced by current research areas . . . . .	271
7.1.4	The Mesh has wide application . . . . .	272
7.1.5	The Mesh is feasible and practical . . . . .	272
7.1.6	The Mesh could be implemented incrementally . . . . .	273
7.1.7	The Mesh is greater than the sum of its parts . . . . .	273
7.1.8	Meshoil is a unique language . . . . .	273
7.1.9	Meshoil is object-oriented . . . . .	273
7.1.10	Meshoil is a general-purpose programming language . . . . .	274
7.1.11	Meshoil is an empowering language for the user . . . . .	274
7.1.12	Meshoil contains features not found in other languages . . . . .	275
7.1.13	Meshoil can be used as a universal usability language . . . . .	275
7.2	Future work . . . . .	276
7.2.1	Immediate focus . . . . .	276

7.2.2	The longer term . . . . .	276
<b>A</b>	<b>Meshoil language description</b>	<b>277</b>
A.1	Syntax . . . . .	278
A.1.1	Meshoil's five XML elements . . . . .	278
A.1.2	<a> for annotation . . . . .	279
A.1.3	<c> for control . . . . .	279
A.1.4	<d> for data . . . . .	280
A.1.5	<b> for brick . . . . .	280
A.1.6	Brick nesting . . . . .	281
A.1.7	Brick structure . . . . .	282
A.1.8	Bricks with multiple <c> and <d> elements . . . . .	284
A.1.9	Brick names . . . . .	284
A.1.10	Brick reuse . . . . .	285
A.1.11	Brick namespacing . . . . .	285
A.1.12	Spin . . . . .	286
A.1.13	Spin controls and method controls . . . . .	287
A.1.14	Program structure . . . . .	288
A.2	with bricks . . . . .	290
A.2.1	Spin controls . . . . .	290
A.2.2	Connection with the outside world . . . . .	291
A.2.3	Flexible format in with bricks . . . . .	292
A.2.4	Defining identical controls . . . . .	292
A.3	do bricks . . . . .	293
A.3.1	Assigning control values . . . . .	293
A.3.2	The order of processing bricks . . . . .	293
A.3.3	Spin controls . . . . .	294
A.3.4	Method controls . . . . .	295
A.3.5	IF/ELSE IF/ELSE logic . . . . .	297
A.3.6	Nested IF logic . . . . .	298
A.3.7	Group spin controls . . . . .	298
A.3.8	Timing spin controls . . . . .	300
A.3.9	Flexible format in do bricks . . . . .	301
A.4	Programming in Meshoil . . . . .	302
A.4.1	Simplest program . . . . .	302
A.4.2	Automatic datatyping . . . . .	302
A.4.3	Defining controls . . . . .	303
A.4.4	Concurrency and the scope of programming constructs . . . . .	304
A.4.5	Local methods . . . . .	305
A.5	Abstract API layer . . . . .	307
A.5.1	Calling API methods . . . . .	307
A.5.2	Abstract methods . . . . .	308
A.5.3	Platform-specific coding . . . . .	310
A.5.4	Building a GUI . . . . .	312
A.6	Using namespacing . . . . .	315
A.6.1	Nested namespacing . . . . .	315
A.6.2	Absolute naming within namespaces . . . . .	315
A.6.3	Working with multiple namespacing schemes . . . . .	316

## CONTENTS

A.7	Examples of more complex expressions . . . . .	319
A.7.1	Logical expressions . . . . .	319
A.7.2	Numerical expressions . . . . .	320
A.7.3	String-based expressions . . . . .	321
<b>B</b>	<b>Discussion on Meshoil</b>	<b>323</b>
B.1	Language design . . . . .	323
B.1.1	How easy is it to program in Meshoil? . . . . .	323
B.1.2	Increasing the descriptive power of a hierarchy . . . . .	323
B.1.3	Trading execution efficiency for usability? . . . . .	323
B.2	Beyond the functionality of a conventional language . . . . .	325
B.2.1	Replacing programming constructs with more generic equivalents . . . . .	325
B.2.2	Confluent conditional expressions . . . . .	326
B.3	OO programming principles . . . . .	329
B.3.1	Abstraction . . . . .	329
B.3.2	Objectification . . . . .	330
B.3.3	Code reuse . . . . .	330
B.3.4	Inheritance . . . . .	331
B.3.5	Encapsulation . . . . .	333
B.3.6	Polymorphism . . . . .	333
<b>C</b>	<b>Tools</b>	<b>335</b>
C.1	The Mesh engine . . . . .	335
C.1.1	The main window . . . . .	335
C.1.2	Control windows . . . . .	336
C.1.3	Implementation . . . . .	338
C.2	XML-based tools . . . . .	339
C.2.1	The Meshoil tester . . . . .	339
C.2.2	The Meshoil viewer . . . . .	342
	<b>Glossary</b>	<b>343</b>
	<b>References</b>	<b>347</b>
	<b>Index</b>	<b>351</b>

# Abstract

The Internet is a vastly under-utilised resource, only used for half of the IT story.

Describe the Internet in two words and many might say ‘sharing knowledge’. But sharing information is more accurate. It’s just that all the principle ways we use the Internet—the Web, email and media streaming—happen to be examples where information is in the form of knowledge.

But IT—*Information* Technology—has another side: the realm of software programming where information means the dynamic control of how things work. The Internet is the driving force in the IT industry, so why isn’t it also known for sharing control? True, there are examples of specialised, one-off software applications interfacing with each other via the Internet, but there has yet to be any systematic and universal attempt to exploit the potential of the Internet for control-IT in the way we have seen it for knowledge-IT.

Taking the strengths of the Web model as a starting point, this thesis proposes a parallel, dynamic world to the Web called *The Mesh*. In the same way that the Web seamlessly connects databases of the world to provide a global font of knowledge, the Mesh would connect software of the world to provide a global means of control. The Mesh would embody all the successful, empowering features of the Web. Everyone would have a say in how things work, mirroring Web 2.0’s user-generated content but for software instead of media.

In being a universally integrated design approach for device control, the Mesh would encompass a number of research areas working on the control issue at the big picture level. It would address the problems of universal usability and ubiquitous computing. It would also provide solutions in agent-based systems and grid computing.

But many features of the Mesh would simply be unique. They would change the way we go about software design, leading to new opportunities for users, programmers and manufacturers alike.

The key to everything is design simplicity.

A concept demonstrator has been developed as an integral part of this research project. It shows that the Mesh is both feasible and practical. Examples of programs run in the concept demonstrator are discussed, showing exactly how the Mesh would be built and how it would work.

## CONTENTS

# Keywords

## General

- Grid computing
- Interface agents
- Internet applications
- Markup languages
- Object-oriented programming
- Ubiquitous computing
- Universal design
- Universal usability
- User interface description languages
- XML.

## This thesis

- Abstract API layer
- Abstract hardware layer
- Atomic-level networking
- Building-brick design flexibility
- Code filtering
- Code integrity through accreditation
- Cost-free user flexibility
- Cut and paste programming
- Engineering integration by layering
- Futures programming
- Ghost-in-the-machine API

## CONTENTS

- Ghost processing
- Globally-unified hardware control
- Hardware reuse
- Meshoil programming language
- Open-plan thinking
- Point-accurate interoperation
- Program morphing
- Responsibility optimisation
- Software-team scalability
- Superprogramming
- The Mesh
- Transparent programming environment
- Universal agent
- User-generated software
- Virtual hardware
- Virtual logic.

# Statement of originality

This work contains no material which has been accepted for the award of any other degree or diploma in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text.

I give consent to this copy of my thesis, when deposited in the University Library, being made available for loan and photocopying, subject to the provisions of the Copyright Act 1968.

---

Martin Lumisden Strange

---

Date

## CONTENTS

# Acknowledgments

First and foremost, I am much indebted to Dr Matthew Sorell for his constant support and encouragement on this long and interesting research journey. I have benefitted immensely from his guidance, wisdom and knowledge. His depth of understanding of engineering realities has provided me with a much needed anchor on many an occasion.

I am also indebted to Associate Professor Michael Liebelt. Without his and Dr Sorell's support, I would not have had the opportunity to receive a faculty divisional scholarship and undertake this PhD. Being given the time and resources to develop one's own ideas in a supportive environment is an enormous privilege, one that I have much enjoyed and appreciated.

I would also like to thank Brad Alexander for his help 'above and beyond the call of duty'. His timely input on critical programming issues helped point me in the right direction a number of times, saving me considerable time and effort.

Finally, I would like to thank my family for their love and support including: my wife, Angela, who has maintained the faith while having to endure the selfish demands of her husband's PhD mistress; my father, Tom, whom I have always admired for his open, enquiring and worldly mind; and my mother, Mary, who continues to amaze me with her unstructured but brilliant mind that, in a single leap of insight, manages to get right to the heart of an issue—how does she do it?!

## CONTENTS

# List of Figures

1.1	The Mesh . . . . .	3
1.2	The universal design concept . . . . .	5
1.3	How does nature handle design? . . . . .	18
1.4	A building-brick approach . . . . .	19
2.1	Meshoil’s four XML elements for writing code . . . . .	24
2.2	Brick nesting . . . . .	24
2.3	Meshoil user interface program for controlling a domestic fan . . . . .	27
2.4	Running the fan user interface (1 of 3) . . . . .	30
3.1	Main control window for simulating the operation of two fans . . . . .	35
3.2	Hardware controls of fan1 . . . . .	37
3.3	Simulation settings: Standard (1-speed, pan) user interface . . . . .	38
3.4	Standard (1-speed, pan) user interface (1 of 2) . . . . .	39
3.5	Simulation settings: Simplified (1-speed) user interface . . . . .	41
3.6	Simplified (1-speed) user interface . . . . .	42
3.7	The conventional trade-off . . . . .	43
3.8	The software designer’s current headache . . . . .	44
3.9	Expanded user solution space . . . . .	45
3.10	Simulation settings: Simplified (1-speed, pan, LED) user interface . . . . .	46
3.11	Standard (1-speed, pan, LED) user interface (1 of 2) . . . . .	47
3.12	Simulation settings: Advanced (3-speed, pan) user interface . . . . .	50
3.13	Advanced (3-speed, pan) user interface (1 of 3) . . . . .	51
3.14	Simulation settings: Convert 3-speed to 5-speed plugin . . . . .	54
3.15	Convert 3-speed to 5-speed plugin (1 of 4) . . . . .	55
3.16	Simulation settings: DIY user interface—Web 2.0 style . . . . .	60
3.17	Plugin for toggling the fan on and off . . . . .	61
3.18	User Interfaces For You Meshpoint . . . . .	62
3.19	Simulation settings: Thermostat plugin . . . . .	64
3.20	Thermostat plugin (1 of 3) . . . . .	65
3.21	Simulation settings: Panning timeout plugin . . . . .	70
3.22	Panning timeout plugin (1 of 3) . . . . .	71
3.23	Simulation settings: Safety cutout plugin . . . . .	75
3.24	Safety cutout plugin . . . . .	76
3.25	Wilson Electric Meshpoint . . . . .	79
3.26	Simulation settings: Latest design . . . . .	82
3.27	Simulation settings: Stopwatch user interface . . . . .	85

## LIST OF FIGURES

3.28	Stopwatch user interface (1 of 2)	87
3.29	Simulation settings: All user interfaces	89
3.30	Simulation settings: Remote control	93
3.31	Remote control from other fan plugin	95
3.32	Simulation settings: fan2 as a hand-held remote controller	97
3.33	Simulation settings: Bi-directional remote control	100
3.34	Simulation settings: Cloned fans	101
3.35	Clone of other fan operation plugin	102
3.36	Simulation settings: Restricted fan usability	103
3.37	Only usable if other fan is on plugin	104
3.38	Simulation settings: Alternate panning	105
3.39	Start panning if other fan stops panning plugin	106
3.40	Simulation settings: Advanced (5-speed, pan) ghost processor user interface	107
3.41	Program running as a single thread	108
3.42	Program multithreaded for increased executional efficiency	108
3.43	Ghost processing	110
3.44	Remote program management	112
3.45	Complete code outsourcing	113
3.46	Simulation settings: Faulty advanced (3-speed, pan) user interface	114
3.47	Faulty code for black and white key panning operation	116
4.1	Simple languages have many advantages	121
5.1	The user interface program at the end of design stage 1	129
5.2	The user interface program at the end of design stage 2 (1 of 2)	130
5.3	The user interface program at the end of design stage 3 (1 of 3)	132
5.4	The user interface program at the end of design stage 4 (1 of 5)	136
5.5	The user interface program at the end of design stage 5 (1 of 7)	142
5.6	Running the safety fan user interface (1 of 3)	149
5.7	Mapping the knobs and keys to radio functions	153
5.8	Radio user interface (1 of 8)	154
5.9	Running the radio user interface (1 of 9)	162
5.10	Dashboard controls and floor pedals	172
5.11	Model of vehicle dynamics (1 of 2)	174
5.12	Running the sports car simulator (1 of 6)	176
5.13	Sports car with safety fan and radio	182
5.14	Calculator user interface (1 of 13)	186
5.15	Running the calculator user interface (1 of 3)	199
5.16	Performing calculations in Reverse Polish notation (1 of 3)	203
5.17	The Divide method in the method library	206
5.18	Using the Java API to support the user interface	207
5.19	Handling error codes returned after a method call	208
5.20	Running the 3-in-1 calculator user interface (1 of 8)	209
5.21	Handling character entry on the mobile phone keypad (1 of 2)	220
5.22	Running the mobile phone/emergency pager user interface (1 of 10)	224
5.23	The Profiles brick supporting everything to do with profiles	234
6.1	Simulating thermometer operation (1 of 6)	242

6.2	Running the thermometer simulation (1 of 7)	247
6.3	Lego's RCX programmable brick	254
6.4	Meshoil equivalent of the robot program (1 of 5)	259
6.5	Running the robot simulation (1 of 3)	263
7.1	Systematic application of open-plan thinking	269
A.1	Meshoil's four XML elements for writing code	278
A.2	<a> for annotation	279
A.3	<c> for control	279
A.4	<d> for data	280
A.5	<b> for brick	280
A.6	Brick nesting	281
A.7	Brick structure	283
A.8	Bricks with multiple <c> and <d> elements	284
A.9	Brick names	284
A.10	Brick reuse	285
A.11	Brick namespacing	285
A.12	Spin	286
A.13	Spin controls	287
A.14	Program structure	288
A.15	Spin controls in with bricks	291
A.16	Direction spin reflects hardware capability	291
A.17	Flexible format in with bricks	292
A.18	Defining identical controls	292
A.19	Assigning control values	293
A.20	The order of processing bricks	293
A.21	Spin controls in do bricks	295
A.22	Using method controls	296
A.23	IF/ELSE IF/ELSE logic	297
A.24	Nested IF logic	298
A.25	Group spin controls	299
A.26	Timing spin controls	300
A.27	Flexible format in do bricks	301
A.28	Simplest program	302
A.29	Automatic datatyping	302
A.30	Defining controls	303
A.31	Concurrency and the scope of programming constructs	304
A.32	Local methods	306
A.33	Calling API methods	307
A.34	Abstract methods (1 of 2)	309
A.35	Platform-specific coding (1 of 2)	311
A.36	Building a GUI (1 of 2)	313
A.37	Nested namespacing	315
A.38	Absolute naming within namespaces	316
A.39	Working with multiple namespacing schemes	317
A.40	Logical expressions	319
A.41	Numerical expressions	320

## LIST OF FIGURES

A.42 String-based expressions . . . . .	322
B.1 Emergency shutdown procedure—executionally inefficient . . . . .	326
B.2 Emergency shutdown procedure—code duplication . . . . .	327
B.3 Emergency shutdown procedure—executionally efficient without code duplication . . . . .	328
B.4 Abstraction . . . . .	329
B.5 Objectification . . . . .	330
B.6 Code reuse . . . . .	330
B.7 Control inheritance . . . . .	331
B.8 Spin inheritance . . . . .	332
B.9 Encapsulation . . . . .	333
B.10 Polymorphism . . . . .	334
C.1 Main window of the Mesh engine . . . . .	335
C.2 Example of the Mesh engine running . . . . .	337

# List of Tables

1.1	Current research areas working on the control issue at the big picture level . . . . .	6
1.2	Emulating the main design features of the Web . . . . .	15
1.3	Open-plan thinking . . . . .	18
1.4	Tools created for the concept demonstrator . . . . .	20
3.1	Examples of problems identified in automated accreditation of software . . . . .	118
6.1	Half the complexity for the Meshoil solution . . . . .	253
7.1	Derivation of unique Mesh principles . . . . .	270
7.2	Addressing design issues faced by current research areas . . . . .	271
7.3	Internet dependency of Mesh principles . . . . .	272
A.1	Spin controls in <i>with</i> bricks . . . . .	290
A.2	Spin controls in <i>do</i> bricks . . . . .	294
A.3	Method controls in <i>do</i> bricks . . . . .	295
A.4	Examples of the generalised notation for an <i>any</i> -based group spin control . . . . .	299
C.1	Buttons in the main window of the Mesh engine . . . . .	336
C.2	Java classes used in the Mesh engine application . . . . .	338

LIST OF TABLES