

**VIABLE SOFTWARE**  
**THE**  
**INTELLIGENT CONTROL PARADIGM**  
**FOR**  
**ADAPTABLE AND ADAPTIVE ARCHITECTURE**

**CHARLES EDWARD HERRING, JR.**

Bachelor of Science, *Cum Laude*

Master of Computer Science

This dissertation is submitted in fulfilment  
of the requirements for the degree of

**DOCTOR OF PHILOSOPHY**



**THE UNIVERSITY OF QUEENSLAND**

Department of Information Technology and Electrical Engineering

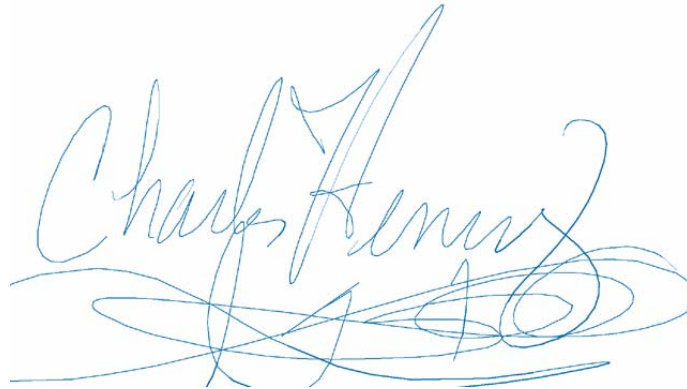
Brisbane, Australia

2002



## DECLARATION

The work presented in this thesis is, to the best of my knowledge and belief, original, except as acknowledged in the text. I hereby declare that I have not submitted this material either in whole or in part, for a degree at this or any other university.

A handwritten signature in blue ink, appearing to read "Charles Henry", with a large, stylized flourish underneath.



## ABSTRACT

The Intelligent Control Paradigm for software architecture is the result of this work. The Viable Software Approach is developed as an instance of the paradigm. The approach uses the Viable System Model as the basis for software system architecture. The result is a model-based architecture and approach for developing software systems by piecemeal *adaptation* with the goal that they become *adaptive* systems at runtime. Software built in this manner is called Viable Software. Viable Software represents a unifying class of self-controlling software that is an “intelligent” control system.

Cybernetics, Control Theory, and Complexity Theory are the background for this work, and aspects relevant to this work are presented. These results are related to software architecture and software engineering. Rationale for the selection of the Viable System Model as a basis for software systems is given. The Viable System Model is described. The model is restated as an Alexanderian “pattern language” to make it more accessible to software engineering. A Viable Software Approach is proposed and expressed in the form of a Product Line Architecture that arranges the Viable System Model, the Viable Software Architecture, a Viable Component Framework, and a Component Transfer Protocol into a system for generative programming. An important result is the formalisation of the pattern of the Viable System into the interface specifications of the Viable Component.

Three case studies illustrate the approach. The first is an analysis and extension of the Groove collaboration system. This study shows how the approach is used to map an existing system into the Viable Software Architecture and add fuzzy-adaptive user interface controllers. The second study presents the design and detailed software construction of an adaptive camera controller as part of a smart environment. The final study shows how a Business-to-Business e-Commerce system can be evolved and an expert system-based controller developed to implement business contracts.



## **ACKNOWLEDGEMENTS**

Prof Simon Kaplan for gentle guidance and generous support.

Dr Robert Colomb for serving as auditor.

Dr Clemens Szyperski, Microsoft Research, for enthusiastic encouragement.

Dr Zoran Milosevic, DSTC Pty Ltd, for continuous support, many valuable discussions, collaboration on the e-Commerce case study, and for tutoring on the fine points of UML.

John Mansfield and Blaize Rhodes for comments and corrections.

Mr David Barbagallo, CEO, and the late Prof Melfyn Lloyd, Research Director, DSTC Pty Ltd, for financial support.



**DEDICATION**

*TO MY PARENTS*

**CHARLES EDWARD HERRING, SR.**

**&**

**SAMMIE SPRINGER HERRING**



# CONTENTS

<b>CHAPTER 1 DEALING WITH COMPLEXITY .....</b>	<b>1</b>
1.1 THESIS.....	3
1.2 THESIS ORGANIZATION .....	4
1.3 SUMMARY OF CONTRIBUTIONS .....	6
<b>CHAPTER 2 BACKGROUND AND RELATED WORK .....</b>	<b>9</b>
2.1 CYBERNETICS AND GST .....	11
2.1.1 <i>Background of the Viable System Model</i> .....	11
2.1.2 <i>The Law of Requisite Variety</i> .....	12
2.1.3 <i>Homeostasis, Ultrastability, and Adaptation</i> .....	15
2.1.4 <i>Self-Organisation and Complexity</i> .....	16
2.2 COMPLEXITY THEORY .....	18
2.2.1 <i>The Santa Fe Institute and Complexity Theory</i> .....	18
2.2.2 <i>Adaptation, Hidden Order, and Emergence</i> .....	19
2.2.3 <i>Summary</i> .....	22
2.3 CONTROL THEORY MODELS.....	23
2.3.1 <i>Control: Classical, Modern and Contemporary</i> .....	24
2.3.2 <i>Feedback Control</i> .....	25
2.3.3 <i>Adaptive, Learning, and Supervisory Control</i> .....	25
2.3.4 <i>Intelligent Autonomous Control</i> .....	27
2.3.5 <i>Summary</i> .....	30
2.4 SOFTWARE ARCHITECTURE .....	31
2.4.1 <i>Definitions and Terminology</i> .....	32
2.4.2 <i>Shaw's Control Paradigm</i> .....	34
2.4.3 <i>Kokar's Self-Controlling Software</i> .....	38
2.4.4 <i>Other related work</i> .....	41
2.4.5 <i>Summary</i> .....	42
2.5 SOFTWARE ENGINEERING.....	43
2.5.1 <i>Viable System Model for Software Process</i> .....	43
2.5.2 <i>Adaptive Development Approaches</i> .....	46
2.5.3 <i>Adaptive Architectures and Programming</i> .....	49
2.5.4 <i>Self-Stabilising and Self-Adaptive Software</i> .....	51
2.6 SUMMARY .....	53
<b>CHAPTER 3 VIABLE SYSTEM MODEL .....</b>	<b>57</b>
3.1 OVERVIEW OF THE VSM .....	57
3.2 BASIC CONCEPTS .....	60
3.3 SYSTEM-IN-FOCUS .....	62
3.4 REGULATION.....	64
3.5 HOMEOSTATICS IN THE VSM .....	67
3.6 OPERATIONAL CONTROL .....	68
3.7 ADAPTATION.....	70
3.8 CLOSURE.....	72
3.9 ORGANISATION, MANAGEMENT AND AUTONOMY .....	73
3.10 SUMMARY .....	75
<b>CHAPTER 4 THE VIABLE SYSTEM PATTERN LANGUAGE .....</b>	<b>77</b>

4.1	SEPARATE CONTROL (1)	80
4.2	OPERATIONAL CONTROL (2)	82
4.3	REGULATORY CENTRE (3)	84
4.4	SPORADIC AUDIT (4)	86
4.5	ADAPTIVE CONTROL (5)	88
4.6	SUPERVISORY CONTROL (6)	90
4.7	ALERTS (7)	92
4.8	RECURSIVE COMPOSITIONS (8)	94
4.9	HOMEOSTATIC LOOP (9)	96
4.10	SUMMARY AND CONCLUSION	97
4.11	ACKNOWLEDGEMENTS	100
<b>CHAPTER 5 THE VIABLE SOFTWARE APPROACH</b>		<b>101</b>
5.1	SOFTWARE ARCHITECTURE: NATURE AND REPRESENTATION	102
5.2	THE VIABLE SOFTWARE ARCHITECTURE	106
5.3	FROM ARCHITECTURE TO DESIGN: A FRAMEWORK	112
5.4	TOWARD A DOMAIN-SPECIFIC PRODUCT LINE	120
5.5	AN EXAMPLE: MODEL-VIEW-CONTROLLER REVISITED	125
5.6	DISCUSSION	130
<b>CHAPTER 6 CASE STUDY: GROOVE</b>		<b>135</b>
6.1	OVERVIEW OF THE GROOVE SYSTEM	135
6.2	MAPPING GROOVE INTO THE VSA	139
6.2.1	<i>Groove Transceiver</i>	139
6.2.2	<i>Shared Spaces</i>	141
6.2.3	<i>Tools Sets</i>	142
6.2.4	<i>Web Services</i>	143
6.3	IMPLEMENTING ADAPTIVE FEATURES	145
6.3.1	<i>Groove Transceiver</i>	147
6.3.2	<i>Shared Spaces, Tool Sets and Tools</i>	153
6.3.3	<i>Web Services and Groove.net</i>	157
6.4	SUMMARY AND DISCUSSION	158
<b>CHAPTER 7 CASE STUDY: A SMART CAMERA CONTROLLER</b>		<b>161</b>
7.1	THE SMART ENVIRONMENT SETTING	161
7.2	A SMART CAMERA CONTROLLER	164
7.2.1	<i>Requirements Analysis</i>	165
7.2.1.1	Clearly identify the system (the “System-In-Focus”) under analysis, define its boundaries, and give it a name	165
7.2.1.2	Develop a model of the process of the SmartCam by identifying inputs, outputs, and critical control variables.	165
7.2.1.3	Identify system interfaces depending on the context	166
7.2.1.4	Determine how and in what format plans and activities will be represented for use by the Regulatory System.	168
7.2.1.5	Determine the requirements for system audit	168
7.2.1.6	Analyse the adaptive requirements of the system	168
7.2.1.7	Determine the supervisory level user interface	169
7.2.2	<i>Architecture</i>	170
7.2.3	<i>Design and Implementation</i>	171
7.3	SUMMARY	177
<b>CHAPTER 8 CASE STUDY: E-COMMERCE</b>		<b>179</b>
8.1	EVOLVING AN E-COMMERCE SYSTEM	180
8.1.1	<i>e-Commerce Reference Model and Reference Requirements</i>	181
8.1.2	<i>e-Commerce Application Requirements</i>	182
8.1.3	<i>e-Commerce Domain Reference Architecture</i>	182
8.1.4	<i>Phase 1: Operations</i>	184
8.1.4.1	Purchasing	184
8.1.4.2	Shipping	185

8.1.4.3	Contracting .....	185
8.1.4.4	Accounting .....	185
8.1.4.5	Auditing .....	185
8.1.4.6	Scheduling .....	186
8.1.4.7	Phase 1 Implementation .....	186
8.1.5	<i>Phase 2 Planning</i> .....	188
8.1.5.1	Operations Analysis .....	188
8.1.5.2	Market Analysis and Exchange Web Site .....	189
8.1.5.3	Vendor System .....	190
8.1.6	<i>Phase 3: Executive</i> .....	190
8.1.7	<i>Summary</i> .....	192
8.2	RETAILER.COM AND E-CONTRACTS .....	193
8.2.1	<i>Implementing B2B Electronic Contracts</i> .....	194
8.2.2	<i>The B2B Framework and Protocols</i> .....	196
8.2.3	<i>Stability Analysis of the B2B Federation</i> .....	201
8.3	NOTE ON B2B HOMEOSTATICS .....	203
8.4	SUMMARY AND DISCUSSION .....	205
<b>CHAPTER 9 CONCLUSION AND CONTRIBUTIONS .....</b>		<b>209</b>
9.1	CONTRIBUTIONS TO COMPUTER SCIENCE .....	216
9.2	CONTRIBUTIONS TO THE VIABLE SYSTEM MODEL .....	218
9.3	FUTURE WORK .....	218
<b>APPENDIX A A LOOK AT LIVING SYSTEMS THEORY .....</b>		<b>221</b>
A.1	OVERVIEW OF LST .....	221
A.2	MODEL-VIEW-CONTROLLER AND AGENT ARCHITECTURE .....	225
A.3	DESIGN PATTERNS .....	226
A.4	MICROSOFT'S DISTRIBUTED COMPONENT ARCHITECTURE .....	227
A.5	INFORMATION TECHNOLOGY APPLICATIONS .....	230
A.6	SUMMARY .....	232
A.7	BIBLIOGRAPHY .....	233
<b>APPENDIX B VSM APPLICATIONS UPDATE .....</b>		<b>235</b>
B.1	TEAM SYNTTEGRITY .....	235
B.2	OTHER GROUPS THAT USE THE VSM .....	236
B.3	RECENT LITERATURE ON THE VSM .....	238
B.4	BIBLIOGRAPHY .....	241
<b>APPENDIX C SMARTCAM DOCUMENTATION .....</b>		<b>245</b>
C.1	CLASS DEFINITIONS AND BEHAVIOURS .....	245
	<i>SmartCam Plant and Accountant</i> .....	247
	<i>SmartCam Operations, Regulator, and Auditor</i> .....	251
	<i>SmartCam Planning</i> .....	256
	<i>SmartCam Executive</i> .....	258
	<i>SmartCam Controller and MSMQBus</i> .....	261
C.2	SMARTCAM MESSAGE SCHEMAS .....	262
	<i>Accounting</i> .....	264
	C.2.1.1    SCAcctRptPeriod .....	264
	C.2.1.2    SCAcctRptPeriod Instance .....	264
	<i>Auditor</i> .....	264
	C.2.1.3    SCAuditRptError .....	264
	C.2.1.4    SCAuditRptError Instance .....	265
	C.2.1.5    SCAuditSporadic .....	265
	<i>SCAuditRequest</i> .....	265
	<i>Executive</i> .....	266
	C.2.1.6    SCExecResources .....	266
	C.2.1.7    SCExecResources Instance .....	266
	C.2.1.8    SCExecMsgStart .....	267
	C.2.1.9    SCExecMsgStart Instance .....	267
	C.2.1.10   SCExecPolicySOP .....	268

	C.2.1.11	SCExecPolicySOP Instance .....	268
<i>Operations</i>		.....	272
	C.2.1.12	SCOpsMsgCommand .....	272
	C.2.1.13	SCOpsMsgCommand Instance.....	272
	C.2.1.14	SCOpsMsgStartCmd.....	272
	C.2.1.15	SCOpsMsgStartCmd Instance .....	273
	C.2.1.16	SCOpsStartParams .....	274
	C.2.1.17	SCOpsStartParams Instance .....	274
	C.2.1.18	SCOpsPlan .....	275
	C.2.1.19	SCOpsPlan Instance.....	276
	C.2.1.20	SCOpsStateChange.....	277
	C.2.1.21	SCOpsStateChange Instance .....	277
	C.2.1.22	SCOpsRptAuditErr .....	278
	C.2.1.23	SCOpsRptRoutine.....	278
<i>Planning</i>		.....	279
	C.2.1.24	SCPlanMsgAdvice.....	279
	C.2.1.25	SCPlanMsgAdvice Instance .....	279
	C.2.1.26	SCPlanOpsUpdate .....	280
	C.2.1.27	SCPlanPlanParams.....	280
	C.2.1.28	SCPlanPlanParams Instance .....	281
	C.2.1.29	SCPlanRptRoutine.....	281
<i>Plant</i>		.....	282
		<i>SCPlantAck</i> .....	282
	C.2.1.30	SCPlantAck Instance .....	283
	C.2.1.31	SCPlantAlgedonic.....	283
	C.2.1.32	SCPlantMsgModel.....	283
	C.2.1.33	SCPlantMsgModel Instance .....	284
	C.2.1.34	SCPlantStateChng.....	284
	C.2.1.35	SCPlantStateChng Instance .....	285
	C.2.1.36	SCPlantRptRoutine .....	285
	C.2.1.37	SCPlantRptRoutine Instance .....	286
	C.2.1.38	SCPlantSelfTest .....	286
C.3		EXAMPLE PROTOCOL SEQUENCE .....	288
<b>APPENDIX D RETAILER.COM DESIGN AND IMPLEMENTATION .....</b>			<b>293</b>
D.1		OVERVIEW OF FUNCTIONALITY OF RETAILER.COM .....	294
D.2		CONTRACT FOR SERVICES: TERMS AND CONDITIONS.....	299
D.3		SOFTWARE IMPLEMENTATION.....	303
D.4		SUMMARY .....	316
<b>BIBLIOGRAPHY .....</b>			<b>319</b>

# FIGURES

FIGURE 1.1 RELATION OF HUMAN CONTROL AND COMPUTING .....	2
FIGURE 2.1 CONCEPT MAP OF RELATED WORK.....	10
FIGURE 2.2 VARIETY AND REGULATION .....	13
FIGURE 2.3 CYBERNETIC STRUCTURAL MODEL (HOWLAND).....	14
FIGURE 2.4 ASHBY'S HOMEOSTAT .....	16
FIGURE 2.5 MODELS AND COMPLEXITY .....	23
FIGURE 2.6 OPEN AND CLOSED-LOOP CONTROLLERS .....	25
FIGURE 2.7 ADAPTIVE CONTROLLERS.....	26
FIGURE 2.8 LEARNING CONTROLLER .....	27
FIGURE 2.9 INTELLIGENT AUTONOMOUS CONTROL FUNCTIONAL ARCHITECTURE .....	28
FIGURE 2.10 ADAPTIVE-RECONFIGURABLE CONTROLLER .....	39
FIGURE 2.11 SELF-ADAPTING SOFTWARE METHODOLOGY .....	48
FIGURE 2.12 CONCEPT MAP OF RELATED WORK (2).....	53
FIGURE 3.1 NEUROPHYSIOLOGICAL CONTROL: SOMATIC, SYMPATHETIC, AND PARASYMPATHETIC.....	58
FIGURE 3.2 VIABLE SYSTEM MODEL (SIMPLIFIED) OF AN ORGANISATION WITH FOUR DIVISIONS .....	59
FIGURE 3.3 RECURSIVE DIMENSIONS OF A VIABLE SYSTEM.....	61
FIGURE 3.4 THE SYSTEM-IN-FOCUS (INSET: UNFOLDING) .....	62
FIGURE 3.5 SYSTEMS 1 AND 2 (REGULATORY CENTRE) .....	65
FIGURE 3.6 DETAIL OF A HOMEOSTATIC LOOP .....	68
FIGURE 3.7 SYSTEM 3: OPERATIONAL CONTROL .....	69
FIGURE 3.8 SYSTEM 3* AND TWO SYSTEM 1'S (1A AND 1B).....	70
FIGURE 3.9 SYSTEM 4: PLANNING AND ADAPTATION .....	71
FIGURE 3.10 CLOSURE OF THE VIABLE SYSTEM: POLICY .....	72
FIGURE 3.11 THE COMPLETE VIABLE SYSTEM WITH ALGEDONIC SIGNAL.....	73
FIGURE 5.1 VIABLE PRODUCTION LINE ARCHITECTURE.....	104
FIGURE 5.2 VIABLE SOFTWARE PRODUCT LINE.....	105
FIGURE 5.3 DOMAIN-SPECIFIC PRODUCT LINE ARCHITECTURE .....	106
FIGURE 5.4 THE CONTROL PARADIGM (STAGE 1).....	108
FIGURE 5.5 SEPARATE REGULATOR (COORDINATION, STAGE 2).....	109
FIGURE 5.6 AUDITOR (STAGE 3).....	109
FIGURE 5.7 ADAPTATION (STAGE 4) .....	110
FIGURE 5.8 SUPERVISION (STAGE 5) .....	111
FIGURE 5.9 COMPOSITION OF SYSTEMS (RECURSIVE, SYSTEMS-OF-SYSTEMS).....	112
FIGURE 5.10 A SYSTEM WITH TWO EMBEDDED SYSTEMS (ENVIRONMENT NOT SHOWN).....	113
FIGURE 5.11 THE VIABLE COMPONENT .....	114
FIGURE 5.12 THE CONNECTION PATTERN .....	116
FIGURE 5.13 INTERFACE CONTRACT PATTERN.....	117
FIGURE 5.14 INTERFACE ESTABLISHMENT SEQUENCE.....	118
FIGURE 5.15 MODEL-VIEW-CONTROLLER .....	126
FIGURE 5.16 "MODEL-VIEW-CONTROLLER" AS CLOSED-LOOP CONTROL SYSTEM .....	127
FIGURE 5.17 MIGRATION OF HUMAN SKILLS INTO SOFTWARE SYSTEMS.....	130
FIGURE 5.18 AIS DSSA REFERENCE ARCHITECTURE .....	132
FIGURE 5.19 COGNITIVELY RICH AGENT ARCHITECTURE.....	133
FIGURE 6.1 GROOVE SHARED SPACE ARCHITECTURE .....	136
FIGURE 6.2 GROOVE TRANSCIEVER OPENING WINDOW.....	137
FIGURE 6.3 GROOVE SHARED SPACE .....	137
FIGURE 6.4 GROOVE FRAMEWORK ARCHITECTURE .....	138
FIGURE 6.5 TRANSCIEVER AND SHARED SPACES.....	140

FIGURE 6.6 SHARED SPACE WITH TOOL SET AND STORAGE .....	141
FIGURE 6.7 TOOL SET AND TOOLS .....	142
FIGURE 6.8 GROOVE.NET AND TRANSCEIVERS .....	144
FIGURE 6.9 FUZZY SET MEMBERSHIP FUNCTIONS .....	150
FIGURE 6.10 GRAPHICAL REPRESENTATION OF FUZZY INFERENCE .....	152
FIGURE 7.1 THE SMART LECTURE ROOM AND EQUIPMENT LAYDOWN .....	162
FIGURE 7.2 VIABLE SOFTWARE ARCHITECTURE DIAGRAM FOR SMART LECTURE ROOM.....	163
FIGURE 7.3 SMARTCAM ROLE-BASED ARCHITECTURE.....	170
FIGURE 7.4 SMARTCAM CLASS DIAGRAM.....	171
FIGURE 7.5 SMARTCAM SETUP PROTOCOL.....	172
FIGURE 7.6 ROUTINE OPERATION AND STATE CHANGE (OBSERVATION).....	174
FIGURE 7.7 SNAP-SHOT OF SMARTCAM RUNNING IN SIMULATION MODE.....	176
FIGURE 8.1 VIABLE SOFTWARE APPROACH PRODUCT LINE.....	181
FIGURE 8.2 RETAILER.COM AND FIRST RECURSION IDENTIFYING B2C AND B2B SUBSYSTEMS.....	182
FIGURE 8.3 RETAILER.COM'S B2B ARCHITECTURE .....	183
FIGURE 8.4 PHASES OF IMPLEMENTATION AND THREE DIMENSIONS OF CONTROL .....	192
FIGURE 8.5 THE VIABLE COMPONENT AND INTERFACES .....	196
FIGURE 8.6 B2B FEDERATION.....	200
FIGURE 8.7 HOMEOSTATS (AFTER BEER).....	204
FIGURE 9.1 UBIQUITOUS COMPUTING.....	211
FIGURE A.1 PROTOTYPICAL UNIT (INFORMATION PROCESSING SUBSYSTEMS ONLY).....	223
FIGURE A.2 MVC MEETS LST .....	225
FIGURE A.3 COM INTERFACE STRUCTURE .....	228
FIGURE C.1 SMARTCAM CLASS DIAGRAM .....	245
FIGURE C.2 SMARTCAM PACKAGE DIAGRAM SHOWING DISTRIBUTED COMPONENTS .....	246
FIGURE C.3 SMARTCAM PLANT PROPERTIES .....	248
FIGURE C.4 SMARTCAM PLANT: METHODS .....	250
FIGURE C.5 SMARTCAM ACCOUNTANT.....	250
FIGURE C.6 SMARTCAM OPERATIONS PROPERTIES .....	252
FIGURE C.7 SMARTCAM OPERATIONS METHODS.....	253
FIGURE C.8 SMARTCAM REGULATOR .....	255
FIGURE C.9 SMARTCAM AUDITOR .....	256
FIGURE C.10 SMARTCAM PLANNING PROPERTIES .....	257
FIGURE C.11 SMARTCAM PLANNING METHODS .....	258
FIGURE C.12 SMARTCAM EXECUTIVE PROPERTIES .....	259
FIGURE C.13 SMARTCAM EXECUTIVE METHODS.....	260
FIGURE C.14 SMARTCAM CONTROLLER.....	261
FIGURE C.15 SMARTCAM MSMQBUS .....	261
FIGURE D.1 RETAILER.COM CLASS DIAGRAM.....	293
FIGURE D.2 OPENING MENUS AND SIMULATION DISPLAY.....	294
FIGURE D.3 CONTRACTS AND ACTIVITIES (EXECUTIVE, PLANNING, AND OPERATIONS CONSOLES) .....	296
FIGURE D.4 AUDITOR AND SCHEDULER ACTIVITIES .....	297
FIGURE D.5 ACCOUNTANT CONSOLE (SHOWING SIMULATED CUSTOMERS) .....	298
FIGURE D.6 SEQUENCE OF ACTIVITIES TO SET UP A CONTRACT .....	299
FIGURE D.7 EXAMPLE CONTRACT FOR SERVICES .....	300
FIGURE D.8 FLOW OF RULE BASE EVALUATIONS .....	304
FIGURE D.9 EXECUTIVE RULE BASE: CLASS DEFINITIONS.....	307
FIGURE D.10 EXECUTIVE RULE BASE: DEMONS AND RULES .....	308
FIGURE D.11 PLANNING RULE BASE .....	310
FIGURE D.12 OPERATIONS CLASS DEFINITION.....	311
FIGURE D.13 OPERATIONS DEMON.....	312
FIGURE D.14 OPERATIONS RULES .....	313
FIGURE D.15 SCHEDULER RULE BASE.....	314
FIGURE D.16 AUDITOR RULE BASE.....	315

# TABLES

TABLE 2.1 ADAPTIVE STRUCTURES, OPERATORS, AND PERFORMANCE MEASURES .....	21
TABLE 3.1 VSM COMPONENTS AND PRIMARY RELATIONSHIPS .....	76
TABLE 6.1 RULES FOR FUZZY CONTROLLER.....	151
TABLE 7.1 SELECTED MESSAGE TYPES AND DESCRIPTIONS.....	173
TABLE 7.2 SMARTCAM’S POLICY (SET OF RULES) .....	173
TABLE 7.3 PLAN SHOWING ADDITIONS AND UPDATES OF ACTIVITIES.....	175
TABLE 9.1 TAXONOMY OF SYSTEM COMPLEXITY.....	214
TABLE A.1 GOF PATTERN CORRELATED WITH LST SUBSYSTEMS .....	226
TABLE C.1 MESSAGE TYPES AND DESCRIPTIONS .....	263
TABLE C.2 INITIAL MESSAGE SEQUENCE (PART 1).....	288
TABLE C.3 DESCRIPTION (PART 1) .....	289
TABLE C.4 INITIAL MESSAGE SEQUENCE (PART 2).....	290
TABLE C.5 DESCRIPTION (PART 2) .....	291
TABLE D.1 CONTRACT DETAILS BASED ON QoS.....	302



# Chapter 1

## Dealing with Complexity

We do not set out to build complex systems: we grow them by accident and of necessity. Most software systems begin life with the minimum functionality needed to meet their requirements - or they should. However, software systems are embedded in complex environments, and most importantly, environments that include humans. The introduction of new software systems, modification of existing ones, and changes in the environment itself creates reciprocal adaptations among interacting systems. These evolutionary and coevolutionary<sup>1</sup> forces drive simple systems into becoming ever more complex. This is the general setting for almost all “non-trivial,” “non-traditional” or advanced software systems we seek to build today.

Software systems as described above include business-to-business (B2B) e-commerce, smart (e.g. ambient, ubiquitous, pervasive) environments [Weiser 1991], military command and control systems, autonomous robots, multi-agent systems, adaptive user interfaces and the Internet itself; in which most of the systems mentioned are embedded. To illustrate, Tennenhouse [Tennenhouse 2000] proposes “Proactive Computing” to address the trends he sees developing wherein the number of networked, embedded, sensors, actuators and real-time computers vastly outnumber the human users. His view of proactive computing calls for getting humans out of the loop and placing them *above the loop* in supervisory and policy setting roles. He thinks this proliferation of diverse systems calls for the “urgent need to lay the intellectual groundwork for their principled

---

<sup>1</sup> Evolution in two or more evolutionary entities brought about by reciprocal selective effects between the entities. Ehrlich, P. and P. Raven (1964). "Butterflies and plants: a study in coevolution." Evolution **18**: 586-608.

design and analysis.” The evolution of the relationship of human control to computers necessitating this approach is shown in Figure 1.1. Tennenhouse’s Proactive Computing is one proposed solution to the general class of problems that motivates the research described here.

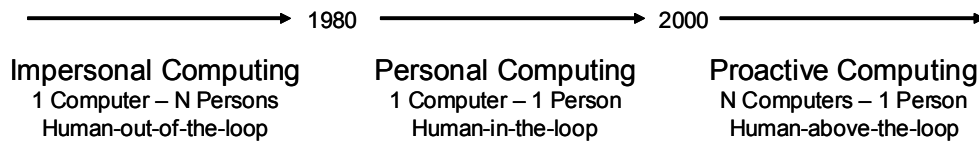


Figure 1.1 Relation of Human Control and Computing

Therefore, the motivation for this work is to address the following question: What “principled design and analysis” approaches can lead to software systems that may be coevolved with other systems in a complex environment? Further, as the role of humans in relation to software systems is all-important, any solution must make explicit the nature of their involvement as both users and developers.

The Intelligent Control Paradigm of software as described in this thesis is one answer to the questions and problems posed above. The Intelligent Control Paradigm is an extension of Shaw’s Control Paradigm [Shaw 1995] with the goal of achieving self-controlling software systems. The Viable System Model (VSM) [Beer 1984] was chosen as a starting point in specifying an instance of the Intelligent Control Paradigm.

The Viable System Model is a model of intelligent control. Accordingly, a main result of this research is the proposition that the concept of *viability* captures the chief quality desired, in fact the necessary and sufficient quality, for a next generation of software systems. A *viable system* has the capability, or potential, to successfully deal with the complexity of its environment: to survive, to maintain its identity. **A *viable software system* is one that proves to be adaptable over time, by humans (or other agents) to become an adaptive software system.** To a large degree, the viability of a software system is predicated on its architecture: architecture constrains the strategies the system may employ in dealing with the complexity faced in the environment. What constitutes this quality of viability, why it is essential, and how it can be realized in software is the subject of this thesis.

## 1.1 Thesis

No algorithm has been found to generate a new type of living organism, a functional yet aesthetically pleasing building, an effective human organisation, or the architecture for a complex software application. Systems of this class, or the knowledge required to build them, are generated by trial and error learning over time. John Holland calls the successful systems that emerge in his Complex Adaptive System<sup>2</sup> (CAS) simulation environment *Echo* “persistent patterns [Holland 1995].” Stewart Brand, in summarizing his photo-essay *How Buildings Learn* [Brand 1997], says the book is an assemblage of “steps toward an adaptive architecture.” He stresses that adaptive buildings are based on proven designs, are cautious about innovation, conservative, conventional, and even ordinary: “Honouring the future begins with honouring the past.” Brand relies heavily on the work of the architect Christopher Alexander. Alexander’s work in developing a pattern language for designing and building regions, cities, towns and homes is based on rediscovering and documenting the successful “unconscious” designs that have been evolved by humans since prehistory [Alexander, Ishikawa et al. 1977].

The researchers mentioned above have one thing in common: they are holistic or “systems thinkers” searching for general principles of adaptive systems. In that regard, the work on this thesis began by casting a wide net, one that fell outside of computer science *per se* in search of general principles, conceptual models, formal, and informal systems that could shape the basis for the design of a next generation of organic software systems. The net closed in around the past and present work related to what are now called “complex adaptive systems.” After reviewing the work in this area, the *Viable System Model* of Stafford Beer was selected as a starting point in developing the approach to software architecture described here.

Beer developed the Viable System Model (VSM) to better understand and improve the efficiency of human organization. He based the model on insights from neurophysiology,

---

<sup>2</sup> Complex Adaptive System is a term introduced by Holland to describe a large collection of interacting adaptive components as typified by his Genetic Algorithms and Echo Simulation Holland, J. H. (1999). Emergence: From Chaos to Order, Helix Books.

Holland, J. M. (1992). Adaptation in Natural and Artificial Systems, Bradford Books.

Holland, J. M. (1995). Hidden Order: How Adaptation Builds Complexity, Helix Books..

## *Chapter 1 - Dealing With Complexity*

cybernetics, and control theory [Beer 1984]. Beer did not invent a new model. He discovered and documented the persistent pattern of living organizations: natural and human engineered. His long-term goal has been the development of a discipline of “management cybernetics” with the VSM as its methodology [Beer 1956; Beer 1979; Beer 1981; Beer 1985]. This thesis applies the VSM to the problem of software architecture. In this approach, the VSM is the conceptual model from which high-level architectural models are derived. Based on these high-level models, component object models and component frameworks can be designed for specific application domains.

The goal in selecting the VSM as a basis for software architectures is to achieve -- in software -- the qualities inherent in the living systems it describes. Beer carefully chose the word *viability* to capture and convey the essential quality of living systems. Now, real systems are embedded in many other, larger systems. For example, a software system is embedded in an organization, a market economy, a user community, a legal system and so on. A software system must be viable, capable of survival, in all of these dimensions. This thesis is concerned with the technical aspects of software: software architecture, design, and implementation. The result of this work is the following definition of *software viability*:

Software Viability is the quality a software system has if it can be *adapted* over time by humans (design-time adaptable) toward becoming an *adaptive* (intelligent supervisory-adaptive-control) system (run-time adaptive).

This work has been an experiment; and one statement of the thesis presented here is to say it is an experiment in the application of a high-level, conceptual model for software systems architecture in order to demonstrate that this definition of viability can be beneficial in software. The overall result of this work is the Intelligent Control Paradigm for adaptable and adaptive software architectures as expressed in the form of the Viable Software Approach.

## **1.2 Thesis Organization**

The thesis is organized as follows. The next chapter, Chapter 2, provides background and related work. Chapter 2 brings together in one place the major concepts from cybernetics, systems thinking, complexity, control theory, software architecture, and software engineering on which the thesis rests. The primary goal of Chapter 2 is to lay

the foundation for the presentation of the Viable System Model and the Viable Software Approach. In that regard, the important work of Ashby is described in detail as it is the basis for Beer's Viable System Model. Chapter 2 situates the approach in relation to other software research. Then, Beer's model of the "viable" organisation, The Viable System Model, is presented in Chapter 3. It is a necessary to have the VSM presented here, as it is central to this thesis. (Appendix B gives an update on recent applications of the VSM based on a literature search conducted as part of this thesis.) Chapter 4 casts the Viable System Model in the form of a pattern language following Alexander. One goal of the pattern language is to make the model more accessible to the practising programmer as patterns are now a standard technique in software engineering [Gamma, 1995]. Chapter 5 is the core contribution of the thesis. In this chapter the Viable Software Approach (VSA) is developed. The central concept of software viability is defined and made specific in the form of a Product Line Architecture approach to developing complex adaptive software systems. The Product Line approach is itself a framework in which architecture, software frameworks, and related constructs and artefacts are arranged to enable a process for the production of domain-specific applications. The architecture, framework, and component specifications embody the structure of the VSM and capture the patterns of the viable system. A key feature of the VSA, and inherent in the definition of software viability, is piecemeal "migration" of human control skills into software systems. An introductory example is included in Chapter 5 to make these abstract concepts more concrete. The example is an analysis of the Model-View-Controller design pattern from the viewpoint of the Viable Software Approach.

Three domains-specific case studies are developed to demonstrate the VSA approach. Chapter 6 shows how the VSA is used for analysis of existing systems. The *Groove* Internet-based, peer-to-peer, collaboration system was chosen because its architecture was well documented, and because it represents a large class of interactive systems. First, Groove's architecture is mapped into a Viable Software Architecture. The restated system architecture clearly shows where adaptive controllers are to be added. Fuzzy controllers are developed at several levels within the architecture to provide adaptive features. Next, in Chapter 7, the emerging area of "Smart Environments" is explored and software for a smart camera controller is described. A scenario is developed showing

how the diverse hardware and software systems that must work together to create such an environment might be organised to support a university campus and a “smart lecture room” in particular. The VSA is applied to develop an architecture for the smart lecture room. One component, “a smart camera” is singled out for detailed design and software implementation. The smart camera software system is documented in Appendix C . A third case study focusing on e-Commerce is described in Chapter 8. There, a B2B scenario outlines how the VSA supports adaptable system growth. The area of “electronic contracts” for B2B is explored in depth and a prototype implementation documented in Appendix D Finally, Chapter 9 gives a summary of the work, identifies contributions, and discusses possible future work

### **1.3 Summary of Contributions**

The Intelligent Control Paradigm of software architecture is the main contribution of this work. One instance of this paradigm is presented in the form of the Viable Software Approach. The Viable Software Approach contains a number of contributions. Firstly, the identification of technical software viability as stated above. This definition of technical software viability brings together the key ideas of the approach. Namely, that software systems need to be both adaptable and adaptive. An adaptable software system is one that can be grown or evolved by humans over time without meeting some fundamental problem that leads to a “dead-end.” Next, there is a direction or gradient to the growth process; a viable software system gains adaptive capabilities over time, piecemeal, as appropriate. The idea of viability as the chief quality of software represents an advancement over previous notions of “goodness” as expressed by the low-level “ilities” of software engineering

A novel Product Line Architecture (PLA) based on a suite of meta-constructs is described. In this PLA, the meta-model of the VSM captures the essential characteristics of complex adaptive systems. Next, the Viable Software Architecture, expressed as a role-based meta-architecture, provides a template for domain-specific architectures. A viable component framework is specified based on the unique set of interfaces identified by the architecture. A major feature of the component interface definition is that it supports recursive system composition. An abstract Component Transfer Protocol is provided as guidance in the development of domain-specific assembly protocols.

The case studies represent a contribution in their own right. Briefly, the Groove study shows how to map an existing architecture into another framework and extend it using fuzzy control techniques. The smart environment study provides a domain-specific architecture for that important class of application; and the smart camera prototype gives a concrete instance of a software design based on the VSA architecture. Finally, the B2B case study shows how to evolve a complex adaptive B2B system. The B2B contract prototype show how expert system approaches can be used within the VSA to implement advanced business functions.

A final claim, based partially on the experience of the case studies, is that the Viable Software Architecture represents a unification of many diverse architectural styles. The VSA can be described as a Multi-Agent System (MAS) architecture. It can also be described as a coordination framework. The VSA could well be the basis for an operating system. Blackboard architectures, hierarchical cellular communication architectures, and even subsumption architectures fall within its purview. Further, the VSA shows how diverse tools and techniques, including many from AI, can be organised within the overall information systems context.



# Chapter 2

## Background and Related Work

The goal of this chapter is to place the thesis in context, and in that regard, it is necessarily two fold. Firstly, it must provide background for the ideas on which the Viable System Model (VSM) is based, as these will be necessary for the presentation of the model in the next chapter and in the remainder of this thesis. Secondly, this chapter shows how the work is related to the current state of developments in software architecture and software engineering. As an aid in accomplishing this objective, a map of the conceptual terrain covered in the course of this research is shown in Figure 2.1. The subject of this thesis, Viable Software Systems, is at the centre of a universe of ideas divided into six regions and corresponding to the following disciplines: Biology, Control Theory, Cybernetics, Complexity, Software Architecture, and Software Engineering. The text labels in the figure represent key disciplines, theories, and models that relate to the development of this thesis. There are two types of labels in the figure: one refers to a specific development (e.g. Cybernetics) with the primary author's name in parenthesis underneath (e.g. Wiener); the other refers to a general area of work, or sub-discipline, (e.g. Fuzzy Control) and is italicised. The four arrows indicate the cross currents of ideas flowing between the disciplines that contribute to the development of theories and models as indicated. The two horizontal arrows show the direct evolution of ideas that form the basis for this thesis: Cybernetics leading up to the development of the VSM; and the path of research in software architecture starting with the "Control Paradigm" leading to the central result of the current thesis – the Intelligent Control Paradigm. The arrows on the sides show other related work contributing to the development of the Viable Software approach.