

Copyright is owned by the Author of the thesis. Permission is given for a copy to be downloaded by an individual for the purpose of research and private study only. The thesis may not be reproduced elsewhere without the permission of the Author.

A Modelling Language for Rich Internet Applications

A thesis presented in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy
in
Computer Science

at Massey University, Turitea,
New Zealand.

Jevon Michael Wright

2011

Copyright © 2011 by Jevon Wright

Copyright is owned by the author of this thesis. Permission is given for a copy to be downloaded by an individual for the purpose of research and private study only. This thesis may not be reproduced elsewhere without the permission of the author.

Abstract

This thesis presents the Internet Application Modelling Language (IAML), a modelling language to support the model-driven development of Rich Internet Applications (RIAs). This definition includes a visual syntax to support the graphical development of IAML model instances, and the underlying metamodel satisfies the metamodelling and viewpoint architectures of the Model Driven Architecture.

While there are many existing modelling languages for web applications, none of these languages were found to be expressive enough to describe fundamental RIA concepts such as client-side events and user interaction. This thesis therefore presents IAML as a new language that reuses existing standards where appropriate. IAML is supported by a proof-of-concept CASE tool within the Eclipse framework, and released under an open source license to encourage industry use. This reference implementation successfully integrates a number of different model-driven technologies to demonstrate the expressiveness of the modelling language.

The IAML metamodel supports many features not found in other web application modelling languages, such as Event-Condition-Action rules; the expression of reusable patterns through *Wires*; and a metamodel core based on first-order logic. Through the implementation of the RIA benchmarking application *Ticket 2.0*, the concepts behind the design of IAML have been shown to simplify the development of real-world RIAs when compared to conventional web application frameworks.

Acknowledgments

This thesis would not have been possible without the support of many important people.

In particular, to my supervisors Jens Dietrich and Giovanni Moretti – thank you for all of your advice, encouragement and inspiration throughout my undergrad degree and my doctoral studies. I am gracious for the patience and support that you have both given me.

To my office mates at Massey that have helped me throughout these years – Graham Jenson, Fahim Abbasi, Yuliya Bozhko and others – thank you for your feedback, ideas and your constructive criticisms; and the innumerable breaks for caffeine, whether in coffee, chocolate or cola form.

To the support staff at Massey – including Patrick Rynhart, Dilantha Punchihewa, and Michele Wagner – thank you for helping me out with the stress and effort necessary to deal with the administrative side of research.

To all my bandmates – Aaron Badman, Aaron Shirriffs, Ian Luxmoore, Josh Williamson, Matt Carkeek, Steve Starr, Wayne Bryant – thank you for supporting me throughout my busy schedules, and for giving me a noisy outlet to express my creativity far away from the thesis.

To all of my family, especially my mum and dad – thank you for your never-ending and reliable support and patience throughout all of my decisions, and encouraging my independence.

And finally to my best friend and fiancé, Krystle Chester, you have helped me and supported me through more than I could have imagined, and completing this thesis would not have been possible without your love, patience and mews. <3

Contents

Abstract	i
Acknowledgments	iii
Contents	v
List of Tables	vii
List of Figures	ix
List of Listings	xi
1 Introduction	1
1.1 Web Applications	1
1.2 Modelling	3
1.3 Research Questions	3
1.4 Research Design	4
1.5 Research Method	6
1.6 Thesis Outline	7
2 Background	9
2.1 Rich Internet Applications	9
2.1.1 Classifications of Rich Internet Applications	11
2.1.2 Interactivity	11
2.1.3 Enterprise Software	12
2.1.4 Security Risks of Rich Internet Applications	12
2.2 Frameworks	13
2.3 Features of Rich Internet Applications	14
2.3.1 Feature Categories	14
2.3.2 Detailed Modelling Requirements	17
2.4 Existing Web Application Modelling Languages	20
2.4.1 WebML	20
2.4.2 UML	21
2.4.3 UWE	22
2.4.4 WebDSL	23
2.4.5 Web Information Systems	24
2.4.6 Older Languages	24

2.4.7	Evaluation	24
2.5	Benchmarking Application	24
2.6	Metrics	26
2.6.1	Web Application Metrics	26
2.6.2	Metamodelling Metrics	28
2.6.3	System Metrics	29
2.7	Software Engineering Principles within the Development of Modelling Languages	30
2.7.1	Modelling Language Design Approaches	30
2.7.2	Software Process Models	31
2.7.3	Test Driven Development	33
2.7.4	Open Source	33
2.8	Conclusion	33
3	Modelling	35
3.1	Modelling	35
3.1.1	Metamodels	36
3.1.2	Domain-Specific Languages	36
3.1.3	Model-Driven Development	38
3.1.4	Model-Driven Engineering	38
3.1.5	Model Driven Architecture	39
3.1.6	Model Spaces	43
3.1.7	Model Transformations	44
3.1.8	Query/Views/Transformations Language	46
3.1.9	The Syntax and Semantics of Metamodels	48
3.1.10	A Formal Definition of Metamodels	49
3.1.11	The Bigger Picture of Modelling Approaches	50
3.1.12	Metamodelling Environments	51
3.2	Model Completion	51
3.2.1	Documented Conventions	52
3.2.2	Model Completion Concepts	52
3.2.3	Model Completion Semantics	53
3.3	Model Instance Verification	54
3.3.1	Validation vs. Verification	55
3.3.2	A Formal Definitions of Constraints	55
3.3.3	Discussion	57
3.4	Visual Modelling	58
3.4.1	Visual Metaphors	58
3.4.2	Visual Metaphors in Existing Models	59
3.4.3	Visual Modelling Software	59
3.4.4	Visual Notation	59
3.5	Conclusion	61

4	Rich Internet Application Modelling Concepts	63
4.1	Managing Complexity through Structural Decomposition	63
4.1.1	Modularity	63
4.1.2	Hierarchical Modelling	64
4.1.3	Aspects	65
4.1.4	Discussion	66
4.2	Events	66
4.2.1	UML	66
4.2.2	AORML	67
4.2.3	URML	67
4.2.4	Kaitiaki	68
4.2.5	Event Algebra	68
4.2.6	DOM Events	69
4.3	Operation Modelling	70
4.3.1	UML Activity Diagrams	70
4.3.2	Business Process Modelling Notation	71
4.3.3	Predicate Modelling	71
4.4	Type Systems	72
4.4.1	Primitive Types	72
4.4.2	Derived Types	73
4.4.3	Type Systems in the Model Driven Architecture	74
4.5	Domain Modelling	75
4.5.1	ER Diagrams	75
4.5.2	UML Class Diagrams	76
4.5.3	XML Schema	76
4.5.4	EMF Ecore	77
4.6	Design Patterns for Data Access	77
4.6.1	Database Broker	77
4.6.2	Iterator	77
4.7	Lifecycle Modelling	79
4.7.1	Implementation-Level Examples	79
4.8	Users and Access Control	80
4.8.1	Access Control Lists	80
4.8.2	Discretionary Access Control	82
4.8.3	Mandatory Access Control	82
4.8.4	Role-based Access Control	83
4.9	Describing Reusable Patterns within a Metamodel	83
4.9.1	Independence of the Reusable Pattern Metamodel	83
4.9.2	Implementation of the Reusable Pattern Metamodel	83
4.10	Conclusion	84
5	The Internet Application Modelling Language	85
5.1	Applying Software Engineering Principles	85
5.1.1	Requirements Planning	85
5.1.2	Modelling Language Development Approach	86

5.1.3	Software Process Model	87
5.1.4	Evaluation	88
5.1.5	Design Goals	88
5.2	Metamodel Design Principles	89
5.2.1	Addressing Security Risks through Modelling Language Design	89
5.2.2	Language Design Principles	89
5.2.3	Hierarchical Modelling Approach	91
5.2.4	Guidelines for Metamodel Refactoring	92
5.3	Language Overview	93
5.3.1	Package Overview	95
5.4	Metamodel Core	96
5.4.1	Logic Model	97
5.4.2	Function Model	100
5.4.3	Event-Condition-Action Model	101
5.4.4	An example decomposition of a Simple Condition	103
5.4.5	Wires Model	105
5.4.6	Constructs Model	105
5.5	Type System	106
5.5.1	Primitive Types	107
5.5.2	Additional Built-in Primitive Types	109
5.5.3	Value Instances	110
5.5.4	Client-side Input Validation	111
5.6	Domain Modelling	112
5.6.1	Domain Types	112
5.6.2	Design Decisions on Modelling Domain Types	114
5.6.3	Domain Iterators	116
5.6.4	Domain Instances	119
5.6.5	Design Decisions on Modelling Domain Instances	120
5.7	Events	122
5.7.1	onChange	122
5.7.2	onInput	122
5.7.3	onAccess	123
5.7.4	onInit	123
5.7.5	onClick	123
5.7.6	onSent	123
5.7.7	onFailure	124
5.7.8	onIterate	124
5.7.9	Client-side Events	124
5.8	Operations	125
5.8.1	Modelling Complex Behaviour	125
5.8.2	Activity Operations	128
5.8.3	Activity Predicates	130
5.8.4	Replacing the Default Behaviour of Model Elements	131
5.9	Wires	131

5.9.1	Sync Wire	132
5.9.2	Set Wire	133
5.9.3	Detail Wire	133
5.9.4	Autocomplete Wire	134
5.10	Users and Access Control	136
5.10.1	Roles and Permissions	136
5.10.2	Login Handlers	137
5.10.3	Access Control Handlers	138
5.10.4	OpenID	139
5.11	Scopes	140
5.11.1	Internet Application	140
5.11.2	Session	141
5.11.3	Failure Handlers	141
5.11.4	Gates	142
5.11.5	Query Parameter	143
5.11.6	Discussion	143
5.12	Messaging	143
5.12.1	Email	144
5.12.2	Additional Messaging Types	144
5.13	User Interface Modelling	145
5.13.1	Existing Approaches in Modelling User Interface Types	145
5.13.2	Visible Thing	146
5.13.3	Sample Representations	150
5.13.4	Frame	150
5.14	Internet Application	150
5.15	Visual Modelling Metaphors	151
5.15.1	Overview Layer	151
5.15.2	Navigation Layer	152
5.15.3	User Interface Layer	152
5.15.4	Domain Modelling Layer	152
5.15.5	Operation Modelling Layer	152
5.16	Unification of Type Instantiation and Classification	153
5.16.1	Guidelines for Applying Instance Unification in Metamodels	155
5.16.2	Applying Instance Unification to the IAML Metamodel	156
5.17	Conclusion	157
6	Implementation Technologies	159
6.1	Common Comparison Criteria	159
6.1.1	Execution Environment Comparison Properties	159
6.1.2	Open Source Comparison Criteria	160
6.2	Metamodelling Environments	161
6.2.1	Eclipse Modeling Framework	162
6.2.2	ArgoUML	163
6.2.3	Whitehorse	164
6.2.4	Marama	166

6.2.5	Summary	167
6.3	Graphical Modelling	167
6.3.1	Graphical Editor Framework	169
6.3.2	Graphical Modeling Framework	169
6.3.3	ArgoUML	170
6.3.4	Whitehorse	170
6.3.5	Marama	171
6.3.6	Summary	172
6.4	Code Generation	173
6.4.1	XSLT	173
6.4.2	Java Emitter Templates	174
6.4.3	openArchitectureWare	175
6.4.4	Xpand	175
6.4.5	Summary	176
6.5	Model Completion	177
6.5.1	Jena	178
6.5.2	Jess	179
6.5.3	Drools	180
6.5.4	Summary	180
6.6	Model Instance Verification	181
6.6.1	Drools	182
6.6.2	Jena	184
6.6.3	OWL 2 Full	185
6.6.4	CrocoPat	185
6.6.5	Alloy	187
6.6.6	openArchitectureWare	188
6.6.7	OCL	189
6.6.8	NuSMV	190
6.6.9	Summary	191
6.7	Conclusion	192
7	Proof-of-Concept Implementation	195
7.1	Introduction	195
7.1.1	Implementation License	195
7.2	<i>IAML Metamodel</i>	197
7.3	<i>IAML Model</i>	197
7.3.1	<i>Model Instance</i>	197
7.3.2	<i>Model Migration</i>	198
7.4	<i>Graphical Editor</i>	198
7.4.1	<i>Model Edit</i>	199
7.4.2	<i>Diagram Editors</i>	199
7.4.3	<i>Diagram Definitions</i>	201
7.4.4	<i>Diagram Extensions</i>	205
7.4.5	<i>Diagram Actions</i>	209
7.5	<i>Model Completion</i>	210

7.5.1	<i>Completion</i>	211
7.5.2	<i>Rule Set</i>	211
7.5.3	<i>Drools</i>	214
7.6	<i>Code Generation</i>	214
7.6.1	<i>openArchitectureWare</i>	214
7.6.2	<i>Templates</i>	215
7.6.3	<i>Runtime Library</i>	216
7.6.4	<i>Platform Configuration</i>	217
7.6.5	<i>Output Formatter</i>	217
7.6.6	<i>Generator</i>	218
7.7	<i>Model Verification</i>	219
7.7.1	Model Instance Verification with Checks	220
7.7.2	Model Instance Verification with OCL in the EMF Validation Framework	221
7.7.3	Model Instance Verification with CrocoPat	221
7.7.4	Model Instance Verification with NuSMV	222
7.8	<i>Model Actions</i>	222
7.9	Tests	223
7.9.1	Model-driven Code Coverage	224
7.10	Conclusion	225
8	Evaluation	227
8.1	Feature Comparison	227
8.2	Modelling Requirements	229
8.3	Benchmarking Application Implementation	231
8.3.1	Implementation in Symphony	232
8.3.2	Implementation in IAML	233
8.3.3	Re-implementation in Symphony	252
8.3.4	System Metrics Evaluation	254
8.4	Metamodelling Metrics	255
8.5	Visual Model Evaluation	257
8.5.1	Notation Information Capacity	258
8.5.2	Cognitive Dimensions Evaluation	259
8.6	Conclusion	259
9	Conclusions and Future Research	261
9.1	Research Contributions and Conclusions	261
9.1.1	Requirements for Modelling Rich Internet Applications	261
9.1.2	A Benchmarking Application for Rich Internet Applications	261
9.1.3	A Modelling Language for Rich Internet Applications	262
9.1.4	Model Completion	263
9.1.5	Model Instance Verification	264
9.1.6	Evaluation of Model-Driven Technologies	264
9.1.7	Other Contributions	265
9.2	Answers to Research Questions	265
9.3	Future Research	267

9.3.1	Further Evaluation of Existing Web Modelling Languages	267
9.3.2	Modelling Full RIAs	268
9.3.3	Improved Graphical Editor View Mappings	268
9.3.4	Integrating Textual Expression Languages in the Visual Editor	269
9.3.5	Extraction of Reusable Components	269
9.3.6	Code Generation Templates for Additional Platforms	270
9.3.7	Incremental Transformations	270
9.3.8	Extensibility of the Proof-of-Concept Implementation of IAML	270
9.3.9	User Evaluations on the Proof-of-Concept Implementation of IAML	271
9.3.10	Identifying Metamodel Refactoring Patterns	271
9.4	Summary	273
A	Use Cases	275
A.1	Actors	275
A.2	List of Use Cases	276
UC-01	View Data	277
UC-02	Update Data	277
UC-03	Pagination	278
UC-04	User Action Auditing	278
UC-05	Debug Mode	279
UC-06	Server Transaction Support	279
UC-07	Local Data Storage	280
UC-08	Server Data Access	281
UC-09	Persistent Client Data	281
UC-10	Temporary Server Data	282
UC-11	Uploading Files	283
UC-12	Restore Server Session	284
UC-13	User Authorisation	285
UC-14	Password Reset	286
UC-15	Session Support	286
UC-16	Account Registration	287
UC-17	Automatic User Authorisation	288
UC-18	Static Views (HTML)	289
UC-19	Asynchronous Form Validation	290
UC-20	Client Form Validation	291
UC-21	Server Form Validation	292
UC-22	Multiple Browser Support	293
UC-23	Mobile Phone Support	293
UC-24	Remote Data Source	294
UC-25	Active Remote Data Source	294
UC-26	Data Feeds	295
UC-27	Web Service	295
UC-28	Back/Forwards Button Control	296
UC-29	Opening New Windows	296
UC-30	Client-Side Application	297

UC-31	Communication with Software	298
UC-32	Mobile Phone Communication	299
UC-33	E-mailing Users	300
UC-34	E-mail Unsubscription	301
UC-35	Persistent Errors	302
UC-36	User Content Access Control	302
UC-37	Private User Content Access Control	303
UC-38	User Collaboration	304
UC-39	Interactive Map	305
UC-40	Drag and Drop	305
UC-41	Client Timer Support	306
UC-42	Server Timer Support	307
UC-43	Page Caching	307
UC-44	Offline Application Support	308
UC-45	Loading Time Support	308
UC-46	Flash MP3 Support	309
UC-47	Flash Communication Support	310
UC-48	Internationalisation Support	310
UC-49	Logout Control	311
UC-50	Single Sign-In Solutions	311
UC-51	User Redirection	312
UC-52	Keyboard Shortcuts	313
UC-53	Undo/Redo Support	313
UC-54	Browser-Based Chat	314
UC-55	Pop-up Window Support	315
UC-56	Incompatible Client Warning	315
UC-57	Dynamic Objects	316
UC-58	Store Data in Local Database	317
UC-59	Store Resources Locally	318
UC-60	Multiple Client Threads	319
UC-61	Multiple Server Threads	320
UC-62	Communication with Plugins	321
UC-63	Scheduled Events	321
UC-64	Custom API Publishing	322
UC-65	Runtime Interface Updates	322
UC-66	Out-of-Order Events	323
UC-67	Backwards-Compatible Scripting	323
UC-68	Spellchecking	324
UC-69	Autocomplete	324
B	Documentation By Example: Sync Wire	325
C	OpenBRR Evaluations of Model-driven Technologies	331
D	Model Checking using NuSMV: Implementing the <i>Infinitely Redirects</i> Constraint	335

E	Description of the Attached Media	339
E.1	Media Contents	339
E.2	IAML Tutorial	340
E.2.1	Creating a New Model Instance	340
E.2.2	Editing the Model Instance	340
E.2.3	Generating the Application	343
E.2.4	Configure the Web Server	343
E.2.5	Access the Generated Application	343
F	XMI Representation of Ticketiaml	347
G	Language-specific Metrics of the Ticket 2.0 Implementations	357
H	Cognitive Dimensions Evaluation of IAML	361
I	Model Element Reference	367
I.1	Access Control Handler	368
I.2	Accessible	369
I.3	Action	369
I.4	Action Edge Source	370
I.5	Activity Node	370
I.6	Activity Operation	371
I.7	Activity Parameter	373
I.8	Activity Predicate	374
I.9	Arithmetic	375
I.10	Autocomplete Wire	376
I.11	Boolean Property	377
I.12	Builtin Operation	377
I.13	Builtin Property	379
I.14	Button	380
I.15	Can Be Synced	381
I.16	Cancel Node	382
I.17	Cast Node	383
I.18	Changeable	383
I.19	Complex Term	384
I.20	Condition Edge Destination	385
I.21	Constraint Edge	385
I.22	Constraint Edge Destination	386
I.23	Constraint Edges Source	386
I.24	Contains Functions	386
I.25	Contains Operations	387
I.26	Contains Values	387
I.27	Contains Wires	388
I.28	Data Flow Edge	389
I.29	Data Flow Edge Destination	389
I.30	Data Flow Edges Source	390

I.31	Decision Node	390
I.32	Detail Wire	391
I.33	Domain Attribute	392
I.34	Domain Attribute Instance	392
I.35	Domain Feature	393
I.36	Domain Feature Instance	394
I.37	Domain Instance	394
I.38	Domain Iterator	395
I.39	Domain Source	400
I.40	Domain Type	401
I.41	ECA Rule	402
I.42	EXSD Data Type	403
I.43	Email	404
I.44	Event	405
I.45	Execution Edge	407
I.46	Execution Edge Destination	407
I.47	Execution Edges Source	408
I.48	Extends Edge	408
I.49	Extends Edge Destination	409
I.50	Extends Edges Source	409
I.51	External Value	410
I.52	Finish Node	410
I.53	Frame	411
I.54	Function	412
I.55	Gate	413
I.56	Generated Element	414
I.57	Generates Elements	415
I.58	Input Form	416
I.59	Input Text Field	417
I.60	Internet Application	418
I.61	Iterator List	419
I.62	Join Node	419
I.63	Label	420
I.64	Login Handler	421
I.65	Map	425
I.66	Map Point	425
I.67	Message	426
I.68	Named Element	426
I.69	Operation	427
I.70	Operation Call Node	427
I.71	Parameter	427
I.72	Parameter Edge Destination	430
I.73	Parameter Edges Source	430
I.74	Parameter Value	431

I.75	Permission	431
I.76	Predicate	431
I.77	Provides Edge	432
I.78	Provides Edge Destination	432
I.79	Provides Edges Source	433
I.80	Query Parameter	433
I.81	Requires Edge	433
I.82	Requires Edge Destination	434
I.83	Requires Edges Source	435
I.84	Role	435
I.85	Schema Edge	436
I.86	Scope	436
I.87	Select Edge	438
I.88	Session	439
I.89	Set Node	440
I.90	Set Wire	440
I.91	Simple Condition	444
I.92	Split Node	445
I.93	Start Node	446
I.94	Sync Wire	446
I.95	Temporary Variable	449
I.96	Value	449
I.97	Visible Thing	451
I.98	Wire	452
I.99	Wire Destination	452
I.100	Wire Source	453
I.101	Wireable	453
I.102	XQuery Function	454
I.103	XQuery Predicate	454

Bibliography	455
---------------------	------------

List of Tables

1.1	A brief history of Web application technologies	2
2.1	Fundamental modelling requirements of Rich Internet Applications (1)	18
2.2	Fundamental modelling requirements of Rich Internet Applications (2)	19
2.3	Existing modelling language support for the general feature categories of modelling Rich Internet Applications	25
2.4	Selected metamodelling metrics for metamodels within the Eclipse Modeling Framework	28
2.5	Overall system metrics	29
2.6	Language system metrics	29
3.1	A list of popular metamodels and their meta-metamodels	40
3.2	The relationships between different modelling approaches	51
3.3	Information encoding capacity of different visual variables	60
4.1	Selected event algebra operators for composing complex events	69
4.2	JUnit lifecycle methods and annotations	79
4.3	Possible lifecycle layers of Rich Internet Applications and their potential use	81
4.4	Example of an Access Matrix	82
5.1	Additional Requirements for <i>Full RIAs</i>	86
5.2	Associations between first-order logic elements and IAML elements	98
5.3	A selection of permitted, conditional and prohibited datatype casts in IAML	109
5.4	Sample visual representations of interface modelling elements in IAML	149
5.5	Visual Metaphors for a Rich Internet Application modelling language	152
6.1	Requirement comparisons between existing metamodelling environments	167
6.2	Requirement comparisons between existing graphical modelling environments	172
6.3	Requirement comparisons between existing code generation environments	176
6.4	Requirement comparisons between existing model completion rule engines	181
6.5	Requirement comparisons between existing model verification environments (1)	192
6.6	Requirement comparisons between existing model verification environments (2)	193
6.7	A summary of the suitability of verification languages for addressing each verification approach category	193
7.1	Generated IAML diagram editors and their associated OSGi bundle IDs	201
7.2	Shape styles for the visual representation of IAML model elements	203

7.3	Background colours for the visual representation of IAML model elements	204
7.4	Summary of the rules used for model completion on IAML model instances	213
7.5	The set of platform-specific configuration properties used by the IAML <i>Code Generator</i> component	218
8.1	Reuse of existing metamodels in the IAML metamodel	228
8.2	A comparison of IAML against existing modelling language support for the general feature categories of modelling Rich Internet Applications	229
8.3	Evaluation of the modelling requirements of <i>Basic RIAs</i> against IAML (1)	230
8.4	Evaluation of the modelling requirements of <i>Basic RIAs</i> against IAML (2)	231
8.5	Legend to the modelling requirements evaluation of IAML	231
8.6	Comparing two Ticket 2.0 implementations using system metrics: overall development effort	254
8.7	Comparing two Ticket 2.0 implementations using system metrics: manual effort	254
8.8	Using metamodel metrics to evaluate the IAML metamodel against other similar metamodels implemented using the Eclipse Modeling Framework	256
8.9	Selected metamodeling metrics for metamodels within the Eclipse Modeling Framework	256
8.10	Evaluation of the information encoding capacity of IAML visual notation	259
C.1	Evaluations of model-driven technologies against OpenBRR ratings (1)	332
C.2	Evaluations of model-driven technologies against OpenBRR ratings (2)	333
C.3	Evaluations of model-driven technologies against OpenBRR ratings (3)	334
G.1	File-based metrics of the development of Ticketsf within Symfony 1.0.12	358
G.2	File-based metrics of the development of Ticketsf-mini within Symfony 1.4.8	359
G.3	File-based metrics of the development of Ticketiaml within IAML 0.6	360

List of Figures

1.1	The general methodology of design science research	5
2.1	A reasonable architecture for Rich Internet Applications	10
2.2	A sample WebML <i>hypertext model</i>	21
2.3	A sample UWE <i>navigation structure model</i>	22
2.4	The conceptual structure of the <i>Ticket 2.0</i> benchmarking application	27
3.1	The Metamodelling Architecture of MDA	39
3.2	The Viewpoint Architecture of MDA	41
3.3	The layers of UML under the MDA	42
3.4	Using both architectures of the MDA to describe the model-driven development of RIAs	44
3.5	Dealing with different Modelling Spaces	45
3.6	The Model Transformation Pattern in MDA	46
3.7	Using change models for round-trip engineering	47
3.8	The model completion process within model-driven development	52
3.9	Definition of the rule program <i>C</i> for the <i>default checkbox rule</i> convention	54
3.10	Model completion $C(A)$	54
4.1	Managing complexity through horizontal partitioning	64
4.2	Managing complexity through hierarchical modelling	64
4.3	Managing complexity through aspect-oriented modelling	65
4.4	Event-Condition-Actions in UML activity diagrams	67
4.5	An AORML Interaction Pattern diagram	67
4.6	Modelling Reaction Rules in URML	68
4.7	Visual Event Handler definitions in Kaitiaki	68
4.8	Operation modelling using UML activity diagrams	70
4.9	Operation modelling of an online shopping business process using BPMN	71
4.10	The built-in datatype hierarchy of XML Schema Datatypes	73
4.11	Representing XML Schema Datatypes within the metamodelling architecture of MDA using modelling spaces	74
4.12	A sample ER Diagram and Data Object Table	75
4.13	A sample UML class diagram	76
4.14	The Database Broker design pattern represented as a UML sequence diagram	78
4.15	The Iterator design pattern using parameterised types	78
4.16	Component lifecycle events in OSGi	81

5.1	The hybrid modelling language process model used in the development of IAML . . .	87
5.2	Hierarchical Modelling in IAML	91
5.3	An overview of some of the important concepts of modelling RIAs using IAML . . .	94
5.4	The generated implementation of the IAML overview model instance	94
5.5	Dependencies between packages of related model elements within IAML	96
5.6	Layered architecture diagram illustrating the layered design of packages within the IAML metamodel	97
5.7	A partial syntax of first-order logic	98
5.8	UML class diagram for the <i>Core</i> Package: Logic Model	99
5.9	UML class diagram for the <i>Core</i> Package: Function Model	100
5.10	UML class diagram for the <i>Core</i> Package: Event-Condition-Action Model	102
5.11	Ticketiaml: An <i>ECA Rule</i> connecting an <i>Event</i> to an <i>Operation</i>	103
5.12	The <i>Simple Condition</i> decomposition of a single predicate using variables and domain types as terms, represented in terms of UML syntax	104
5.13	The model developer view of the <i>Simple Condition</i> decomposition	104
5.14	VisualAge for Smalltalk: Using Connections	105
5.15	UML class diagram for the <i>Core</i> Package: Wires Model	105
5.16	UML class diagram for the <i>Core</i> Package: Constructs Model	106
5.17	Adapting the metamodelling architecture of MDA to the use of primitive type systems	107
5.18	UML class diagram for the <i>Domain</i> package of IAML	112
5.19	Ticketiaml: The defined <i>Domain Attributes</i> of the <i>Domain Type Event</i>	113
5.20	If domain types are defined using metamodelling, the resulting models are <i>incompatible</i> in terms of the MDA metamodelling architecture	115
5.21	Defining instances using references instead of instantiation is MDA-compatible . . .	116
5.22	UML class diagram for the <i>Domain Instances</i> package of IAML	118
5.23	Ticketiaml: Selecting an instance of the <i>Event Domain Type</i> through a <i>Domain Iterator</i> connected to a <i>Domain Source</i>	119
5.24	Ticketiaml: Creating a new instance of an <i>Event</i> using a <i>Domain Iterator</i>	121
5.25	Representing complex behaviour within IAML using two <i>ECA Rules</i>	126
5.26	UML class diagram for the <i>Activity</i> Package: Operations Model	128
5.27	The same complex behaviour of Figure 5.25 expressed with an <i>Activity Operation</i> .	129
5.28	UML class diagram for the <i>Activity</i> Package: Execution Model	130
5.29	UML class diagram for the <i>Activity</i> Package: Data Flow Model	131
5.30	UML class diagram for the <i>Activity</i> Package: Predicates Model	132
5.31	UML class diagram for the <i>Wires</i> package of IAML	132
5.32	Ticketiaml: Connecting a <i>Detail Wire</i> to a <i>Domain Iterator</i>	134
5.33	An instance of autocomplete in Gmail	135
5.34	Autocomplete implemented in IAML using an <i>Autocomplete Wire</i>	135
5.35	The user interface for an <i>Autocomplete Wire</i> , as generated by the proof-of-concept implementation of IAML	136
5.36	UML class diagram for the <i>Users</i> package of IAML	137
5.37	UML class diagram for the <i>Access Control</i> Package	138
5.38	Ticketiaml: Protecting access to a <i>Scope</i> via an <i>Login Handler</i> and <i>Access Control Handler</i>	139

5.39	UML class diagram for the <i>Scopes</i> package of IAML	141
5.40	Ticketiaml: Protecting access to a <i>Scope</i> via an entry <i>Gate</i>	142
5.41	UML class diagram for the <i>Messaging</i> package of IAML	144
5.42	Instantiation of user interface elements in WebML	145
5.43	Instantiation of user interface elements in UWE	146
5.44	UML class diagram for the <i>Visual</i> package of IAML	147
5.45	Ticketiaml: Designing the user interface for the <i>Browse Events</i> page using <i>Visible Things</i>	151
5.46	The consequences of unifying instance types within the metamodelling architecture of the MDA	153
5.47	A proposed refactoring of the IAML metamodel to unify the instantiation of <i>Domain Instances</i> and <i>Visible Things</i>	154
6.1	Using ArgoUML to define a UML class diagram	165
6.2	Defining a metamodel instance using Marama	166
6.3	The GMF Framework Wizard in Eclipse	170
6.4	The <i>Shape Designer</i> of Marama showing the shape design and corresponding concrete view	171
7.1	Overall UML component diagram of the Proof-of-concept Implementation of IAML	196
7.2	UML component diagram of the <i>IAML Model Component Decomposition</i>	197
7.3	UML component diagram of the <i>Graphical Editor Component Decomposition</i> . . .	199
7.4	Implementation of a graphical editor for IAML model instances using the Graphical Modeling Framework	200
7.5	Illustrating generated and overridden elements in the IAML editor	208
7.6	UML component diagram of the <i>Model Completion Component Decomposition</i> . . .	210
7.7	UML component diagram of the <i>Code Generation Component Decomposition</i> . . .	215
7.8	UML component diagram of the <i>Model Verification Component Decomposition</i> . . .	219
7.9	Visualisation of model instance constraint violations within a diagram editor	220
7.10	Illustrating code coverage of code generation templates	225
8.1	A screenshot of the <i>Browse Events</i> page implemented in Ticketiaml	251
8.2	A screenshot of the <i>View Event</i> page implemented in Ticketiaml	251
8.3	A screenshot of the <i>Browse Events</i> page implemented in Ticketsf-mini	253
8.4	A screenshot of the <i>View Event</i> page implemented in Ticketsf-mini	253
8.5	Comparison of the number of classes (<i>NoC</i>) and abstract classes (<i>NoAC</i>) of the IAML metamodel against other similar metamodels	257
8.6	Comparison of the total number of references (<i>TNoR</i>) and attributes (<i>TNoA</i>) of the IAML metamodel against other similar metamodels	258
9.1	Two approaches in defining a <i>Complex Term</i> that references a <i>Value</i> contained within a <i>Changeable</i> as a <i>Parameter</i>	269
B.1	Ticketiaml: The completed contents of the <i>edit event Domain Iterator</i>	327
B.2	Ticketiaml: The completed contents of the <i>Event-typed Domain Instance</i>	327
B.3	Ticketiaml: The completed contents of the <i>Edit Event Input Form</i>	328

B.4	Ticketiaml: The completed contents of the <i>title</i> Input Text Field , within the <i>Edit Event</i> Input Form , to implement synchronisation	328
B.5	A screenshot of the <i>Edit Event</i> page implemented in Ticketiaml	329
E.1	Creating a new IAML model instance through the Eclipse wizard menu (1)	341
E.2	Creating a new IAML model instance through the Eclipse wizard menu (2)	341
E.3	The IAML visual editor for a new IAML model instance	342
E.4	The default contents of the “Home” Frame within a new IAML model instance	342
E.5	The tool palette of the IAML visual editor	342
E.6	The IAML visual editor representing a Frame containing two Input Text Fields connected by a Sync Wire	344
E.7	Generating the source code for an IAML model instance (1)	344
E.8	Generating the source code for an IAML model instance (2)	344
E.9	Screenshot of the generated tutorial application accessed through Mozilla Firefox	345

List of Listings

1	Part of a web application implemented in WebDSL	23
2	Implementation of the <i>acyclical class inheritance</i> constraint of UML class diagrams in OCL	49
3	The IDL definition of the MouseEvent interface	69
4	EAnnotation documentation in Ecore	164
5	Internet Application code generation using a JET template	174
6	Internet Application code generation using an OAW Xpand Template	175
7	Example rule implementation in Jena	178
8	Example rule implementation in Jess	179
9	Example rule implementation in Drools	180
10	Implementation of <i>infinitely redirects</i> in Drools	183
11	Implementation of <i>infinitely redirects</i> in Jena	184
12	Implementation of <i>infinitely redirects</i> in OWL 2 Full	186
13	Implementation of <i>infinitely redirects</i> in CrocoPat	187
14	Implementation of <i>infinitely redirects</i> in Alloy	188
15	Implementation of <i>infinitely redirects</i> in openArchitectureWare: Xtend	188
16	Implementation of <i>infinitely redirects</i> in openArchitectureWare: Checks	189
17	Implementation of <i>infinitely redirects</i> in OCL	189
18	Implementation of <i>infinitely redirects</i> in LTL	191
19	One Drools rule used in the model completion implementation of the Sync Wire model element	212
20	Implementation of the <code>runFrameEvents</code> code generation template in Xpand	216
21	Implementation of the <code>containingScope</code> model extension in Xtend	216
22	Implementation of a Checks constraint for Sync Wires	220
23	Implementation of an OCL constraint for Values in the EMF Verification Framework	221
24	Configuration details to host generated applications through the Apache HTTP Server	344