

Verification and Validation in Systems Engineering

Mourad Debbabi · Fawzi Hassaïne · Yosr Jarraya ·
Andrei Soeanu · Luay Alawneh

Verification and Validation in Systems Engineering

Assessing UML/SysML Design Models

Dr. Mourad Debbabi
Computer Security Laboratory
Concordia Institute for Information
Systems Engineering (CIISE)
Faculty of Engineering and Computer Science
Concordia University
1455, Boulevard de Maisonneuve West, EV 7-642
Montreal, Quebec, H3G 1M8 Canada
debbabi@ciise.concordia.ca

Dr. Fawzi Hassaine
Capabilities for Asymmetric and
Radiological Defence and Simulation
Defence Research and Development
Canada – Ottawa
3701 Carling Avenue
Ottawa, ON, Canada K1A 0Z4
fawzi.hassaine@drdc-rddc.gc.ca

Dr. Yosr Jarraya
Computer Security Laboratory
Concordia Institute for Information
Systems Engineering (CIISE)
Faculty of Engineering and Computer Science
Concordia University
1455, Boulevard de Maisonneuve West, EV 7-642
Montreal, Quebec, H3G 1M8 Canada
y_jarraya@encs.concordia.ca

Andrei Soeanu
Computer Security Laboratory
Concordia Institute for Information
Systems Engineering (CIISE)
Faculty of Engineering and
Computer Science
Concordia University
1455, Boulevard de Maisonneuve West,
EV 7-642
Montreal, Quebec, H3G 1M8 Canada
a_soeanu@ciise.concordia.ca

Luay Alawneh
Department of Electrical and
Computer Engineering
Faculty of Engineering and Computer Science
Concordia University
1455, Boulevard de Maisonneuve West,
EV 13-173
Montreal, Quebec, H3G 1M8 Canada
l_alawne@encs.concordia.ca

ISBN 978-3-642-15227-6 e-ISBN 978-3-642-15228-3

DOI 10.1007/978-3-642-15228-3

Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2010936711

ACM Codes: D.2, C.4, K.6

© Springer-Verlag Berlin Heidelberg 2010

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilm or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

The use of general descriptive names, registered names, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

Cover design: KuenkelLopka GmbH, Heidelberg

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

To our families and friends!

Preface

At the dawn of the 21st century and the information age, communication and computing power are becoming ever increasingly available, virtually pervading almost every aspect of modern socio-economical interactions. Consequently, the potential for realizing a significantly greater number of technology-mediated activities has emerged. Indeed, many of our modern activity fields are heavily dependant upon various underlying systems and software-intensive platforms. Such technologies are commonly used in everyday activities such as commuting, traffic control and management, mobile computing, navigation, mobile communication. Thus, the correct function of the forenamed computing systems becomes a major concern. This is all the more important since, in spite of the numerous updates, patches and firmware revisions being constantly issued, newly discovered logical bugs in a wide range of modern software platforms (e.g., operating systems) and software-intensive systems (e.g., embedded systems) are just as frequently being reported.

In addition, many of today's products and services are presently being deployed in a highly competitive environment wherein a product or service is succeeding in most of the cases thanks to its quality to price ratio for a given set of features. Accordingly, a number of critical aspects have to be considered, such as the ability to pack as many features as needed in a given product or service while concurrently maintaining high quality, reasonable price, and short time -to- market. Consequently, modeling is becoming a key activity in product development since it enables a controlled design process that can allow for design reuse and easy feature integration in a predictable timeline. Also, the economical globalization and multinational cooperation trends are active drivers pushing for standardized modeling and engineering practices. Standardization could potentially facilitate the deployment and adoption of cost-effective verification and validation methodologies that could, in turn, be harmonized with current business goals.

The overall objective in modern product development for maximizing immediate economical gains can cause conflicts between business goals and engineering practices. Consequently, many aspects such as ergonomics, various environmental concerns, thorough verification and validation, or computational efficiency are frequently ignored. This is especially the case in those areas where the effects are not immediately apparent such as in software applications (e.g., operating systems, web

browsers, office suites) or in software-intensive systems (e.g., computer and mobile networking, portable/wearable electronics, and pervasive computing).

Although we can observe an ever-increasing awareness regarding ergonomics and environmental issues, the same can hardly be said about the necessity for a thorough verification and validation methodology for modern software and software-intensive systems. Nowadays, most of the systems' verification and validation is usually performed through testing and simulation; these techniques although very convenient for certain categories of systems do not provide necessarily for a complete screening of the possible states of a system.

Thus, it becomes apparent that an appropriate science for conscience setting should always accompany, to some extent, any development initiative. Furthermore, in such a setting, it is not so difficult to acknowledge that the most precious resource that human society benefits from is not any physical or material datum but rather the human resources themselves. As such, one may argue that as human society becomes increasingly technology dependant, the importance of assuring robust, bug-free, and high-quality software and systems is also equally increasing.

Systems engineering has marked an impressive comeback in the wake of the new challenges that emerged in this modern era of complex systems design and development. As the main objective of systems engineers is to enable the realization of successful systems, verification and validation represent an important process that is used for the quality assessment of the engineered systems and their compliance with the requirements established at the origin of the system development. Furthermore, in order to cope with the growing complexity of modern software and systems, systems engineering practices have undergone a fundamental transition to a model-based approach. In these settings, systems engineering community and standardization bodies developed interest in using and/or developing some specific modeling language that supports the cooperative work and the exchange of information among systems engineering practitioners.

This volume investigates the available means that can be employed in order to provide a dedicated approach toward the automatic verification and validation of systems engineering design models expressed in standardized modeling languages. It also provides a bird's eye view of the most prominent modeling languages for software and systems engineering, namely the unified modeling language (UML) and the more recent systems modeling language (SysML). Moreover, it innovates with the elaboration of a number of quantitative and qualitative techniques that synergistically combine automatic verification techniques, program analysis, and software engineering quantitative methods, applicable to design models described in modern modeling languages such as UML and SysML. Here is the way, the rest of this book is organized: [Chap. 1](#) presents an introduction to the verification and validation problem, systems engineering, various related standards, and modeling languages along with the model-driven architecture (MDA) concept. In [Chap. 2](#), we discuss the paradigm of architecture frameworks adopted and extended by defense organizations but also employed in the commercial arena as enterprise architecture frameworks. Moreover, in [Chap. 3](#), we provide an overview of the UML 2.0 modeling language in the historical context leading to its emergence. Then, we present in

[Chap. 4](#) the more recent SysML modeling language, the chronological account of its adoption and the commonalities and specific differences between SysML and UML. [Chapter 5](#) describes the verification, validation, and accreditation concepts. Therein, we review noteworthy assessment methodologies based on software engineering techniques, formal verification, and program analysis. [Chapter 6](#) describes an effective and synergistic approach for the verification and validation of systems engineering design models expressed in standardized modeling languages such as UML and SysML. Moreover, [Chap. 7](#) demonstrates the relevance and usefulness of software engineering metrics in assessing structural system aspects captured by UML class and package diagrams. [Chapter 8](#) details the automatic verification and validation of UML behavioral diagrams. Computational models are derived from state machine, sequence, or activity diagrams and are matched against logical properties that capture verification and validation requirements. The corresponding model-checking verification methodology is also described. [Chapter 9](#) discusses the mapping of SysML activity diagrams annotated with probabilities to Markov decision processes (MDP) that can be assessed by probabilistic model checking procedures. Furthermore, [Chap. 10](#) details the performance analysis of SysML activity diagrams annotated with time constraints and probability artifacts using a discrete-time Markov chain model. [Chapter 11](#) is dedicated to the semantic foundations of SysML activity diagrams. We define a probabilistic calculus that we call activity calculus (AC). The latter algebraically captures the formal semantics of the activity diagrams using the operational semantics framework. In [Chap. 12](#), we establish the soundness of the translation of SysML activity diagrams into PRISM specifications. This ensures that the code generated by our algorithm correctly captures the behavior intended by the SysML activity diagram given as input. Finally, a discussion of the presented work together with some concluding remarks are sketched as conclusion in [Chap. 13](#).

Acknowledgments

We would like to express our deepest gratitude to all the people who contributed to the realization of this work. Initially, our research on the verification and validation of systems engineering design models has been supported thanks to a research contract from the Defence Research & Development Canada, the R&D establishment for Canada's Department of National Defence, under the Collaborative Capability Definition, Engineering and Management (Cap-DEM) project. We also acknowledge support from the Natural Sciences and Engineering Research Council (Discovery Grant) of Canada and Concordia University (Research Chair Tier I). We would like also to express our gratitude to the members of the Computer Security Laboratory of Concordia University who helped in reviewing the preliminary versions of this book.

Contents

1	Introduction	1
1.1	Verification and Validation Problem Statement	2
1.2	Systems Engineering	3
1.3	Systems Engineering Standards	5
1.4	Model-Driven Architecture	6
1.5	Systems Engineering Modeling Languages	8
1.5.1	UML 2.x: Unified Modeling Language	8
1.5.2	SysML: Systems Modeling Language	9
1.5.3	IDEF: Integration Definition Methods	10
1.6	Outline	11
2	Architecture Frameworks, Model-Driven Architecture, and Simulation	15
2.1	Architecture Frameworks	16
2.1.1	Zachman Framework	16
2.1.2	Open Group Architecture Framework	17
2.1.3	DoD Architecture Framework	18
2.1.4	UK Ministry of Defence Architecture Framework	25
2.1.5	UML Profile for DoDAF/MODAF	25
2.2	AP233 Standard for Data Exchange	26
2.3	Executable Architectures or from Design to Simulation	26
2.3.1	Why Executable Architectures?	27
2.3.2	Modeling and Simulation as an Enabler for Executable Architectures	28
2.4	DoDAF in Relation to SE and SysML	31
2.5	Conclusion	35
3	Unified Modeling Language	37
3.1	UML History	37

3.2	UML Diagrams	38
3.2.1	Class Diagram	39
3.2.2	Component Diagram	40
3.2.3	Composite Structure Diagram	41
3.2.4	Deployment Diagram	42
3.2.5	Object Diagram	43
3.2.6	Package Diagram	43
3.2.7	Activity Diagram	44
3.2.8	Activity Diagram Execution	47
3.2.9	Use Case Diagram	48
3.2.10	State Machine Diagram	49
3.2.11	Sequence Diagram	53
3.2.12	Communication Diagram	55
3.2.13	Interaction Overview Diagram	56
3.2.14	Timing Diagram	57
3.3	UML Profiling Mechanisms	58
3.4	Conclusion	59
4	Systems Modeling Language	61
4.1	SysML History	61
4.2	UML and SysML Relationships	62
4.3	SysML Diagrams	63
4.3.1	Block Definition Diagram	64
4.3.2	Internal Block Diagram	65
4.3.3	Package Diagram	66
4.3.4	Parametric Diagram	66
4.3.5	Requirement Diagram	67
4.3.6	Activity Diagram	69
4.3.7	State Machine Diagram	71
4.3.8	Use Case Diagram	72
4.3.9	Sequence Diagram	72
4.4	Conclusion	73
5	Verification, Validation, and Accreditation	75
5.1	V&V Techniques Overview	76
5.1.1	Inspection	77
5.1.2	Testing	77
5.1.3	Simulation	78
5.1.4	Reference Model Equivalence Checking	79
5.1.5	Theorem Proving	79
5.2	Verification Techniques for Object-Oriented Design	79
5.2.1	Design Perspectives	80
5.2.2	Software Engineering Techniques	80
5.2.3	Formal Verification Techniques	81
5.2.4	Program Analysis Techniques	82

5.3	V&V of Systems Engineering Design Models	83
5.4	Tool Support	88
5.4.1	Formal Verification Environments	88
5.4.2	Static Analyzers	90
5.5	Conclusion	92
6	Automatic Approach for Synergistic Verification and Validation	95
6.1	Synergistic Verification and Validation Methodology	96
6.2	Dedicated V&V Approach for Systems Engineering	99
6.2.1	Automatic Formal Verification of System Design Models	99
6.2.2	Program Analysis of Behavioral Design Models	100
6.2.3	Software Engineering Quantitative Techniques	101
6.3	Probabilistic Behavior Assessment	101
6.4	Established Results	102
6.5	Verification and Validation Tool	103
6.6	Conclusion	105
7	Software Engineering Metrics in the Context of Systems Engineering .	107
7.1	Metrics Suites Overview	107
7.1.1	Chidamber and Kemerer Metrics	107
7.1.2	MOOD Metrics	108
7.1.3	Li and Henry's Metrics	109
7.1.4	Lorenz and Kidd's Metrics	109
7.1.5	Robert Martin Metrics	109
7.1.6	Bansiya and Davis Metrics	110
7.1.7	Briand et al. Metrics	110
7.2	Quality Attributes	111
7.3	Software Metrics Computation	111
7.3.1	Abstractness (A)	112
7.3.2	Instability (I)	112
7.3.3	Distance from the Main Sequence (DMS)	113
7.3.4	Class Responsibility (CR)	113
7.3.5	Class Category Relational Cohesion (CCRC)	114
7.3.6	Depth of Inheritance Tree (DIT)	114
7.3.7	Number of Children (NOC)	114
7.3.8	Coupling Between Object Classes (CBO)	115
7.3.9	Number of Methods (NOM)	116
7.3.10	Number of Attributes (NOA)	117
7.3.11	Number of Methods Added (NMA)	117
7.3.12	Number of Methods Overridden (NMO)	118
7.3.13	Number of Methods Inherited (NMI)	118
7.3.14	Specialization Index (SIX)	119
7.3.15	Public Methods Ratio (PMR)	119

7.4	Case Study	120
7.5	Conclusion	123
8	Verification and Validation of UML Behavioral Diagrams	125
8.1	Configuration Transition System	125
8.2	Model Checking of Configuration Transition Systems	127
8.3	Property Specification Using CTL	129
8.4	Program Analysis of Configuration Transition Systems	130
8.5	V&V of UML State Machine Diagram	131
8.5.1	Semantic Model Derivation	132
8.5.2	Case Study	134
8.5.3	Application of Program Analysis	138
8.6	V&V of UML Sequence Diagram	141
8.6.1	Semantic Model Derivation	141
8.6.2	Sequence Diagram Case Study	142
8.7	V&V of UML Activity Diagram	145
8.7.1	Semantic Model Derivation	145
8.7.2	Activity Diagram Case Study	145
8.8	Conclusion	152
9	Probabilistic Model Checking of SysML Activity Diagrams	153
9.1	Probabilistic Verification Approach	153
9.2	Translation into PRISM	155
9.3	PCTL* Property Specification	160
9.4	Case Study	161
9.5	Conclusion	166
10	Performance Analysis of Time-Constrained SysML Activity Diagrams	167
10.1	Time Annotation	167
10.2	Derivation of the Semantic Model	169
10.3	Model-Checking Time-Constrained Activity Diagrams	170
10.3.1	Discrete-Time Markov Chain	172
10.3.2	PRISM Input Language	172
10.3.3	Mapping SysML Activity Diagrams into DTMC	173
10.3.4	Threads Identification	173
10.4	Performance Analysis Case Study	176
10.5	Scalability	181
10.6	Conclusion	187
11	Semantic Foundations of SysML Activity Diagrams	189
11.1	Activity Calculus	189
11.1.1	Syntax	190
11.1.2	Operational Semantics	194

11.2	Case Study	200
11.3	Markov Decision Process	203
11.4	Conclusion	203
12	Soundness of the Translation Algorithm	205
12.1	Notation	205
12.2	Methodology	206
12.3	Formalization of the PRISM Input Language	206
12.3.1	Syntax	207
12.3.2	Operational Semantics	208
12.4	Formal Translation	210
12.5	Case Study	213
12.6	Simulation Preorder for Markov Decision Processes	215
12.7	Soundness of the Translation Algorithm	217
12.8	Conclusion	222
13	Conclusion	223
	References	227
	Index	241

List of Figures

1.1	MDA layers (source [71])	7
2.1	DoDAF views and corresponding areas of concern (source [63])	19
2.2	DMSO HLA FEDEP (source [237])	29
2.3	IEEE 1516.3 seven-step HLA FEDEP (source [237])	30
2.4	Conceptual view of an HLA federation	31
2.5	DoDAF element relations (source [63])	31
3.1	UML 2 diagram taxonomy (source [186])	38
3.2	Class diagram example	40
3.3	Component diagram example	41
3.4	Composite structure diagram example	42
3.5	Deployment diagram example	43
3.6	Class diagram (<i>top</i>) and object diagram (<i>bottom</i>)	44
3.7	Package diagram example	44
3.8	Activity control flow artifacts	46
3.9	Activity control flow patterns	48
3.10	Use case diagram example	49
3.11	State machine hierarchical clustering	50
3.12	State machine components	50
3.13	Syntax of sequence diagrams	54
3.14	Sequence diagram example (source [72])	55
3.15	Communication diagram example (source [72])	56
3.16	Interaction overview diagram example	57
3.17	Timing diagram example	58
4.1	UML and SysML relationship	63
4.2	SysML diagram taxonomy (source [187])	64
4.3	Block definition diagram example (source [190])	65
4.4	Internal block diagram example (source [190])	66
4.5	SysML package diagram example (source [190])	67
4.6	Parametric diagram example (source [190])	68
4.7	Requirement diagram example (source [190])	70
4.8	SysML activity diagram example (source [190])	71
4.9	SysML state machine example (source [190])	72
4.10	SysML use case diagram example (source [187])	73

6.1	Synergistic verification and validation proposed approach	98
6.2	Detailed description of our methodology	98
6.3	Architecture of the verification and validation environment	103
6.4	Environment screenshot: metrics assessment	104
6.5	Environment screenshot: model checking	104
6.6	Environment screenshot: property specification	105
7.1	Examples of class and package diagrams	121
8.1	Basic computation trees and their corresponding CTL properties (source [6])	129
8.2	CTL syntax	130
8.3	Case study: ATM state machine diagram	135
8.4	CTS of the ATM state machine	137
8.5	State machine counterexample	138
8.6	Corrected ATM state machine	139
8.7	Dataflow subgraphs	140
8.8	Control flow subgraph	140
8.9	ATM sequence diagram example	143
8.10	The CTS of the ATM sequence diagram example	144
8.11	Sequence diagram counterexample for property (8.6)	144
8.12	ATM activity diagram example	147
8.13	CTS of the ATM cash withdrawal activity diagram	149
8.14	Activity diagram counterexample for property (8.8)	150
8.15	Activity diagram counterexample for property (8.10)	150
8.16	Corrected ATM activity diagram example	151
9.1	Probabilistic model checking of SysML activity diagrams	155
9.2	Translation algorithm of SysML activity diagrams into MDP – part 1	158
9.3	Translation algorithm of SysML activity diagrams into MDP – part 2	159
9.4	Function generating PRISM commands	160
9.5	Case study: digital camera activity diagram – flawed design	162
9.6	PRISM code for the digital camera case study – part 1	163
9.7	PRISM code for the digital camera case study – part 2	164
9.8	Case study: digital camera activity diagram – corrected design	165
10.1	Digital camera activity diagram-flawed design	169
10.2	Proposed approach	171
10.3	PRISM code for the digital camera activity diagram example – part 1	177
10.4	PRISM code for the digital camera activity diagram example – part 2	178
10.5	Reachability graph of the flawed activity diagram	179
10.6	Effect of widely separated timescales on MTBDD size	185
10.7	Effect of added concurrency and non-downscaled time values on the MTBDD size	185
10.8	Effect of widely separated timescales on model checking	186

10.9	Effect of added concurrency and non-downscaled time values on model-checking: memory consumption	186
10.10	Effect of added concurrency and non-downscaled time values on model-checking: time consumption	187
11.1	Unmarked syntax (<i>left</i>) and marked syntax (<i>right</i>) of activity calculus	191
11.2	Marking pre-order definition	193
11.3	Mapping activity diagram constructs into AC syntax	194
11.4	Case study: activity diagram of money withdrawal	195
11.5	Semantic rules for initial	196
11.6	Semantic rules for action prefixing	196
11.7	Semantic rules for final	196
11.8	Semantic rules for fork	196
11.9	Semantic rules for non-probabilistic guarded decision	197
11.10	Semantic rules for probabilistic decision	197
11.11	Semantic rules for merge	197
11.12	Semantic rules for join	197
11.13	Case study: activity diagram of a banking operation	200
11.14	Derivation run leading to a deadlock – part 1	201
11.15	Derivation run leading to a deadlock – part 2	202
12.1	Approach to prove the correctness of the translation	206
12.2	Syntax of the PRISM input language – part 1	207
12.3	Syntax of the PRISM input language – part 2	208
12.4	Semantic inference rules for PRISM’s input language	209
12.5	PRISM code for the SysML activity diagram case study	215
12.6	Example of simulation relation using weight function	216

List of Tables

1.1	IDEF methodologies	10
2.1	Three different description types for the same product (source [259])	17
2.2	Information systems analogs for the different types of descriptions (source [259])	17
2.3	DoDAF 1.5 operational view (source [243])	20
2.4	DoDAF 1.5 systems and services – part 1 (source [243])	22
2.5	DoDAF 1.5 systems and services – part 2 (source [243])	23
2.6	DoDAF 1.5 technical standards (source [243])	24
2.7	DoDAF 1.5 all views (source [243])	25
2.8	The AV products mapping	32
2.9	The OV products mapping	33
2.10	The SV products mapping – part 1	34
2.11	The SV products mapping – part 2	35
2.12	The TV products mapping	35
4.1	Correspondence between SysML and UML (source [227])	63
4.2	Requirement diagram syntax elements	69
7.1	Package diagram metrics	122
7.2	Class diagram inheritance-related metrics	122
7.3	Class diagram general metrics	123
8.1	CTL modalities	130
8.2	Statistics related to model checking memory foot print	141
9.1	Comparative assessment of flawed and corrected design models	166
10.1	Thread allocation for the flawed design	178
10.2	Comparative table of the assessment results for the flawed and corrected designs	181
10.3	Comparative table of the performance model reachability graph complexity	181
10.4	Time intervals bounds values per experiment	182
10.5	Experimental results: varying time constants with no added concurrency (C.0)	183
10.6	Experimental results: varying time constants and adding one new concurrent module (C.1)	184

10.7	Experimental results: varying time constants and adding two new concurrent modules (C.2).....	184
10.8	Model-checking performance results – part 1	184
10.9	Model-checking performance results – part 2	184
10.10	Model-checking performance results – part 3	185

Acronyms

AP233	application protocol 233
ATM	automated teller machine
CASE	computer-aided software engineering
CBO	coupling between object classes
CCRC	class category relational cohesion
CPU	central processing unit
CR	class responsibility
CTL	computation tree logic
CTS	configuration transition system
DIT	depth of inheritance tree
DMS	distance from the main sequence
DMSO	Defense Modeling and Simulation Organization
DoD	Department of Defense
DoDAF	Department of Defense Architecture Framework
DTMC	discrete-time Markov chain
EFFBD	enhanced function flow block diagram
EIA	Electronic Industries Alliance
FSM	finite state machine
HDL	hardware description language
ICAM	integrated computer-aided manufacturing
IDEF	integrated definition language
IEEE	Institute of Electrical and Electronics Engineers
INCOSE	International Council on Systems Engineering
ISO	International Organization for Standardization
IT	information technology
LCA	least common ancestor
LTL	linear time logic
MDA	model-driven architecture
ML	modeling language
MOF	MetaObject Facility
M&S	modeling and simulation
NAVMSMO	Navy Modeling and Simulation Management Office

NMA	number of methods added
NMI	number of methods inherited
NMO	number of methods overridden
NOA	number of attributes
NOC	number of children
NOM	number of methods
OCL	object constraint language
OMG	Object Management Group
OMT	object modeling technique
PCTL	probabilistic computation tree logic
PDA	personal digital assistant
PMR	public methods ratio
PRISM	probabilistic symbolic model checker
QA	quality assurance
R&D	research and development
RPG	Recommended Practices Guide
SIX	Specialization Index
SOS	structural operational semantics
SoS	systems on top of systems
SMV	symbolic model verifier
SPT	schedulability performance and time
STEP	standard for the exchange of product model data
SysML	system modeling language
TCTL	timed computation tree logic
TeD	telecommunications description language
TOGAF	The Open Group Architecture Framework
UML	unified modeling language
V&V	verification and validation
VV&A	verification, validation, and accreditation
WebML	Web modeling language
XML	extensible markup language