An OMG® Value Delivery Modeling Language™ Publication

Value Delivery Modeling Language (VDML)

Version 1.1

OMG Document Number: formal/18-09-03

Release date: October 2018

Normative reference: https://www.omg.org/spec/VDML/

Machine Consumable file:

Normative: https://www.omg.org/spec/VDML/20180201/vdml.xmi

USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

The material in this document details an Object Management Group specification in accordance with the terms, conditions and notices set forth below. This document does not represent a commitment to implement any portion of this specification in any company's products. The information contained in this document is subject to change without notice.

LICENSES

The companies listed above have granted to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. Each of the copyright holders listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification.

Subject to all of the terms and conditions below, the owners of the copyright in this specification hereby grant you a fully-paid up, non-exclusive, nontransferable, perpetual, worldwide license (without the right to sublicense), to use this specification to create and distribute software and special purpose specifications that are based upon this specification, and to use, copy, and distribute this specification as provided under the Copyright Act; provided that: (1) both the copyright notice identified above and this permission notice appear on any copies of this specification; (2) the use of the specifications is for informational purposes and will not be copied or posted on any network computer or broadcast in any media and will not be otherwise resold or transferred for commercial purposes; and (3) no modifications are made to this specification. This limited permission automatically terminates without notice if you breach any of these terms or conditions. Upon termination, you will destroy immediately any copies of the specifications in your possession or control.

PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

GENERAL USE RESTRICTIONS

Any unauthorized use of this specification may violate copyright laws, trademark laws, and communications regulations and statutes. This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright herein may be reproduced or used in any form or by any means--graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems--without permission of the copyright owner.

DISCLAIMER OF WARRANTY

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE OBJECT MANAGEMENT GROUP AND THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO

RESTRICTED RIGHTS LEGEND

TRADEMARKS

COMPLIANCE

OMG's Issue Reporting Procedure

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page http://www.omg.org, under Documents, Report a Bug/Issue.

Contents

Table of Figures ix
Table of Tables xii
Preface xiii
1 Scope 1
1.1 Introduction 1
2 Conformance 3
2.1 Full VDML Conformance 3
2.2 VDML Metamodel Conformance 3
2.3 VDML Collaboration Modeling Conformance 3
3 References 5
3.1 Normative References 5
3.2 Non-Normative References 5
4 Terms and Definitions 8
5 Symbols 8
6 Additional Information 9
6.1 Acknowledgements 9
6.1.1 Submitting Organizations 9
6.1.2 Participants 9
6.1.3 Supporting organizations 9
6.2 IPR and Patents 10
6.3 Guide to the Specification 10
7 VDML Metamodel 11
7.1 Overview of VDML 11
7.1.1 VDML Model 12
7.1.2 Value and Value Proposition 12
7.1.3 Capability Definition 13
7.1.4 Collaboration 13
7.1.5 Community 14
7.1.6 Business Network 14
7.1.7 Organization Unit (Org Unit) 15
7.1.8 Capability Method 17
7.1.9 Activity 18
7.1.10 Port 20
7.1.11 Resources and Stores 21
7.1.12 Measures 22

7.1.13 Scenarios and contexts 22
7.1.14 Staff Collaborations 25
7.1.15 Model Integration 25
7.2 VDML Class Definitions 27
7.2.1 Collaboration and Value Creation 27
7.2.1.1 Collaborations and participants 27
7.2.1.1.1 Actor Class 28
7.2.1.1.2 Person Class 28
7.2.1.1.3 Collaboration Class 28
7.2.1.1.4 Participant Class (Abstract) 29
7.2.1.1.5 Role Class 30
7.2.1.1.6 CalendarService Class 30
7.2.1.2 Activity networks 30
7.2.1.2.1 Activity Class 31
7.2.1.2.2 ResourceUse Class 32
7.2.1.2.3 Assignment Class 34
7.2.1.2.4 DeliverableFlow Class 35
7.2.1.2.5 BusinessItem Class 36
7.2.1.2.6 Store Class 37
7.2.1.2.7 Pool Class 39
7.2.1.3 ValueAdds and ValuePropositions 39
7.2.1.3.1 ValueProposition Class 40
7.2.1.3.2 ValuePropositionComponent Class 41
7.2.1.3.3 ValueAdd Class 42
7.2.1.3.4 ValueElement Class (Abstract) 42
7.2.2 Collaboration Sub-Types 43
7.2.2.1 BusinessNetworks 44
7.2.2.1.1 BusinessNetwork Class 44
7.2.2.1.2 Party Class 44
7.2.2.2 Communities 44
7.2.2.2.1 Community Class 45
7.2.2.2.2 Member Class 45
7.2.2.3 OrgUnits and Capabilities 45
7.2.2.3.1 OrgUnit Class 46
7.2.2.3.2 Position Class 46
7.2.2.3.3 CapabilityOffer Class 47
7.2.2.3.4 ReleaseControl Class 47
7.2.2.4 CapabilityMethods 48

7.2.2.4.1 CapabilityMethod Class 48
7.2.2.4.2 Performer Class 49
7.2.3 Models and Scenarios 50
7.2.3.1 ValueDeliveryModels 50
7.2.3.1.1 ValueDeliveryModel Class 50
7.2.3.2 Scenarios and AnalysisContexts 51
7.2.3.2.1 AnalysisContext Class (Abstract) 52
7.2.3.2.2 Scenario Class 53
7.2.3.2.3 DelegationContext Class 54
7.2.4 Core Elements 55
7.2.4.1 VdmlElements 55
7.2.4.1.1 VdmlElement Class (Abstract) 55
7.2.4.1.2 Attribute Class 56
7.2.4.1.3 Annotation Class 56
7.2.4.1.4 MeasurableElement Class (Abstract) 56
7.2.4.1.5 MeasuredCharacteristic Class 56
7.2.4.2 Expressions 57
7.2.4.2.1 Expression Class 57
7.2.4.2.2 Operand Class 57
7.2.4.3 PortContainers 57
7.2.4.3.1 Port Class (Abstract) 58
7.2.4.3.2 OutputPort Class 59
7.2.4.3.3 InputPort Class 59
7.2.4.3.4 PortContainer Class (Abstract) 60
7.2.4.4 PortDelegations 60
7.2.4.4.1 PortDelegation Class (Abstract) 61
7.2.4.4.2 InputDelegation Class 61
7.2.4.4.3 OutputDelegation Class 61
7.2.5 Libraries 62
7.2.5.1 BusinessItemLibrary 62
7.2.5.1.1 BusinessItemLibrary Class 63
7.2.5.1.2 BusinessItemDefinition Class 63
7.2.5.1.3 BusinessItemCategory Class 64
7.2.5.1.4 BusinessItemLibraryElement Class (Abstract) 64
7.2.5.2 ValueLibrary 64
7.2.5.2.1 ValueLibrary Class 65
7.2.5.2.2 ValueDefinition Class 65
7.2.5.2.3 ValueCategory Class 66

7.2.5.3 CapabilityLibrary 66
7.2.5.3.1 CapabilityLibrary Class 67
7.2.5.3.2 CapabilityDefinition Class 68
7.2.5.3.3 CapabilityCategory Class 68
7.2.5.3.4 Capability Class (Abstract) 69
7.2.5.3.5 CapabilityDependency Class 69
7.2.5.4 PracticeLibrary 70
7.2.5.4.1 PracticeLibrary Class 71
7.2.5.4.2 PracticeDefinition Class 71
7.2.5.4.3 PracticeCategory Class 72
7.2.5.5 RoleLibrary 72
7.2.5.5.1 RoleLibrary Class 73
7.2.5.5.2 RoleDefinition Class 73
7.2.5.5.3 RoleCategory Class 73
7.2.6 Integration with SMM (Structured Metrics Metamodel) 74
7.2.6.1 Packages 74
7.2.6.2 SMM Main Concepts 75
8 Notation 77
8.1 General 77
8.2 Role Collaboration 77
8.3 ValueProposition Exchange 79
8.4 Activity Network 80
8.5 Collaboration Structure 86
8.6 CapabilityLibrary 89
8.7 Capability Heatmap 90
8.8 Capability Management 91
8.9 Measurement Dependency 94
Annexes 96
Annex A: Glossary 97
Annex B: Alignment with Existing Business Modeling Techniques 101
B.1 Overview 101
B.2 Value Networks 101
B.3 REA (Resources Events Agents) 103
B.4 e^3value 106
B.5 Capability Maps 107
B.6 Value Stream 108
B.7 Business Model 109

B.7.1 Lindgren 109
B.7.2 Osterwalder 110
B.8 Possession, Ownership, Availability (POA) 111
B.9 VDML Support for BMM Strategic Planning 113
B.10 VDML Support for Balanced Scorecard and Strategy Map 118
B.11 VDML Relationship to BPMN 121
Annex C: Use Cases 123

Table of Figures

Figure 7-1: VDML Viewpoints 11
Figure 7-2: Capability Offers 17
Figure 7-3: Activity structure 19
Figure 7-4: Two uses of a collaboration 23
Figure 7-5: Scenarios and context trees 24
Figure 7-6: Collaborations 28
Figure 7-7: Activities 31
Figure 7-8: Assignments 34
Figure 7-9: DeliverableFlows 35
Figure 7-10: BusinessItems 37
Figure 7-11: Stores 38
Figure 7-12: Values and ValuePropositions 40
Figure 7-13: BusinessNetworks 44
Figure 7-14: Communities 45
Figure 7-15: OrgUnits and Capabilities 46
Figure 7-16: CapabilityMethods 48
Figure 7-17: ValueDeliveryModels 50
Figure 7-18: Scenarios and AnalysisContexts 52
Figure 7-19: VdmlElements 55
Figure 7-20: Expressions 57
Figure 7-21: PortContainers 58
Figure 7-22: PortDelegations 61
Figure 7-23: BusinessItemLibraries 62
Figure 7-24: ValueLibraries 65
Figure 7-25: CapabilityLibraries 67
Figure 7-26: PracticeLibraries 71
Figure 7-27: RoleLibraries 72
Figure 7-28: VDML Metamodel package 74
Figure 7-29: SMM main concepts 75
Figure 8-1: Role shape as oval 77
Figure 8-2: Role shape with expand button 77
Figure 8-3: DeliverableFlow shape for Tangible 78
Figure 8-4: DeliverableFlow shape for Intangible 78
Figure 8-5: DeliverableFlow for Tangible, connecting two Roles 78
Figure 8-6: Role Collaboration diagram (BusinessNetwork example) 79
Figure 8-7: ValueProposition shape 79
Figure 8-8: Role providing a ValueProposition 79
Figure 8-9: Role receiving a ValueProposition 79
Figure 8-10: ValueProposition Exchange diagram (example) 80
Figure 8-11: Swim-lane shape for Role (in Activity Network diagram) 80
Figure 8-12: Activity shape 80
Figure 8-13: Activity shape, with expand button 80
Figure 8-14: Store shape 81
Figure 8-15: Pool shape 81
Figure 8-16: Connector shape for DeliverableFlow (in Activity Network diagram) 81

Figure 8-17: Connector shape for internalPortDelegation (in Activity Network diagram) 81
Figure 8-18: shape of OutputPort, on boundary of Activity 82
Figure 8-19: Shape of OutputPort, with Condition, on boundary of Activity 82
Figure 8-20: Shape of OutputPort, with ValueAdd, on boundary of Activity 83
Figure 8-21: Shape of OutputPort, with ValueAdd and Condition, on boundary of Activity 83
Figure 8-22: Shape of InputPort, on boundary of Activity 83
Figure 8-23: Shape of InputPort, with Condition, on boundary of Activity 83
Figure 8-24: Shape of InputPort, receiving roleResource, on boundary of Activity 83
Figure 8-25: Shape of InputPort, receiving role Resource, and with Condition, on boundary of Activity 83
Figure 8-26: Shape of OutputPort, on boundary of Store 84
Figure 8-27: Shape of OutputPort, with Condition, on boundary of Store 84
Figure 8-28: Shape of OutputPort, with ValueAdd, on boundary of Store 84
Figure 8-29: Shape of OutputPort, with ValueAdd and Condition, on boundary of Store 84
Figure 8-30: Shape of InputPort, on boundary of Store 84
Figure 8-31: Shape of InputPort, with Condition, on boundary of Store 84
Figure 8-32: Shape of Collaboration InputPort, connected to internalPortDelegation 85
Figure 8-33: Shape of Collaboration OutputPort, connected to internalPortDelegation 85
Figure 8-34: Shape of Collaboration OutputPort, with ValueAdd, and connected to internalPortDelegation 85
Figure 8-35: Activity Network diagram (simple example) 85
Figure 8-36: Activity Network diagram (simple example) 86
Figure 8-37: Role Collaboration and Activity Network as synchronized views (example) 86
Figure 8-38: Collaboration shape 86
Figure 8-39: BusinessNetwork shape 87
Figure 8-40: OrgUnit shape 87
Figure 8-41: CapabilityMethod shape (in Collaboration Structure and Capability Management diagrams) 87
Figure 8-42: Community shape 87
Figure 8-43: Role containment connector 87
Figure 8-44: Collaboration structure, with Role of parent Collaboration assigned to sub-Collaboration 88
Figure 8-45: Role Assignment connector 88
Figure 8-46: Actor assigned to Role 88
Figure 8-47: Collaboration Structure diagram (example) 89
Figure 8-48: Capability shape (in CapabilityLibrary diagram) 89
Figure 8-49: Capability hierarchy 89
Figure 8-50: Capability shape with expand button (in CapabilityLibrary diagram) 89
Figure 8-51: Expanded parent Capability, with sub-Capability 90
Figure 8-52: CapabilityLibrary diagram (example) 90
Figure 8-53: Capabilities with heatIndex about HeatThreshold (in Capability Heatmap) 91
Figure 8-54: CapabilityOffer shape 91
Figure 8-55: Shape of connector between CapabilityOffer and a capabilityResource or method 91
Figure 8-56: CapabilityOffers on boundary of OrgUnit, with expand button (in: Capability Management diagram) 91
Figure 8-57: OrgUnit expanded (in: Capability Management diagram) 92
Figure 8-58: CapabilityOffers of OrgUnit with capabilityResource and method from other OrgUnit 92

Figure 8-59: CapabilityOffers of OrgUnit with capabilityResource and method from other OrgUnit (not shown) 93
Figure 8-60: Connector shape for dependency of CapabilityMethod on other CapabilityOffer(s) 93
Figure 8-61: Dependencies of CapabilityMethod on CapabilityOffers of methodOwner and other OrgUnits 93
Figure 8-62: Capability Management diagram (example) 94
Figure 8-63: MeasuredCharacteristic shape (in Measurement Dependency diagram) 94
Figure 8-64: Shape of MeasurementRelationship, with "positive" influence 95
Figure 8-65: Shape of MeasurementRelationship, with "negative" influence 95
Figure 8-66: MeasurementDependency diagram (example) 95
Figure B-1: Value Network Map or Graph 102
Figure B-2: Example of an REA Model 104
Figure B-3: Example of an e^3 value Model 106
Figure B-4: Capability Heat Map 108
Figure B-5: The Business Model Cube (Lindgren) 109
Figure B-6: The Business Model Canvas (Osterwalder) 110
Figure B-7: Example of a POA Model 112
Figure B-8: Overview of BMM 115
Figure B-9: Strategic planning process 116
Figure B-10: VDML models for BSC/SM 119
Figure B-11: Modeling transformation phases 120

Table of Tables

Table 1.1 - Business Challenges and VDML Solutions 1
Table B.1 - Mapping of VNA Concepts to VDML Concepts 102
Table B.2 - Mapping of REA Concepts to VDML Concepts 105
Table B.3 - Mapping of e^{3}value Concepts to VDML Concepts 106
Table B.4 - Mapping of Business Model Cube Concepts to VDML Concepts 109
Table B.5 - Mapping of Business Model Canvas Concepts to VDML Concepts 111
Table B.6 - Mapping of POA Concepts to VDML Concepts 112

Preface

OMG

Founded in 1989, the Object Management Group, Inc. (OMG) is an open membership, not-for-profit computer industry standards consortium that produces and maintains computer industry specifications for interoperable, portable, and reusable enterprise applications in distributed, heterogeneous environments. Membership includes Information Technology vendors, end users, government agencies, and academia.

OMG member companies write, adopt, and maintain its specifications following a mature, open process. OMG's specifications implement the Model Driven Architecture® (MDA®), maximizing ROI through a full-lifecycle approach to enterprise integration that covers multiple operating systems, programming languages, middleware and networking infrastructures, and software development environments. OMG's specifications include: UML® (Unified Modeling Language™); CORBA® (Common Object Request Broker Architecture); CWM™ (Common Warehouse Metamodel); and industry-specific standards for dozens of vertical markets.

More information on the OMG is available at http://www.omg.org/.

OMG Specifications

As noted, OMG specifications address middleware, modeling and vertical domain frameworks. All OMG Specifications are available from the OMG website at:
http://www.omg.org/spec

All of OMG's formal specifications may be downloaded without charge from our website. (Products implementing OMG specifications are available from individual suppliers.) Copies of specifications, available in PostScript and PDF format, may be obtained from the Specifications Catalog cited above or by contacting the Object Management Group, Inc. at:

OMG Headquarters
109 Highland Avenue
Needham, MA 02494
USA
Tel: +1-781-444-0404
Fax: +1-781-444-0320
Email: *pubs@omg.org*

Certain OMG specifications are also available as ISO standards. Please consult http://www.iso.org

Issues

The reader is encouraged to report any technical or editing issues/problems with this document by completing the Issue Reporting Form listed under OMG specifications on the main web page.

This page intentionally left blank.

1 Scope

1.1 Introduction

The purpose of VDML is to provide a standard modeling language for analysis and design of the operation of an enterprise with particular focus on the creation and exchange of value. It provides an abstraction of the operation of an enterprise that is appropriate for business executives, along with representation of supporting detail for business analysts to link strategy and business models to the activities, roles, and capabilities that run the enterprise. The target users are business people—executives, business architects, analysts and managers. Information systems analysts and designers may use VDML models as specifications for the design of supporting information systems.

VDML is designed to address several critical business challenges:

1) It creates a robust way to model both tangible and intangible value flows,
2) It provides the capacity to model complex collaborations and business networks,
3) It provides a flexible way to model business activities to more readily support continuous transformation in environments of high variability and,
4) It supports more effective shared capabilities optimization and deployment. Table 1.1, below, highlights these challenges and VDML solutions.

Table 1.1 - Business Challenges and VDML Solutions

Business Challenge	How VDML Meets the Challenge
Model both tangible and intangible value flows.	• Models creation and exchange of both tangible (formal) and intangible (informal or ad hoc) deliverables and values exchanged between roles and their benefits. • Compatible with intangible asset or resource management. • Can model both proposed and realized value. • Supports value stream analysis to address customer values.
Model complex collaborations and business networks	• Models traditional organizations. • Models communities, informal collaborations and business networks. • Allows for drill down for analysis of performance and responsibility and considering alternative demand scenarios. • Clarifies responsibilities of roles in collaborations. • Provides a basis for using network analytics to assess business models and value flows.
Model business transformations in environments of high variability	• Provides a higher level of abstraction than business processes for an enterprise perspective on transformation plans and priorities. • Models multiple business entities and activities as an ecosystem. • Provides operational definition of critical business frameworks (Osterwalder, balanced scorecard, value stream). • Enables shared understanding of business requirements for business transformation and support from BPM and IT. • Provides a way to model non routine, highly variable or non standardized activities.
Model shared capabilities and their deployment across multiple activities and lines of business	• Models organizational responsibilities and performance of capabilities in multiple contexts. • Provides a tighter linkage to value creation.

	• Helps to more effectively identify, build, change and deploy shared capabilities across multiple collaborations, business units, and activities.

Given these challenges, VDML is designed to bring together the organization or network structure with the creation and exchange of value and defines the capabilities that produce value. In particular, the VDML metamodel has been developed to support value chain, value stream and capability analysis, and has been refined to support Value Network Analysis (VNA), e^3value modeling, REA (Resource Event Agent) analysis and an owner/investor business model. Further, it is designed to support modeling of the human collaborations and role-based interactions both within organizations and within networks that are required to support value delivery. VDML can demonstrate the management of resources, assignment of people and roles, exchanges with business partners and performance measures that help identify problems and opportunities to improve the business.

Central to VDML is the concept of value. *Value is a measurable factor of benefit delivered to a recipient in association with a deliverable.* Examples of value include the fitness of a product for a purpose, a measure of product reliability, a probability of production defects, a commitment to future delivery of another deliverable, a measure of product or brand prestige, information that provides a business advantage, or any other feature or benefit that would affect the desirability of a product, service or economic exchange. VDML is designed to support the optimization of stakeholder value for both internal and external facing business activities. VDML supports value measurement from both operational and recipient satisfaction perspectives. It supports modeling and analysis of both tangible and intangible business value and is compatible with resource management frameworks utilizing intangible assets as well as traditional financial assets and resources.

Also central to VDML is the ability to model collaborative business relationships and role based business networks. At the level of business partner interactions, VDML can represent the net exchange of value between business entities (e.g., companies, agencies or consumers) or go into the detail of transactional exchanges for achieving exchange agreements and managing the exchange of economic resources. These value exchange models can support analysis of the overall effect of exchanges between multiple enterprises that enable each of the participants to realize a perceived net gain to sustain the relationship. Such analyses also can be important for understanding the costs, risks, and delays of these transactions and the viability of the business model.

For analysis of business operations, VDML supports a perspective of value-driven business design by focusing on the activities and flow of deliverables that produce products or services and the delivery of tangible and intangible values. The delivery of value can focus on end customers and external stakeholders as well as value delivered to internal customers or the enterprise entity.

VDML can provide a framework and generate requirements for the design of business processes, but it provides a view that is different from BPMN and other process modeling tools by focusing on the consumption and production of deliverables and the statistical performance and contributions of value by activities including cost, quality and duration. As such it provides a vital link between business strategy and enterprise value models and business processes. However, VDML avoids the detail of the operational control aspects of business processes. The focus is on delivery of value and the means to that end.

VDML scales from key operational activities to full industry level business models and large scale business networks. It is appropriate for commercial and non-commercial endeavors as well as government agencies. A VDML model can extend from product concept to full commercialization, delivery and customer support. It can support the capture of measurements to assess the impact of performance of specific activities on the values of end products or services. It aligns these concepts with business capabilities that can be managed, shared and optimized from an enterprise perspective, and it links all of these with the responsible people, collaborations, and organizations that manage and perform these capabilities.

The measurements of a VDML model represent statistical figures for delivery of a product or service, for a market segment, a product line, a product mix or a line of business or other operational variations. Where a line

of business involves similar but different product configurations, the measurements can represent a particular product mix. Different scenarios may be used to analyze different products or product mixes that, for the most part, use the same capabilities.

More detailed analysis that reflects variability of products and operating circumstances would require simulation. VDML does not directly support simulation, but a VDML model can capture the fundamental data needed to support simulation such as System Dynamics, Monte Carlo Simulation and Discrete Event Simulation. Changes to a VDML model would then be directly incorporated into the simulation and results can be incorporated to update or create a VDML scenario. While these simulation techniques are outside the scope of VDML, it is expected that some implementers may want to include such capabilities with their products. The design of the VDML metamodel is intended to be compatible with extensions to include simulation.

This specification includes a limited, normative, graphical notation. It is expected that notation will evolve as users identify new ways to view their robust models. Implementations may implement non-normative views that are familiar to users of existing techniques. It is expected that the graphical displays will be complemented by tabular displays, some of which are suggested by the use cases.

This specification does not define the various measurements applicable to performance and value delivery analysis. It is expected that these will be domain/industry-dependent and that industry or professional groups will establish shared libraries to be imported as a basis for models in that industry. This specification incorporates the SMM (Structured Metrics Metamodel) specification to represent the measure libraries and the measurable properties of model elements.

The scope of a particular VDML model (a model created using VDML) will depend very much on the purpose of the model. The expectation is that a VDML model can be applied to address specific problems, but then can be maintained and grown to provide a sustainable, integrated abstraction of the operation of the enterprise so that little need be added to the model to address problems and opportunities that are encountered in the future.

2 Conformance

2.1 Full VDML Conformance

A conformant implementation supports the normative graphical notation and imports and exports models that conform to the VDML metamodel (Sub-clause 7.2) and the SMM (Structured Metrics Metamodel).

2.2 VDML Metamodel Conformance

A product can claim “VDML Metamodel Conformance” if it can import and export XMI that is consistent with the VDML metamodel (Sub-clause 7.2) and the SMM (Structured Metrics Metamodel) but does not conform to the VDML notation Clause 8). This is a subset of Full VDML Conformance (Sub-clause 2.1).

2.3 VDML Collaboration Modeling Conformance

This subset of the VDML metamodel can be implemented for modeling organizational structures and relationships. A VDML Collaboration Modeling implementation conforms if it can import and export XMI that is consistent with the SMM (structured Metrics Metamodel) and with VDML Metamodel Conformance (Sub-clause 2.2) but implements only the following classes:

- `ValueDeliveryModel` (7.2.3.1.1)
- `VdmlElement` (7.2.4.1.1)
- `MeasurableElement` (7.2.4.1.4)

- `Attribute` (7.2.4.1.2)
- `Annotation` (7.2.4.1.3)
- `MeasuredCharacteristic` (7.2.4.1.5)
- `PortContainer` (7.2.4.3.4)
- `Collaboration` (7.2.1.1.3)
- `Role` (7.2.1.1.5)
- `RoleDefinition` (7.2.5.5.2)
- `RoleLibrary` (7.2.5.5.1)
- `RoleCategory` (7.2.5.5.3)
- `Participant` (7.2.1.1.4)
- `Actor` (7.2.1.1.1)
- `Person` (7.2.1.1.2)
- `Assignment` (7.2.1.2.3)
- `OrgUnit` (7.2.2.3.1)
- `Position` (7.2.2.3.2)
- `Community` (7.2.2.2.1)
- `Member` (7.2.2.2.2)
- `BusinessNetwork` (7.2.2.1.1)
- `Party` (7.2.2.1.2)

This level of conformance is a subset of VDML Metamodel Conformance (Sub-clause 2.2) and thus does not include conformance with VDML notation (Sub-clause 8). Associations between the above classes and any other VDML classes allow multiplicity of zero and may be ignored.

3 References

3.1 Normative References

BMM, *Business Motivation Model*, version 1.1, May 2010, OMG Document Number: ptc/2010-05-01, http://www.omg.org/spec/BMM/1.1 .

BPMN, *Business Process Model and Notation*, version 2.0, June 2010, OMG Document Number: dtc/2010-06-05, http://www.omg.org/spec/BPMN/2.0 .

CMMN, *Case Management Model and Notation*, version 1.0, May 2014, OMG Document Number: formal/2014-05-05, http://www.omg.org/spec/CMMN/1.0/ .

MOF, *Meta Object Facility*, version 2.4.1, June 2013, OMG Document Number: formal/2013-06-01, http://www.omg.org/spec/MOF/2.4.1 .

SMM, *Structured Metrics Metamodel*, version 1.2, November 2017, OMG Document Number: ptc/17-11-02, http://www.omg.org/spec/SMM/1.2 .

SoaML, *SOA Modeling Language*, version 1.0, Release Date: March 2012, http://www.omg.org/spec/SoaML/1.0/ .

UML, *Unified Modeling Language*, version 2.4.1, June 2013, OMG Document Number: formal/2011-08-06, http://www.omg.org/spec/UML/2.4.1 .

3.2 Non-Normative References

Allee, V., *The Future of Knowledge: Increasing Prosperity through Value Networks*, Butterworth-Heinemann 2003.

Allee, V., *Value Network Analysis and Value Conversion of Tangible and Intangible Assets*, Journal of Intellectual Capital, Volume 9, issue 1, pp 5-24, January 2008, http://www.vernaallee.com/images/VAA-VNAandValueConversionJIT.pdf .

Allee, V., *Value Networks and the True Nature of Collaboration*, ValueNet Works, 2011, http://www.valuenetworksandcollaboration.com.

Ballantyne, D., Varey, R.J., Frow, P. and Payne, A., *Service-dominant logic and value propositions: Re-examining our mental models*, Otago Forum 2, Paper no: 5, 2008, http://www.business.otago.ac.nz/marketing/events/OtagoForum/Final%20forum%20papers/Otago%20Forum%20Paper%205_Ballantyne.pdf .

BPMM, *Business Process Maturity Model*, version 1.0, Object Management Group, Release Date: June 2008, http://www.omg.org/spec/BPMM/1.0/PDF/ .

Brodie, L. and Gilb, T., *Values for Value,* AgileRecord, October 2010, http://www.gilb.com/dl448 .

Cummins, Fred A., *Building the Agile Enterprise with SOA, BPM and MBM*, Morgan Kaufman, 2009.

Cummins, Fred A., Building the Agile Enterpise (blog) includes several posts on VDML, 2011-2013.

Cummins, Fred A., and Henk de Man, *VDML support for the Business Architecture Guild BizArch Viewpoint*, OMG document number bmi/2013-11-02, November, 2013, http://www.omg.org/cgi-bin/doc?bmi/2013-11-02 .

Cummins, Fred A., and Henk de Man, *Analysis of the Relationships between VDML and BPMN*, OMG document number bmi/2013-11-01, November, 2013, http://www.omg.org/cgi-bin/doc?bmi/2013-11-01 .

Gane, C. and Sarson, T, *Structured Systems Analysis: Tools and Techniques*, Prentice-Hall Software Series, Prentice-Hall, Englewood Cliffs, New Jersey, 1979.

Geerts, G. L. and W. E. McCarthy, *An Ontological Analysis of the Primitives of the Extended REA Enterprise Information Architecture*. The International Journal of Accounting Information Systems, March 2002.

Gilb, T., *Value Delivery in Systems Engineering*, October 2007, Published and used by INCOSE with permission, http://www.gilb.com/dl137

Gilb, T. and Gilb, K., *Done should mean value delivered to Stakeholders*, AgileRecord, October 2011, http://www.gilb.com/dl484 .

GoldSim, *Summary of Major New Features and Changes*, version 10.1, GoldSim Technology Group, February 2010, http://www.goldsim.com/downloads/Documents/Version101Summary.pdf .

GoldSim, *Probabilistic Simulation Environment, User's Guide*, version 10.5, GoldSim Technology Group, December 2010, http://www.goldsim.com/downloads/Documents/Version101Summary.pdf .

Gordijn, J. and Akkermans, H., *Value based requirements engineering: Exploring innovative e-commerce ideas*. In Requirements Engineering Journal, Vol. 8(2):114-134, 2003, http://e3value.few.vu.nl/docs/bibtex/pdf/Gordijn2003e3value.pdf, or a popular version of it: http://e3value.few.vu.nl/docs/bibtex/pdf/Gordijn2001e3value.pdf .

Grant, R. M.,*The Resource-Based Theory of Competitive Advantage: Implications for Strategy Formulation*. California Management Review, March 1991.

Harmon, Paul, *Business Process Change: A Guide for Business Managers and BPM and Six Sigma Professionals*, Morgan Kaufman, 2007.

Hruby P., Kiehn J. And Scheller C.: *Model-Driven Design Using Business Patterns*, Springer-Verlag, 2006.

IBM, *Component Business Modeling*, http://www.haifa.ibm.com/projects/software/cbm/index.html .

ITIL, *Service Design*, ITIL version 3, August 2011, http://www.best-management-practice.com/Publications-Library/IT-Service-Management-ITIL/ITIL-2011-Edition/Service-Design/ .

Johnson, M. W., Christensen, C. M., and Kagermann, H., *Reinventing Your Business Model, Harvard Business Review on Business Model Innovation*, Harvard Business School Publishing Corporation, 2010.

Jones, D. and Womack, j., *Seeing the Whole*. Lean Enterprise Institute, March 2003, See an on-line chapter in http://www.lean.org/Library/Seeing_the_Whole_Part1.pdf. Online version of Final Draft, Journal of Intellectual Capital, Volume 9, No. 1, 2008.

Lindgren, P. and Jørgensen,R., M.-S. Li, Y. Taran, K. F. Saghaug, *Towards a new generation of business model innovation model*, presented at the 12th International CINet Conference: Practicing innovation in times of discontinuity, Aarhus, Denmark, 10-13 September 2011

Martin, J., *The Great Transition: Using the Seven Disciplines of Enterprise Engineering*, AMACOM. New York,1995.

McCarthy, W. E., *The REA accounting model: A generalized framework for accounting systems in a shared data environment*, Accounting Review, July 1987.

McFarland, Daniel A., *The Pursuit of Organizational Intelligence*, Blackwell Publishers, Oxford, UK, 1999.

Osterwalder, A., *The Business Model Ontology- A Proposition in a Design Science Approach*, Thesis, University of Lausanne, 2004, http://www.hec.unil.ch/aosterwa/PhD/Osterwalder_PhD_BM_Ontology.pdf .

Osterwalder, A. and Pigneur, Y., *Business Model Generation: A Handbook for Visionaries, Game Changers, and Challengers*, John Wiley & Sons, 2010.

PMBOK, *A Guide to the Project Management Body of Knowledge (PMBoK Guide)*, Project Management Institute (PMI), 2000, http://www.cs.bilkent.edu.tr/~cagatay/cs413/PMBOK.pdf

Porter, M. E., *Competitive Advantage: Creating and Sustaining Superior Performance*, The Free Press, New York, 1985.

Prahalad, C.K and Hamel, G., *The Core Competence of the Corporation*. Harvard Business Review, May/June 1990.

Rother, M. and Shook, J., *Learning to See*. Lean Enterprise Institute, 1998.

Scheller. C.V., Hruby, P. *Is POA the Precise Semantics of REA?*, 3rd International Workshop on Value Modeling and Business Ontologies, Stockholm, Sweden, 2009.

Scheller, C.V., Hruby, P. *Modeling Services and Intellectual Property Rights Using POA* (Possession, Ownership, Availability), 5th International Workshop on Value Modeling and Business Ontologies, Gent, The Netherlands, 2011.

SoaML, *SOA Modeling Language*, version 1.0, Release Date: March 2012, http://www.omg.org/spec/SoaML/1.0/ .

SOA-RM, *Reference Model for Service Oriented Architecture 1.0*, OASIS, October 2006, http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.html.

Sowa, J. F. and Zachman, J. A., *Extending and formalizing the framework for information systems architecture*, IBM Systems Journal, vol 31, no 3, 1992, http://www.zachman.com/images/ZI_PIcs/ibmsj1992.pdf .

Stabell, C.B., and Fjeldstad, O.D., *Configuring Value for Competitive Advantage: On Chains, Shops and Networks*, Strategic Management Journal, 19(5), 413-417, 1998, http://www.agbuscenter.ifas.ufl.edu/5188/miscellaneous/configuring_value.pdf .

Teece, D. J., Pisano, G. and Shuen, A., *Dynamic Capabilities and Strategic Management*. Strategic Management Journal, August 1997.

Vervest, P. H.M., Van Liere, D. W. and Zheng, Li (Eds.), *The Network Experience, New Value from Smart Business Networks*, Springer, 2009, http://www.erim.eur.nl/ERIM/publications/book_releases/Release?p_item_id=5157588&p_pg_id=93

Weill, P. and M. R. Vitale (2001). *Place to space: Migrating to eBusiness Models*. Boston, Harvard Business School Press.

Whittle, R., and Myrick, C.B., *Enterprise Business Architecture: The Formal Link Between Strategy and Results*, CRC Press, 2005.

Wikipedia, *Value Chain*, http://en.wikipedia.org/wiki/Value_chain .

Zachman, J. A., *A framework for information systems architecture*, IBM Systems Journal, vol 26, no 3, 1987, http://www.cesames.net/wp-content/uploads/2010/04/ibmsj2603e.pdf .

Zollo, M. and Winter, S. G., *Deliberate Learning and the Evolution of Dynamic Capabilities*. Organization Science, 2002, Vol. 13, No. 3.

4 Terms and Definitions

Terms and definitions are included in Annex A: Glossary.

5 Symbols

Symbols are consistent with the MOF and UML specifications.

6 Additional Information

6.1 Acknowledgements

This sub clause identifies the organizations and representatives that are formal submitters of this specification as well as those who participated in the development of the specification and those who have an interest in the result and support adoption.

6.1.1 Submitting Organizations

Cordys Corporation B.V.
Henk de Man, henkdman@gmail.com
Mudigonda Rajender, mrajender@gmail.com
CSC
Pavel Hruby phruby@csc.com
Victor L. Harrison 1vharrison7@gmail.com
Klaus Loehnert kloehner@csc.com

6.1.2 Participants

In addition to the submitters, the following people contributed directly to the development of this specification.

Verna Allee, verna.allee@valuenetworks.com

Arne Berre, Arne.J.Berre@sintef.no

Fred Cummins, fred.a.cummins@gmail.com

Larry Hines, Larry.Hines@microfocus.com

Peter Lindgren, pel@production.aau.dk

Pete Rivett, pete.rivett@adaptive.com

6.1.3 Supporting organizations

Aalborg University
Peter Lindgren pel@production.aau.dk
Adaptive
Pete Rivett pete.rivett@adaptive.com
Agile Enterprise Design
Fred Cummins fred.a.cummins@gmail.com
AT&T
Jenny Huang, jh2873@att.com
BizAgi, Ltd.
Jesus Sanchez Jesus.Sanchez@bizagi.com
Eindhoven Technical University
Rik Eshuis, Ph.D
Fujitsu
Hiroshe Miyazaki miyazaki.hir-02@jp.fujitsu.com
InPaqt
Felix Janszen, felix.janszen@inpaqt.nl
Mega International
Antoine Lonjon, antoine.lonjon@mega.com
Ministery of Defense, Netherlands
J.C. van Es, jc.v.es@mindef.nl

Oelan
H. (Hans) van Bommel, hans.vanbommel@oelan.nl
Princeton Blue
Vitaly Khusidman, vitaly.khusidman@princetonblue.com.
REA Technology
Christian Vibe Scheller, cvs@reatechnology.com.
SINTEF
Arne Berre, Arne.J.Berre@sintef.no
Strategic Value Partners
Neal McWhorter, nealmcwhorter@strategicvaluepartners.com
ValueNet Works
Verna Allee verna.allee@valuenetworks.com
Vlastuin Group
J.A.J.G.M. Lentjes, j.lentjes@vlastuin.nl

6.2 IPR and Patents

The submitters contributed this work to OMG on a Non-Assert basis.

6.3 Guide to the Specification

Clauses 1 - 6 are introductory topics, and clause 7 contains the metamodel specification that defines the structure of a compliant metamodel. Clause 8 includes VDML Notation. Additional information is provided by the following Annexes:

- Annex A: Glossary
- Annex B: Alignment with Existing Business Modeling Techniques
- Annex C: Use Cases

7 VDML Metamodel

7.1 Overview of VDML

This clause specifies the normative, VDML metamodel. It starts with an overview to provide a general understanding of the concepts to be modeled and the semantics and relationships of the model elements. The next sub-clause describes the details of the metamodel elements and their attributes and relationships.

VDML provides a medium for the consistent representation and integration of business concepts and viewpoints for business leaders to develop understanding, consensus and what-if scenarios, along with the ability to exchange the resulting models between different but complaint modeling tools. Like BPMN, VDML includes concepts of activities, roles, flows and participants, but VDML provides a higher level of abstraction of business activity to focus on statistical characteristics of activities, resources, deliverables and value contributions - along with concepts of business capabilities and extended organizational relationships - to provide an enterprise-level perspective on the operation of the business. This enterprise perspective supports recognition and understanding of problems and opportunities in the context of market demand and enterprise optimization, and it highlights the relative importance of potential changes to the structure and capabilities of the enterprise.

The VDML modeling concepts include an aggregation of concepts that occur in different enterprise-level, business modeling and analysis techniques. As a result, a VDML model can support multiple viewpoints—different abstractions of the design of an enterprise. Figure 7-1 depicts a number of viewpoints that have been considered in the development of VDML. Several of these viewpoints are established business modeling techniques; Annex B: Alignment with Existing Business Modeling Techniques discusses the alignment of the concepts of a number of established techniques to VDML.

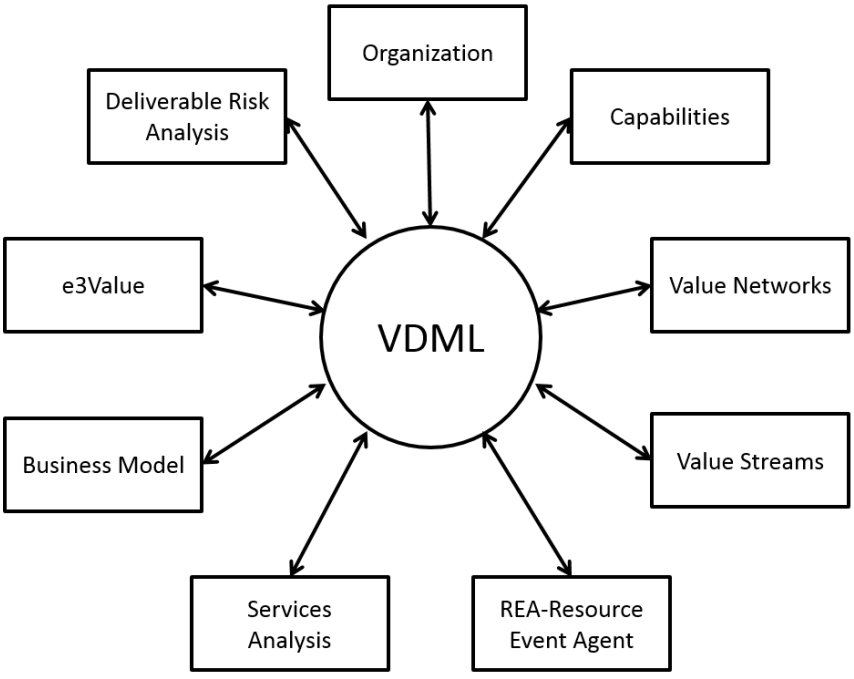

Figure 7-1: VDML Viewpoints

This specification includes minimal normative, graphical notation. It is expected that notations of existing viewpoints will evolve and new views will be developed when more robust models can be developed using VDML. Two substantial use cases are included in separate documents to illustrate the graphical notation and application of VDML. See Annex C: Use Cases.

VDML, version 1, does not include a simulation capability, but the ability to extend VDML for Discrete Event Simulation, Monte Carlo Simulation and System Dynamics has been considered in the metamodel. The following sub-clauses will include discussions of optional extensions to some elements to support simulation. These extensions are normative, but are not required for compliance with VDML, version 1.

The following sub-clauses will describe a number of aspects of modeling with VDML to help potential users understand the benefits of VDML modeling, and to help implementers understand the semantics and context of the model elements defined in the metamodel clause. Each of these sub-clauses describes the business concepts and interactions of a cluster of closely related model elements. Some elements will be mentioned in multiple clusters since these clusters are all interrelated.

7.1.1 VDML Model

A `ValueDeliveryModel` element contains, directly or indirectly, all of the elements of a particular VDML model. A VDML model may have multiple `Scenarios` where each `Scenario` has different `Measurements` associated with the model elements, representing different operating circumstances and configurations. These `Measurements` may be entered by a user, computed by the model or imported from production operations or a simulation. Different `Scenarios` can also delegate to different `Collaborations` as sub-`Collaborations` (i.e., services). A common `Scenario` defines `Measurements` that are "common" across all `Scenarios` in the model (i.e., they apply unless modified for a specific `Scenario`). A `ValueDeliveryModel` is the unit of model exchange. A model exchange between different VDML tools will include one or more `Scenarios` including the common.

7.1.2 Value and Value Proposition

The creation and exchange of value is a fundamental driver of analysis using VDML. A value is a measurable benefit delivered to a recipient in association with a business item/deliverable. The `Measurement` represents the degree to which the property is present and may be either an objective or subjective measure. A value may represent a feature that is intrinsic in the deliverable such as its composition, its performance, or its weight, or other benefits conveyed by the deliverable to the recipient such as price, a commitment to future purchases, a warranty, an environmental impact of the product or trustworthiness. Different recipients will have different opinions regarding their level of satisfaction with the particular value (different `Measurements`), but they should all agree on the operational `Measurement` of the value contribution.

A deliverable will typically convey multiple values, and an exchange may involve multiple deliverables. Together the bundle of deliverables and values will determine the level of appreciation of the recipient. A `ValueProposition` embodies the values associated with the deliverable(s) and provides a transformation from a `Measurement` of each value concerned to a level of satisfaction of that value for the particular recipient (e.g., customer or market segment). These may be combined in a weighted average to represent the overall level of satisfaction. This overall satisfaction, as well as some of the values, may be compared to the `ValueProposition` of a competitor. The `ValueProposition` provides insight for identification of values that must be added or maintained and those that are candidates to improve competitive position. Each `ValueProposition` represents the perspective of a recipient. Since recipients may not explicitly define their preferences, each `ValueProposition` may represent estimates of the recipient's satisfaction by the modeling business entity.

`ValueAdd` elements represent value properties contributed by different activities participating in the delivery of the product or service. A contribution can be positive or negative. For a product or line of business, the

ValueAdd elements for the same type of value are aggregated to determine the impact on the ValueProposition. A value that is considered below recipient expectations or market demand suggests a need for improvement. The ValueAdd elements for that value can help identify activities and supporting Capabilities that might be improved to enhance recipient value.

Different ValueProposition Scenarios may be defined for different recipients. Each may incorporate variations in production operations, different product features, or a different set of values of interest and each will have their particular satisfaction levels for the values. Recipients include different customer market segments, internal customers of services and the enterprise owners or stockholders. When a value proposition represents the interests of business leaders or stockholders, the model may represent a business future state so the recipient values can be viewed as transformation objectives.

7.1.3 Capability Definition

Business capabilities are fundamental to the delivery of a product or service. Sharing of business capabilities can be a competitive advantage for product cost or quality as well as agility of the enterprise in adapting to new technology or market opportunities. VDML provides a framework in which Capabilities can be identified and optimized from an enterprise perspective.

Deliverables such as products or services are produced by using business Capabilities to perform Activities that add value. A CapabilityOffer represents the ability of an organization to perform a particular type of work and may involve people with particular skills and knowledge, intellectual property, defined practices, operating facilities, tools and equipment. VDML provides for specification of capabilities (CapabilityDefinition) in a taxonomy (CapabilityLibrary) that specifies broad classifications, broken down into very specific Capabilities that, potentially, are applicable to multiple products or lines of business. The taxonomy is the basis for a Capability map display or a Capability "heat map" where certain Capabilities are highlighted for improvement.

A specific Capability may be performed by one Actor or a team of actors. A Capability may also rely on other supporting Capabilities to perform specific Activities. These supporting Capabilities typically are sharable in different contexts to achieve economies of scale or consistent controls for multiple uses.

The purpose of the capability taxonomy is to promote consistency in the definitions of similar Capabilities, and to provide the opportunity to recognize where the same or similar Capabilities are being performed by different organizations. This presents the opportunity for consolidation to realize economies of scale. It also provides reference to existing Capabilities (CapabilityOffers) when a capability is needed to respond to changing market demands or to develop a new product or line of business.

Capabilities are owned by organization units (OrgUnits). The OrgUnit either has or can obtain the necessary resources to deliver the Capability. This typically involves a Pool of people with the necessary skills and knowledge along with facilities, practices, tools and equipment that are needed and possibly services providing supporting Capabilities.

7.1.4 Collaboration

Collaboration is the fundamental organizational concept of VDML. A Collaboration represents the interaction of multiple Participants for a shared purpose. Each Participant is in a Role that represents that Participant's relationship to the rest of the Participants and the shared purpose. Each Participant may be an Actor (a human or automaton), or it may be another Collaboration. Participants are assigned to one or more Roles, which allows any given Participant to engage in

multiple Collaborations. Those Roles, in turn may be assigned as Participants in other Collaborations.

For example, a department in an enterprise is a collaboration. It may engage persons in Position Roles as well as groups of persons in subordinate team OrgUnit Collaborations. A Role, such as an engineer in a department, may be engaged by a committee, another Collaboration, or a specific set of Activities. So, in the committee example, a person in the engineering Role, as an engineer, also takes on the Role of representative of the department as a member of the committee.

A Role will be filled by a Participant that has the Capability required to perform the associated Activity(s). That Capability may be provided by an Actor (person or machine) or another Collaboration of Participants needed to achieve the needed Capability. The capability taxonomy (CapabilityLibrary) provides a link to a CapabilityOffer(s) for identification of an OrgUnit(s) that can provide the needed Capability.

A Collaboration may define the Activities performed by Participants in Roles. The interactions between Roles may be represented as an Activity network. While the generic Collaboration element can stand alone to represent any business collaboration or Activity network, it has been specialized to four more specific types of Collaboration: Community, BusinessNetwork, OrgUnit and CapabilityMethod. A Community is a loose association of members that share a common interest. A BusinessNetwork represents Collaboration between Parties that are typically economically independent and that participate in an exchange in the marketplace. An OrgUnit (organization) is a Collaboration that is a component of an organization structure and is responsible for defined resources. A CapabilityMethod is a Collaboration for specification of the operation of a sharable Capability (i.e., a service).

7.1.5 Community

Members of a Community are individuals or organizations that have come together due to a shared interest. This includes professional organizations, standards organizations, political action groups, voters in a particular jurisdiction, and market segments representing shared interests in certain products or services. A community may provide an opportunity to share expertise, to join together for advocacy, or to develop standard practices. Some communities are associated with a formal organization that provides business functions and support for activities of the members. The Community may be viewed as a branch of the more formal organization structure consisting of Org Units, discussed below. A Community may be formed by persons with shared interests from across a business organization to share experiences and collaborate on innovation. Roles within the Community are Member Roles.

7.1.6 Business Network

A BusinessNetwork is a Collaboration between independent economic entities, participating in an exchange of products, services and usually money. Participants may include companies, government agencies, or other institutions (all VDML Collaborations) as well as individuals (e.g., retail customers). They collaborate for their mutual benefit, providing and receiving deliverables with associated values.

These Participants play Party Roles in a BusinessNetwork. Each Participant contributes to the BusinessNetwork Collaboration because they perceive that they realize a net gain from exchanging deliverables in their Party Role. A BusinessNetwork model may be configured to engage specific Participants with known interests and capabilities. In such a model it is possible to develop more detailed representations of the operation of each Participant through their Party Roles. Alternatively, Party Roles may involve classes of Participants such as a market segment represented by a

Community where it is understood that in a specific business transaction, only one member of the Community would participate. This abstraction is essential for reducing the scale of a VDML model where an enterprise may engage thousands of similar business partners and millions of similar customers.

Each Party Role may exchange deliverables with some or all of the other Party Roles. Each of the Party Roles performs Activities that produce or consume deliverables. These Activities may delegate to other Collaborations, typically CapabilityMethods, to specify the internal operations in greater detail. Of course, for a Participant other than the enterprise being modeled, there may be no model details available to represent the internal business operation and different Parties. In addition, a BusinessNetwork may model typical exchanges between the enterprise being modeled and many other entities. For such a BusinessNetwork, certain Roles can be filled by a Community representing entities that have similar interests and participate in the BusinessNetwork in the same way with the understanding that only one Member of the Community participates in the Party Role for a particular exchange. For example, a manufacturer may engage in a BusinessNetwork with dealers. The "dealer" Party Role may be assigned to a Community of dealers representing the potential Participants.

A BusinessNetwork may be composed of more specific BusinessNetworks as where a company engages in separate but related business transactions, for example, the sales of printers and the associated sales of printer cartridges. A BusinessNetwork may also represent a very broad set of relationships that depict a target business as a member of an ecosystem. This is useful for identifying and understanding the less obvious business relationships and effects that may be factors in the success of the business including the impact on professional, environmental and social values. Such models may focus on product lifecycles rather than individual product production and sales. See an example in Figure B-1.

In a BusinessNetwork model, deliverables and associated ValuePropositions that are sent and received by each Party (Role) determine the overall value of participation for each Party and the viability of the network. The net value of an exchange can be expressed for a Party in a ValueProposition representing the business value to that Party. That net value can then be computed from the ValuePropositions sent and received. Each ValueProposition provided may be assigned an economic value that reflects the cost of the deliverable(s) with associated values, and each ValueProposition received may be assigned an economic value based on the recipient's value requirements and competitive pricing. It is at this level that the interaction of multiple lines of business may be evaluated such as in a business strategy where printers are sold at a loss to drive sales of cartridges.

7.1.7 Organization Unit (Org Unit)

VDML includes a representation of organizational structure because (1) organizations manage and maintain the implementation of business capabilities, (2) organizations are collaborations where specific people work together for a shared purpose, (3) organizations are accountable for the management and utilization of resources and the operating performance of their capabilities, and (4) organization structure is a fundamental aspect of any enterprise transformation.

An OrgUnit is the building block of organizations. For example, the top tiers of a company, a business unit, or a department are all OrgUnit Collaborations. These are persistent Collaborations that generally have specific persons in Position Roles. OrgUnit is the only Collaboration type that is an owner of resources such as people, machines, intellectual property, etc., and is thus the only Collaboration that provides sharable capabilities.

An OrgUnit has Position Roles. Position Roles are formally defined and are usually identified with the job classification of an appropriate participant. Position Roles may define the budgeted positions, so some may be vacant. The Participant in a Position may be an employee or a contractor, so a Position Role might be filled by an employee Role of the company or a contractor Role of a contract

Collaboration. From time-to-time, a Position Role may be assigned to a Performer Role of a CapabilityMethod that defines a specific Activity to be performed by the person in the Position Role. Individual assignments are not required in a VDML model (version 1), but would be modeled to support a simulation.

An OrgUnit may be modeled as having specific Activities and deliverables, but, in general, an OrgUnit will provide multiple Capabilities including internal administrative and support operations that are not part of its primary business purpose. Some of these Capabilities are provided by Actors (humans or machines) in OrgUnit Positions. Other more complex Capabilities require more specific Collaborations of some or all of the OrgUnit Participants. These Collaborations are performed as the need arises and can be defined with CapabilityMethods that specify the Activities and Roles to be filled for each application of a Capability.

Position Roles within an OrgUnit may also fill Roles in other Collaborations thus linking the OrgUnit with other Collaborations including other OrgUnits. For example, an engineer in a Position Role within an engineering group my fill a Role as a member of a technical committee that has representatives from other groups to coordinate product design efforts. A person in an OrgUnit Position Role may participate in a Performer Role such as the submitter (Role) in a purchase request Collaboration of the Purchasing department.

An OrgUnit generally exists in a hierarchical, organization structure that defines a chain of responsibility over personnel and other resources and operations and has associated financial budgets and accounts. Thus a company has divisions, the divisions have departments, the departments have groups or teams, etc. Each ad hoc committee or project team is typically formed from Positions in more persistent OrgUnits, and each has a reporting relationship to a persistent OrgUnit. OrgUnit is also used for other Collaborations that represent persistent relationships between specific Participants such as a standing review committee, a task force or a project team.

An OrgUnit will typically have a manager or leader responsible for the management of the OrgUnit and its resources and operations. The leader is a Participant in the Collaboration; however, being in a manager Position Role, that person also participates as a member of the parent OrgUnit Collaboration. So a department may have groups (Collaborations) with a manager in each group (individual) in a Position Role where that Position manager is also a Participant in another Position Role of the parent department. However, some organizations are not simple hierarchies. For example, another member of a group may be the financial analyst that also has a Position Role in the department of the chief financial officer. A project team may have Positions filled by persons in Position Roles in multiple OrgUnits—Roles in Roles.

An OrgUnit typically brings together people and other resources that work together to provide a Capability represented as a CapabilityOffer. The Capability may be as broad as the product engineering capability of a department, or may be more specific such as the engine design group. An OrgUnit will, from time to time, engage sub-groups of its members to produce specific results. These ad hoc Collaborations may be defined as CapabilityMethods, or may simply be project teams. An OrgUnit may also offer Capabilities that are not defined in further detail. This may indicate that the details are simply not modeled, or that the Capability is configured ad hoc to meet a specific requirement.

Each Capability offered by an OrgUnit is identified by a CapabilityOffer. A CapabilityOffer can identify the CapabilityMethod and/or Pool of Actors that provide the Capability as well as significant resources that are used to deliver the Capability. Positions and managed resources that support Capabilities of an OrgUnit are in Stores or Pools (discussed later) associated with the CapabilityOffers that use them.

7.1.8 Capability Method

Some activity patterns of `Collaborations` are used over and over again, producing specific deliverables and associated values. An `OrgUnit` may offer multiple `Capabilities` that each apply a different, repeatable `Activity` pattern to support each `CapabilityOffer`. Each pattern is specified with a `CapabilityMethod`. Each use of a `CapabilityMethod` will have a set of `Measurements` and `Assignments` of `Performers` for that use, specified by its `DelegationContext` (discussed later).

When a requirement for use of a `CapabilityMethod` occurs, people are assigned to certain `Roles`, they perform defined `Activities`, they use and consume resources, they create certain deliverables and they produce a result. The details of this work can be represented with an `Activity` network with `Roles`, deliverables and `ValueAdds` that are required to deliver a desired result. Any of the `Collaboration` types discussed above can represent its work with an `Activity` network, but a `CapabilityMethod` can be offered as a shared `Capability`, owned by a responsible `OrgUnit` and supported by people and resources owned by an `OrgUnit`.

A `CapabilityMethod` defines a sharable `Activity` network. This is similar to a business process definition that is used over and over to produce a result with potentially different input circumstances and Role Assignments for each occurrence. However, a `CapabilityMethod` focuses on the statistical aspects of `Activities`, the flow of deliverables and contributions of value, and thus provides a higher level of abstraction of what the business does, how it performs and who is responsible rather than the detail of the exceptions, variations and control mechanisms involved in orchestrating work on individual work products.

Consequently, an `Activity` network can also represent a case type for case management where the process is adaptive. For a case model, VDML `Activity` `Measurements` reflect the statistical occurrences of an `Activity` for cases in the same scenario. A VDML model of case management is closer to a CMMN (Case Management Model and Notation) model than a BPMN (Business Process Model and Notation) process. In a CMMN case, `Activities` are linked by dependencies, typically representing the availability of a result or a change in state of a business item.

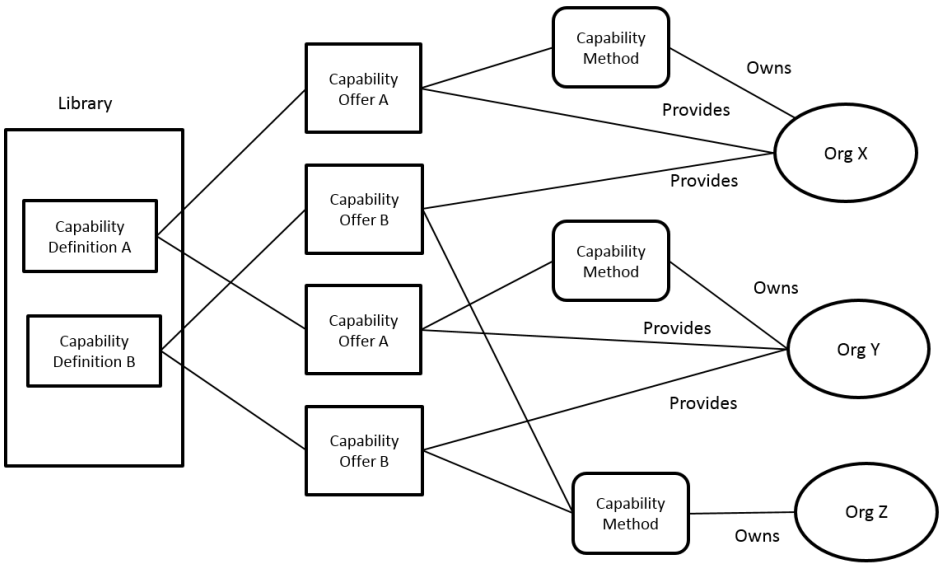

Figure 7-2: Capability Offers

A CapabilityMethod is associated with a CapabilityOffer for the OrgUnit that has resources to provide the Capability. Typically, the Roles of the CapabilityMethod will be filled by persons in Positions of the *provider* OrgUnit. The CapabilityMethod is also associated with an *owner* OrgUnit that has the responsibility and authority to develop and modify the CapabilityMethod design. This recognizes that a CapabilityMethod may be developed and maintained by one organization, and it may be used by multiple OrgUnits to provide the Capability in different contexts. This will be typical of an administrative function, as well as methods defined by a department and performed by different groups within the department.

Figure 7-2 illustrates the relationships between a CapabilityDefinition (library), CapabilityOffers, CapabilityMethods and the owner and provider OrgUnits. The graphical elements shown here are not normative (for illustration purposes only). The CapabilityLibrary has two CapabilityDefinitions, A and B. Each is offered by both OrgUnit X and OrgUnit Y as indicated by the "provides" association.

OrgUnits X and Y each have a CapabilityMethod for Capability A; consequently they each own a CapabilityMethod and provide Capability A.

OrgUnit Z owns a CapabilityMethod for Capability B that can be used (provided by) OrgUnit X or OrgUnit Y to provide Capability B. OrgUnit Z is responsible for the design of the CapabilityMethod for Capability B but not the supporting resources. OrgUnit X and OrgUnit Y are responsible for the operation of the CapabilityMethod for Capability B and the supporting resources.

The CapabilityOffers will have associated elements for resources that support the Capabilities in OrgUnit X and OrgUnit Y (not shown).

A CapabilityMethod, or any Collaboration providing a Capability, supports Measurements for a typical delivery of the Capability. However, a CapabilityMethod is designed to be shared and will be engaged by Activities in one or more Collaborations to perform supporting work.

Thus a CapabilityMethod may be engaged by an Activity of another Collaboration as a "sub-Collaboration." The OrgUnit that performs the CapabilityMethod is engaged in the Role of the delegating Activity, and it uses the CapabilityMethod to provide the desired Capability. The Role may appear in multiple Activities, so the OrgUnit must have CapabiityOffers that support the requirements of each of those Activities.

7.1.9 Activity

Activities define work to be done by Participants in Roles within a Collaboration. A collaboration has a set of roles to which participants are assigned. Each Activity is performed by one Role within the Collaboration.

Within a Collaboration, the same Role may perform multiple Activities and may provide different Capabilities for each of the Activities.

An Activity identifies the type of Capability required to perform the Activity by reference to a CapabilityDefinition, and the Role to be filled by a Participant that provides that Capability.

In some cases the Participant will be an Actor (human or automaton); in other cases it will be filled by a Collaboration (usually an OrgUnit using a CapabilityMethod). Note that if a Role performs multiple Activities, and the Activities require different Capabilities, then the selected Participant must be capable of providing each of the Capabilities required.

Figure 7-3 depicts a number of elements related to an Activity. In the example, the Performer Role is filled by a Position Role (a Position in an OrgUnit), indicating that it is filled by a Participant in a Position, typically an Actor. (This diagram is for illustration only and is not intended to be normative.)

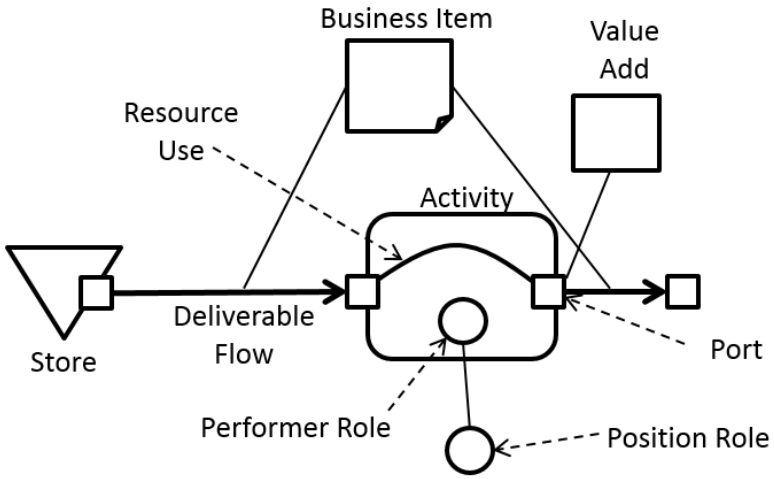

Figure 7-3: Activity structure

An Activity consumes and produces BusinessItems as deliverables. A BusinessItem is anything that can be acquired or created, that conveys information, obligation or other forms of value. For example, it includes parts, products, units of fluids, orders, emails, notices, contracts, currency, assignments, devices, property and other resources that can be conveyed from a provider to a recipient.

A BusinessItem flows between Activities and Stores, and it may flow through a delegation to a sub-Collaboration.

Flow of BusinessItems into and out of Activities as well as Stores is depicted by DeliverableFlows. Flow from one Activity to another indicates that the receiving Activity requires the BusinessItem as input. The value contributions of each Activity are represented as ValueAdds. As long as the input BusinessItem is essentially the same thing as the output, the VDML BusinessItem element remains the same, occurring in multiple DeliverableFlows.

An Activity can have DeliverableFlows from one Activity or Store and to another Activity or Store (but not from a Store to a Store). In the figure, the BusinessItem is on both the input and output DeliverableFlows, indicating that the input and output are essentially the same except the output has some added value. This could be a part going through progressive stages of production. The ValueAdded to the BusinessItem is associated with the same OutputPort as the DeliverableFlow of the BusinessItem. The ResourceUse, internal to the Activity, connects to InputPorts and OutputPorts of the Activity to indicate the resources used by a particular output (there could be additional InputPorts and OutputPorts). This use of graphical elements is for illustration only and is not normative.

An Activity is expected to contribute to one or more values to its deliverable(s) and thus to the overall result of the Collaboration (although not all values may be represented in a particular model). These contributions are represented by ValueAdd elements. Value contributions are associated with Measurements that have a positive or negative impact on the market value or desirability of the end product or service. ValueDefinitions are captured in a library (ValueLibrary), as are Characteristics

(SMM Library). Types of value can include product features and qualities as well as Activity performance characteristics such as duration, cost and defect rate that affect recipient (customer) satisfaction. Specific Measurements of interest are at the discretion of the modeler and are generally determined by the ValueProposition expected by the recipient/customer.

Together, the Roles of a Collaboration and their Activities produce the desired result and associated values. The ValueAdds of a Collaboration may be summarized for their impact on a unit of production of the end product or service. Consequently, the Measurements associated with an Activity and its ValueAdds are based on one unit of production or can be expressed in a way that is meaningful for considering the value of the end product or service such as cost per unit. However, the Measurements are averages or statistical measures reflecting variances over some time period rather than the Measurements of one, selected unit of production. Where there is variability in the result, such as different configurations of automobiles on a production line, the Measurements will represent a particular product mix with statistical variance. Where it is important to represent different or more specific product mixes, the same Collaboration can be used to represent different Scenarios. Scenarios are discussed in more detail, below.

The flows of a VDML Activity network are always directed toward completion, so all ValueAdds contribute to the end result. Conversely, all values can be traced back to their contributors. This is possible, because VDML is not representing the actual paths of each unit of production, but rather the statistical use of various Activities that contribute to results achieved over some period of time.

The set of results may include contributions of some Activities that are only active for some units of production due to product features, operating exceptions, defects, repairs, sample testing, machine failures, and so on. The resultant Measurements represent a typical unit of production, but may include statistical Measurements of variance.

Repairs and rework may be a particular concern for large, complex products. A modeler may choose to represent Activities for repair and rework and divert some percentage of production along this path. Conversely, the modeler may simply add a rework factor into each Activity that will be applied to adjust operating Measurements to reflect this additional work. Scrap Activities might receive some percentage of production and direct it to a salvage operation, so the cost of production goes into the end product less the recovery from salvage.

Time-to-delivery may be an important measure for customer satisfaction. This may be simply the sum of durations of Activities or Stores acting as buffers to determine the total time for delivery. However, the production process may be paced as with a production line, so that while some Activities are shorter in duration, they do not save time for delivery. To express this, an Activity, typically an initial Activity, may specify a recurrence interval that may affect the duration and operating efficiency of subsequent Activities.

Note that various factors such as product mix, rework and recurrence intervals may be applied to the same model in the context of different Scenarios so that the effects can be compared and alternatives can be evaluated.

7.1.10 Port

Ports are the connection points for inputs and outputs to Collaborations, Activities and Stores (specializations of PortContainer). DeliverableFlows link Ports between PortContainers. ValueAdds are associated with OutputPorts. OutputPorts define the association of DeliverableFlows to ValueAdds, distinguishing between the ValueAdds associated with different BusinessItems that are output.

Ports also provide for the association of InputPorts to OutputPorts within an Activity through association with a ResourceUse. A ResourceUse may specify characteristics of use of resources from an InputPort such as how much resource is used, and/or how long a resource is in use. ResourceUse may also clarify dependencies between inputs (resources) and outputs (deliverables) that may affect costs and durations associated with the Activity.

A ResourceUse can also define the allocation of DeliverableFlows among alternative Ports. So where inputs to an Activity come from multiple, alternative DeliverableFlows, a ResourceUse will determine the choice of alternative. This may be expressed as a preference, or as percentage allocation, or for a simulation, this could apply selection criteria to individual transactions.

7.1.11 Resources and Stores

A resource is something that is used or consumed by an Activity to deliver its value. This includes parts, intellectual property, energy, a person, knowledge assets, a machine, a tool, and so on. The VDML user should not attempt to represent all resources used by all Activities, but should focus on those resources that are important to support a high-level analysis. So in most cases, production facilities, small parts, supplies, heat, light and power will not be of concern. The cost of such resources can be included in Capability overhead or additional Activity operating cost if they represent significant factors. However, parts of high value or machines that must be scheduled probably are of sufficient interest to be modeled.

Resources are held in a Store and are delivered as BusinessItems over a DeliverableFlow from the Store to a target Activity. A resource may be replenished by DeliverableFlow of a BusinessItem from a source Activity. An Activity may forward a resource as a Business Item with a subsequent DeliverableFlow or delegate to a sub-Collaboration.

A resource may be consumable or reusable. A consumable resource is no longer available after it is consumed by an Activity; a reusable resource is used by one or more Activities and then returned to the Pool (a specialization of Store). A resource also may be fungible or non-fungible. A non-fungible resource is uniquely identifiable and cannot be replaced by another resource of the same type. Fasteners and other interchangeable parts are fungible. For example, an engine configured for a particular automobile assembly is not fungible.

A Store may hold non-fungible resources to be matched to corresponding business items. For example, a Store may hold engines and match them to the corresponding automobiles. In VDML (without simulation) most individual business items are represented as typical of the model BusinessItem, so a Store of non-fungible resources can specify an average Store size or holding period to represent the effect of matching to corresponding BusinessItems. In a simulation implementation, the Store would match individual, incoming BusinessItems.

Reusable resources are managed by a Pool and are assigned for use. A Pool is a specialization of Store that can track the availability and assignments of individual resources. Reusable resources are associated with a responsible OrgUnit and individual resources may be assigned to Pools based on their Capabilities. A Pool tracks the total number of resources for its Capability as well as the number that are available for use. The modeler may explicitly represent the individual BusinessItems for each resource, or just capture the Pool size and average number available, depending on the level of detail required.

A CapabilityOffer can identify the Pool of an OrgUnit that provides reusable resources for that Capability. A Pool is owned by the associated OrgUnit that is accountable for managing the Pool. A reusable resource, represented by a BusinessItem, will be used by an Activity for some duration, and may be passed to one or more subsequent Activities before being returned. The cumulative duration of these uses will determine the consumption of available resource time.

This, along with the rate of production, will determine if the Pool of resources will always have resources available or will introduce some additional wait-time for assignment or release of a resource.

In simulation, the availability of each, reusable resource must be managed by a Pool as a BusinessItem. Each resource may have scheduled times when it is unavailable. An individual resource may also be available to perform one of multiple Capabilities and thus may be assigned to multiple Pools. When a resource is assigned for use, it is unavailable until returned to the Pool. The availability must be associated with the specific resource since a resource could provide multiple Capabilities competing for assignments. Availability of a resource may also be controlled based on a CalendarService. This allows availability to reflect scheduled work time or other factors that may remove a resource from availability. The CalendarService as well as attributes and relationships of the Pool and the BusinessItems required to support Discrete Event Simulation are optional in VDML version 1.

7.1.12 Measures

VDML incorporates SMM (Structured Metrics Metamodel) in order to leverage the SMM metamodel and support the use of SMM Libraries to define Measures to be applied to VDML model elements. In SMM, a Measure is a method that is applied to characterize an attribute of something by assigning a comparable quantification or qualification. A Measure is applied to a Characteristic, such as weight of a part, to determine a Measurement that expresses the value of the MeasuredCharacteristic for a particular VDML model element.

SMM also provides for different Measurements to be expressed for the same thing in the context of different Observations. In VDML, an Observation is associated with an AnalysisContext. Thus a VDML element may have different Measurements for the same MeasuredCharacteristic in different AnalysisContexts.

VDML MeasuredCharacteristics reflect statistical Measurements per unit of production. A VDML model may include Collaborations having different units of production. For example, an automobile may be the unit of production for a final assembly line, but for a tire manufacturing operation one tire is the unit of production, and five tires are needed for one automobile. On the other hand, a product design is the unit of production of the Collaboration that produces a design for production of many automobiles. These Collaborations are related, but their units of production are different, so where these Collaborations interact, there must be appropriate adjustments for the ValueAdd Measurements to produce consistent results.

For example, a tire production Collaboration will yield a cost per tire. The automobile production cost must incorporate the cost of 5 tires. From a product lifecycle perspective, the product costs must include the cost of product development and design revisions (additional design Collaborations) (along with other sales and marketing costs) prorated over the expected production for the lifecycle of the product.

7.1.13 Scenarios and contexts

In VDML, the operation of the business is represented by the interaction of multiple Collaborations. The Role that performs an Activity in a Collaboration can be assigned to a sub-Collaboration to provide the desired Capability. When the sub-Collaboration is an OrgUnit, the OrgUnit can apply a CapabilityMethod identified by the OrgUnit's CapabilityOffer to deliver the Capability.

Whenever an Activity of a Collaboration delegates to another Collaboration in order to engage a shared Capability, that particular use of the sub-Collaboration may be one of many. For each use of the Collaboration, there will be different Measurements of performance, depending on the particular circumstances of that use. SMM supports the capture of multiple sets of Measurements by associating these with different Observations. VDML uses this SMM facility by defining each use of a Collaboration

as an AnalysisContext where each AnalysisContext is associated with an SMM Observation. In VDML, a root AnalysisContext for a VDML model is specialized as a Scenario, and a context for a delegation is specialized as a DelegationContext.

A DelegationContext defines aspects of the particular delegation to the sub-Collaboration referenced as the contextCollaboration. When an OrgUnit uses a CapabilityMethod, the OrgUnit is assigned to the Role that performs the delegating Activity and the CapabilityMethod is linked to the DelegationContext as the contextCollaboration—the context defined for that application of the CapabilityMethod. If there is no CapabilityMethod, or the Role is not assigned to an OrgUnit, then the sub-Collaboration is both in the Role, and in the DelegationContext (through contextCollaboration).

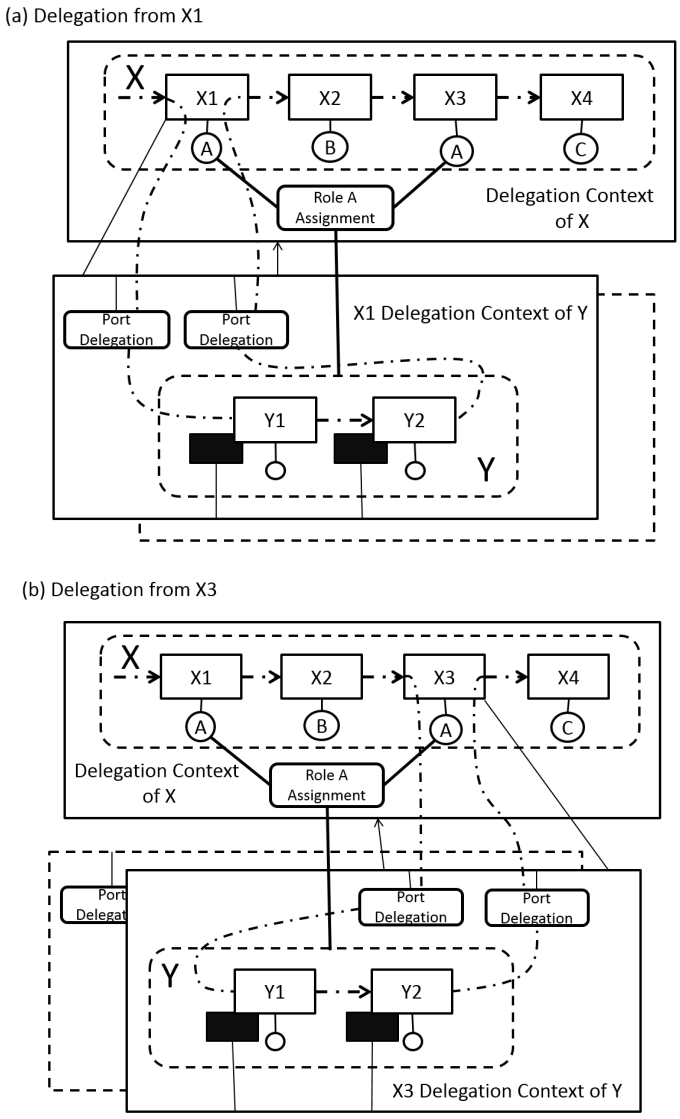

Figure 7-4: Two uses of a collaboration

An AnalysisContext may also define Role Assignments as context-dependent. Thus the Role Assignments in one occurrence of a Collaboration may be different from the Role Assignments in another occurrence of the same Collaboration. This is particularly important when the Participant in

a Role is identified by an input to the Collaboration. In the alternative, a Role Assignment is associated with the Collaboration (the owner of the Role), and the Assignment is context-independent.

The InputPorts and OutputPorts of the delegating Activity are linked to InputPorts and OutputPorts of the sub-Collaboration (e.g., CapabilityMethod) using InputDelegations and OutputDelegations (which are context-based). Consequently, the ValueAdd elements leading into the calling Activity are input to the sub-Collaboration, and the ValueAdd elements of the sub-Collaboration become outputs of the calling Activity.

Figure 7-4 illustrates two uses of the Y Collaboration within one use of the X Collaboration. This diagram is not normative—for illustration only. These are all in the context of one Scenario (not shown). Collaboration Y is assigned Role A in the depicted DelegationContext of Collaboration X. Diagram (a) illustrates the DelegationContext of Y when used in Activity X1; Diagram (b) illustrates the DelegationContext of Y when used in Activity X3. Each DelegationContext identifies the SMM Observation (not shown) and the DelegationContexts for Y identify the PortDelegation and Measurement elements associated with that delegation. The illustration shows one input and one output linked through PortDelegations. The dark rectangles associated with Y1 and Y2 represent Measurements, one for each use of each Activity of the Y Collaboration. ValueAdd elements are not shown.

VDML takes this concept a step further to allow an entire VDML model to have multiple Scenarios, different sets of Measurements to reflect different operating situations. For example, one Scenario might represent Measurements for a particular product mix, and another Scenario could represent the Measurements for an alternative product mix. Different Scenarios could be used to evaluate differences in the use of different operating assumptions or sub-Collaborations, or for current and future configurations and Measurements.

A Scenario element represents the root AnalysisContext for the set of Measurements for a particular situation. The Scenario and the DelegationContexts of that Scenario form an AnalysisContext tree of sub-Collaborations engaged in that Scenario as depicted in Figure 7-5 (non-normative graphic).

The delegations of one Scenario can differ from those of another Scenario thus forming a different tree.

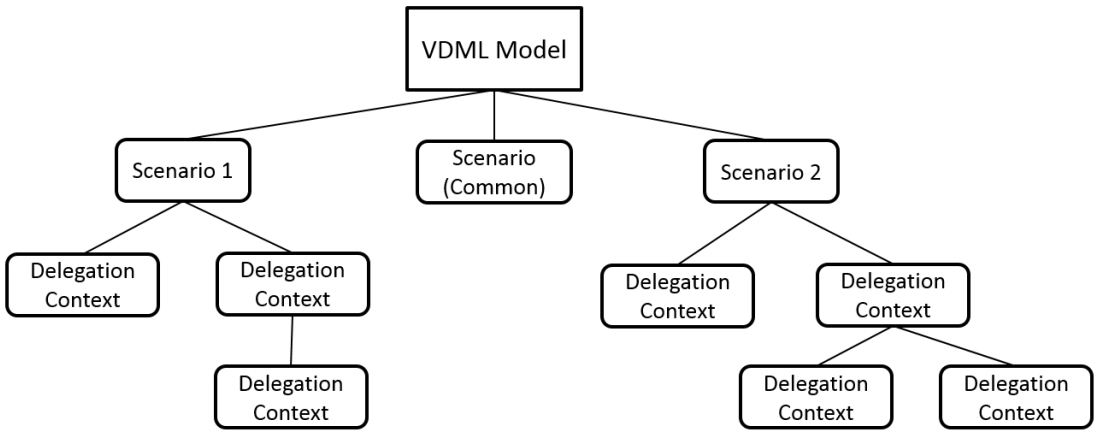

Figure 7-5: Scenarios and context trees

Collaborations and other elements that do not occur as sub-Collaborations in the Scenario are

included in the root, Scenario context. The root Scenario context includes Stores that occur in DeliverableFlows that cross from one Collaboration to another (crossing DelegationContexts), as when the product of a Collaboration is delivered as an input to another Collaboration.

Any Scenario in a VDML model may involve delegation (engagement of sub-Collaborations), apart from the "common" Scenario, which only serves to provide Measurements that are "common" across all Scenarios in that VDML model.

7.1.14 Staff Collaborations

VDML models will tend to have a primary focus on the “line” or “value stream” operations that contribute directly to the delivery of a product or service and associated values to an end customer. However, the business also requires “staff” operations that maintain and change the line operations. These may be generally classified as support Capabilities, consistent with the value chain modeling concepts of Porter (1985). They produce value for internal customers. This includes the staff Activities in an organization that maintain the day-to-day operating Capabilities as well as those that support capability transformation initiatives.

VDML cannot model the transition of Collaborations and Activities to accomplish a business transformation. However, in addition to the mainstream business, (1) it can model the consequences of transformation (by comparing the “current state” model to the “future state” model), and (2) it can model the work of day-to-day maintenance and support that keeps the business running and sustains efficiencies.

The work of maintenance and support involves a Collaboration and sub-Collaborations with Actors contributing to repairs and preventive maintenance. This is no different from other Collaborations except that elements of the mainstream business model are the subject matter of the work. The elements affected will be referenced by BusinessItems in the DeliverableFlows.

The current business model and the future business model of a transformation can be modeled as different Scenarios of the same VDML model if the future structure is not significantly different. Scenarios can also be used to represent stages of progression of the transformation to consider the transitional impact on the continued operation of the business.

The day-to-day staff operations to maintain and support the operation of the business are more closely linked to the line operations. Maintenance and repair of equipment is a primary example. The monitoring of equipment, preventive maintenance, and timely response to failure will have a significant impact on line operations. The deliverables are operational machines, but the values realized are reduced downtime and possibly reduced production defects. Consequently, production performance Measurements of the operational Capability will reflect the values delivered by the machine maintenance and repair Capability. The unit of production of machine maintenance will not be the same as the unit of production of the primary operations, so values of the machine maintenance Activities will need to be adjusted when applied to the primary value stream.

Other Capabilities that may not be considered production operations should be considered as part of a product lifecycle. These may include product engineering, purchasing, product distribution and field support as well as marketing and sales. These may have different units of production, but they can affect product values. These aspects of the enterprise are less likely to be modeled or integrated into a model in the short term, but they can certainly be modeled as separate value streams serving internal customers.

7.1.15 Model Integration

VDML may be used to model different aspects of an enterprise. For example, a VDML model might be developed for each line of business. Another model might be developed for product engineering, or field support. At some point it may be desirable to integrate these models for a more comprehensive representation

of how the business works. Some of the products may be bundled for sale so the values will be merged. Some operations, such as product engineering, will contribute values that are not provided by production operations such as ease-of-use and mean-time-to-failure. Integration of models supports better understanding of the interactions of these different business aspects.

The most straight-forward integration will occur in a `BusinessNetwork` between autonomous `Parties`—`Participants` that assume `Roles` in the `BusinessNetwork`. This integration is defined in terms of the `DeliverableFlows` between the `Parties`. Integration of two lines of business may occur in a `Collaboration` between the line-of-business value streams that brings together the `BusinessItems` and `ValueAdds` of both and reconciles unit-of-production differences to provide an integrated `ValueProposition`. This integration might be viewed as providing a VDML model that contains a `Scenario` with a `BusinessNetwork` as its `contextCollaboration`, which `BusinessNetwork` contains `Activities` that delegate (via `delegationContexts` in that `Scenario`) to the line-of-business `Collaborations`, which may be contained in separate VDML models.

More often, integration will involve the integration of `Capabilities` of one model into a scenario of the other model—bringing in `CapabilityMethods`, `Stores` and `Pools` and, possibly, other `Collaborations`. Each model may have its own `CapabilityLibrary`, `BusinessItemLibrary`, `ValueLibrary`, `RoleLibrary` and organization structure model that may or may not be consistent with the other model(s). Integration will involve reconciliation of the following points of intersection:

1) Reconciliation of libraries: `CapabilityLibrary`, `BusinessItemLibrary`, `ValueLibrary` and `RoleLibrary`. Each of these must be a superset of the corresponding libraries of the models being integrated. Duplicate names and names of duplicated definitions may need to be reconciled.
2) Reconciliation of the organization structures as a superset of the `OrgUnits` in the two models with overlaps removed. Overlapping `OrgUnits` may have duplicated `CapabilityOffers` so duplicated `CapabilityMethods`, `Stores` and `Pools` will need to be removed. References to names removed for duplicated elements will need to be reconciled.
3) A branch of a delegation tree in one `Scenario` may then be linked as an activity engaged by delegation from an `Activity` in the other `Scenario`. This requires that delegated `DeliverableFlows`, `BusinessItems` and `ValueAdds` be reconciled as with any delegation. This will likely affect the `Measurements` of both the delegated branch and the receiving `Scenario`, since there will be a propagation of effect of changes in `DeliverableFlows` in both input flows and output flows. The branch is then engaged in different contexts in each scenario and it will have measurements associated with each scenario.
4) If a `Collaboration` in one model produces an input to a `Collaboration` in the other model through a shared `Store` (i.e., a side effect `DeliverableFlow`) then these inputs and outputs also must be reconciled.

These linkages can be facilitated by the modeling environment implementation. The user should be able to (1) select a `Scenario` branch—the `Collaboration` of interest and all of the delegations below it, and (2) identify the `DelegationContext` and `Activity` of the receiving `Scenario` that will delegate to the selected branch. The implementation should then identify and facilitate the delegation bindings (`PortDelegations`) as for any sub-`Collaboration`, and identify the inputs and outputs through stores to be reconciled between the new branch and the parent model.

7.2 VDML Class Definitions

This sub-clause defines the details of each of the VDML metamodel classes under the following topics:

- Collaboration and value creation. It defines the concepts associated with `Collaboration` including `Activity` networks and `deliverables` along with the contributions of `Activities` and aggregation in `ValueProposition`.
- Collaboration sub-types. It defines the four sub-types of `Collaboration`: `OrgUnit`, `BusinessNetwork`, `Community` and `CapabilityMethod`.
- Models and scenarios. It defines the scope of a `ValueDeliveryModel` (VDML model) and the representation of `Scenarios` within a model. A `Scenario` is a set of `Measurements` associated with elements of a `ValueDeliveryModel` in a particular situation; a model may have multiple `Scenarios`. Within a `Scenario`, a `Collaboration` may be engaged in different `AnalysisContexts` and thus have a set of `Measurements` associated with each context.
- Core elements. It defines elements that represent primitive concepts in a `ValueDeliveryModel`.
- Libraries. Libraries are collections of business concept specifications. Libraries may be shared across multiple `ValueDeliveryModels`.
- Integration with SMM. It defines elements that represent primitive `Measurements` for values, performance characteristics and other measurable aspects. `Measurements` are defined by the SMM specification.

7.2.1 Collaboration and Value Creation

`Collaborations` define the fundamental structure of a `ValueDeliveryModel`. `Collaborations` involve `Participants` in `Roles` working together to perform `Activities`. `Participants` may be individuals or other `Collaborations`. Their `Activities` receive and produce `deliverables` for which they may contribute values. Values are aggregated and embodied by `ValuePropositions` for `recipients` of the end products.
This sub-clause covers the following:

- `Collaborations` and `Participants` describe the basic elements and relationships of `Collaborations`.
- `Activity` networks describe the elements and relationships of `Activities` and `Stores` that occur within `Collaborations` including the `Assignment` of `Roles` and the flow of `BusinessItems`.
- Values and `ValuePropositions` describe the contributions of `ValueAdds` by `Activities` and their aggregation in `ValuePropositions` that address `recipients` of end product.

7.2.1.1 Collaborations and participants

Figure 7-6 describes the core structure of a `ValueDeliveryModel`. `Collaborations` bring together `Participants` in assigned `Roles` to perform `Activities`. `Participants` may be people or organizations or other `Roles` of people or organizations. So a person in a manager `Role` in one department (`Collaboration`) may, as manager, participate as a member (`Role`) in a planning committee (`Collaboration`) —a `Role` in a `Role`.

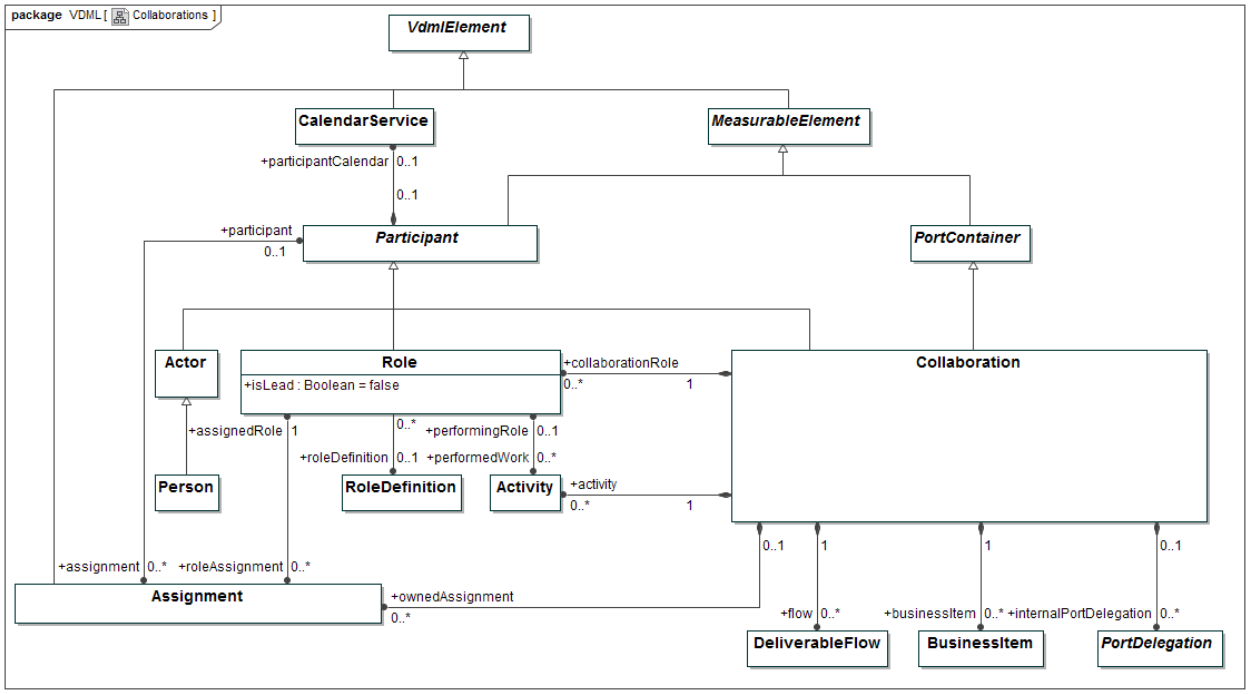

Figure 7-6: Collaborations

7.2.1.1.1 Actor Class

An individual (indivisible) `Participant`, which might be human (a `Person`) or non human (e.g., a software agent or machine).

SuperClass
`Participant`

Property	Description
scenario: Scenario [0..*]	`Scenarios` that refer to the `Actor` as a `contextCollaboration` (see 7.2.3.2.2).

7.2.1.1.2 Person Class

A human `Actor`.

SuperClass
`Actor`

7.2.1.1.3 Collaboration Class

Collection of `Participants` joined together for a shared purpose or interest.

`Participants` are assigned `Roles` that are specific to and contained in the `Collaboration`. A `Participant` might be named in the model (as instance of the class `Participant`), or might be dynamically determined from `roleResource` (see 7.2.1.2.3).

VDML distinguishes `OrgUnits`, `Communities`, `BusinessNetworks` and `CapabilityMethods` as sub-types of `Collaboration`.

SuperClass
`PortContainer`

Property	Description
collaborationRole: Role [0..*]	Roles specific to and contained in the Collaboration.
ownedAssignment: Assignment [0..*]	Assignment of Collaboration Roles to Participants. These Assignments are specific to and contained in the Collaboration, i.e., they are not context-dependent.
activity: Activity [0..*]	Activities that are contained in the Collaboration and performed by Roles in the Collaboration.
flow: DeliverableFlow [0..*]	DeliverableFlows as contained in the Collaboration.
businessItem: BusinessItem [0..*]	BusinessItems as contained in the Collaboration.
internalPortDelegation: PortDelegation [0..*]	Delegations of Ports of the Collaboration to Ports of PortContainers (Activities or Stores) inside the Collaboration. This enables that the internal structure of a Collaboration need not be visible to the Activity that delegates its work to the Collaboration.
delegationContext: DelegationContext [0..*]	DelegationContexts that refer to the Collaboration as their contextCollaboration (see 7.2.3.2.3).
scenario: Scenario [0..*]	Scenarios that refer to the Collaboration as a contextCollaboration (see 7.2.3.2.2).

Constraints

- PortDelegations that are owned by a Collaboration MUST delegate Ports of the Collaboration (as PortContainer) to Ports of Activities that are contained in the Collaboration.

7.2.1.1.4 Participant Class (Abstract)

Anyone or anything that can fill a Role in a Collaboration. Participants can be Actors (human or automatons) or Collaborations or Roles of Actors or Collaborations.

SuperClass
MeasurableElement

Property	Description
assignment: Assignment [0..*]	Assignments of Roles to the Participant.
participantCalendar: CalendarService [0..1]	Calendar that determines the availability of the Participant, to perform work. When a Participant is assigned to a Role, and that Role has a calendarService, the Participant's calendarService overrides the Role's calendarService. If a Participant is a Role and the Collaboration that contains the Role has a calendarService, the Participant's calendarService overrides the Collaboration's calendarService.

7.2.1.1.5 Role Class

A Role is an expected behavior pattern or Capability profile associated with participation in a Collaboration.

SuperClass
Participant

Property	Description
isLead: Boolean = false	Indicates, if "true," whether the Role is a leader in the Collaboration.
roleDefinition: RoleDefinition [0..1]	Association to a RoleDefinition, as contained in a RoleLibrary that is applied to enforce consistency in the definition of Roles. Multiple Roles that are associated with the same RoleDefinition, are considered similar from the perspective of the library.
roleAssignment: Assignment [0..*]	Assignments that assign the Role to Participants.
performedWork: Activity [0..*]	Activities that are performed by the Role.
port: Port [0..*]	Ports for which the Role is responsible to handle the inputs or outputs that they represent.
providedProposition: ValueProposition [0..*]	ValuePropositions for which the Role is the provider (see 7.2.1.3.1).
receivedProposition: ValueProposition [0..*]	ValuePropositions for which the Role is the recipient (see 7.2.1.3.1).

Constraints

- If an Assignment, that assigns a Role to a participant, is contained in a Collaboration, the Collaboration MUST also contain the Role that is assigned.
- An InputPort that provides the roleResource to which a Role is assigned MUST be contained by an Activity that is performed by that Role.
- Ports to which a Role refers (via port), MUST be Ports of Stores.

7.2.1.1.6 CalendarService Class

A CalendarService can be used to determine resource availability with more precision. This is relevant for simulation. Specification of calendar structure itself is not in scope of VDML.

SuperClass
VdmlElement

7.2.1.2 Activity networks

Activities define the work of Roles of Participants in a Collaboration. They are linked in networks by DeliverableFlows through which they receive and produce or modify BusinessItems (inputs and outputs). A network may also receive BusinessItems from Stores or deliver them to Stores where the Resources may be held. DeliverableFlows are connected to Activities and Stores through InputPorts and OutputPorts.

An Activity may identify a required capability by reference to a CapabilityDefinition and engage an organization (OrgUnit) that provides that capability by selecting from organizations that provide a CapabilityOffer for the required capability.

Figure 7-7 focuses on Activities and associated elements.

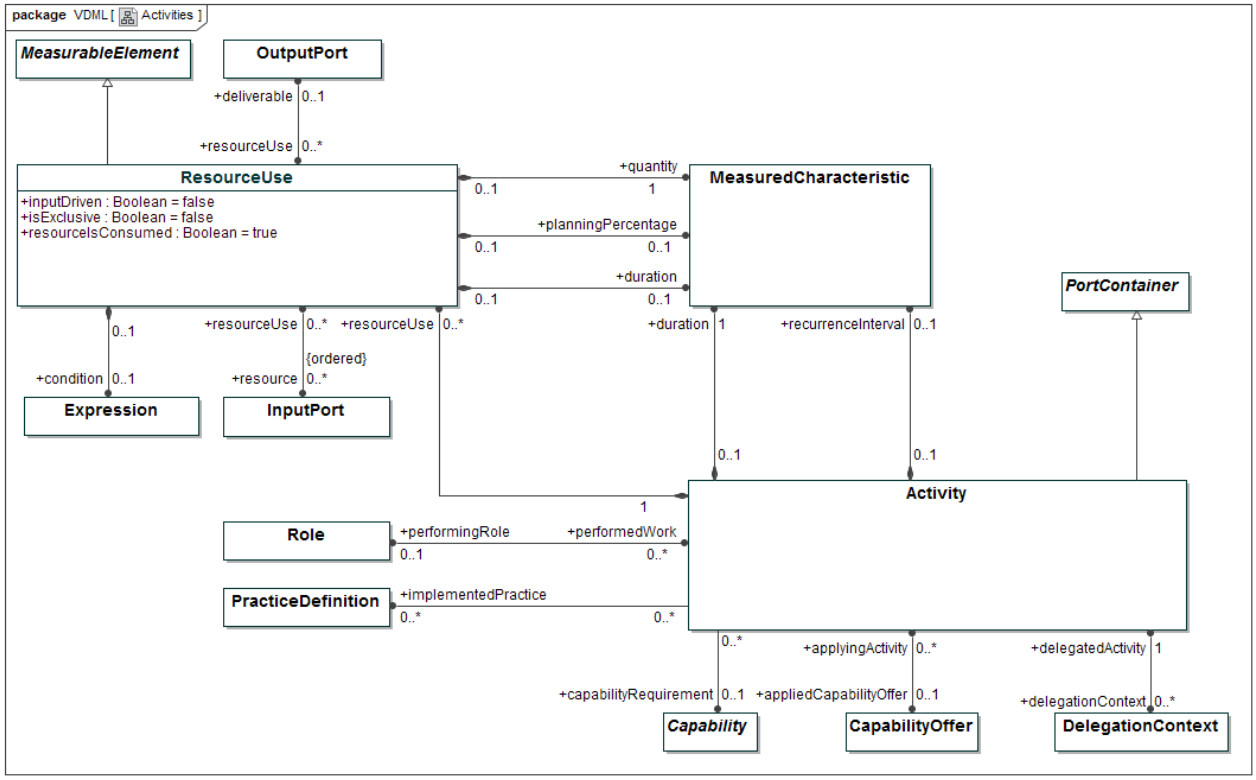

Figure 7-7: Activities

7.2.1.2.1 Activity Class

Work contributed to a Collaboration by a Participant in a Role of the Collaboration. A Role may be filled by another Collaboration and a Role may contribute to multiple Activities in the same Collaboration. The Participant in the Role might be named in the model (as instance of the class Participant), or might be dynamically determined from roleResource (see 7.2.1.2.3).

SuperClass
PortContainer

Property	Description
performingRole: Role [0..1]	The Role in the Collaboration that performs the Activity.
capabilityRequirement: Capability [0..1]	The Capability that is required by the Activity to perform its work, and which is defined via association to a Capability, as contained in a CapabilityLibrary that is applied to enforce consistency in the definition of Capabilities.
appliedCapabilityOffer: CapabilityOffer [0..1]	The CapabilityOffer that is applied to perform the work. It matches the Capability that is required by the Activity. When more than one CapabilityOffer is available in the business, that matches this Capability, a choice has to be made, and is persisted in the model via this property.
delegationContext: DelegationContext [0..1]	The AnalysisContext, as set by the Activity,

	in which a Collaboration (typically a CapabilityMethod) is analyzed, to which the Activity delegates its work (i.e., that the Activity uses as sub-Collaboration) .
resourceUse: ResourceUse [0..*]	Specifications of the use or consumption of a resource, received as input, by the Activity.
implementedPractice: PracticeDefinition [0..*]	Indications of which practices are implemented by means of the Activity, via association to a PracticeDefinition, as contained in a PracticeLibrary, that is applied to enforce consistency in the definition of practices. The same practices might also require other Activities to implement them.
duration: MeasuredCharacteristic [1]	The average duration of an Activity.
recurrenceInterval: MeasuredCharacteristic [0..1]	The time interval between two successive recurrences of the Activity. As any MeasuredCharacteristic, it can be associated with a Measurement that can be stochastically determined, which is also useful in e.g., Discrete Event Simulation. The interval can also be considered equivalent to "takt time" or "cadence time" in Lean Value Stream Maps (see Rother et al. (1998)). The Activity that has the recurrenceInterval (i.e., the scheduled Activity) maybe called the "pacemaker" in Lean Value Stream Maps.

Constraints

- The Role that performs an Activity MUST be contained in the Collaboration that also contains the Activity.
- The durations of each of the ResourceUses of an Activity MUST NOT be longer than the duration of the Activity.
- The offset of Ports of the Activity MUST NOT be longer than the duration of the Activity.
- For each Port of an Activity, the sum of its offset and the longest duration of a ResourceUse that relates to it, MUST NOT be longer than the duration of the Activity.
- An Activity MUST NOT have a recurrenceInterval, when it receives inputs other than from Stores.
- ResourceUse of an Activity MUST relate to Ports of that Activity.
- When an Activity contains more than one InputPort to receive roleResource, i.e., InputPorts to which the Role is assigned that performs the Activity, the Activity MUST have a ResourceUse that relates to the set of these InputPorts (actually making them alternatives to each other), or each DelegationContext in which the Activity-containing Collaboration is used, MUST specify a contextBasedAssignment (see 7.2.3.2.1) that assigns the Role to the roleResource that is received on one of these InputPorts.
- When an Activity delegates to a CapabilityMethod (via delegationContext), and it has its appliedCapabilityOffer defined, the appliedCapabilityOffer of the Activity MUST be a supportedCapability of the CapabilityMethod.

7.2.1.2.2 ResourceUse Class

Specifies the use or consumption of a resource within an Activity, to which the resource serves as

input. This may involve the specification of how much resource is used, the duration during which it is used, as well as, possibly other Measurements. A ResourceUse may also specify alternative sources for a resource. In VDML a resource is considered anything that is "used" or "consumed" in the production of a deliverable.

SuperClass
MeasurableElement

Property	Description
resource: InputPort [0..*] {ordered}	The resource for which the use is specified. When more than one resource is specified, these resources serve as alternatives to each other. Preferences within the set of alternatives are implied by the ordering of the set of resources.
resourceIsConsumed: Boolean = true	Specifies whether the resource is consumed, or whether it is re-usable after Activity completion.
isExclusive: Boolean = false	Specifies whether more than one resource can be used from a set of alternative resources. If "true" only a one resource can be used (typically the one with highest preference). If "false", more than one resource can be used from a set of alternative resources.
quantity: MeasuredCharacteristic [1]	The quantity of the resource that is required to perform the Activity.
deliverable: OutputPort [0..1]	ResourceUse might be dependent on an output of the Activity. If specified, the quantity of the ResourceUse specifies how much resource is required per unit of the deliverable. Example: An Activity that assembles a car requires four wheels. Each car (deliverable) requires four wheels (resource).
inputDriven: Boolean = false	If "false" (the default situation) the ResourceUse quantity specifies how much input (i.e., resource) is required, possibly dependent on an output (i.e., deliverable), and instances of the Activity, that use resource from the Store, can only be created based on other inputs (e.g., a sales order) than the resource that is related to the ResourceUse (the input can be said to be "pulled" by the Activity). If "true", the quantity specifies how much output results from processing an input (e.g., in de-assembly), and an Activity instance will be created as soon as resource is available in the Store that provides the input (the input can be said to "push" Activity).
condition: Expression [0..1]	Specifies the condition under which the resource is used.
planningPercentage: MeasuredCharacteristic [0..1]	Specifies probability of use of the resource.
duration: MeasuredCharacteristic [0..1]	The average duration of use of a re-usable resource, or, when the resource is consumed, the average time it takes to produce a deliverable from the resource, by the Activity.

Constraints

- A ResourceUse MUST NOT have a duration when the resource is consumed (i.e., resourceIsConsumed = true), unless the deliverable is also specified for the ResourceUse.
- The unit of the Measure (as specified by SMM) that determines the Measurement of planningPercentage, MUST be "percent".
- ResourceUse MUST NOT be defined in relation to resources that are not received from Stores.

7.2.1.2.3 Assignment Class

Figure 7-8 defines the association of a Participant to an Activity through an Assignment. The Participant may be determined through receipt of a BusinessItem that identifies a Participant. Assignments are context-dependent, as will be described in the sub-clause on Scenarios and contexts.

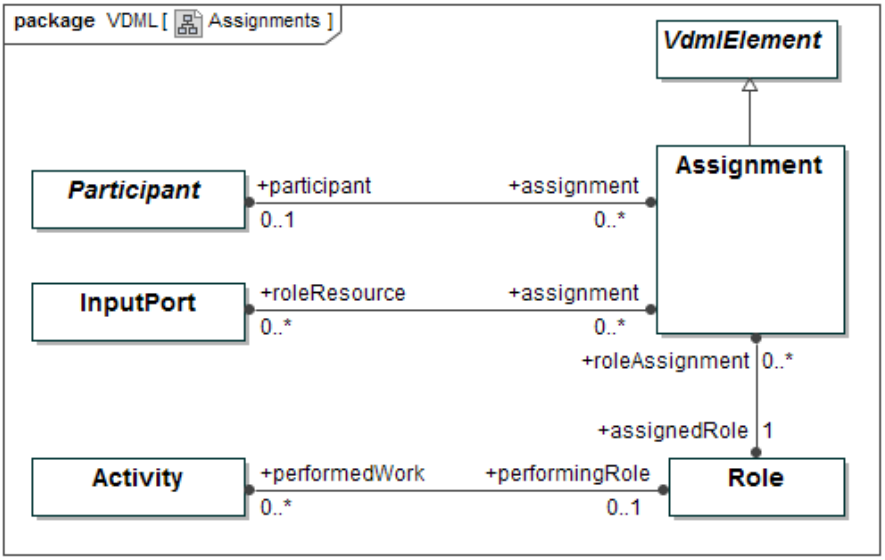

Figure 7-8: Assignments

An Assignment specifies how a Role in a Collaboration is or can be filled. An Assignment might be structurally defined in the model, as Assignment of a Role to a Participant, such as an OrgUnit, Position or Actor. In that case the Assignment is contained by the Collaboration that also contains the Role. An Assignment might also be DelegationContext-specific. In that case the Assignment is contained by a DelegationContext in which the Collaboration that contains the Role is used. ContextBasedAssignments are typically, though not necessarily, controlled dynamically in "run-time". The assignee to which the Role is assigned dynamically, is typically be defined by a resource that the Activity, that is performed by the Role, might obtain from a Pool of resources.

SuperClass
VdmlElement

Property	Description
assignedRole: Role [1]	The Role that is assigned by the Assignment
participant: Participant [0..1]	The Participant to which the Role is assigned
roleResource: InputPort [0..*]	The resources that are received through the InPutPorts and to which the Role is assigned. These resources are identified by the BusinessItems associated with the DeliverableFlows that connect to the InputPorts or to the InputPorts that are delegated to these InputPorts (see 7.2.4.4.2). The BusinessItems denote "classes of things" from which the Participant in the Role is determined dynamically.

Constraints

- An Assignment MUST NOT assign a Role to both a Participant and roleResource.
- A Role, that is not a Position, MUST NOT be assigned to more than one Participant, unless the additional Assignments to Participants are AnalysisContext-contained, whereby there MUST NOT be more than one such contextBasedAssignment per AnalysisContext (see 7.2.3.2).
- When a Position is assigned to multiple Participants, independent of any AnalysisContext (see 7.2.3.2), each of these Participants MUST be a Position or Member or Actor, whereby such a Position or Member MUST NOT, directly, or indirectly through other Assignments, be assigned to a Collaboration.
- When a Position is assigned to multiple Participants in the same AnalysisContext (see 7.2.3.2), each of these Participants MUST be a Position or Member or Actor, whereby such a Position or Member MUST NOT, directly, or indirectly through other Assignments, be assigned to a Collaboration.
- A Role MUST NOT be assigned itself (directly, or indirectly via other Assignments).
- A Role MUST not be assigned its containing Collaboration.
- BusinessNetworks and CapabilityMethods MUST NOT be Participants in Positions (i.e., Roles in OrgUnits) or Members (i.e., Roles in Communities).
- BusinessNetworks MUST NOT be Participants in Performers (i.e., Roles in CapabilityMethods).

7.2.1.2.4 DeliverableFlow Class

Figure 7-9 shows the linkage of a DeliverableFlow to OutputPorts and InputPorts on sending and receiving Activities or Stores. The DeliverableFlow conveys one or more BusinessItems.

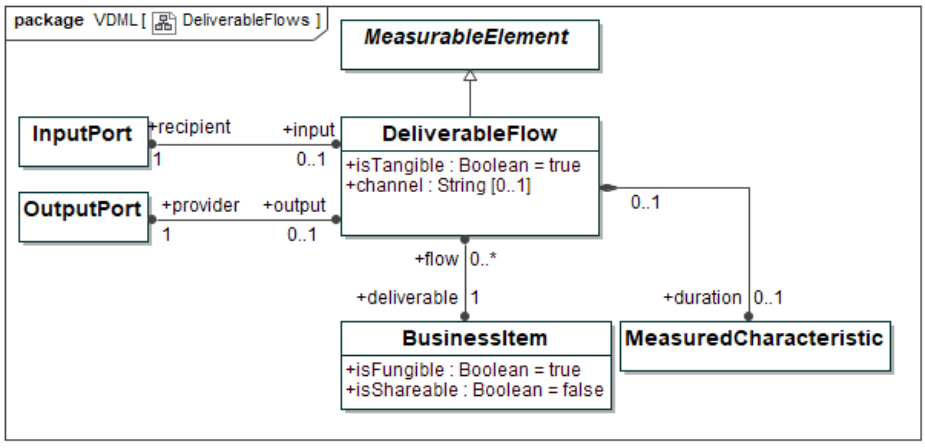

Figure 7-9: DeliverableFlows

A DeliverableFlow is a transfer of a deliverable from a provider (or producer) to a recipient (or consumer). A deliverable is a BusinessItem that is provided by a provider, i.e., produced by an Activity or delivered from a Store.

SuperClass
MeasurableElement

Property	Description
deliverable: BusinessItem [1]	Product or service, modeled as BusinessItem, produced by an Activity or delivered from a Store, and that can be conveyed to another Activity or Store.
isTangible: Boolean = true	If "true", the deliverable represents something that is contracted, mandated or expected by the recipient and which may generate revenue. If "false," the deliverable, as "intangible," represents something that is unpaid or non-contractual or that make things work smoothly or efficiently and help build relationships (see Allee (2008)).
recipient: InputPort [1]	Identifies the InputPort that receives the deliverable that is transferred via the DeliverableFlow.
provider: OutputPort [1]	Identifies the OutputPort that provides the deliverable that is transferred via the DeliverableFlow.
duration: MeasuredCharacteristic [0..1]	Represents the average delay that deliverables are subject to, when transferred from recipient to provider. This delay is caused by unbalance due to e.g., recipient's capacity to process the deliverable, or to differences between provider's and recipient's batchSizes.
channel: String [0..1]	Mechanism to execute a DeliverableFlow, such as e-mail, face-to-face conversation, SOAP, REST, physical transportation, postal service, telephone, fax, FTP, etc.

Constraints

- A DeliverableFlow MUST connect Ports of two Activities, or a Port of an Activity with a Port of a Store.
- A DeliverableFlow that connects to a Port of a Store MUST not have duration.
- A DeliverableFlow MUST NOT connect Ports of Activities that are contained in different Collaborations.

7.2.1.2.5 BusinessItem Class

A BusinessItem is anything that can be acquired or created, that conveys information, obligation or other forms of value and that can be conveyed from a provider to a recipient. For example, it includes parts, products, units of fluids, orders, emails, notices, contracts, currency, assignments, devices, property and other resources.

BusinessItems are classes of things that, dependent on the context in which they occur, might represent resources or deliverables. For example, a BusinessItem might be resource that is used by one Activity, and produced as deliverable by another.

Figure 7-10 indicates the association of a BusinessItem to a BusinessItemDefinition in the BusinessItemLibrary.

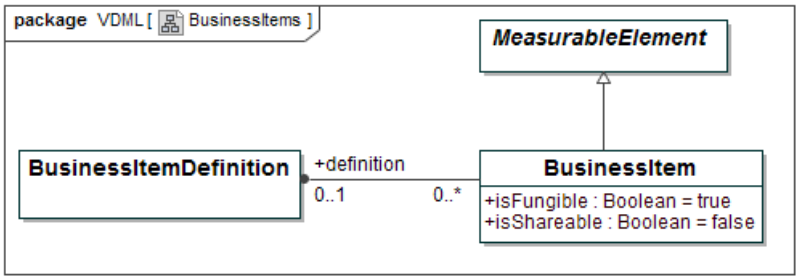

Figure 7-10: BusinessItems

SuperClass
`MeasurableElement`

Property	Description
isFungible: Boolean = true	If "true", instances of the `BusinessItem` are interchangeable, otherwise only a particular instance can satisfy a need
isShareable: Boolean = false	If "true", instances of the `BusinessItem` can be used simultaneously in multiple locations.
definition: BusinessItemDefinition [0..1]	Association to a `BusinessItemDefinition`, as contained in a `BusinessItemLibrary` that is applied to enforce consistency in the definition of `BusinessItems`. Multiple `BusinessItems` that are associated with the same `BusinessItemDefinition`, are considered similar from the perspective of the library.
flow: DeliverableFlow [0..*]	`DeliverableFlows` that convey the `BusinessItem` (see 7.2.1.2.4).
store: Store [0..*]	`Stores` in which the `BusinessItem` is stored (see 7.2.1.2.6).
method: CapabilityMethod [0..*]	`CapabilityMethods` to which the `BusinessItem` serves as `methodResource` (see 7.2.2.4).

7.2.1.2.6 Store Class

Figure 7-11 shows the relationship between a `Store` and a `BusinessItem` representing the `Resources` that are held by that `Store`. A `Store` is specialized to a `Pool` if the `resources` are reusable and thus may be tracked and returned to the `Pool` after use.

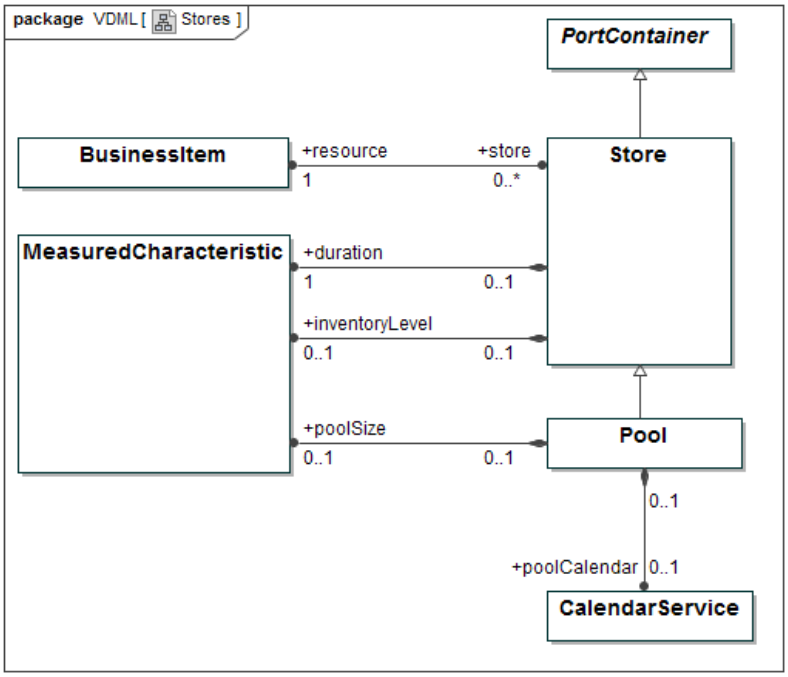

Figure 7-11: Stores

A `Store` is a container of `resource`. The `resource` that is stored is identified by a `BusinessItem`.

SuperClass

`PortContainer`

Property	Description
resource: BusinessItem [1]	The `BusinessItem` that identifies the `resource` that is being stored.
storeOwner: OrgUnit [1]	The `OrgUnit` that owns the `Store` (see 7.2.2.3).
inventoryLevel: measuredCharacteristic [0..1]	The average number of instances of the `resource` kept in `Store`. It might result from simulation. It's `Measurement`, like of any `MeasuredCharacteristic` might also be stochastically determined. Inventory level is essential to simulation.
duration: measuredCharacteristic [1]	The average time during which a `resource` is kept in `Store`.
supportedCapability: CapabilityOffer [0..*]	The `CapabilityOffers` that the `resource` in the `Store` supports.
storeContext: AnalysisContext [0..*]	`AnalysisContexts` that refer to the `Store` as their `contextStore` (see 7.2.3.2.1).

Constraints

- The `BusinessItem` that is received in the `Store`, or provided by the `Store`, via `DeliverableFlows`, MUST be the same as the `BusinessItem` that is associated with the `Store` as `resource`.

7.2.1.2.7 Pool Class

A `Pool` is a `Store` that contains re-usable `resource`, i.e., `resource` that is returned to the `Pool` after having been used, so that it is again available for use.

SuperClass
`Store`

Property	Description
poolSize: measuredCharacteristic [0..1]	The average number of `resource` instances that reside in the system, i.e., that are in the `Pool` (counted by `inventoryLevel`), or are in use by `Activities`.
position: Position [0..*]	`Positions` that are assigned, directly, or indirectly via other `Roles`, to `Actors` that are considered members of the `Pool`.
poolCalendar: CalendarService [0..1]	Calendar that determines the availability of the `resource`, when residing in the `Pool`, to perform work. The `CalendarService` that is assigned to a `Position` that is associated to the `Pool`, overrides the `poolCalendar`, as far as that `Position` is concerned.

Constraints

- The `inventoryLevel` of a `Pool` MUST NOT be bigger than the `poolSize` of that `Pool`.
- `Positions` that are associated with a `Pool` MUST NOT be assigned, directly, or indirectly via other `Roles`, to `Collaborations`.

7.2.1.3 ValueAdds and ValuePropositions

Figure 7-12 shows the structure that defines a `ValueProposition`. A `ValueProposition` expresses a `recipient`'s levels of satisfaction with the associated values. A `recipient` may be an individual entity or a representative of a market segment. `OutputPorts` on an `Activity` or `Store` identify `ValueAdd` elements representing the value contributions of the `Activity` or `Store`, which value contributions maybe incremental, or they may be cumulative until that point in the value stream. A `ValuePropositionComponent` may aggregate value (from `ValueAdd` element(s) and/or other `ValuePropositionComponent(s)`), and transform its `valueMeasurement` to a `recipient satisfactionLevel`.

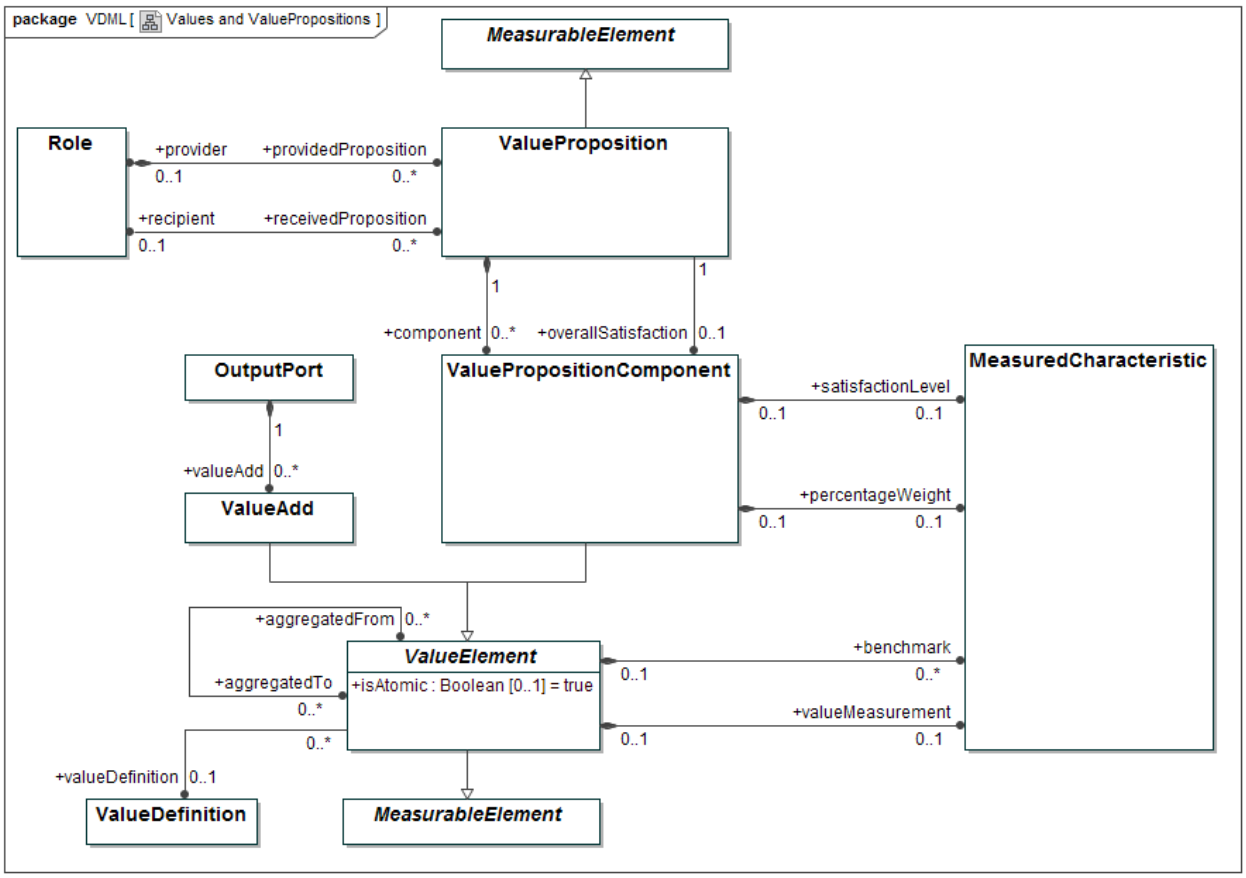

Figure 7-12: Values and ValuePropositions

The concept of "Value" as is adopted in VDML can be defined as "a measurable factor of benefit, of interest to a `recipient`, in association with a `BusinessItem`".

7.2.1.3.1 ValueProposition Class

A `ValueProposition` is the expression of the values offered to a `recipient` evaluated in terms of the `recipient`'s level of satisfaction.

SuperClass
`MeasurableElement`

Property	Description
component: ValuePropositionComponent [0..*]	The components that constitute the `ValueProposition`.
overallSatisfaction: ValuePropositionComponent [0..1]	Optional `component` that expresses the "overall satisfaction" of the `recipient` of the `ValueProposition`, with the `components` of that `ValueProposition`. It is important that the satisfaction of the `recipient` (typically a customer) with the values that are embodied by the `components` of the `ValueProposition` can be determined and expressed for the `ValueProposition` as a whole, in a way that enables further aggregation (see 7.2.1.3.4)

	of that overallSatisfaction into other ValueElements. The overallSatisfaction is typically computed as the weighted average satisfaction of components of the ValueProposition, based on their satisfactionLevel and percentageWeight properties (see 7.2.1.3.2).
provider: Role [0..1]	The Role that provides the ValueProposition. This Role optionally contains the ValueProposition in the metamodel.
recipient: Role [0..1]	The Role that receives the ValueProposition, as provided by the provider.

Constraints

- The overallSatisfaction of a ValueProposition MUST be a component of that ValueProposition.
- The overallSatisfaction of a ValueProposition MUST NOT itself have percentageWeight.
- The provider of a ValueProposition MUST NOT be the recipient of that ValueProposition.

7.2.1.3.2 ValuePropositionComponent Class

A part of a ValueProposition that expresses the perspective of the recipient of the ValueProposition on a particular value, as associated with a BusinessItem that is delivered to that recipient. This perspective includes the relative importance of that value to the recipient, expressed as a percentageWeight. It might also include the level of satisfaction of the recipient, with that value, based on a ranking or grading of that value (see SMM (2017)). As with any MeasurableElement, the user is enabled to add MeasuredCharacteristics if more Measurements are required. A ValuePropositionComponent, as ValueElement (see 7.2.1.3.4), might be aggregated from ValueAdd(s) (see 7.2.1.3.3) and/or other ValuePropositionComponent(s). It typically aggregates from ValueAdds on the OutputPorts of Activities in the network of Activities that result into the BusinessItem that is delivered (as deliverable) to the recipient of the ValueProposition.

SuperClass
ValueElement

Property	Description
percentageWeight: MeasuredCharacteristic [0..1]	The relative importance of the value, as embodied by the ValuePropositionComponent, to the recipient, expressed as a percentage. This property, together with the property that expresses the recipient's satisfactionLevel, may guide the provider in establishing priorities for improvement of Capabilities that contribute to value delivery.

satisfactionLevel: measuredCharacteristic [0..1]	The MeasuredCharacteristic that contains the level of satisfaction of the recipient with the value, as embodied by the ValuePropositionComponent, and is typically based on a ranking or grading of the valueMeasurement of that value (see SMM (2017)). This property, together with the percentageWeight property, may guide the provider in establishing priorities for improvement of Capabilities that contribute to value delivery.

Constraints

- The unit of the Measure (as specified by SMM) that determines the Measurement of percentageWeight of the ValuePropositionComponent MUST be "percent."

7.2.1.3.3 ValueAdd Class

A ValueAdd represents the value contribution of a PortContainer (i.e., an Activity, Store or Collaboration) and is associated with an OutputPort of that PortContainer. A ValueAdd, as ValueElement, might be aggregated from other ValueAdds, e.g., a ValueAdd of a Collaboration as aggregated from ValueAdds of Activities that are contained in the Collaboration. It might also be aggregated from ValuePropositionComponents, such as of ValuePropositions from suppliers.

The valueMeasurement of a ValueAdd, that is a leaf of a ValueElement aggregation structure (see 7.2.1.3.4), is typically dependent on or derived from Measurements of one or more MeasuredCharacteristics of the PortContainer that contains the OutputPort that carries the ValueAdd, or of elements that are associated with that PortContainer. These MeasuredCharacteristics will often identify performance Characteristics, such as aspects of costs, time (duration) or quality.

SuperClass
ValueElement

7.2.1.3.4 ValueElement Class (Abstract)

A ValueElement is a generalization of ValueAdd and ValuePropositionComponent. It also enables value aggregation, i.e., aggregation of a ValueElement from/to other ValueElements. Value aggregation is also a means to trace value, as embodied by a ValuePropositionComponent, back to the sources of value in the value stream, i.e., to the Activities, sub-Collaborations and Stores, and possibly supplier values (modeled as components of suppliers' ValuePropositions) that contribute to that value.

SuperClass
MeasurableElement

Property	Description
valueDefinition: ValueDefinition [0..1]	The associated ValueDefinition, as contained in a ValueLibrary, used to define the type of value that the ValueElement represents.
isAtomic: Boolean [0..1] = true	This optional property can be used to enforce whether or not a ValueElement can be aggregated from ValueElements of the same type (i.e., that refer to the same ValueDefinition). This is particularly

	useful to enforce consistency during the development of a model. When a ValueElement isAtomic, this means that it is a leaf in the structure of aggregation of ValueElements of the same type of value (i.e., of ValueElements that refer to the same ValueDefinition in the ValueLibrary). When a ValueElement is not atomic (i.e., isAtomic equals "false"), this means that it is an internal node in the structure of aggregation of ValueElements of the same type of value. For example, my_profit, my_cost and my_revenue are ValueElements, referring respectively to profit, cost and revenue as ValueDefinitions, whereby my_profit is aggregated from my_cost and _my_revenue, and my_cost is aggregated from more granular cost elements, referring to the same ValueDefinition (cost). ValueElement my_profit would be atomic, but a cost element that aggregates from other cost elements would not be atomic.
aggregatedFrom: ValueElement [0..*]	Represents ValueElement objects that are aggregated.
aggregatedTo: ValueElement [0..*]	Represents ValueElement objects that aggregate ValueElement.
valueMeasurement: MeasuredCharacteristic [0..1]	The MeasuredCharacteristic which associated Measurement contains the result of measuring of the value that is embodied by the ValueElement.
benchmark: MeasuredCharacteristic [0..*]	The MeasuredCharacteristic that defines a benchmark for the Measurement of value that is embodied by the ValueElement. A benchmark is typically measured based on a grading or ranking of valueMeasurement (see SMM (2017)).

Constraints

- When valueDefinition of a ValueElement is empty, isAtomic MUST be empty.
- The valueDefinition of a ValueElement that isAtomic MUST be different from the valueDefinition of aggregatedFrom of the ValueElement.
- At least one of the ValueDefinition-referring aggregatedFrom of a non-atomic (i.e., for which isAtomic equals “false”) ValueElement MUST refer to the same ValueDefinition as that non-atomic ValueElement refers to.

7.2.2 Collaboration Sub-Types

VDML defines four different specializations of Collaboration: BusinessNetwork, Community, Org Unit and CapabilityMethod. These specializations add specific semantics and place certain restrictions on the structure of each type including the types of Roles that can participate and the Roles each Collaboration type can fill.

7.2.2.1 BusinessNetworks

A `BusinessNetwork` is a `Collaboration` between independent business (or economic) entities, potentially companies, agencies, individuals or anonymous members of `Communities` of independent business entities, participating in an economic exchange.

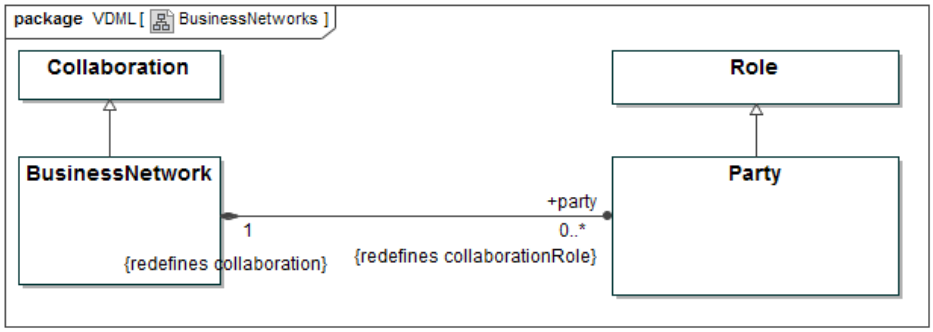

Figure 7-13: BusinessNetworks

7.2.2.1.1 BusinessNetwork Class

A `BusinessNetwork` can only have `Party Roles`. A `BusinessNetwork` can have nested `BusinessNetworks` within it.

SuperClass
`Collaboration`

Property	Description
party: Party [0..*]	`Roles` specific to and contained in the `BusinessNetwork`.

7.2.2.1.2 Party Class

A `Party` identifies a `Role` in a `BusinessNetwork`.

SuperClass
`Role`

7.2.2.2 Communities

A `Community` is a loose association of `Participants` with some shared purpose or interest. For example, a `Community` may be used to represent a market segment or a membership organization. `Communities` are restricted to having only `Member Roles`.

Figure 7-14: Communities

7.2.2.2.1 Community Class

A `Community` is a loose `Collaboration` of `Participants` with similar characteristics or interests that work together for some shared purpose such as sharing knowledge. Examples are `Communities` of interest or expertise within or outside the organization, industry membership organizations, and market segments.

SuperClass
`Collaboration`

Property	Description
member: Member [0..*]	`Roles` specific to and contained in the `Community`.

7.2.2.2.2 Member Class

A `Member` identifies a `Role` in a `Community`.

SuperClass
`Role`

7.2.2.3 OrgUnits and Capabilities

`OrgUnits` represent the structure of an organization. They exist to manage people and `resources`. `OrgUnits` can have `Capabilities` that typically define how the `OrgUnit` uses its people and `resources`. An `OrgUnit` makes its `Capabilities` available through `CapabilityOffers` that references associated `CapabilityDefinitions`. Note that multiple `OrgUnits` may offer the same `Capability`.

A `CapabilityOffer` may identify `Stores` and/or `Pools` from which it draws `resources` where `Pools` are people with particular skills.

The `Activity` network for a `CapabilityOffer` may be defined by a `CapabilityMethod` that identifies the `Roles` and their `Activities`.

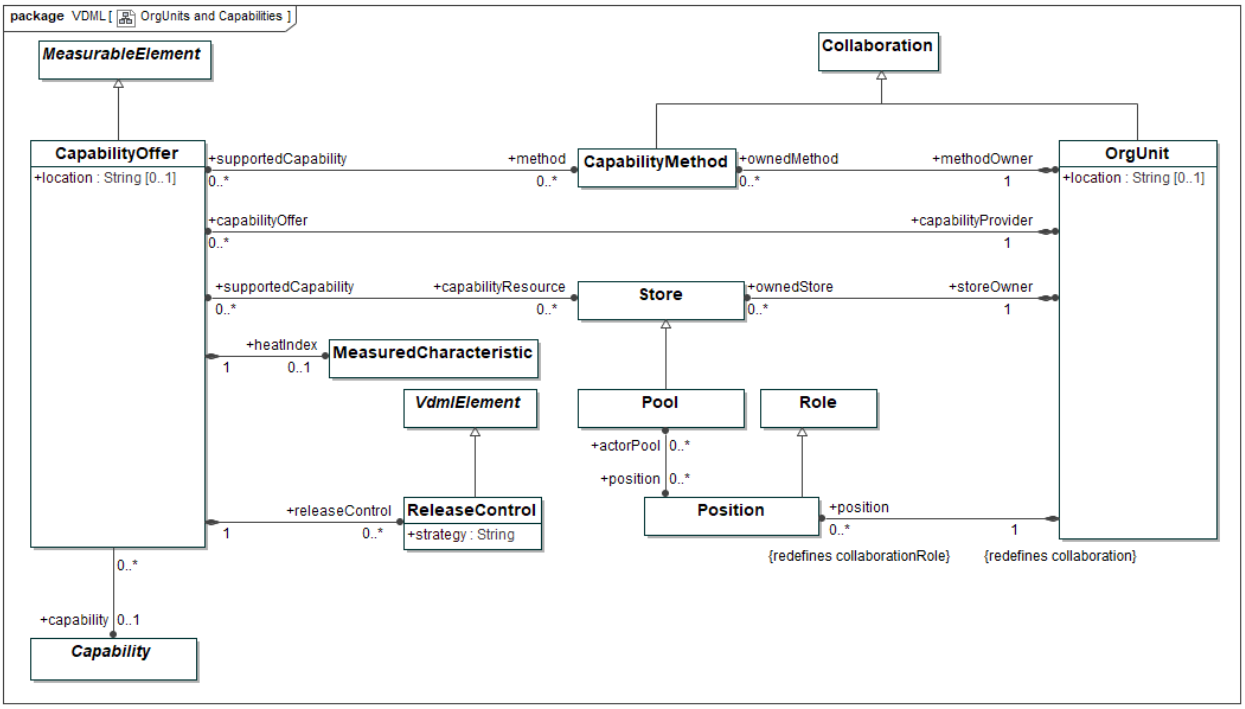

Figure 7-15: OrgUnits and Capabilities

7.2.2.3.1 OrgUnit Class

An administrative or functional organizational `Collaboration`, with responsibility for defined `resources`, including a `Collaboration` that occurs in the typical organization hierarchy, such as business units and departments (and also the company itself), as well as less formal organizational `Collaboration` such as a committee, project, or task force.

SuperClass
`Collaboration`

Property	Description
position: Position [0..*]	`Roles` specific to and contained in the `OrgUnit`.
capabilityOffer: CapabilityOffer [0..*]	The `CapabilityOffers` as owned managed and provided by the `OrgUnit`.
ownedMethod: CapabilityMethod [0..*]	The `CapabilityMethods` that are owned by the `OrgUnit`. `CapabilityMethods` might be owned by other `OrgUnits` than the ones that provide the `CapabilityOffers` that are supported by the `CapabilityMethods`.
ownedStore: Store [0..*]	The `Stores` of `resources` that are owned by the `OrgUnit`.
location: String [0..1]	The (optionally defined) `location` where the `OrgUnit` resides. `Location` may be used as geographic location. The way in which `location` is defined is left to the user.

7.2.2.3.2 Position Class

A `Position` identifies a `Role` in an `OrgUnit`.

SuperClass
`Role`

Property	Description
actorPool: Pool [0..*]	The Pools that the Actors, that fill the Positions (directly or indirectly via other Roles), are considered to be members of.

7.2.2.3.3 CapabilityOffer Class

A CapabilityOffer represents the ability to perform a particular kind of work and deliver desired value, by applying resources that are managed together, possibly based on formalized methods (CapabilityMethods).

SuperClass
MeasurableElement

Property	Description
capability: Capability [0..1]	The Capability that is offered, and which is defined via association to a Capability, as contained in a CapabilityLibrary, that is applied to enforce consistency in the definition of Capabilities.
capabilityProvider: OrgUnit [1]	The OrgUnit that owns manages and provides the CapabilityOffer.
method: CapabilityMethod [0..*]	The CapabilityMethods that support the Capability that is offered.
capabilityResource: Store [0..*]	The resources that support the Capability.
releaseControl: ReleaseControl [0..*]	The strategies to control the priority of the work to be performed by the CapabilityOffer.
location: String [0..1]	The (optionally defined) location where the CapabilityOffer resides. Location may be used as geographic location. The way in which location is defined is left to the user. Location of a CapabilityOffer can be used as basis to choose between multiple CapabilityOffers that offer the Capability that is required by an Activity.
applyingActivity: Activity [0..*]	The Activities to which the CapabilityOffer is applied (see 7.2.1.2.1).
heatIndex: MeasuredCharacteristic [0..1]	A Measurement that is compared with the heatThreshold as defined for the Scenario (see 7.2.3.2.2). When the heatIndex is beyond the heatThreshold, the CapabilityOffer is assumed to require business innovation / transformation management focus. When one or more CapabilityOffers have heatIndex value beyond the heatThreshold, the associated Capability maybe highlighted on a "heatmap" (see 8.7).

7.2.2.3.4 ReleaseControl Class

A ReleaseControl defines the strategy to control the priority of the work to be performed by a CapabilityOffer. Examples of such strategies are "first come first served," "shortest processing time first," "Activities for similar deliverable Characteristics first," "serving demand of highest priority customer first," "Activities that are on critical path, given demand fulfillment due date, first," etc.

At any moment in time, the work to be performed by a CapabilityOffer is represented by a set of Activity instances, from possibly multiple Activities (from possibly different Collaborations) that require the CapabilityOffer. The ReleaseControl strategy determines the "next" Activity instance that can start, from the subset of these instances that could start, i.e., for which the resources (including the roleResource), that are required to start, are available. Once a particular Activity instance is started, it will start using these resources, and the ReleaseControl will determine which Activity instance will start next, etc.

When no is defined for a CapabilityOffer, the assumed strategy is "first come first served."

releaseControl is mainly used to support Discrete Event Simulation, though it might generally provide insight in how the CapabilityOffer is applied and how its resources are organized to perform the work. In the absence of simulation it essentially provides annotation.

SuperClass
VdmlElement

Property	Description
strategy: String	The strategy that determines the priority for work release, as specified by the ReleaseControl. It may ontain a descriptive string or it may contain a URL that specifies an operation which is specified outside the model
scenario: Scenario [0..*]	The Scenarios according to which the ReleaseControls are applied.

7.2.2.4 CapabilityMethods

A CapabilityMethod can define the activity networks by which an OrgUnit delivers a capability. When an Activity engages an OrgUnit to fill a role and provide an offered capability, the CapabilityMethod defines how the OrgUnit will perform that Activity. The Activity provides inputs to the CapabiltyMethod, and receives results from the CapabilityMethod through the PortDelegation mechanism discussed later.

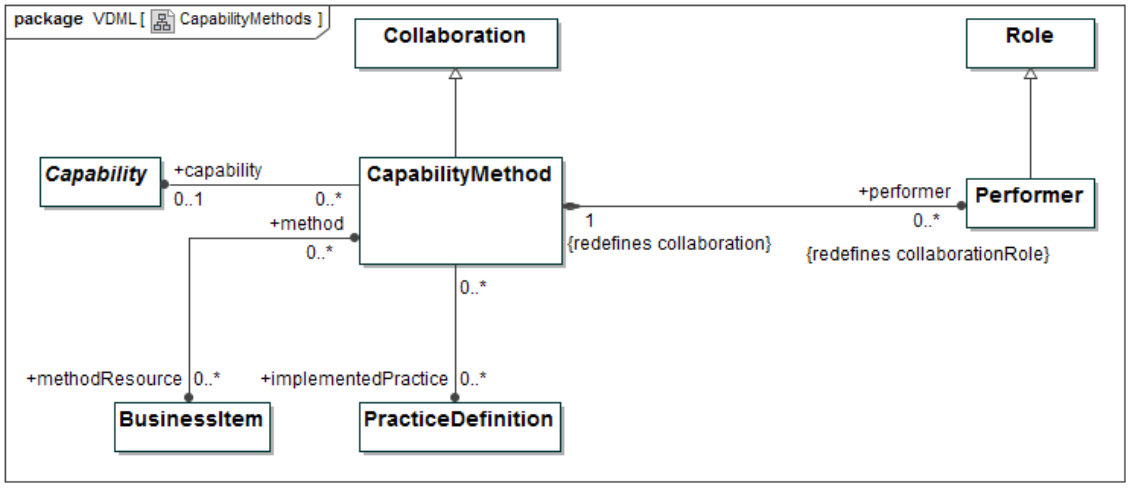

Figure 7-16: CapabilityMethods

7.2.2.4.1 CapabilityMethod Class

A Collaboration specification that defines the Activities, DeliverableFlows, BusinessItems, capabilityRequirements and Roles that deliver a Capability and associated

value contributions (defined via ValueAdds, see 7.2.1.3). For each application of the CapabilityMethod, within a Scenario or in multiple Scenarios, there may be distinct Measurements of performance and value contribution, and Role Assignments suitable to the application context. A CapabilityMethod does not own resources but receives them from other sources in the course of performing its Activities.

SuperClass
Collaboration

Property	Description
performer: Performer [0..*]	Roles specific to and contained in the CapabilityMethod.
capability: Capability [0..1]	The Capability that is provided through the CapabilityMethod, and which is defined via association to a Capability, as contained in a CapabilityLibrary that is applied to enforce consistency in the definition of Capabilities.
methodOwner: OrgUnit [1]	The OrgUnit that owns the CapabilityMethod. OrgUnits that apply the CapabilityMethod, to support their CapabilityOffers, need not be methodOwner.
supportedCapability: CapabilityOffer [0..*]	The CapabilityOffers that the CapabilityMethod supports.
methodResource: BusinessItem [0..*]	Resources that support, strengthen or accelerate the CapabilityMethod, and which cannot be controlled at the Activity level, i.e., for which use or consumption via InputPorts, counting of use or consumption and inventory control do not apply. Typical examples are knowledge resources such as patents, or licenses of applications. A business process execution engine is an example of such an application.
implementedPractice: PracticeDefinition [0..*]	Indications of which practices are implemented by means of the CapabilityMethod, via association to a PracticeDefinition, as contained in a PracticeLibrary that is applied to enforce consistency in the definition of practices. The same practices might also require other CapabilityMethods to implement them.

Constraints

- When a CapabilityMethod supports more than one CapabilityOffer, possibly provided by different OrgUnits, the Capability as provided by these CapabilityOffers MUST be the same.
- When a CapabilityMethod supports a CapabilityOffer, then, when both refer to a Capability, they MUST refer to the same Capability.

7.2.2.4.2 Performer Class

A Performer identifies a Role in a CapabilityMethod.

SuperClass
Role

7.2.3 Models and Scenarios

A `ValueDeliveryModel` represents the elements and relationships of the design of an enterprise or, more often, a segment of an enterprise. All the elements of a model are directly or indirectly associated with a `ValueDeliveryModel` element. A modeling environment may have multiple `ValueDeliveryModels` that represent different versions of the enterprise design, designs of different segments of the enterprise or even different enterprises. Each model is independent.

Within each model, there may be different `Scenarios` each representing a set of `Measurements` and potentially different delegations under different circumstances.

7.2.3.1 ValueDeliveryModels

Figure 7-17 represents the direct associations of model elements with a `ValueDeliveryModel` element.

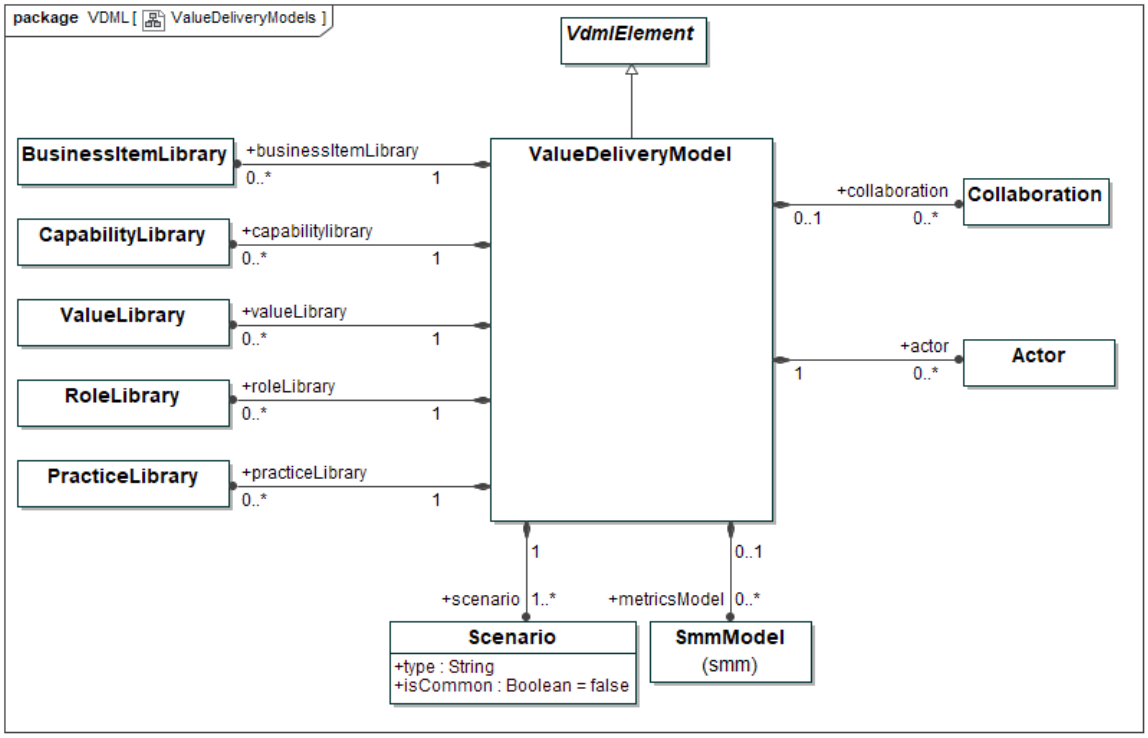

Figure 7-17: ValueDeliveryModels

7.2.3.1.1 ValueDeliveryModel Class

A `ValueDeliveryModel` is a model that supports business analysis and design based on evaluation of performance and stakeholder satisfaction achieved through the `Activities` and interactions of people and organizations using business `Capabilities` to apply `resources` and deliver stakeholder values.

This class represents the entry point into a `ValueDeliveryModel` and provides the top-level container for all the elements of it.

The fact that these elements are contained in a `ValueDeliveryModel` does not prohibit association of them, or elements contained in them with other `ValueDeliveryModels`, or elements that are directly or indirectly contained in them. It is essential to consider that `ValueDeliveryModels` do not prohibit re-use, but rather enable re-use.

SuperClass
AnalysisContext

Property	Description
scenario: Scenario [1..*]	Scenarios that are contained in the ValueDeliveryModel.
metricsModel: SmmModel [0..*]	SmmModels (as specified by SMM) that are contained in and specific to the ValueDeliveryModel.
businessItemLibrary: BusinessItemLibrary [0..*]	BusinessItemLibraries that are contained in and specific to the ValueDeliveryModel.
capabilityLibrary: CapabilityLibrary [0..*]	CapabilityLibraries that are contained in and specific to the ValueDeliveryModel.
valueLibrary: ValueLibrary [0..*]	ValueLibraries that are contained in and specific to the ValueDeliveryModel.
roleLibrary: RoleLibrary [0..*]	RoleLibraries that are contained in and specific to the ValueDeliveryModel.
practiceLibrary: PracticeLibrary [0..*]	PracticeLibraries that are contained in and specific to the ValueDeliveryModel.
collaboration: Collaboration [0..*]	Collaborations that are contained in the ValueDeliveryModel.
actor: Actor [0..*]	Actors that are contained in the ValueDeliveryModel.

Constraints

- A ValueDeliveryModel MUST NOT have more than one common Scenario (i.e., a Scenario with isCommon = true).

7.2.3.2 Scenarios and AnalysisContexts

Figure 7-18 defines the associations of a Scenario, DelegationContexts, and elements of Collaborations. AnalysisContext is an abstract class of elements that form a delegation tree with Scenario at the root. Each element in the tree may have a corresponding SMM Observation; this means it has a distinct set of Measurements for the Collaboration and/or Store elements associated with that context. Each delegation by an Activity to a Collaboration is represented by a DelegationContext that can also define Role Assignments for that delegation and PortDelegation elements that link inputs and outputs of the delegation. Role Assignments can be defined per Scenario as well.

This delegation structure allows a Collaboration to occur in different contexts within a Scenario, and it also allows an Activity to delegate to different Collaborations in different Scenarios.

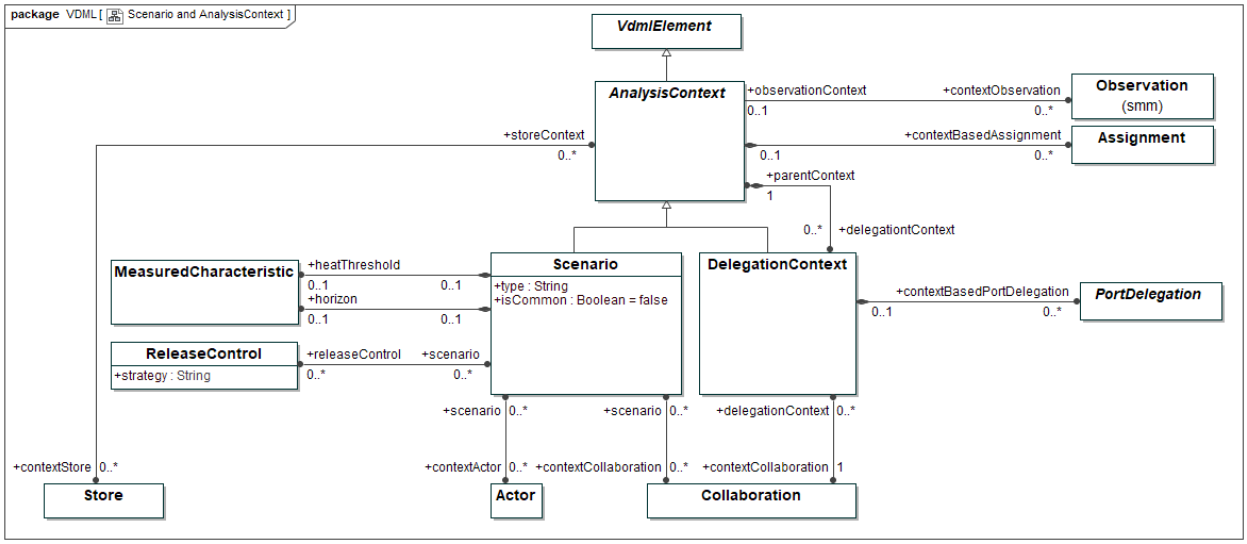

Figure 7-18: Scenarios and AnalysisContexts

7.2.3.2.1 AnalysisContext Class (Abstract)

An `AnalysisContext` defines the set of `Measurements` of a particular use of one or more `Collaborations`, or `Stores` when used as decoupling points between `Collaborations`. It may also define `Assignments` of `Roles` in its `contextCollaboration(s)`.

SuperClass
`VdmlElement`

Property	Description
contextObservation: Observation [0..*]	A `contextObservation` is an `Observation` (as specified by SMM) that contains the `Measurements` (of `MeasuredCharacteristics`) that are specific to the `AnalysisContext` (i.e., the `observationContext`). For purpose of creating a VDML model with embedded `Measurements` a single `Observation` per `AnalysisContext` is sufficient. For other purposes, such as monitoring performance after implementation of the VDML model in the business, additional `Observations` can be used to capture `Measurements` for the same `AnalysisContext`. VDML does not specify how to distinguish between these `Observations`, given their purpose. A VDML tool may do that in various ways. For example by allowing a single `Observation` for modeling purpose, or by requiring that the `whenObserved` property of an `Observation`, as specified by SMM is left empty for the modeling-time `Observation` and is set as the moment of taking a monitoring snapshot for the other `Observations`.
contextBasedAssignment: Assignment [0..*]	`Assignments` of `Roles` that are contained in the `contextCollaborations` of the `Scenario` (when the `AnalysisContext` is a `Scenario`), or that are contained in the `contextCollaboration` of the `DelegationContext` (when the `AnalysisContext`

	is a DelegationContext). These Assignments are AnalysisContext-specific, i.e., context-dependent.
delegationContext: DelegationContext [0..*]	The set of DelegationContexts that are owned by the AnalysisContext.
contextStore: Store [0..*]	The Stores that are analyzed in the AnalysisContext. When the AnalysisContext is a Scenario, the contextStores serve as decoupling buffers between Collaborations that are analyzed under that Scenario, and which Collaborations are related to either the Scenario itself, or to DelegationContexts in the nesting tree of that Scenario.

Constraints

- When an Activity delegates its work to a Collaboration, the DelegationContext in which this delegation occurs, MUST be owned by a DelegationContext in which the Activity-containing Collaboration itself is used, or MUST be owned by a Scenario (as the top of an AnalysisContext tree), when the Activity-containing Collaboration is directly related to that Scenario.
- When an Activity delegates its work in more than one DelegationContext, these DelegationContexts MUST be owned by different AnalysisContexts, being DelegationContexts and/or Scenarios, in which the Activity-containing Collaboration itself is used.

7.2.3.2.2 Scenario Class

A Scenario defines a consistent business use case and set of Measurements of a ValueDeliveryModel by specifying a, possibly recursive, AnalysisContext for elements in scope of that use case. The nesting of AnalysisContexts allows a Collaboration to be used as a sub-Collaboration by more than one Activity, each of which sets its particular DelegationContext and Measurements. It may also, as AnalysisContext, define Assignments of Roles in its contextCollaborations.

SuperClass
AnalysisContext

Property	Description
type: String	The type of the Scenario, e.g., "As-is / monitoring-based," "To-be / estimation-based," "To-be / simulation-based," etc.
isCommon: Boolean = false	If "true," the Scenario represents the common Scenario of a ValueDeliveryModel. The common Scenario may define a Measurement for any MeasuredCharacteristic of any MeasurableElement in a ValueDeliveryModel, which Measurement applies as initial Measurement in any AnalysisContext (other than the common Scenario) in which the MeasurableElement is analyzed, until the contextObservation of that AnalysisContext contains a Measurement for that MeasuredCharacteristic.
contextCollaboration: Collaboration [0..*]	Collaborations, in scope of the Scenario and that serve as top-level Collaborations in that Scenario.

	For an OrgUnit particpant in the calling Collaboration, if there is a CapabilityMethod, it is identified by this association. The Participant in the same Role in the calling Collaboration may apply different CapablityMethods for different Activities.
contextActor: Actor [0..*]	Actors, in scope of the Scenario and for which the Scenario's contextObservation imposes Measurements.
horizon: MeasuredCharacteristic [0..1]	The time distance into the future that the Scenario spans.
releaseControl: ReleaseControl [0..*]	ReleaseControls that apply in the Scenario.
heatThreshold: MeasuredCharacteristic [0..1]	A MeasuredCharacteristic that serves as criterion to determine whether CapabilityOffers for a certain Capability require business innovation / transformation management focus. Such focus is assumed to be required when a CapabilityOffer's heatIndex is beyond the heatThreshold.

Constraints

- The common Scenario (i.e., the Scenario for which isCommon equals "true") MUST not contain DelegationContexts, and MUST not have contextStores, contextCollaborations and releaseControls.
- A Scenario MUST NOT have more than one releaseControl for the same CapabilityOffer.

7.2.3.2.3 DelegationContext Class

AnalysisContext, set by an Activity and in which the Activity delegates its work to a Collaboration. A DelegationContext also defines the PortDelegations of Activity inputs and/or outputs to/from Collaboration inputs and/or outputs, and may, as AnalysisContext, define Assignments of Roles in its contextCollaboration.

SuperClass
AnalysisContext

Property	Description
parentContext: AnalysisContext [1]	The AnalysisContext that contains the DelegationContext.
contextCollaboration: Collaboration [1]	The Collaboration to which work is delegated by an Activity, and which is analyzed in the DelegationContext.
contextBasedPortDelegation: PortDelegation [0..*]	PortDelegations that map Ports of the delegatedActivity to Ports of the contextCollaboration of the DelegationContext. These PortDelegations are DelegationContext-specific.
delegatedActivity: Activity [1]	Activity that delegates its work to the contextCollaboration.

Constraints

- `PortDelegations` that are contained by a `DelegationContext` MUST map `Ports` of the `DelegationContext`'s `delegatedActivity` to `Ports` of the `DelegationContext`'s `contextCollaboration`.

7.2.4 Core Elements

7.2.4.1 VdmlElements

`VdmlElement` is a shared abstract class for primary model elements. It defines the associations to attach `Attributes` and `Annotations`, and its specialization, `MeasuredElement` is the abstract super-class for elements that can have associated `Measurements`.

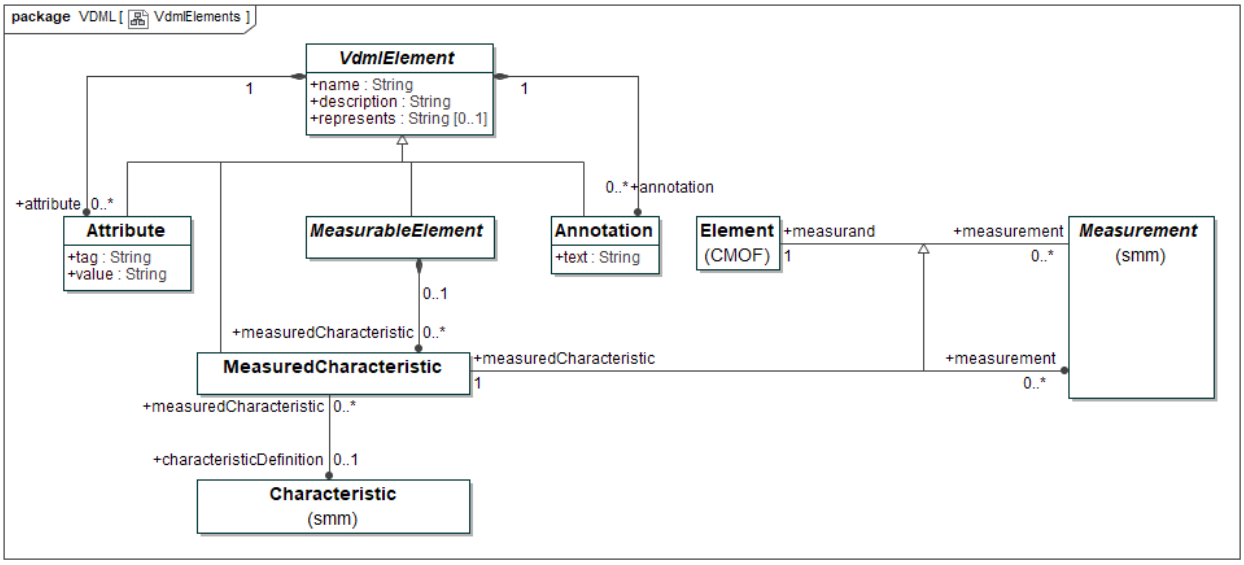

Figure 7-19: VdmlElements

SMM (see SMM (2017)) can associate `Measurements` with any `measurand` (i.e., object that is measured). In VDML we apply `Measurements` more narrowly, by enforcing that they can only be associated with `MeasuredCharacteristics` as defined by VDML. This is achieved by specialization of SMM-defined association between `Element` and `Measurement`, and redefining `measurand` (property owned by that association) to `measuredCharacteristic` (property owned by the specialized association).

7.2.4.1.1 VdmlElement Class (Abstract)

A `VdmlElement` is the root of the hierarchy of all classes in VDML. It is an abstract class.

Property	Description
name: String	The name of the `VdmlElement`.
description: String	A description of the `VdmlElement`.
represents: String [0..1]	A reference to something that the `VdmlElement` represents, such as a model element in any MOF-based model, an object in an application database, a web page, or anything uri-addressable (optional).
attribute: Attribute [0..*]	User defined attributes of the `VdmlElement`.
annotation: Annotation [0..*]	User defined annotations to the `VdmlElement`.

7.2.4.1.2 Attribute Class

An `Attribute` allows information to be attached to any `VdmlElement` in the form of a name-value pair. `Attributes` provide a simple mechanism to add user defined information to model elements.

SuperClass
`VdmlElement`

Property	Description
text: String	Contains the name of the `Attribute`.
value: String	Contains the current value of the `Attribute`.

7.2.4.1.3 Annotation Class

`Annotations` allow textual descriptions to be attached to any `VdmlElement`.

SuperClass
`VdmlElement`

Property	Description
tag: String	Contains the text of the `Annotation` to the target model element.

7.2.4.1.4 MeasurableElement Class (Abstract)

Abstract class that represents the subset of `VdmlElements` that can have user defined `MeasuredCharacteristics`.

SuperClass
`VdmlElement`

Property	Description
measuredCharacteristic: MeasuredCharacteristic [0..*]	User defined `MeasuredCharacteristics` of the `MeasurableElement` .

7.2.4.1.5 MeasuredCharacteristic Class

`MeasurableElement` property that has `Measurements` as instance values. It is defined based on a `Characteristic` in a `MeasureLibrary`. A `Measure` as defined in the `MeasureLibrary`, against that `Characteristic`, is used to determine the `Measurement(s)` of the `MeasurableElement`.

SuperClass
`VdmlElement`

Property	Description
characteristicDefinition: Characteristic [0..1]	The `Characteristic` as defined in a `MeasureLibrary` (as specified by SMM).
measurement: Measurement [0..*]	`Measurements` that specify the instance values of the `MeasuredCharacteristic`.

Constraints

- When a `MeasuredCharacteristic` is associated with more than one `Measurement`, each `Measurement` MUST be contained in a separate `Observation` (as specified by SMM), as associated with a separate `AnalysisContext` (see 7.2.3.2.1).

7.2.4.2 Expressions

An `Expression` specifies the operational mechanism by which one or more alternative elements are selected. The actual selection of elements would occur during business operations or a simulation. In the absence of simulation, the expression provides the basis for statistical analysis of the selection criteria.

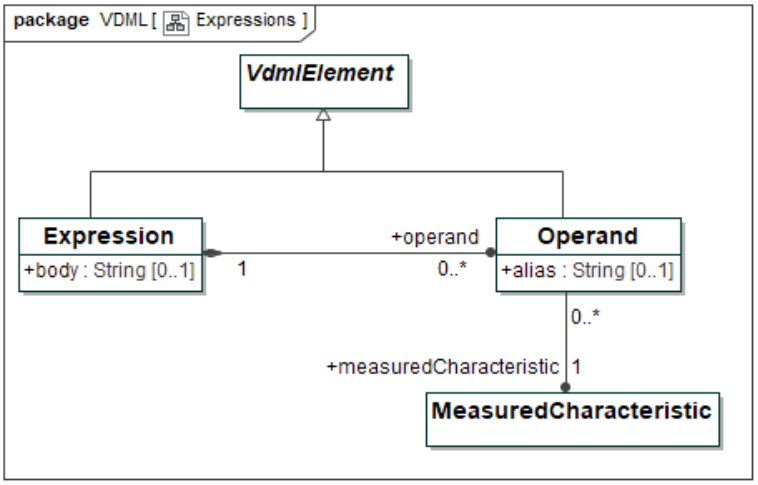

Figure 7-20: Expressions

7.2.4.2.1 Expression Class

An `Expression` defines a statement which will evaluate on a (possibly empty) set of model objects (instances of metamodel objects), when executed in a context. An `Expression` does not modify the environment in which it is evaluated.

SuperClass
`VdmlElement`

Property	Description
body: String [0..1]	Specifies the statement that is evaluated.
operand: Operand [0..*]	The `operands` that are used in the `body` of the `Expression`.

7.2.4.2.2 Operand Class

An `Operand` is an object on which the `body` of an `Expression` is evaluated.

SuperClass
`VdmlElement`

Property	Description
measuredCharacteristic: MeasuredCharacteristic [1]	The `MeasuredCharacteristic` that serves as `operand` in the `Expression`.
alias: String [0..1]	Short substitute for the fully qualified name of the `operand`, in the context of the `Expression`.

7.2.4.3 PortContainers

`Collaborations`, `Activities` and `Stores` can have `inputs` and `outputs`. `Ports` define the connection points for `inputs` and `outputs`, and `PortContainer` is the abstract superclass that associates

Ports with Collaborations, Activities and Stores. Ports, via their related DeliverableFlows, are also associated with the input and output BusinessItems.

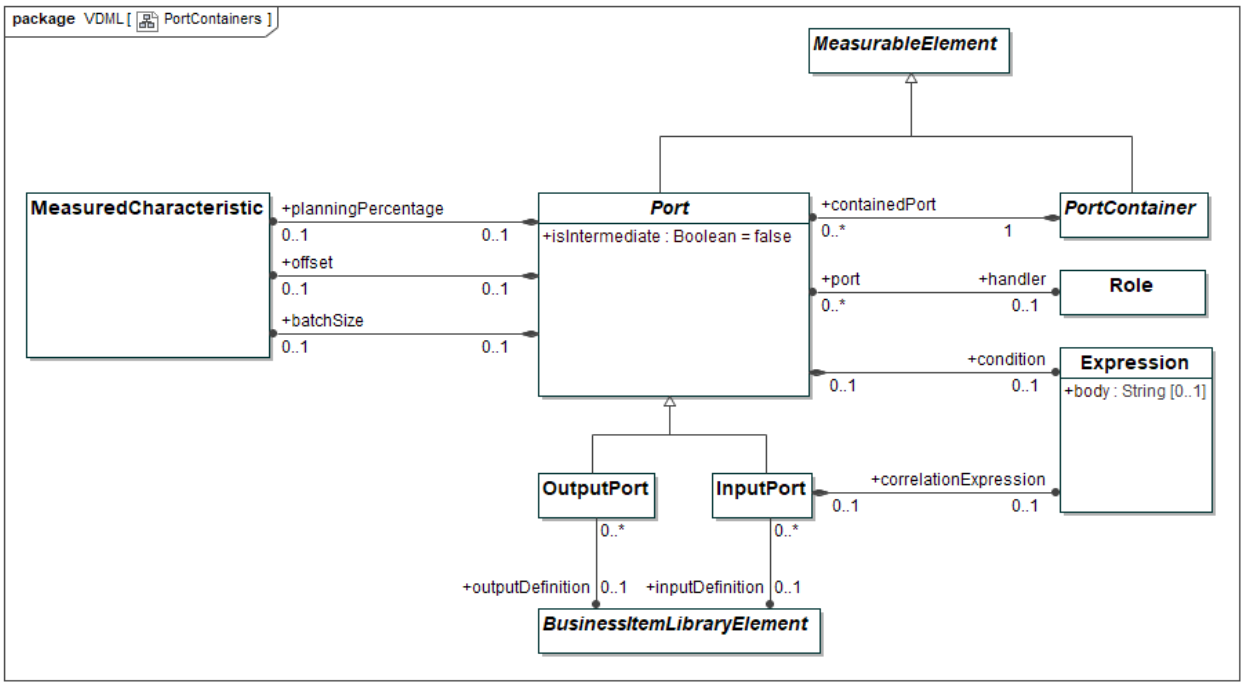

Figure 7-21: PortContainers

7.2.4.3.1 Port Class (Abstract)

A Port is a connection point to a PortContainer, used to handle inputs and outputs (e.g., consume inputs, produce outputs or delegate inputs or outputs to Ports of other PortContainers).

SuperClass
MeasurableElement

Property	Description
isIntermediate: Boolean = false	Specifies whether communication with the PortContainer, via the Port, can occur at any time, or only at the start (for an InputPort) or end (for an OutputPort) of the lifecycle of an instance of the PortContainer. This difference is only relevant for PortContainers that require instances to conduct their behavior, which are Activities and CapabilityMethods.
offset: measuredCharacteristic [0..1]	Specifies, for an InputPort, the elapse of time between start of the Activity and the receipt of the input. For an OutputPort it specifies the elapse of time between the delivery of the output and the completion of the Activity.
planningPercentage: MeasuredCharacteristic [0..1]	Specifies probability of use of the Port.
condition: Expression [0..1]	Specifies the condition under which the Port is used.
batchSize: MeasuredCharacteristic [0..1]	Specifies the number of units of an input or output that are communicated together via a Port.

handler: Role [0..1]	Specifies the Role that is responsible for handling the particular input or output that the Port represents.

Constraints

- PortContainers other than Activities and Collaborations that are CapabilityMethods MUST NOT contain Ports with isIntermediate = false.
- PortContainers that are Collaborations MUST NOT have Ports with a planningPercentage.
- PortContainers that are Collaborations MUST NOT have Ports with a condition.
- PortContainers that are Collaborations MUST NOT have Ports with a batchSize.
- PortContainers that are Collaborations or Stores MUST NOT have Ports with offset.
- When offset is defined on a Port, the Port MUST be defined as intermediate Port (i.e., isIntermediate = true).
- An intermediate Port of an Activity MUST have an offset.
- The unit of the Measure (as specified by SMM) that determines the Measurement of planningPercentage, MUST be "percent."

7.2.4.3.2 OutputPort Class

Port that is used to handle outputs from PortContainers.

SuperClass
Port

Property	Description
outputDefinition: BusinessItemLibraryElement [0..1]	Association to a BusinessItemLibraryElement, as contained in a BusinessItemLibrary, used to define the type of output that the OutputPort can handle.
output: DeliverableFlow [0..1]	DeliverableFlow that refers to the OutputPort as its provider (see 7.2.1.2.4).
resourceUse: ResourceUse [0..*]	Objects that define ResourceUse relative to the OutputPort as deliverable (see 7.2.1.2.2).
valueAdd: ValueAdd [0..*]	ValueAdd objects that represent the values that are delivered by the OutputPort (see 7.2.1.3).
delegatedOutput: OutputDelegation [0..*]	The OutputDelegations that refer to the OutputPort as their target (see 7.2.4.4.3).
outputDelegation: OutputDelegation [0..*]	The OutputDelegations that delegate the OutputPort (see 7.2.4.4.3).

7.2.4.3.3 InputPort Class

Port that is used to handle inputs to PortContainers.

SuperClass
Port

Property	Description
inputDefinition: BusinessItemLibraryElement [0..1]	Association to a `BusinessItemLibraryElement`, as contained in a `BusinessItemLibrary`, used to define the type of `input` that the `InputPort` can handle.
correlationExpression: Expression [0..1]	`Expression` that is evaluated to specify the instance of a non-fungible `BusinessItem` (i.e., `BusinessItem` with `isFungible` = false).
input: DeliverableFlow [0..1]	`DeliverableFlow` that refers to the `OutputPort` as its `recipient` (see 7.2.1.2.4).
resourceUse: ResourceUse [0..*]	Objects that define `ResourceUse` for the `InputPort` as `resource` (see 7.2.1.2.2).
delegatedInput: InputDelegation [0..*]	The `InputDelegations` that refer to the `InputPort` as their `target` (see 7.2.4.4.2).
inputDelegation: inputDelegation [0..*]	The `InputDelegations` that delegate the `InputPort` (see 7.2.4.4.2).
assignment: Assignment [0..*]	The `Assignment` that assigns the `Role` to the `roleResource` that is provided by the `InputPort` (see 7.2.1.2.3)

Constraints

- A `Port` that is not an `InputPort` of an `Activity` and that is not `recipient` of a non-fungible `BusinessItem` from a `Store` MUST NOT have a `correlationExpression`.

7.2.4.3.4 PortContainer Class (Abstract)

A `PortContainer` is a container of `Ports`. VDML distinguishes `Activities`, `Stores` and `Collaborations` as sub-types of `PortContainer`.

SuperClass
`MeasureableElement`

Property	Description
containedPort: Port [0..*]	`Ports` that are contained in the `PortContainer`

7.2.4.4 PortDelegations

Figure 7-22 defines the links between `Ports` in a delegation. Essentially, specializations of `PortDelegation` map the `input` of a delegating `Activity` to the `input` of an engaged `Collaboration`, and the `output` of the engaged `Collaboration` to the `output` of the delegating `Activity`. `InputDelegations` and `OutputDelegations` are also used to map the `input` of a `Collaboration` to the `input` of an `Activity` that is contained in that `Collaboration`, and the `output` of a `Collaboration` to the `output` of an `Activity` that is contained in that `Collaboration` respectively.

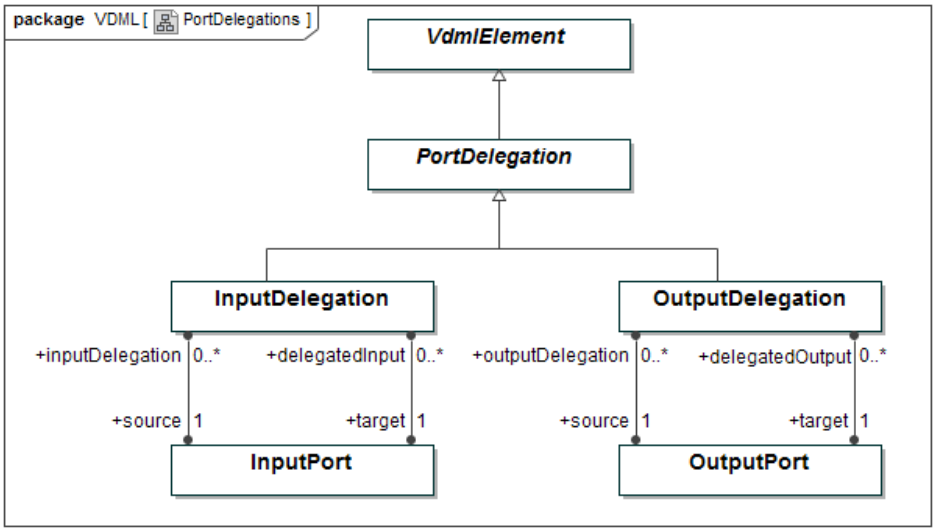

Figure 7-22: PortDelegations

7.2.4.4.1 PortDelegation Class (Abstract)

A `PortDelegation` provides a mapping between a `Port` of an `Activity` and a `Port` of a `Collaboration` to which the `Activity` delegates its work (in a particular `DelegationContext` (see 7.2.3.2.3). A `PortDelegation` can also provide a mapping between a `Port` of a `Collaboration` and a `Port` of an `Activity` that is contained in the `Collaboration`. `Ports` of `Collaborations` can be considered to represent an interface that enables abstraction of the internal work organization of `Collaborations` away from `Activities` that delegate their work to the `Collaboration`.

SuperClass
`VdmlElement`

7.2.4.4.2 InputDelegation Class

A `PortDelegation` that maps an `InputPorts`.

SuperClass
`PortDelegation`

Property	Description
source: InputPort[1]	The `InputPort` that delegates its `input` to the `target InputPort`
target: InputPort[1]	The `InputPort` to which the `source InputPort` delegates its `input`

Constraints

- An `InputDelegation` MUST either map an `InputPort` of an `Activity` to an `InputPort` of a `Collaboration`, or it MUST map an `InputPort` of a `Collaboration` to an `InputPort` of an `Activity` that is contained in the `Collaboration`.
- An `InputPort` of an `Activity` that is contained in a `Collaboration` MUST NOT be mapped, via `InputDelegation`, to more than one `InputPort` of the containing `Collaboration`.

7.2.4.4.3 OutputDelegation Class

A `PortDelegation` that maps `OutputPorts`.

SuperClass
`PortDelegation`

Property	Description
source: OutputPort[1]	The `OutputPort` that delegates its `output` to the `target OutputPort`
target: OutputPort[1]	The `OutputPort` to which the `source OutputPort` delegates it `output`

Constraints

- An `OutputDelegation` MUST either map an `OutputPort` of an `Activity` to an `OutputPort` of a `Collaboration`, or it MUST map an `OutputPort` of a `Collaboration` to an `OutputPort` of an `Activity` that is contained in the `Collaboration`.
- An `OutputPort` of an `Activity` that is contained in a `Collaboration` MUST NOT be mapped, via `OutputDelegation`, to more than one `OutputPort` of the containing `Collaboration`.

7.2.5 Libraries

Each VDML library is a collection of definitions of a particular type of business concept. This ensures that a particular concept has the same name and definition wherever it occurs in a `ValueDeliveryModel`. The names and definitions are user-defined, but it is expected that there will be shared libraries for specific industries so the same names and definitions are used in many companies with local extensions as needed. The library structure provides for a taxonomy of concepts. Elements in these libraries also have associated data that is useful to guide users in "discovering" parts of their business designs, such as `Activity` networks, `resources` that support `Capabilities`, etc.

7.2.5.1 BusinessItemLibrary

Figure 7-23 defines the library structure for `BusinessItemDefinitions`.

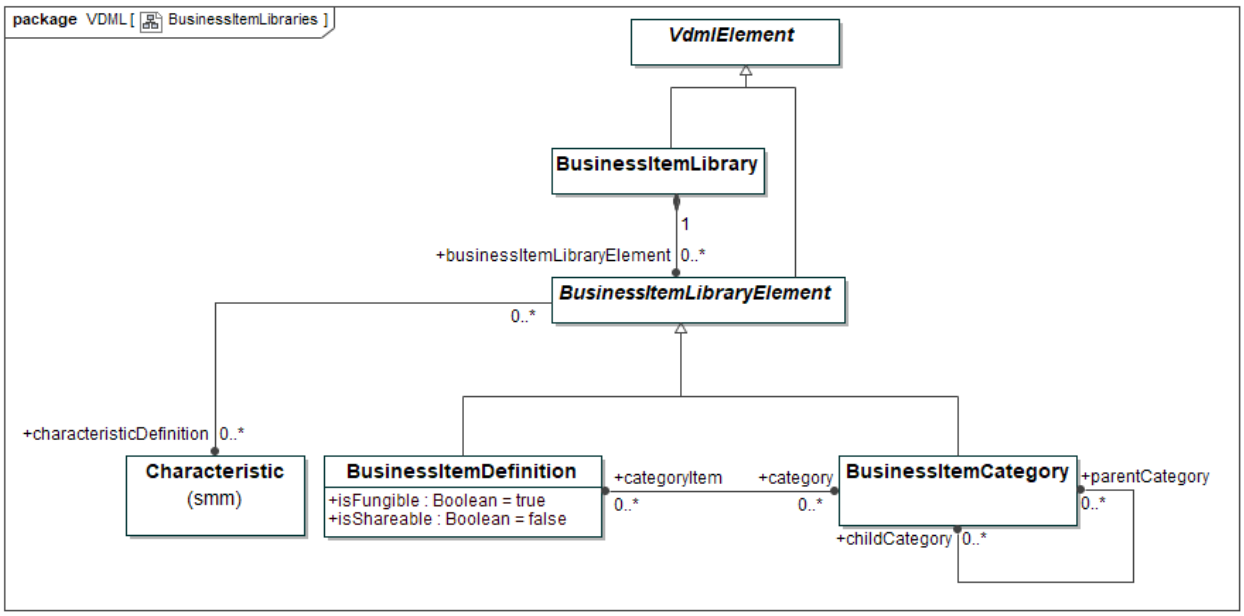

Figure 7-23: BusinessItemLibraries

7.2.5.1.1 BusinessItemLibrary Class

A `BusinessItemLibrary` contains a taxonomy of `BusinessItems`, consisting of `BusinessItemDefinitions` and categories of them, and is applied to enforce consistency in the definition of `BusinessItems`. Multiple `BusinessItems` that are associated with the same `BusinessItemDefinition` are considered similar from the perspective of the `BusinessItemLibrary`.

SuperClass
`VdmlElement`

Property	Description
businessItemLibraryElement: BusinessItemLibraryElement [0..*]	`BusinessItemLibraryElements` that are contained in the `BusinessItemLibrary`

7.2.5.1.2 BusinessItemDefinition Class

A `BusinessItemDefinition` is a standardized definition that is applied to enforce consistency in the definition of `BusinessItems`. Multiple `BusinessItems` that are associated with the same `BusinessItemDefinition` are considered similar from the perspective of the `BusinessItemLibrary`.

SuperClass
`BusinessItemLibraryElement`

Property	Description
isFungible: Boolean = true	If "true", instances of `BusinessItems`, that are defined by the `BusinessItemDefinition`, are interchangeable, otherwise only a particular instance can satisfy a need.
isShareable: Boolean = false	If "true", instances of the `BusinessItems` that are defined by the `BusinessItemDefinition` can be used simultaneously in multiple locations.
category: BusinessItemCategory [0..*]	Zero or more `BusinessItemCategories` to which the `BusinessItemDefinition` belongs.
practiceDefinition: PracticeDefinition [0..*]	`PracticeDefinitions` that specify the use of a `resource` that is defined by the `BusinessItemDefinition` (see 7.2.5.4).
capabilityDependency: CapabilityDependency [0..*]	The `CapabilityDependencies` that refer to the `BusinessItemDefinition` as their `deliverableDefinition` (see 7.2.5.3.5).
supportedCapability: CapabilityDefinition [0..*]	The `CapabilityDefinitions` that refer to the `BusinessItemDefinition` as their `capabilityResourceDefinition` (see 7.2.5.3.2).

Constraints

- If a `BusinessItemDefinition` belongs to a `BusinessItemCategory`, the `BusinessItemCategory` MUST be contained in the same `BusinessItemLibrary` that also contains the `BusinessItemDefinition`.

7.2.5.1.3 BusinessItemCategory Class

A `BusinessItemCategory` is a collection of `BusinessItemDefinitions` similar enough to group them together, and which maybe be part of a hierarchy of `BusinessItemCategories`.

SuperClass
`BusinessItemLibraryElement`

Property	Description
categoryItem: BusinessItemDefinition [0..*]	`BusinessItemDefinitions` that belong to the `BusinessItemCategory`.
parentCategory: BusinessItemCategory [0..*]	Parent `BusinessItemCategories` of the `BusinessItemCategory`, in the hierarchy of `BusinessItemCategories`.
childCategory: BusinessItemCategory [0..*]	Child `BusinessItemCategories` of the `BusinessItemCategory`, in the hierarchy of `BusinessItemCategories`.

Constraints

- A `BusinessItemCategory` MUST be contained in the same `BusinessItemLibrary` as `parentCategories` and `childCategories` (if defined) of the BusinessItemCategory.

7.2.5.1.4 BusinessItemLibraryElement Class (Abstract)

A `BusinessItemLibraryElement` is a generalization of `BusinessItemDefinition` and `BusinessItemCategory`. It has been introduced to enable `InputPorts` (see 7.2.4.3.3) and `OutputPorts` (see 7.2.4.3.2) to refer to either of these.

SuperClass
`VdmlElement`

Property	Description
characteristicDefinition: Characteristic [0..*]	`Characteristics` from `MeasureLibraries` (as specified by SMM) that may suggest `MeasuredCharacteristics` that `BusinessItems` and `Ports` may have that are standardized by reference to the `BusinessItemLibraryElement`. `MeasuredCharacteristics` of these `BusinessItems` and `Ports` may then refer to the `Characteristics` in these `MeasureLibraries`. `Measures`, in these `MeasureLibraries` associated with these `Characteristics`, may then suggest how `Measurement` is achieved.

7.2.5.2 ValueLibrary

Figure 7-24 defines the library structure for `ValueDefinitions` referenced by `ValueElements` elements.

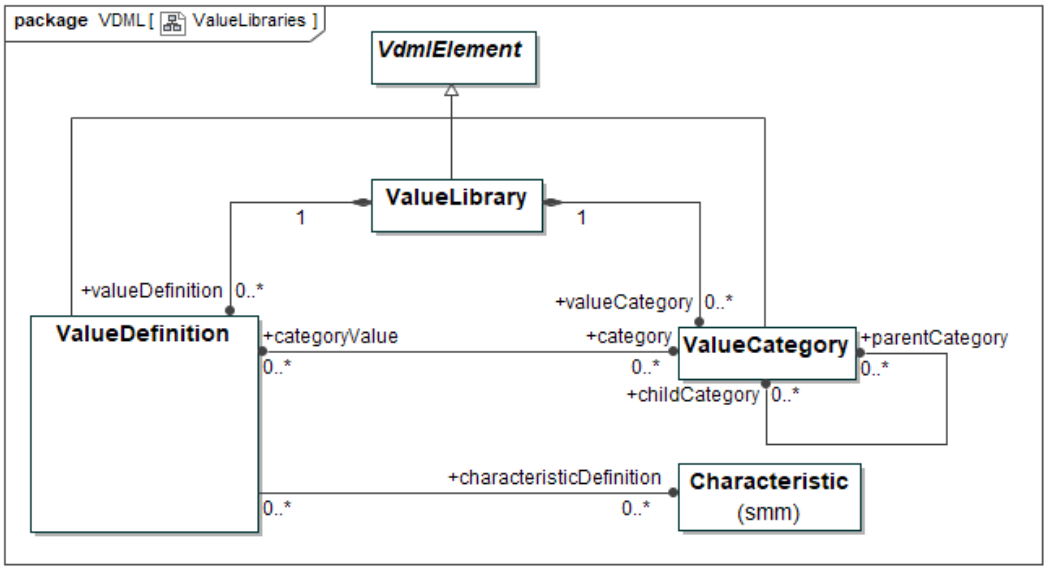

Figure 7-24: ValueLibraries

7.2.5.2.1 ValueLibrary Class

A `ValueLibrary` contains a taxonomy of values, consisting of `ValueDefinitions` and categories of them, and is applied to enforce consistency in the definition of `ValueElements` (see 7.2.1.3.4). Multiple `ValueElements` that are associated with the same `ValueDefinition`, are considered similar from the perspective of the `ValueLibrary`.

SuperClass
`VdmlElement`

Property	Description
valueDefinition: ValueDefinition [0..*]	`ValueDefinitions` that are contained in the `ValueLibrary`
valueCategory: ValueCategory [0..*]	`ValueCategories` that are contained in the `ValueLibrary`

7.2.5.2.2 ValueDefinition Class

A `ValueDefinition` is a standardized definition that is applied to enforce consistency in the definition of `ValueElements` (see 7.2.1.3.4). Multiple `ValueElements` that are associated with the same `ValueDefinition` are considered similar from the perspective of the `ValueLibrary`.

SuperClass
`VdmlElement`

Property	Description
category: ValueCategory [0..*]	Zero or more `ValueCategories` to which the `ValueDefinition` belongs.
characteristicDefinition: Characteristic [0..*]	`Characteristics` from `MeasureLibraries` (as specified by SMM) that may suggest `MeasuredCharacteristics` that `ValueElements` may have that are standardized by reference to the `ValueDefinition`. `MeasuredCharacteristics`

	of these ValueElements may then refer to these Characteristics in these MeasureLibraries. Measures, in these MeasureLibraries associated with these Characteristics, may then suggest how Measurement is achieved.

Constraints

- If a ValueDefinition belongs to a ValueCategory, the ValueCategory MUST belong to the same ValueLibrary that also contains the ValueDefinition.

7.2.5.2.3 ValueCategory Class

A ValueCategory is a collection of ValueDefinitions similar enough to group them together, and which maybe be part of a hierarchy of ValueCategories.

SuperClass
VdmlElement

Property	Description
categoryValue: ValueDefinition [0..*]	ValueDefinitions that belong to the ValueCategory
parentCategory: ValueCategory [0..*]	Parent ValueCategories of the ValueCategory, in the hierarchy of ValueCategories
childCategory: ValueCategory [0..*]	Child ValueCategories of the ValueCategory, in the hierarchy of ValueCategories

Constraints

- A ValueCategory MUST be contained in the same ValueLibrary as parentCategories and childCategories (if defined) of the ValueCategory.

7.2.5.3 CapabilityLibrary

Figure 7-25 defines the library structure for CapabilityDefinitions referenced by CapabilityMethods, CapabilityOffers and Activities.

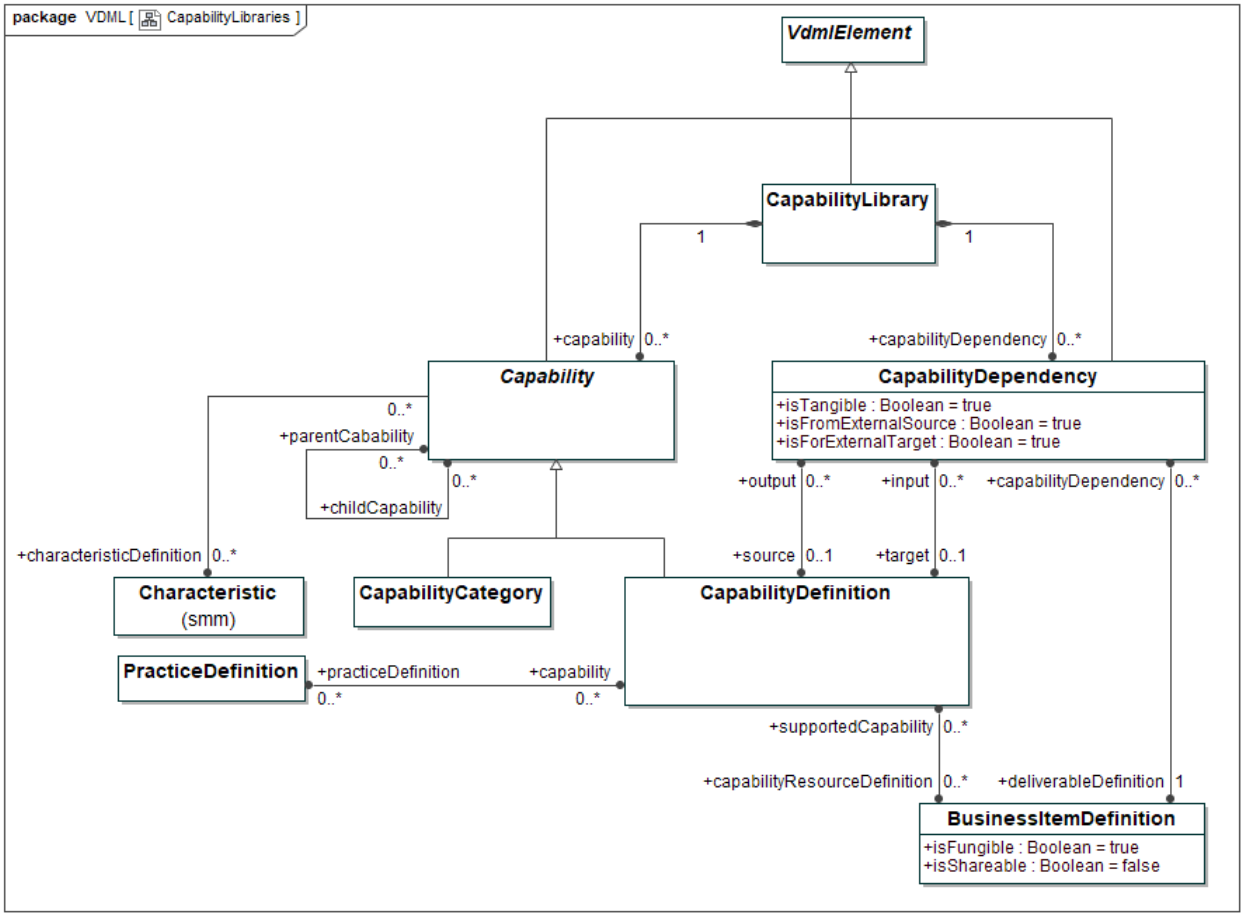

Figure 7-25: CapabilityLibraries

7.2.5.3.1 CapabilityLibrary Class

A `CapabilityLibrary` contains a taxonomy of `Capabilities`, consisting of `CapabilityDefinitions` and categories of them, and is applied to enforce consistency in the definition of `Capabilities`. Multiple `CapabilityOffers` that are associated with the same `Capability` are considered similar from the perspective of the `CapabilityLibrary`. Similarly, multiple `Activities` that are associated with the same `Capability` are considered to have similar requirement from the perspective of the `CapabilityLibrary`.

A `Capability` is the ability to perform a particular kind of work and deliver desired value.

SuperClass
`VdmlElement`

Property	Description
capability: Capability [0..*]	Standardized `Capabilities` that are contained in the `CapabilityLibrary`. These `Capabilities` are either `CapabilityDefinitions` or `CapabilityCategories`.
capabilityDependency: CapabilityDependency [0..*]	`CapabilityDependencies` that are contained in the `CapabilityLibrary`.

7.2.5.3.2 CapabilityDefinition Class

A CapabilityDefinition is a standardized definition that is applied to enforce consistency in the definition of CapabilityOffers of OrgUnits and capabilityRequirements of Activities. Multiple CapabilityOffers that are associated with the same CapabilityDefinition are considered similar from the perspective of the CapabilityLibrary, and multiple Activities that are associated with the same CapabilityDefinition are considered to have similar requirement from the perspective of the CapabilityLibrary.

SuperClass
Capability

Property	Description
output: CapabilityDependency [0..*]	Dependency that other Capabilities, identified by their respective CapabilityDefinitions, might have on the Capability, identified by the CapabilityDefinition, in that they require an output from it, which output is identified by a deliverableDefinition that is related to the CapabilityDependency.
input: CapabilityDependency [0..*]	Dependency that the Capability, identified by the CapabilityDefinition, might have on other Capabilities, identified by their respective CapabilityDefinitions, in that it requires an input from them, which input is identified by a deliverableDefinition that is related to the CapabilityDependency.
capabilityResourceDefinition: BusinessItemDefinition [0..*]	Definition of resources that may be required to support CapabilityOffers that are defined by reference to the CapabilityDefinition.
practiceDefinition: PracticeDefinition [0..*]	Definition of practices that may be implemented, completely or in part, by enforcing the Activities that require Capabilities that are defined by the CapabilityDefinition, or by the CapabilityMethods that support CapabilityOffers that are defined by the CapabilityDefinition.

7.2.5.3.3 CapabilityCategory Class

A CapabilityCategory is a collection of CapabilityDefinitions similar enough to group them together, and which maybe be part of a hierarchy of CapabilityCategories.

SuperClass
Capability

7.2.5.3.4 Capability Class (Abstract)

A Capability element in a CapabilityLibrary is an abstract element that might represent a CapabilityDefinition or a CapabilityCategory, introduced to enable enforcement of standardized Capabilities and standardized capabilityRequirements by reference to either one.

For some Activities, especially in early stages of analysis and design, it is practical to define a capabilityRequirement by reference to a CapabilityCategory. Later on the requirement might be refined by referring to a distinct CapabilityDefinition in the CapabilityCategory. Similarly, some CapabilityOffers might be best defined by reference to a CapabilityCategory, whereas others can be defined by reference to a CapabilityDefinition. Experience with commonly known industry standard CapabilityLibraries have learned that it is often difficult to make a strict distinction between CapabilityDefinitions and CapabilityCategories. For these reasons, reference to an element that represents either a CapabilityDefinition or a CapabilityCategory, is useful.

SuperClass
VdmlElement

Property	Description
characteristicDefinition: Characteristic [0..*]	Characteristics from MeasureLibraries (as specified by SMM) that may suggest MeasuredCharacteristics that Activities may have that define their capabilityRequirement by reference to the Capability, or that CapabilityOffers may have that offer the Capability. MeasuredCharacteristics of these Activities or CapabilityOffers may then refer to these Characteristics in these MeasureLibraries. Measures, in these MeasureLibraries associated with these Characteristics, may then suggest how Measurement is achieved.
parentCapability: Capability [0..*]	Parent Capabilities of the Capability, in the hierarchy of Capabilities.
childCapability: Capability [0..*]	Child Capabilities of the Capability, in the hierarchy of Capabilities.

7.2.5.3.5 CapabilityDependency Class

A CapabilityDependency suggests a possible dependency between two Capabilities, which dependency clarifies that one Capability requires a deliverable that is provided by the other.

A CapabilityDependency might suggest a DeliverableFlow between two Activities, when these Activities require the Capabilities that are defined by the CapabilityDefinitions that are dependent on each other via the CapabilityDependency. When an OrgUnit provides a CapabilityOffer, or a CapabilityMethod supports a CapabilityOffer, for the Capability that is identified by a CapabilityDefinition that relates to a CapabilityDependency, the deliverableDefinition of that CapabilityDependency might suggest a Port that the OrgUnit or the CapabilityMethod might require.

SuperClass
VdmlElement

Property	Description
deliverableDefinition: BusinessItemDefinition [1]	Definition of the `deliverable`, the transfer of which is suggested by the `CapabilityDependency`.
source: CapabilityDefinition [0..1]	`CapabilityDefinition` that defines the `Capability` that is applied to produce the `deliverable`.
target: CapabilityDefinition [0..1]	`CapabilityDefinition` that defines the `Capability` that, when applied uses or consumes the `deliverable`.
isTangible: Boolean = true	If "true", the `deliverable`, as defined by the `deliverableDefinition`, represents something that is contracted, mandated or expected by the `recipient` and which may generate revenue. If "false", the `deliverable`, as "intangible", represents something that is unpaid or non-contractual or that make things work smoothly and help build relationships (see Allee (2008)).
isFromExternalSource: Boolean = true	The `source` of the `CapabilityDependency` is not defined explicitly. This suggests that, when an `Activity` requires a `Capability` that is defined by the `target`, the `deliverable` may be provided by a `Store` or is `target` of an `InputDelegation`.
isForExternalTarget: Boolean = true	The `target` of the `CapabilityDependency` is not defined explicitly. This suggests that, when an `Activity` requires a `Capability` that is defined by the `source`, the `deliverable` maybe received by a `Store`, or is `target` of an `OutputDelegation`.

Constraints

- A `CapabilityDependency` with external source (i.e., `isFromExternalSource` = true) MUST NOT have a `source CapabilityDefinition` connected.
- A `CapabilityDependency` with external target (i.e., `isForExternalTarget` = true) MUST NOT have a `target CapabilityDefinition` connected.

7.2.5.4 PracticeLibrary

A `PracticeLibrary` contains a taxonomy of Practices, consisting of `PracticeDefinitions` and categories of them, and is applied to enforce consistency in the definition of what Practices are implemented by `CapabilityMethods` and/or `Activities`.

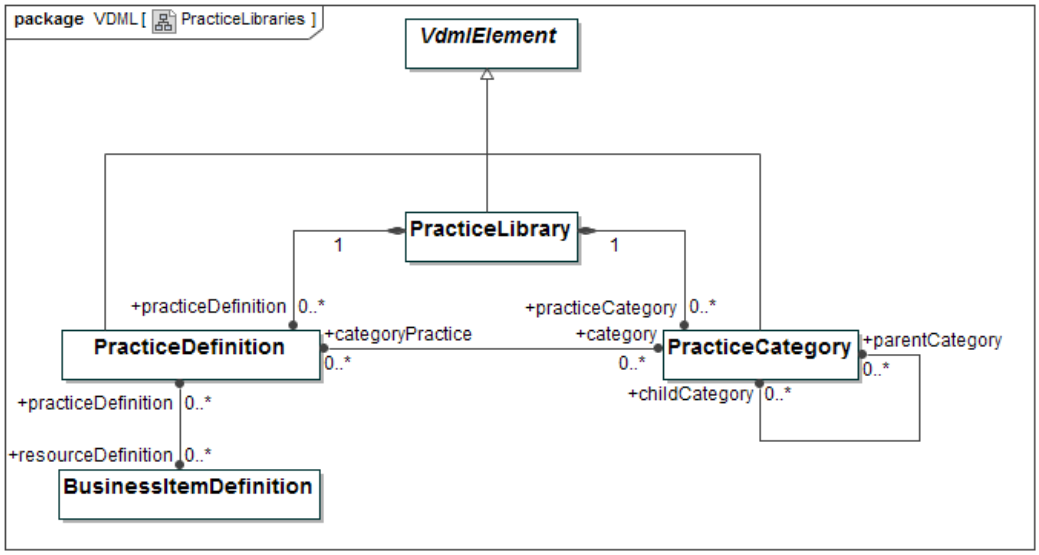

Figure 7-26: PracticeLibraries

7.2.5.4.1 PracticeLibrary Class

A `practice` is a definition of a proven way to handle specific types of work and that have been successfully used by multiple organizations.

SuperClass
`VdmlElement`

Property	Description
practiceDefinition: PracticeDefinition [0..*]	`PracticeDefinitions` that are contained in the `PracticeLibrary`.
practiceCategory: PracticeCategory [0..*]	`PracticeCategories` that are contained in the `PracticeLibrary`.

7.2.5.4.2 PracticeDefinition Class

A `PracticeDefinition` is a standardized definition that is applied to enforce consistency in the definition of what `practices` are implemented by `CapabilityMethods` and/or `Activities`.

SuperClass
`VdmlElement`

Property	Description
category: PracticeCategory [0..*]	Zero or more `PracticeCategories` to which the `PracticeDefinition` belongs.
resourceDefinition: BusinessItemDefinition [0..*]	Definitions of `resources` that `Activities` or `CapabilityMethods` may require in order to comply with the `practice` that is identified by the `PracticeDefinition` that they implement.
capability: capabilityDefinition [0..*]	The `CapabilityDefinitions` that refer to the `PracticeDefinition`.

Constraints

- If a `PracticeDefinition` belongs to a `PracticeCategory`, the `PracticeCategory` MUST belong to the same `PracticeLibrary` that also contains the `PracticeDefinition`.

7.2.5.4.3 PracticeCategory Class

A `PracticeCategory` is a collection of `PracticeDefinitions` similar enough to group them together, and which maybe be part of a hierarchy of `PracticeCategories`.

SuperClass
`VdmlElement`

Property	Description
categoryPractice: PracticeDefinition [0..*]	`PracticeDefinitions` that belong to the `PracticeCategory`.
parentCategory: PracticeCategory [0..*]	Parent `PracticeCategories` of the `PracticeCategory`, in the hierarchy of `PracticeCategories`.
childCategory: PracticeCategory [0..*]	Child `PracticeCategories` of the `PracticeCategory`, in the hierarchy of `PracticeCategories`.

Constraints

- A `PracticeCategory` MUST be contained in the same `PracticeLibrary` as `parentCategories` and `childCategories` (if defined) of the `PracticeCategory`.

7.2.5.5 RoleLibrary

Figure 7-27 shows the meta-model of `RoleLibraries`. A `RoleLibrary` contains a taxonomy of `Roles`, consisting of `RoleDefinitions` and categories of them, and is applied to enforce consistency in the definition of what `Roles` are defined as parts of `Collaborations`.

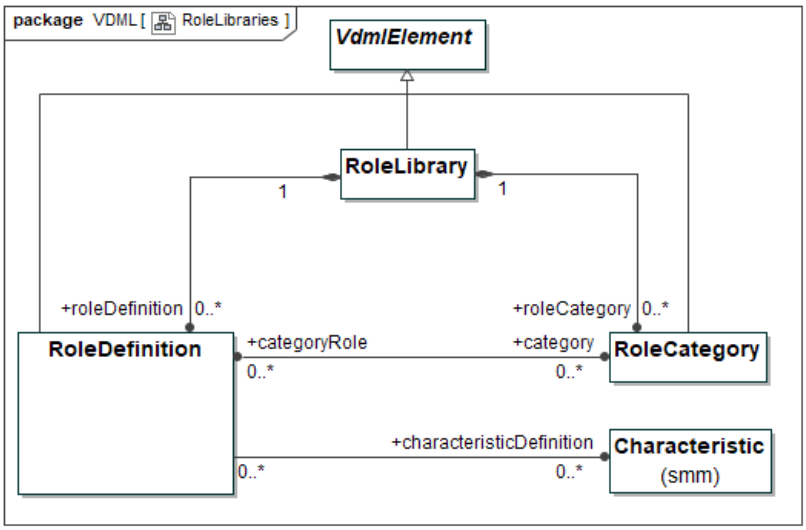

Figure 7-27: RoleLibraries

7.2.5.5.1 RoleLibrary Class

A RoleLibrary contains a taxonomy of Roles, consisting of RoleDefinitions and categories of them, and is applied to enforce consistency in the definition of Roles. Multiple Roles that are associated with the same RoleDefinition are considered similar from the perspective of the RoleLibrary.

SuperClass
VdmlElement

Property	Description
roleDefinition: RoleDefinition [0..*]	RoleDefinitions that are contained in the RoleLibrary.
roleCategory: RoleCategory [0..*]	RoleCategories that are contained in the RoleLibrary.

7.2.5.5.2 RoleDefinition Class

A RoleDefinition is a standardized definition that is applied to enforce consistency in the definition of Roles. Multiple Roles that are associated with the same RoleDefinition are considered similar from the perspective of the RoleLibrary.

SuperClass
VdmlElement

Property	Description
category: RoleCategory [0..*]	Zero or more RoleCategories to which the RoleDefinition belongs.
characteristicDefinition: Characteristic [0..*]	Characteristics from MeasureLibraries (as specified by SMM) that may suggest MeasuredCharacteristics that Roles may have that are standardized by reference to the RoleDefinition. MeasuredCharacteristics of these Roles may then refer to these Characteristics in these MeasureLibraries. Measures, in these MeasureLibraries associated with these Characteristics, may then suggest how Measurement is achieved.

Constraints

- If a RoleDefinition belongs to a RoleCategory, the RoleCategory MUST belong to the same RoleLibrary that also contains the RoleDefinition.

7.2.5.5.3 RoleCategory Class

A RoleCategory is a collection of RoleDefinitions similar enough to group them together, and which maybe be part of a hierarchy of RoleCategories.

SuperClass
VdmlElement

Property	Description
categoryRole: RoleDefinition [0..*]	RoleDefinitions that belong to the RoleCategory.
parentCategory: RoleCategory [0..*]	Parent RoleCategories of the RoleCategory, in the hierarchy of RoleCategories.

childCategory: RoleCategory [0..*]	Child `RoleCategories` of the `RoleCategory`, in the hierarchy of `RoleCategories`.

Constraints

A `RoleCategory` MUST be contained in the same `RoleLibrary` as `parentCategories` and `childCategories` (if defined) of the `RoleCategory`.

7.2.6 Integration with SMM (Structured Metrics Metamodel)

This sub-clause defines the relationship between VDML and SMM. SMM provides the representation of `Measurements` within VDML `Scenarios` (SMM `Observations`).

7.2.6.1 Packages

The VDML metamodel is contained in a single package, called VDML.

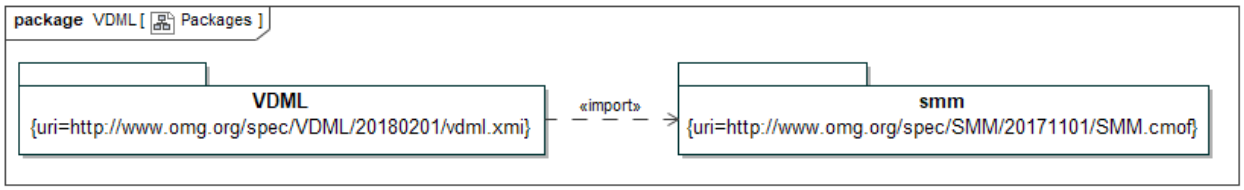

Figure 7-28: VDML Metamodel package

For purpose of support of integrated measurement of performance and value characteristics, VDML depends on the Structured Metrics Metamodel (SMM), as specified in SMM (2017). The SMM metamodel package is imported in the VDML metamodel package, as indicated in the package diagram.

The next sub-clause will provide a high level summary of the main SMM concepts as far as is required to understand VDML.

7.2.6.2 SMM Main Concepts

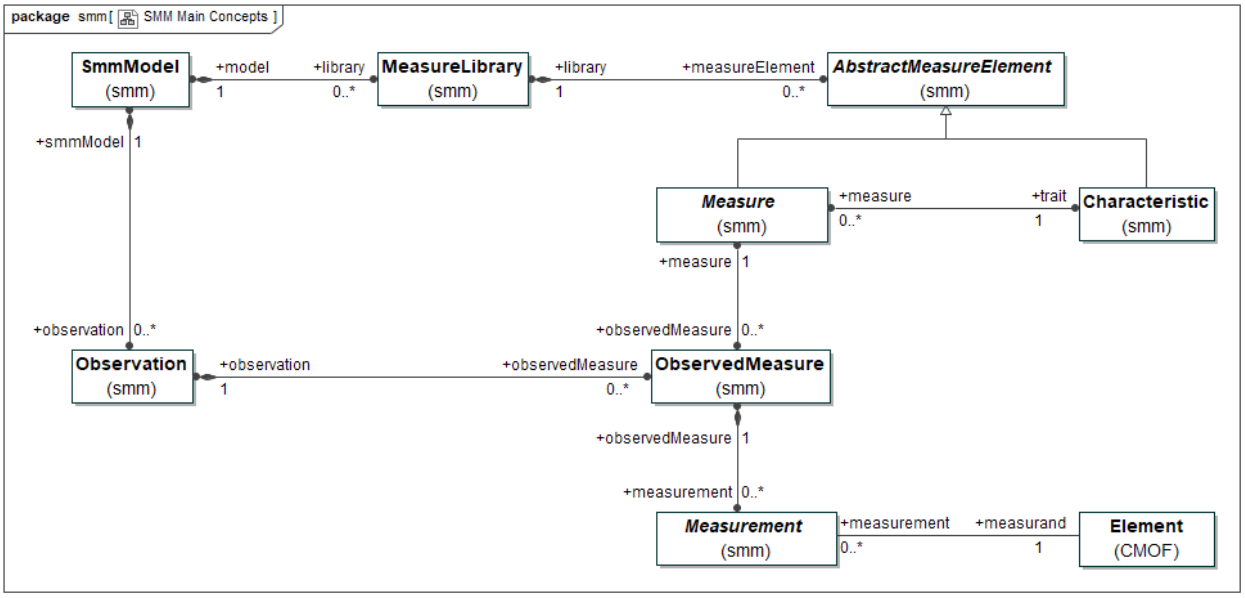

Figure 7-29: SMM main concepts

Details of SMM classes and the MOF (or CMOF) class, as represented in Figure 7-29, as well as, where appropriate, in other diagrams in this document, will not be specified in the VDML specification itself.

The reader can refer to SMM (2017) and MOF (2013) for details.

For convenience, we use the following definitions, as popularized versions of more formal and technical definitions in SMM (2017):

- A `Measure` is method that is applied to characterize an attribute of something by assigning a comparable quantification or qualification.
- A `Measurement` is the result of applying a `Measure`.
- A `Characteristic` is a distinguishing feature or quality that can be qualified or quantified by applying a `Measure`.

Basically, an SMM model contains zero or more `MeasureLibraries` and zero or more `Observations`. A `MeasureLibrary` contains, amongst others, defined `Measures`, which may be (re-)used to determine `Measurements` in different contexts. Each `Measure` is defined against a `Characteristic` (or "`trait`"), which is contained in the `MeasureLibrary` as well. A `unit` is defined per each `Measure`. `Observations` contain `Measurements`, via so-called `ObservedMeasures`, which are applications of `Measures` to obtain these `Measurements`. A `Measurement` contains the result of applying the `Measure` to a `measurand` (i.e., any object that is measured). In SMM a `measurand` is defined as `Element` (as specified by CMOF). `Element` (from CMOF) represents anything that can be modeled. In VDML a `measurand` is more narrowly defined (see 7.2.4.1).

The reader should refer to SMM (2017) for a total and detailed specification of SMM.

This page intentionally left blank.

8 Notation

8.1 General

VDML notation is provided by diagrams that cover the following areas:

- `Role` Collaboration
- `ValueProposition` Exchange
- `Activity` Network
- `Collaboration` Structure
- `CapabilityLibrary`
- `Capability` Heatmap
- `Capability` Management
- `Measurement` Dependency

These are specified in subsequent sub-clauses in this clause. Next to distinct notation elements, also a few examples diagrams are provided that apply these elements in combination. Though some of these example diagrams might apply colors, colors do not imply any normative semantics.

8.2 Role Collaboration

The concepts of `Role`, as contained in a `Collaboration`, is fundamental to VDML, and thus a `Role` is represented on most of the diagrams.

In an `Activity` Network diagram (see 8.4) a `Role` is represented as a swim-lane. In other diagrams, in particular `Role` Collaboration diagram, a `Role` is represented as oval.

The `Role` shape (oval) in Figure 8-1 does not have an expand button, which implies that it is not assigned, or, if it is assigned to an `Actor` (possibly represented as `BusinessItem`) or other `Role`.

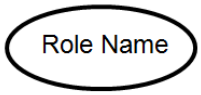

Figure 8-1: Role shape as oval

The `Role` shape in Figure 8-2 has an expand button, to indicate `Assignment` of the `Role` to a `Collaboration`. The `Participant` that fills the `Role` may contain its own `Roles`.

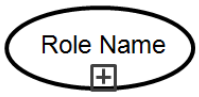

Figure 8-2: Role shape with expand button

Figure 8-3 shows a `DeliverableFlow` with `isTangible` = true (see 7.2.1.2.4). It is shown as solid connector. The `name` along the alongside the connector represents the `name` of the `deliverable` (the `BusinessItem` that is associated as `deliverable`). Optionally a sequence number can be added (behind the double colon), to support “story telling” based on a `Role` Collaboration diagram. A potential sequence number may be derived from the metamodel, but is a convenience feature of the diagram itself, as multiple

Role Collaboration diagrams, on the same underlying Collaboration in the model, might apply different sequence numbers.

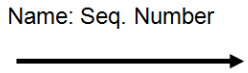

Figure 8-3: DeliverableFlow shape for Tangible

Figure 8-4 shows a DeliverableFlow with isTangible = false (see 7.2.1.2.4). It is shown as dashed connector.

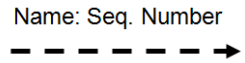

Figure 8-4: DeliverableFlow shape for Intangible

Figure 8-5 represents how, in a Role Collaboration diagram, two Role shapes are connected via a connector that represents a DeliverableFlow.

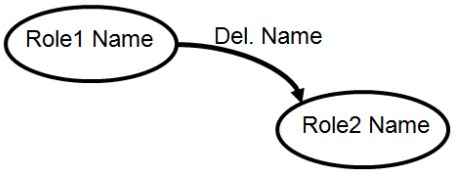

Figure 8-5: DeliverableFlow for Tangible, connecting two Roles

DeliverableFlows connect Ports of two Activities, or a Port of an Activity and a Port of a Store. A Role may perform Activities, defined as its performedWork, and a Role may be responsible for handling inputs or outputs of Stores, which handling is defined by reference to the Store Ports (see 7.2.1.1.5). The detail concerned with Activities, Stores and their Ports is abstracted out of the Role Collaboration diagram. Just the DeliverableFlow connector, as connecting two Roles, is shown. Activity Network diagrams (see 8.4) shows how DeliverableFlows connect Activities or Activities and Stores.

A DeliverableFlow that connects Activities that are performed by the same Role MUST NOT be represented (as connector) in a Role Collaboration diagram. Similarly, a DeliverableFlow that connects an Activity, performed by a Role, and a Store, via a Port for which that same Role is defined as handler (see 7.2.4.3.1), MUST NOT be represented (as connector) in a Role Collaboration diagram. Only connectors that indicate transfer of deliverable between Roles are depicted.

A Role Collaboration diagram might also be used to represent how Roles collaborate (i.e., exchange deliverables) through other Roles in which they serve as Participant.

Figure 8-6 provides, as example, the Role Collaboration diagram for a BusinessNetwork. Transporters place orders, based on which the Manufacturer produces and delivers the product (e.g., trailers). Transporters also provide feedback, in the form of ideas for further innovation of the product. The Manufacturer uses these ideas to apply new innovations and to make these available to the transporters market. The transporters market might be modeled as Community, which fills the Transporter's Party, which is indicated by the expand button on the corresponding Role shape.

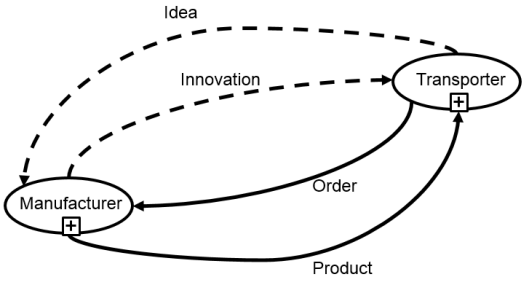

Figure 8-6: Role Collaboration diagram (BusinessNetwork example)

8.3 ValueProposition Exchange

In a `ValueProposition` Exchange diagram, a `ValueProposition` is shown as square (see Figure 8-7). The `ValueProposition` name is placed outside the square.

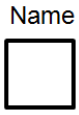

Figure 8-7: ValueProposition shape

Figure 8-8 shows a `Role` providing a `ValueProposition`. The connector represents the association that connects the `Role` and the `ValueProposition` via respectively its `provider`'s and `providedProposition` ends (see 7.2.1.3).

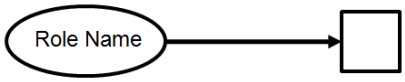

Figure 8-8: Role providing a ValueProposition

Figure 8-9 shows a `Role` receiving a `ValueProposition`. The connector represents the association that connects the `Role` and the `ValueProposition` via respectively its `recipient`'s and `receivedProposition` ends (see 7.2.1.3).

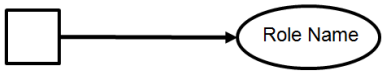

Figure 8-9: Role receiving a ValueProposition

Figure 8-10 shows, as example, the `ValueProposition` Exchange between the `Parties` in the `BusinessNetwork` that is also underlying the `Role` Collaboration diagram in Figure 8-6. Note that the same `Role` might provide and receive multiple `ValuePropositions`, from possibly multiple other `Roles`, thought this simple example only shows two `Roles`, each of which provides just one `ValueProposition`. The name of the `ValueProposition` that is provided by the Manufacturer suggests that it is about value associated with the delivery of trailers (the product) of a certain product family of trailers (here called XTrailer).

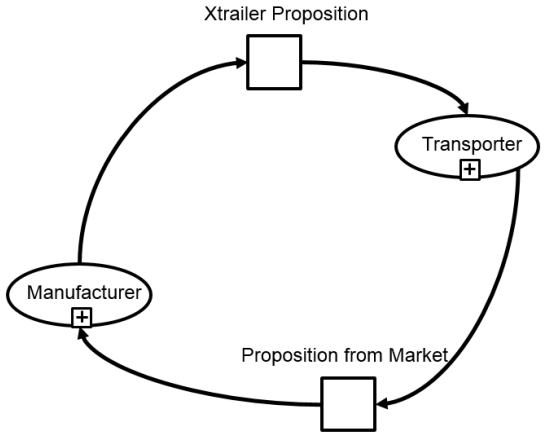

Figure 8-10: ValueProposition Exchange diagram (example)

When compared to the `Role` Collaboration diagram, the `ValueProposition` Exchange diagram provides a more abstract view on the `Collaboration`. `Roles` are depicted in both. A `ValueProposition`, as depicted in a `ValueProposition` Exchange diagram, consists of `ValuePropositionComponents`, which are associated with `ValueAdds` as contained by `OutputPorts` at the `provider`'s end of `DeliverableFlows` that are depicted in the `Role` Collaboration diagram (see Metamodel diagram in Figure 7-12).

8.4 Activity Network

In an `Activity` Network diagram, a `Role` is represented as swim-lane (see Figure 8-11).

Figure 8-11: Swim-lane shape for Role (in Activity Network diagram)

As indicated in Figure 8-12, an `Activity` is shown as rectangle with rounded corners.

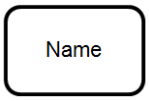

Figure 8-12: Activity shape

An `Activity` shape with expand button (see Figure 8-13) represents an `Activity` that delegates its work to a `Collaboration`.

Figure 8-13: Activity shape, with expand button

Stores and Pools are represented in both Activity Network diagram and Capability Management diagram (see 8.8). A Store is shown as bottom-up pyramid, with its name placed outside the shape (see Figure 8-14).

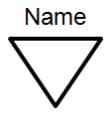

Figure 8-14: Store shape

As indicated in Figure 8-15, a Pool is shown as bottom-up pyramid with re-use marker.

Figure 8-15: Pool shape

In an Activity Network diagram, DeliverableFlows are represented via a solid connector shape (see Figure 8-16). Also here, the name alongside the connector represents the name of the deliverable. Optionally, the text alongside the connector can be extended to not only show the name of the deliverable, but also the state of the deliverable, represented between square brackets. For instance: “patient [hospitalized]”. Whereby that state is modeled as the name of the provider (OutputPort) of the DeliverableFlow. This extended form of text alongside the connector can be used for connectors in a Role Collaboration diagram too. Unlike connectors in a Role Collaboration diagram, in an Activity Network diagram there is no visualization of the distinction between Tangibles and Intangibles, and there is no indication of sequence number of the connector.

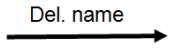

Figure 8-16: Connector shape for DeliverableFlow (in Activity Network diagram)

In an Activity Network diagram, internalPortDelegations (see 7.2.1.1.3) are represented via a dotted connector shape (see Figure 8-17).

Figure 8-17: Connector shape for internalPortDelegation (in Activity Network diagram)

Both DeliverableFlows and internalPortDelegations are used in connection to Ports. The following table shows the shape for a Port, and its variations. These variations depend on whether the corresponding PortContainer is an Activity or Store, or whether it is a Collaboration, and furthermore, whether the Port is used as InputPort (see 7.2.4.3.3) or OutputPort (see 7.2.4.3.2), and whether the Port carries a Condition or planningPercentage (see 7.2.4.3.1), whether the OutputPort carries ValueAdd(s) (see 7.2.4.3.2), and whether the Activity InputPort receives roleResource (see 7.2.1.2.3).

Port Shape	Port Container	Shape description	Shape placement	Input or Output	With Value Add	With Condition and/or planning Percentage	Provides role Resource for Role
	Activity, Store	Small open square	On boundary of shape of Port Container	Either	-	-	-
		Small open square, with splitter		Either	-	√	-
		Small open square, with thick boundary		Input	-	-	√
		Small open square, with splitter and thick boundary		Input	-	√	√
		Small filled square		Output	√	-	-
		Small filled square, with splitter		Output	√	√	-
	Collaboration	Bottom-left open pyramid	Free-floating in Activity Network diagram of Collaboration	Either	-	-	-
		Bottom-left filled pyramid		Output	√	-	-

The following figures show the use of Ports in combination with DeliverableFlows and internalPortDelegations in Activity Network diagrams.

Figure 8-18, Figure 8-19, Figure 8-20, and Figure 8-21 show the possible variations of Activity OutputPort shapes. The solid connector, connecting to the OutputPort, denotes a DeliverableFlow, and the name alongside the connector denotes the name of the deliverable, being the BusinessItem that is associated with the DeliverableFlow.

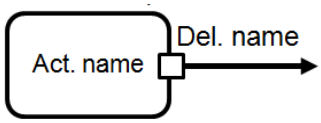

Figure 8-18: shape of OutputPort, on boundary of Activity

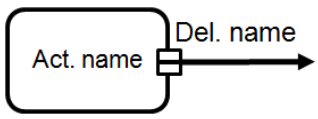

Figure 8-19: Shape of OutputPort, with Condition, on boundary of Activity

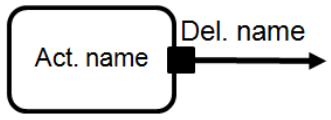

Figure 8-20: Shape of OutputPort, with ValueAdd, on boundary of Activity

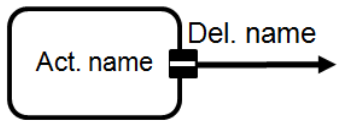

Figure 8-21: Shape of OutputPort, with ValueAdd and Condition, on boundary of Activity

Similarly, Figure 8-22, Figure 8-23, Figure 8-24, and Figure 8-25 show the possible variations of `Activity InputPort` shapes.

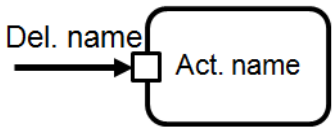

Figure 8-22: Shape of InputPort, on boundary of Activity

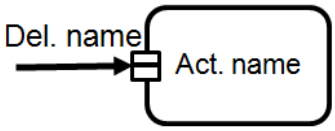

Figure 8-23: Shape of InputPort, with Condition, on boundary of Activity

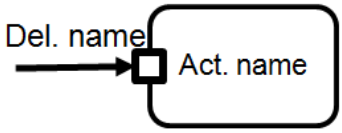

Figure 8-24: Shape of InputPort, receiving roleResource, on boundary of Activity

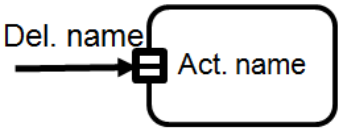

Figure 8-25: Shape of InputPort, receiving role Resource, and with Condition, on boundary of Activity

Figure 8-26, Figure 8-27, Figure 8-28, and Figure 8-29 show the possible variations of `Store OutPort` shapes. Similar shapes are used for `Pools`.

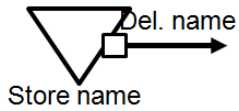

Figure 8-26: Shape of OutputPort, on boundary of Store

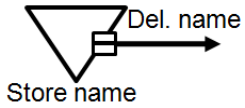

Figure 8-27: Shape of OutputPort, with Condition, on boundary of Store

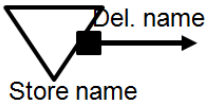

Figure 8-28: Shape of OutputPort, with ValueAdd, on boundary of Store

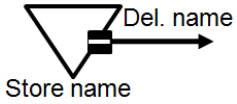

Figure 8-29: Shape of OutputPort, with ValueAdd and Condition, on boundary of Store

Figure 8-30 and Figure 8-31 show the possible variations of `Store InPort` shapes. Note that `Stores` do not apply any `Capability`, and hence do not require a variation of `InputPort` shape that denotes the receipt of `roleResource`.

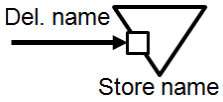

Figure 8-30: Shape of InputPort, on boundary of Store

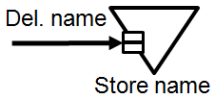

Figure 8-31: Shape of InputPort, with Condition, on boundary of Store

The shape of a `Collaboration InputPort`, as connected via an `internalPortDelegation`, is shown in Figure 8-32. The text that is placed outside the shape indicates the `name` of the `BusinessItemLibraryElement` that is associated as `inputDefinition` (see 7.2.4.3). When an `Activity` Network diagram is shown in a particular `DelegationContext` (see 7.2.3.2.3), the text

indicates the name of the BusinessItem that is associated as deliverable with the DeliverableFlow that connects to the InputPort that is delegated to the Collaboration InputPort.

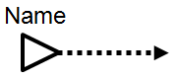

Figure 8-32: Shape of Collaboration InputPort, connected to internalPortDelegation

Figure 8-33 shows the shape of a Collaboration OutPort, as connected via an internalPortDelegation. The text that is placed outside the shape indicates the name of the BusinessItemLibraryElement that is associated as outputDefinition (see 7.2.4.3). When an Activity Network diagram is shown in a particular DelegationContext (see 7.2.3.2.3), the text indicates the name of the BusinessItem that is associated as deliverable with the DeliverableFlow that connects to the OutputPort that is delegated to the Collaboration OutputPort.

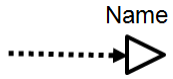

Figure 8-33: Shape of Collaboration OutputPort, connected to internalPortDelegation

Figure 8-34 shows the shape of a Collaboration OutPort, carrying ValueAdd. Also this Port is shown as connected via an internalPortDelegation.

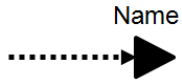

Figure 8-34: Shape of Collaboration OutputPort, with ValueAdd, and connected to internalPortDelegation

Figure 8-35 provides a simple example of an Activity Network diagram.

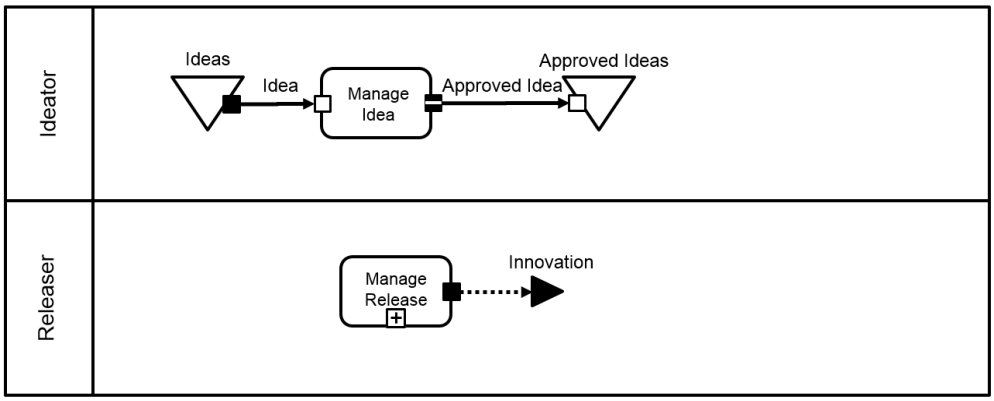

Figure 8-35: Activity Network diagram (simple example)

Another simple Activity Network diagram, demonstrating the application of a partly different set of elements, is given in Figure 8-36.

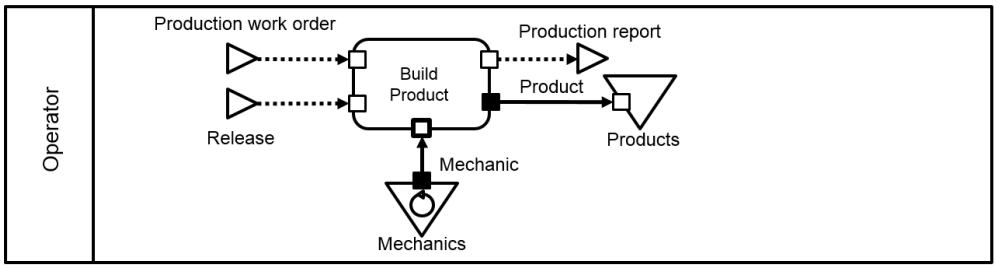

Figure 8-36: Activity Network diagram (simple example)

Note that both a `Role` Collaboration diagram and an `Activity` Network diagram can provide (synchronized) views on the same underlying model of a `Collaboration`. Figure 8-37 extends the `Role` Collaboration diagram example (of Figure 8-6) by also showing the `Activity` Network diagram of the same `BusinessNetwork`. Note that the `DeliverableFlow` that conveys "Order" is not shown in the `Role` Collaboration diagram, as it does not denote transfer of `deliverable` between `Roles`.

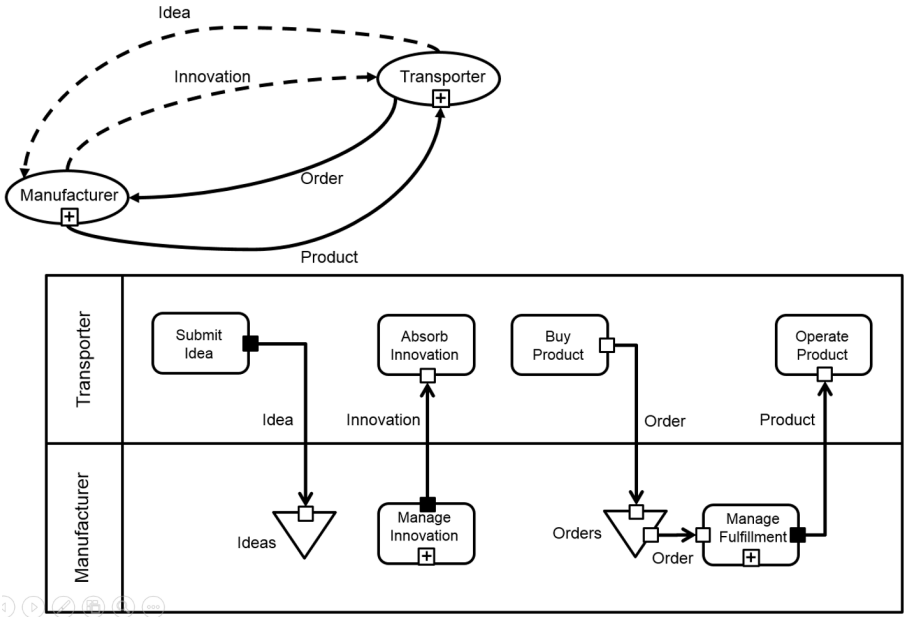

Figure 8-37: Role Collaboration and Activity Network as synchronized views (example)

8.5 Collaboration Structure

Figure 8-38 shows the shape, a squared corner rectangle, by which a `Collaboration` is shown in a `Collaboration` Structure diagram.

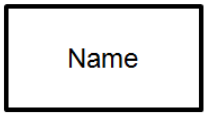

Figure 8-38: Collaboration shape

A BusinessNetwork, being a specialized Collaboration, is shown via a Collaboration shape with a BusinessNetwork marker (see Figure 8-39).

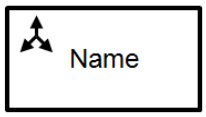

Figure 8-39: BusinessNetwork shape

An OrgUnit, being a specialized Collaboration, is shown via a Collaboration shape with an OrgUnit marker (see Figure 8-40).

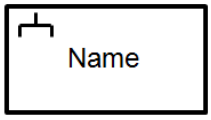

Figure 8-40: OrgUnit shape

A CapabilityMethod, being a specialized Collaboration, is shown via a Collaboration shape with a CapabilityMethod marker (see Figure 8-41).

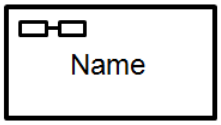

Figure 8-41: CapabilityMethod shape (in Collaboration Structure and Capability Management diagrams)

A Community, being a specialized Collaboration, is shown via a Collaboration shape with a Community marker (see Figure 8-42).

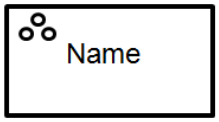

Figure 8-42: Community shape

In a Collaboration Structure diagram, a Role, not assigned to a Collaboration is shown as a Role oval (the same shape as indicated in Figure 8-1).

Containment of a Role in a Collaboration is shown by the solid connector as represented in Figure 8-43.

Figure 8-43: Role containment connector

When a Role, as contained in a Collaboration (e.g., the OrgUnit MyCompany in Figure 8-44), is assigned to another Collaboration (e.g., the OrgUnit R&D in Figure 8-44), the Role containment connector connects to a small oval shape, representing that Role, adjacent to the boundary of the Collaboration at the Participant's end of the connector. The name of the Role is placed outside the Role shape.

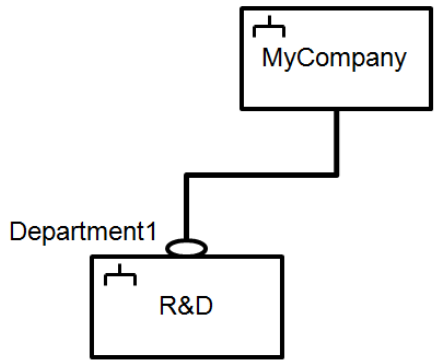

Figure 8-44: Collaboration structure, with Role of parent Collaboration assigned to sub-Collaboration

Figure 8-44 shows how a Collaboration Structure diagram is capable of representing an organization structure, based on the VDML metamodel. Collaboration Structure diagram is more universal than only applicable to organization structure however (see the example in Figure 8-47).

As indicated in Figure 8-45, in a Collaboration Structure diagram, Assignment of a Role to another Role or to an Actor (possibly dynamically determined based on roleResource, see 7.2.1.2.3), is shown as dashed directed connector.

Figure 8-45: Role Assignment connector

Figure 8-46 visualizes Assignment of a Role to an Actor. The Actor is represented by its name only.

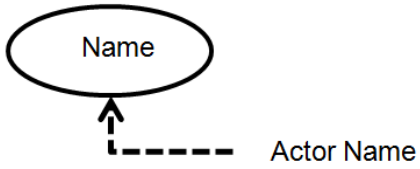

Figure 8-46: Actor assigned to Role

Figure 8-47 provides an example of a Collaboration Structure diagram.

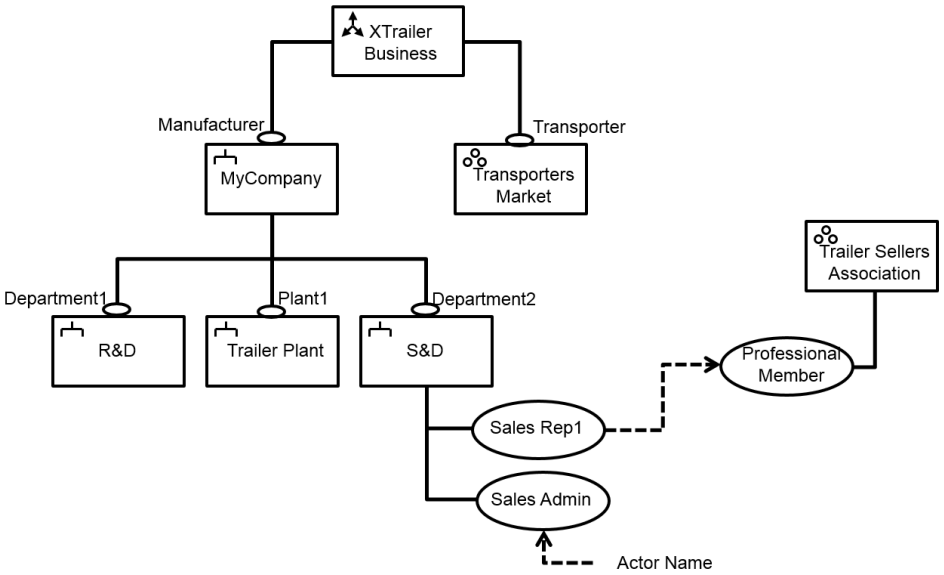

Figure 8-47: Collaboration Structure diagram (example)

8.6 CapabilityLibrary

As indicated in Figure 8-48, in a `CapabilityLibrary` diagram, a `Capability` (see 7.2.5.3) is shown as square corner rectangle.

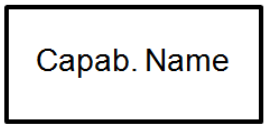

Figure 8-48: Capability shape (in CapabilityLibrary diagram)

The relationship between `parentCapability` and `childCapability` (see 7.2.5.3) is shown via containment of the `childCapability` within the boundary of its `parentCapability` (see Figure 8-49).

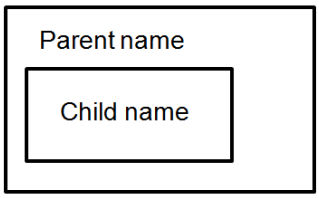

Figure 8-49: Capability hierarchy

Figure 8-50 shows the shape of a `Capability` with expand button.

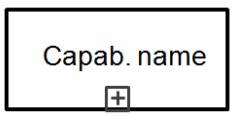

Figure 8-50: Capability shape with expand button (in CapabilityLibrary diagram)

When the shape of a `Capability` is expanded (via the expand button), the shapes of its `childCapabilities` are shown as embedded (i.e., contained within its boundary). Alternatively its `childCapabilities` can be represented on a separate `CapabilityLibrary` diagram.

As Figure 8-51 indicates, a `Capability` shape that shows as expanded, can be collapsed via its expand button (shown with "-"marker, indicating that the `Capability` is shown as expanded).

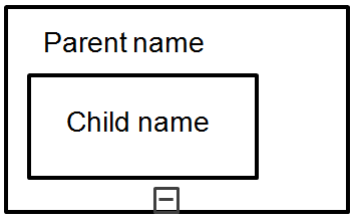

Figure 8-51: Expanded parent Capability, with sub-Capability

Figure 8-52 shows an example of a `CapabilityLibrary` diagram.

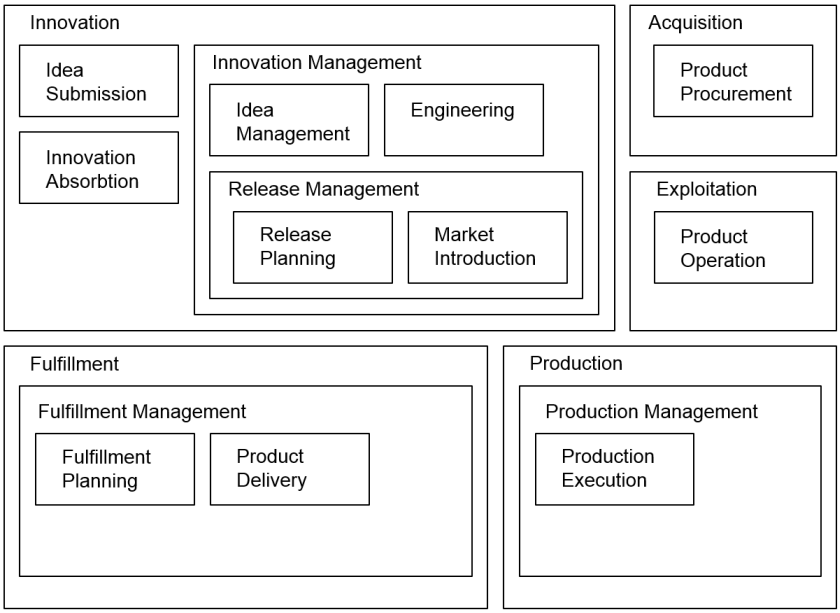

Figure 8-52: CapabilityLibrary diagram (example)

8.7 Capability Heatmap

The notation of a `CapabilityLibrary` diagram can be reused, and further refined, to support `Capability` Heatmaps.

Figure 8-53 visualizes `Capability` hierarchy in a way similar as is indicated in Figure 8-49. The thick boundaries of `Capabilities` "Child1" and "Parent" denote that one or more "Child1"-corresponding `CapabilityOffers` (see 7.2.2.3.3) have their `heatIndex` equal to or above `heatThreshold` (see 7.2.3.2.2).

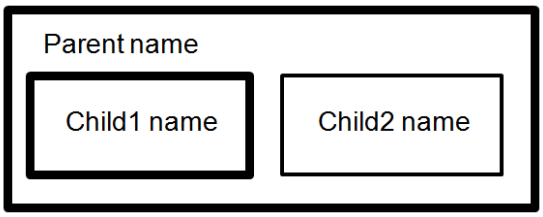

Figure 8-53: Capabilities with heatIndex about HeatThreshold (in Capability Heatmap)

A normal boundary (as for `Capability` "Child2") indicates that `heatIndex` of related `CapabilityOffers` is below `heatThreshold`.

8.8 Capability Management

`CapabilityOffers` are shown in a `CapabilityManagement` diagram. Their shape is a stretched hexagon, with the `CapabilityOffer's name` placed inside.

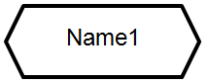

Figure 8-54: CapabilityOffer shape

A dashed connector (see Figure 8-55) is used to represent the association between a `CapabilityOffer` and a `CapabilityMethod` (referenced as `method`) or a `Store` or `Pool` (referenced as `capabilityResource`) that supports it (see 7.2.2.3).

Figure 8-55: Shape of connector between CapabilityOffer and a capabilityResource or method

In a `Capability` Management diagram, an `OrgUnit` is shown as square corner rectangle, with `name` label placed on its boundary. `CapabilityOffers`, representing the `Capabilities` that are provided by the `OrgUnit`, are placed on its boundary as well (see Figure 8-56). An expand button is used to expand the `OrgUnit` (the "+" marker in the expand button in Figure 8-56 indicates that the `OrgUnit` is shown as collapsed.

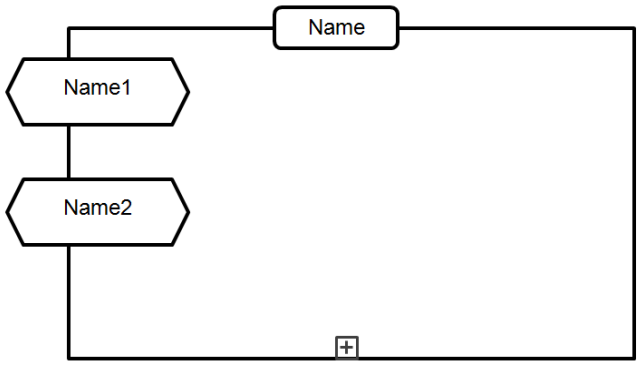

Figure 8-56: CapabilityOffers on boundary of OrgUnit, with expand button (in: Capability Management diagram)

Figure 8-57 shows the same OrgUnit as expanded. When expanded, CapabilityMethods and Stores (or Pools) that support the CapabilityOffers, are shown as contained within the OrgUnit's boundary. When the OrgUnit is expanded, its Position Roles that are assigned to (sub-) OrgUnits, are shown within the OrgUnit's boundary as well, to facilitate navigation to Capability Management diagrams of these sub-OrgUnits. For reasons of diagram scalability, Positions that are associated with Pools (see 7.2.2.3), SHOULD NOT be shown as contained within the OrgUnit's boundary (note that there may be many such Positions). Positions that are not (yet) assigned to Participants MAY be shown inside the boundary.

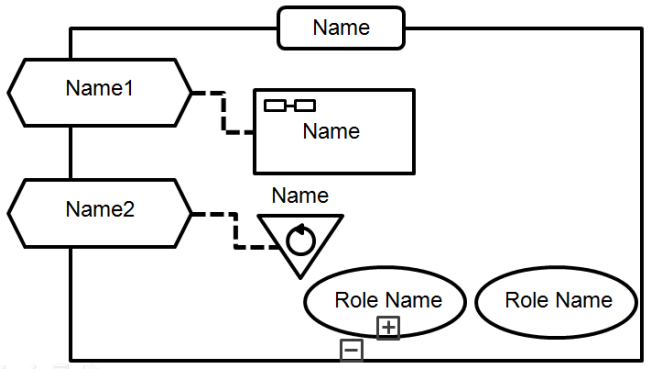

Figure 8-57: OrgUnit expanded (in: Capability Management diagram)

CapabilityOffers of one OrgUnit might be supported by CapabilityMethods or Stores (or Pools) that are owned by other OrgUnits. Figure 8-58 shows this, in a situation where both the OrgUnit that owns a CapabilityOffer and the OrgUnits that own the CapabilityMethod(s) and/or Store(s) (or Pool(s)) that support it, are included in the same Capability Management diagram.

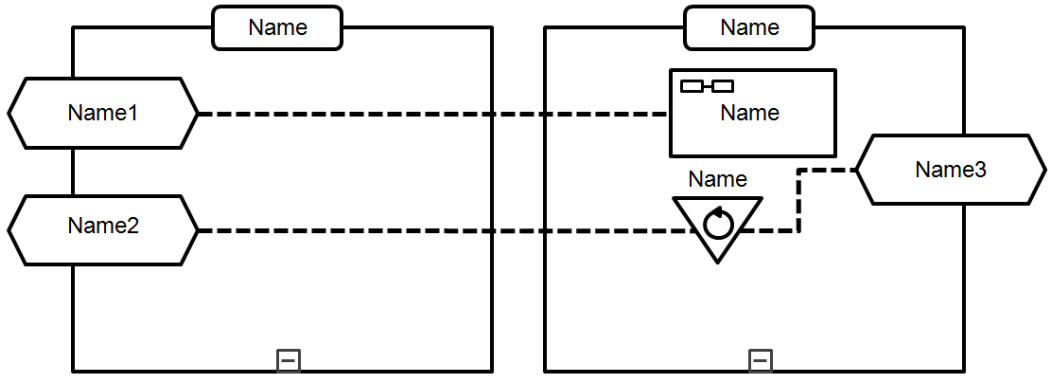

Figure 8-58: CapabilityOffers of OrgUnit with capabilityResource and method from other OrgUnit

An alternative way of modeling is provided in Figure 8-59. Though the CapabilityMethod(s) and/or Store(s) (or Pool(s)) that support CapabilityOffer(s) of another OrgUnit are represented in the Capability Management diagram of the CapabilityOffer(s)-owning OrgUnit, the OrgUnit(s) that own these CapabilityMethod(s) and/or Store(s) (or Pool(s)) are not represented in that diagram.

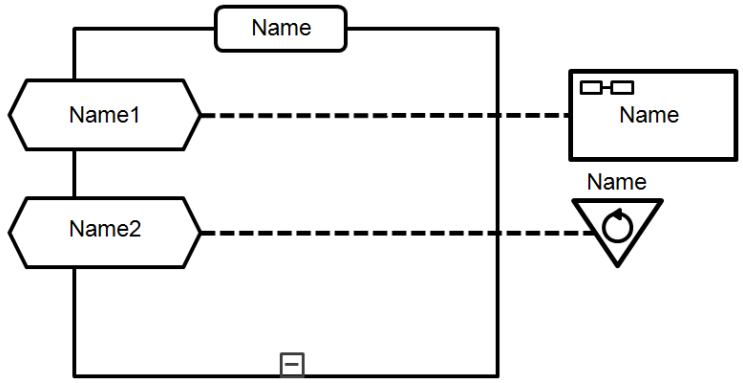

Figure 8-59: CapabilityOffers of OrgUnit with capabilityResource and method from other OrgUnit (not shown)

A `CapabilityMethod`, supporting a `CapabilityOffer`, might depend on other `CapabilityOffers`. It can depend on another `CapabilityOffer` when it contains one or more `Activities`, to which the other `CapabilityOffer` is applied (see 7.2.1.2.1).

Optionally this dependency can be visualized in a `Capability` Management diagram, as a dotted connector (see Figure 8-60).

Figure 8-60: Connector shape for dependency of CapabilityMethod on other CapabilityOffer(s)

Figure 8-61 shows a `Capability` Management diagram, where one `CapabilityMethod` depends on three `CapabilityOffers`, whereby these dependencies are visualized by the dotted connector of Figure 8-60.

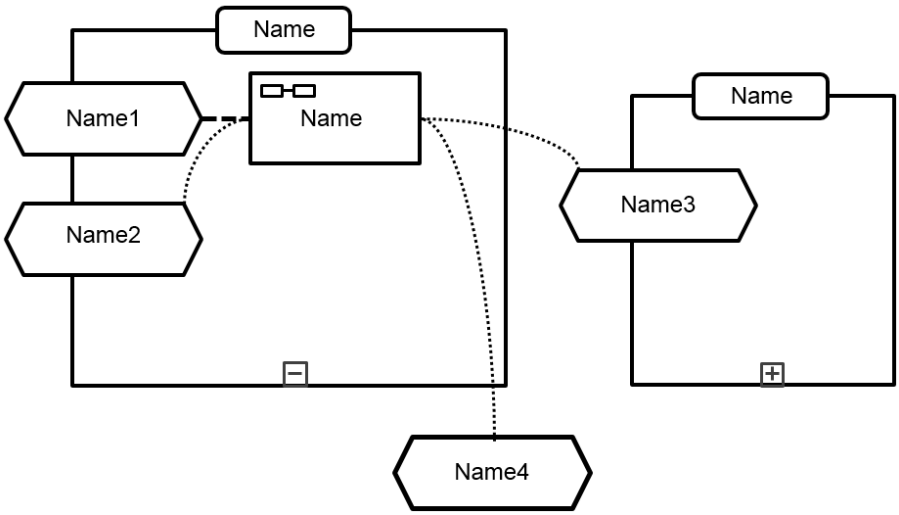

Figure 8-61: Dependencies of CapabilityMethod on CapabilityOffers of methodOwner and other OrgUnits

Figure 8-62 provides an example of a `Capability` Management diagram.

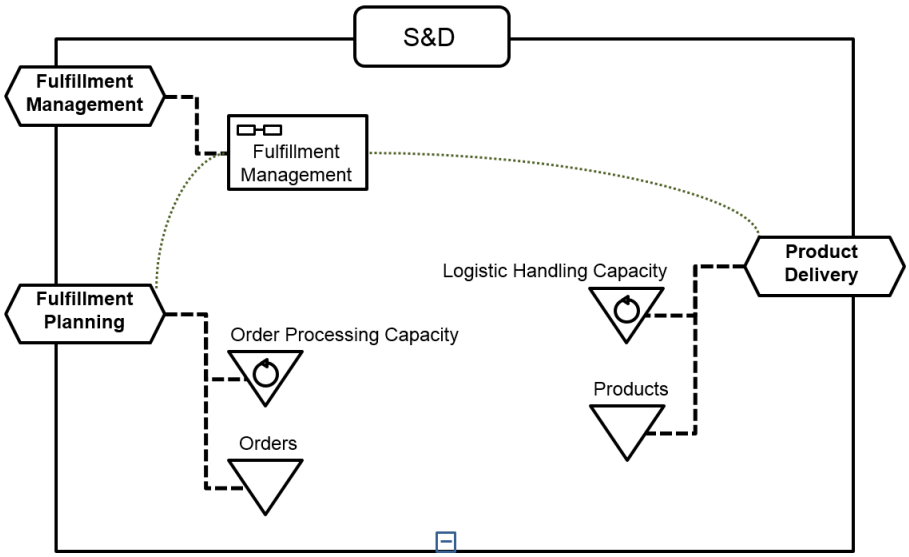

Figure 8-62: Capability Management diagram (example)

8.9 Measurement Dependency

SMM specifies `MeasurementRelationships` between `Measurements`. These relationships represent aggregations, rankings or other transformations, dependent on the particular semantics of the various types of `Measurements` (and their underlying `Measures`) as specified by SMM. VDML associates `Measurements` (as specified by SMM) to `MeasuredCharacteristics` of `MeasuredElements` (e.g., `Activities`, `Ports`, `Stores`, `Collaborations`, `ValueAdds`, `ValuePropositions`...). Different `AnalysisContexts` (see 7.2.3.2.1), via their `Obervation` (as specified by SMM) may enforce different `Measurements` on the same `MeasuredCharacteristic`. Per `AnalysisContext`, or per `Scenario` (including all `AnalysisContexts` in its `AnalysisContext` tree), `MeasurementRelationships` between `MeasuredCharacteristics` can be shown in a `Measurement` Dependency diagram. A `Measurement` Dependency diagram supports understanding of how `Measurements` "influence" other `Measurements`, and thus facilitates detection of root-causes for lack of value contribution or `recipient`'s satisfaction with value (as defined in a `ValueProposition`). A `Measurement` Dependency diagram may also be helpful in visualizing and analyzing simulation results, based on a `Scenario` in a `ValueDeliveryModel`.

In a `Measurement` Dependency diagram, a `MeasuredCharacteristic` (see 7.2.4.1.5) is denoted by a thin-boundary rectangle shape (see Figure 8-63).

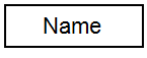

Figure 8-63: MeasuredCharacteristic shape (in Measurement Dependency diagram)

Revision of includes adding a property `influence` to `MeasureRelationship` (the class in SMM that "types" `MeasurementRelationships`). This is a property with enumeration type `Influence`, having enumerated values "positive" and "negative." "Positive" means that, when a `baseMeasurement`'s value increases, the value of the `Measurement` on the other end of the `MeasurementRelationship` will also increase. "Negative" means that, when a `baseMeasurement`'s value increases, the value of the `Measurement` on the other side of the `MeasurementRelationship` will decrease.

In a `Measurement` Dependency diagram, a `MeasurementRelationship` (as specified by SMM), between `Measurements` of two `MeasuredCharacteristics` is denoted by a thin and solid connector. A connector with a marker that contains a "+" symbol denotes a "positive" `influence` (see Figure 8-64).

Figure 8-64: Shape of MeasurementRelationship, with "positive" influence

A connector with a marker that contains a "-" symbol denotes a "negative" `influence` (see Figure 8-65).

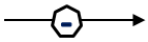

Figure 8-65: Shape of MeasurementRelationship, with "negative" influence

Figure 8-66 provides an example of a `Measurement` Dependency diagram.

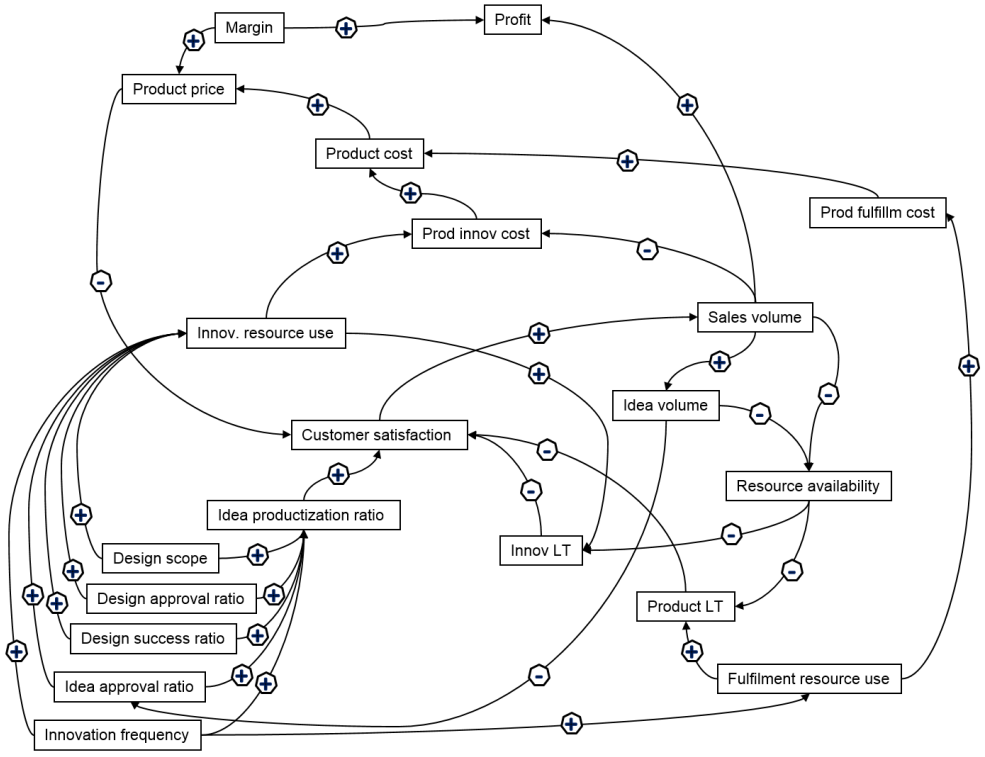

Figure 8-66: MeasurementDependency diagram (example)

Annexes

Normative annexes are integral parts of the standard. An annex's normative status (as opposed to informative) shall be made clear by the way in which it is referred to in the text and under the heading of the annex.

Informative annexes give additional information intended to assist the understanding or use of the standard and shall not contain provisions to which it is necessary to conform in order to be able to claim compliance with the standard. Their presence is optional. An annex's informative status (as opposed to normative) shall be made clear by the way in which it is referred to in the text and under the heading of the annex.

The following Annexes are included:

Annex A: Glossary

Annex B: Alignment with Existing Business Modeling Techniques

Annex C: Use Cases

Annex A: Glossary

(Normative)

Each definition indicates if it is VDML-specific (it's source being VDML), or its origin from another source. Several terms that are listed in the glossary denote commonly known concepts that have origins from outside VDML. VDML avoids introducing new terms unnecessarily, but uses terms that people can relate to. While the concepts are consistent, VDML provides definitions that are clear and in the context of a VDML model. This ensures that both implementers and users will have a clear and consistent understanding of VDML concepts and their relationships.

Activity - Work contributed to a collaboration by a participant in a Role of the collaboration. A role may be filled by another collaboration and a role may contribute to multiple activities in the same collaboration (source: VDML; example of use of term outside VDML: Osterwalder (2004)).

Activity network - A network of activities of participants in a collaboration that are lined by deliverable flows (source: VDML).

Actor - An individual (indivisible) participant, which might be human (a person) or non human (e.g., a software agent or machine) (source: VDML; example of use of term outside VDML: Gordijn and Akkermans (2003)).

AnalysisContext - An AnalysisContext defines a set of measurements associated with a particular use of a collaboration or a store used as a decoupling point between collaborations. When an activity delegates to a collaboration, an AnalysisContext, specialized as Delegation Context,defines the delegations of activity inputs and/or outputs to/from collaboration inputs and/or outputs, and may define assignments of roles within the collaboration (source: VDML).

Attribute - An attribute allows information to be attached to any VDML element in the form of a name-value pair. Attributes provide a simple mechanism to add user defined information to model elements (source: VDML).

Business Item - A business item is anything that can be acquired or created, that conveys information, obligation or other forms of value and that can be conveyed from a provider to a recipient. For example, it includes parts, products, units of fluids, orders, emails, notices, contracts, currency, assignments, devices, property and other resources (source: VDML).

Business Model - A business model describes the rationale of how an organization creates, delivers, and captures value (source: Osterwalder and Pigneur (2010) Lindgren (2011)).

Business Network - A collaboration between independent business (or economic) entities, potentially companies, agencies, individuals or anonymous members of communities of independent business entities, participating in an economic exchange (source: VDML; example of use of term outside VDML: Vervest et al. (2009)).

Capability - Ability to perform a particular kind of work and deliver desired value (source: VDML; examples of use of term outside VDML: Osterwalder (2004), SoaML (2012), ITIL (2011)).

Capability Method - A collaboration specification that defines the activities, deliverable flows, business items, capability requirements and roles that deliver a capability and associated value contributions. For each application of the capability method, within a scenario or in multiple scenarios, there may be distinct measurements of performance and value contributions and role assignments suitable to the application context. A capability method does not own resources but receives them from other sources in the course of performing its activities (source: VDML). An activity does not delegate directly to a capability method but engages it through its organization unit based on a capability offer.

Channel - Mechanism to execute a deliverable flow, such as e-mail, face-to-face conversation, SOAP, REST, physical transportation, postal service, telephone, fax, FTP, etc. (source: VDML).

Characteristic - Distinguishing feature or quality that can be qualified or quantified by applying a measure (popularized version of definition in SMM (2017)).

Collaboration - Collection of participants joined together for a shared purpose or interest (source: VDML; examples of use of term outside VDML: SoaML (2012), BPMN (2011)).

Community - A loose collaboration of participants with similar characteristics or interests (source: VDML; examples of use of term outside VDML: Weill and Vitale (2001)).

Delegation Context - A specialized AnalysisContext, set by an activity and in which the activity delegates its work to a collaboration. A delegation context also defines the delegations of activity inputs and/or outputs to/from collaboration inputs and/or outputs, and may define assignments of roles within the collaboration (source: VDML).

Deliverable - Product or service defined by an associated business item that is produced by an activity or delivered from a store that can be conveyed to another activity or store (source: VDML; example of use of term outside VDML: Allee (2008), ITIL (2011)).

Deliverable Flow - The transfer of a deliverable from a provider (or producer) to a recipient (or consumer) (source: VDML).

Intangible - Deliverable that represents something that is unpaid or non-contractual that makes things work smoothly or efficiently (as opposed to Tangible) (source: VDML; example of use of term outside VDML: Allee (2008)).

Measure - A method that is applied to characterize an attribute of something by assigning a comparable quantification or qualification (popularized version of definition in SMM (2017)).

Measurement - The result of applying a measure (popularized version of definition in SMM (2017)).

Organization - An administrative or functional structure normally interpreted as a network of Organization Units at a higher level in an organizational hierarchy (source: VDML; example of use of term outside VDML: ITIL (2011)).

Organization Unit (or: OrgUnit) - An administrative or functional organizational collaboration, with responsibility for defined resources, including a collaboration that occurs in the typical organization hierarchy, such as business units and departments (and also the company itself), as well as less formal organizational collaboration such as a committee, project, or task force (source: VDML; example of use of term outside VDML: Zachman framework, as introduced by Zachman (1987), and Sowa and Zachman (1992), though a formal definition of the term seems to be omitted).

Participant - Anyone or anything that can fill a role in a collaboration. Participants can be actors (human or automatons) or collaborations or roles of actors or collaborations. They maybe named in the model, or dynamically determined in run-time (source: VDML; example of use of term outside VDML: Allee (2008)).

Pool - A store that contains re-usable resource, i.e., resource that is returned to the pool after having been used, so that it is again available for use (source: VDML; example of use of term outside VDML: PMBOK (2000)).

Practice - Proven way to handle specific types of work and that have been successfully used by multiple organizations (source: VDML; examples of use of term outside VDML: BPMM (2008), ITIL (2011)).

Process - A sequence or flow of Activities in an organization with the objective of carrying out work (source: BPMN (2011)). VDML does not represent process, per se, but represents a process abstraction with a network of activities and flows that represent dependencies and statistical characteristics of a process.

Resource - Anything that is "used" or "consumed" in the production of a deliverable (source: VDML; example of use of term outside VDML: Hruby et al. (2006)).

Role - An expected behavior pattern or capability profile associated with participation in a collaboration (source: VDML; example of use of term outside VDML: Allee (2008)).

Scenario - A scenario defines a consistent business use case and set of measurements of a value delivery model by specifying a, possibly recursive, AnalysisContext for elements in scope of that use case. The nesting of contexts allows a collaboration to be used as a sub-collaboration by more than one activity, each of which sets its particular delegation context and measurements (source: VDML).

Service - A service is a mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description (source: SOA-RM (2006)).

Store - Represents a container of resource. The resource that is stored is identified by a business item (source: VDML; common concept in data flow diagrams (DFD), also known as Gane-Sarson diagrams, as proposed and applied by Gane and Sarson (1979); common construct in simulation systems, such as GoldSim, as explained in GoldSim (2010, 1) and GoldSim (2010, 2); data store in BPMN (2011) is a similar construct, though with a more narrow meaning).

Tangible - Deliverable that represents something that is contracted, mandated or expected by the recipient and which may generate revenue (as opposed to Intangible) (source: VDML; example of use of term outside VDML: Allee (2008)).

Value - A measurable factor of benefit, of interest to a recipient, in association with a business item (source: VDML; example of uses of term outside VDML: Brodie and Gilb (2010), Gilb (2007), Gilb and Gilb (2011)).

Value Chain - Set of activities that an organization carries out to create value for its customers (Porter (1985)).

Value contribution - A measurable effect of an activity that affects the level of satisfaction of one or more values in a value proposition (source: VDML).

Value Delivery Model - Model that supports multiple scenarios for business analysis and design based on evaluation of performance and stakeholder satisfaction achieved through the activities and interactions of people and organizations using business capabilities to apply resources and deliver stakeholder values (source: VDML).

Value Network - Any set of roles and interactions in which participants engage in both tangible and intangible exchanges to achieve economic or social good (Allee (2008)). Or: Any web of relationships that generates both tangible and intangible value through complex dynamic exchanges between two or more individuals, groups or organizations (Allee (2003)).

Value Proposition - Expression of the values offered to a recipient evaluated in terms of the recipient's level of satisfaction (source: VDML; examples of use of term outside VDML: Ballantyne et al. (2008), Osterwalder (2004), Johnson et al. (2010)).

Value Stream - The network of activities that includes resources, value contributions and capabilities to determine a value proposition for a customer who may be the ultimate customer or an internal end user of the result (source: VDML; example of use of term outside VDML: Whittle and Myrick (2005)).

This page intentionally left blank.

Annex B: Alignment with Existing Business Modeling Techniques

(Informative)

B.1 Overview

The following sub clauses describe the alignment of VDML concepts with the following, existing business modeling techniques:

- Value Networks
- Resources, Events, Agents (REA)
- e^3value
- Capability map
- Value Stream
- Cube Business Model
- Possession, Ownership, Availability (POA)
- VDML Support for BMM Strategic Planning
- VDML for Balanced Scorecard and Strategy Map
- VDML Relationship to BPMN

These sub clauses demonstrate the ability of the VDML metamodel to support the models of these techniques. Tables are sorted, alphabetically, by concept names of the particular modeling technique.

B.2 Value Networks

Value Network Analysis (VNA) is an integrative modelling technique for analysis of business activity (Allee 2003, 2008). It defines the specific *Roles* in a collaboration and their interactions that create value through the exchange of *Deliverables*. Roles and deliverables are made visible through visual graphs. Analyses include cost/benefit, value realization, perceived value and internal and external value impact. The goal of the method is to increase and/or optimize value outputs, to leverage financial and non-financial resources (including intangible assets) for improving financial and organizational performance, to find new value opportunities and to improve operational performance and flows of value.

A Value Network Analysis begins with descriptions of contributing *roles* and *value transactions* visualized as a graph or map. Nodes represent roles, and directional arrows between nodes describe transactions. Each transaction has an attribute of tangible or intangible deliverable in the network. Roles are filled by Participants in the network, which can be individuals or firms. Multiple Participants may be candidates for a Role and a Participant may play multiple roles.

Typically solid lines indicate contractual, *tangible* revenue-generating or funding related *deliverables* and their directional transactions. Dashed lines show the critical *intangible* or informal deliverables such as knowledge exchanges and conveyed benefits that build relationships and keep things running smoothly.

Figure B.1 shows a value network for interactions of a technology provider with other business entities. Similar networks describe interactions between roles at different levels of operational detail within a business entity.

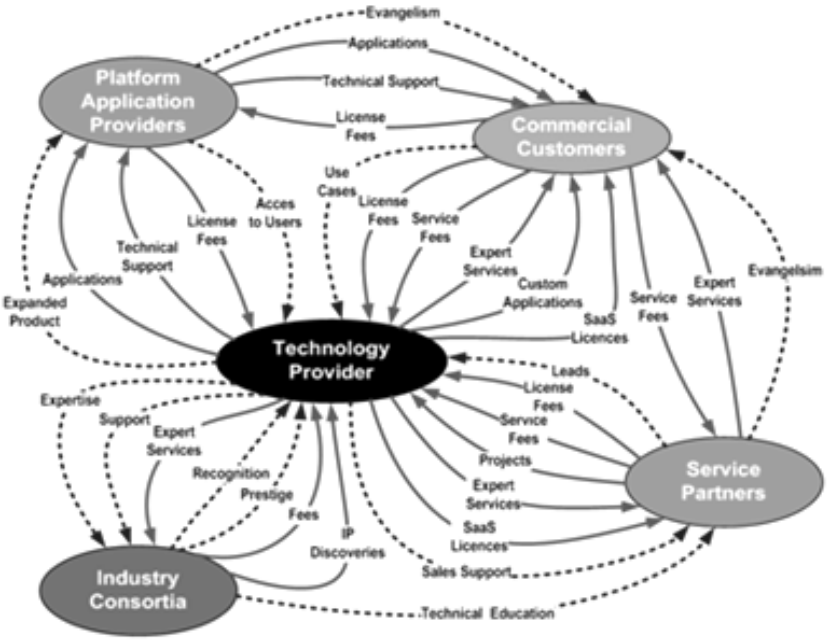

Figure B.1: Value Network Map or Graph

Table B.1 shows the alignment of VNA concepts to VDML concepts. Only corresponding VDML concepts are included in the table. See Annex A: Glossary for further VDML definitions.

Table B.1 - Mapping of VNA Concepts to VDML Concepts

VDML Concept	VNA Concept	Remarks
Activity	Activity	In VNA, an activity defines the boundary and focus of a value network or sub network and can include multiple actions, roles, deliverables sequences and processes. In VDML the boundary of a network of roles and activities is referred to as a collaboration whereas an activity describes work of a single role within the collaboration. (See activity network below).
Attribute	Attribute	In VDML a user-defined, name-value pair associated with a model element. VNA definition is compatible.
Channel	Channel	Definitions are the same.
Deliverable	Deliverable	Definitions are the same.
Intangible	Intangible	Definitions are the same.
Measure	Measure	Definitions are the same.
Measurement	Measurement	Definitions are the same.
Actor	Participant	In VNA Participant and Actor are used interchangeably. In VDML an actor is an entity that does work while a participant can be an actor, a collaboration or a role.
Participant	Participant	Definitions are the same.
Resource	Resource or Asset	Definitions are the same. In VNA resources may include intangible assets such as human competence, brand, relationships, reputation, and methods. Also see Value Realization. In VDML, resources are conveyed by deliverable flows while measurements of values (e.g.,

		duration and quality) are conveyed by value adds/contributions. In VNA resources are made available to roles, who manage them in regard to the deliverables they are responsible for generating or handling as inputs. In VDML, control of resources is more specific, using stores, activities of roles and deliverable flows.
Role	Role	Definitions are the same.
Scenario	Scenario	Definitions are compatible.
Tangible	Tangible	Definitions are the same.
Deliverable Flow	Transaction	Concepts are the same. (See Process)
Value	Value	Definitions are the same.
Activity network	Value network	Definitions are compatible. In VNA, a value network depicts interactions of roles and the flow of deliverables between them. In VDML an activity network depicts the activities of the roles (more detail) and the flow of deliverables between them.
Collaboration	Value Network	Any collaboration can be modeled as a value network. All value networks are collaborations. In VDML, business network, community, organization unit and capability method are specializations of collaboration. In VDML an activity is distinguished from a collaboration such that an activity can delegate to a shared collaboration.
Value proposition	Value realization	In VNA, value realization is when a value input, either tangible or intangible, has a positive impact on or replenishes resources or assets. In VDML a value proposition conveys to a recipient deliverable(s) with a bundle of values that can become inputs to subsequent activities of the recipient.

B.3 REA (Resources Events Agents)

William McCarthy developed the REA model in 1982 as a generalized accounting framework, but later evolved it together with Guido Geerts into an ontology for economic systems, covering value delivery among networks of economic agents Geerts, McCarthy (2002), McCarthy (1987).

Figure B.2 depicts exchanges between three economic agents and a value conversion within an enterprise, in the REA model. It shows the relationships between economic resources, economic events and economic agents. The REA ontology describes the economic principles of trade and production business processes, i.e., the use, consumption, production and exchanges of economic resources. One of the fundamental REA concepts is duality, explaining what resources an agent gives up in order to receive other resources. Duality also represents causality relationship, explaining why economic events happen from the economic point of view. From an agent's entrepreneurial perspective and over the lifetime of the enterprise, the received resources must have a higher value than the provided resources. A business process is a set of economic events related by the duality relationship.

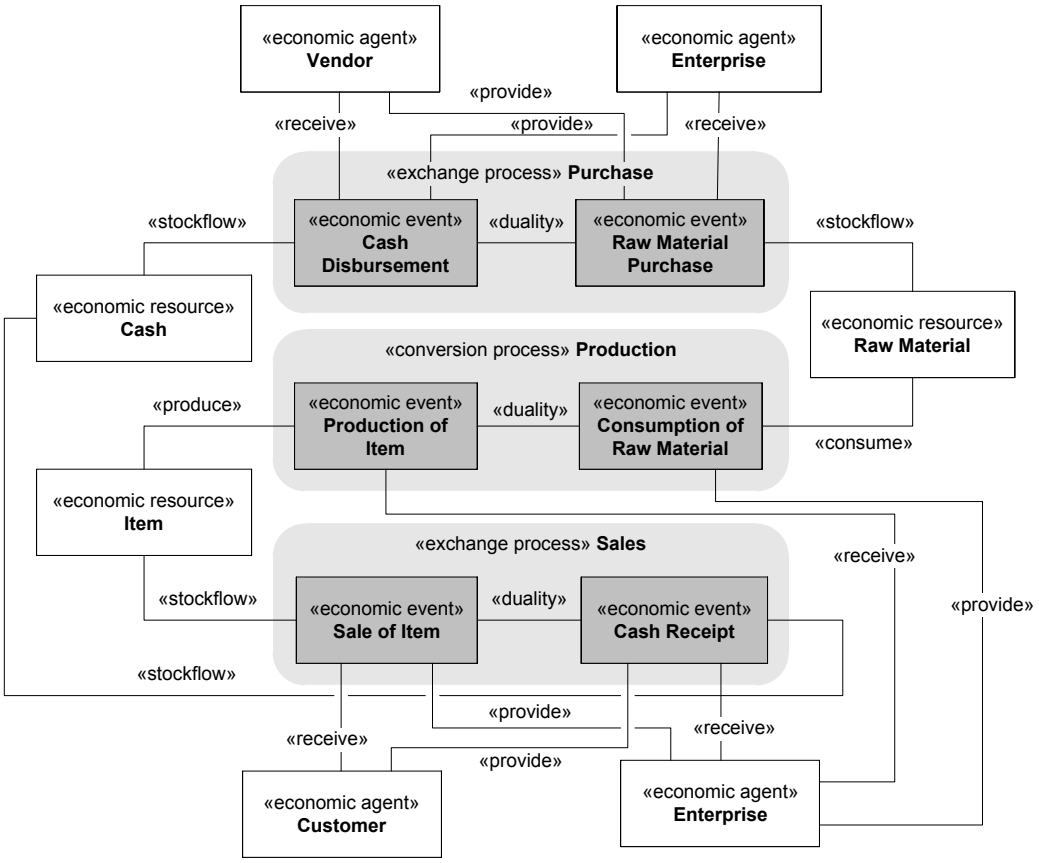

Figure B-2: Example of an REA Model

The REA ontology also contains rules for verifying completeness of the model, i.e., every REA model must specify who are the provider and recipient of every exchanged resource, how an agent receives and gives up each of its resources, and why events happen. The REA ontology does not specify notation – any data modeling technique can be used to describe REA models. Figure B.2 illustrates an REA model in the UML notation.

Scope of the REA model is determined by granularity of economic resources (a resource can contain other resources), in contrast to VDML, where the scope is determined by granularity of a value proposition (defined as a value stream that may incorporate other value streams) where value streams may share capabilities) and collaboration, (a collaboration may contain other collaborations engaged in roles).

The REA model also contains concepts for describing what could or should happen, i.e., commitments, contracts, schedules and policies.

A mapping of REA concepts to VDML concepts appears in Table B.2, below.

Table B.2 - Mapping of REA Concepts to VDML Concepts

VDML Concept	REA Concept	Remarks
Business network	Business process	In REA, a set of economic events related by a duality relationship. In VDML a similar set of activities (events) are related as occurring within the context of a business network.
Business network	Contract	Contract extends the business network concept to a formal agreement, i.e., characteristic of a specific business network.
Business network	Exchange	In VDML, business network collaboration defines the scope of an exchange between parties. In VDML, there can be business networks that are within business networks, so more complex business networks can be composed of more discrete networks.
Value proposition	Commitment	Extends value proposition concept as an obligation, i.e., interpretation of a value proposition as an obligation.
Duality	Duality	A property of an economic exchange by which each contributor to an exchange receives compensation for its contribution. It is an observed property of a VDML business network.
Store or capability	Economic resource	In REA, something of economic value that is purchased, sold, produced, used or consumed. A capability can be represented as a resource that provides a service. In VDML, a store holds resources. A resource flows as a business item. Business items also may convey other things such as orders, specifications, etc., that are input or output of activities.
Party role (Business network)	Economic agent	Consistent with contract party and may be an actor or an organization. Within a business entity, there will be other, more specific economic agents that may be represented as OrgUnits or Actors that are in the organization structure of the primary economic agent (e.g., company).
Activity	Economic event	Economic event may be a single VDML activity or an activity that delegates to a collaboration of more detailed activities. In REA, economic events are atomic and cannot be decomposed – REA model granularity is determined by granularity of economic resources.
Capability method or practice	Policy	In REA, a policy defines restrictions on patterns of activities. There is no directly equivalent element in VDML except that a capability method might be designated as a required method. A planning percentage may be used to determine the percentage of the time a port/deliverable is the output of an activity and thus could represent the effect of business rules. A practice refers to a generally accepted approach to doing a type of work and a capability method may be identified as conforming to a practice, but the details of a practice are not expressed in VDML, per se.
Deliverable flow (Role)	Provide and Receive	In VDML, exchanges between activities are via DeliverableFlows. An abstraction can show DeliveableFlows as between the roles of the activities where DeliverableFlows will not appear between activities for the same role.
Deliverable flow (Store)	Stockflow	Deliverable flow is neither restricted to resources, nor to input or output of store.
Role association	Responsibility	Responsibility is a relationship of a role. A role may be filled by a participant (another role, actor or collaboration).

Value	Value	In REA the focus is on economic value and is determined through the execution of an exchange and will depend on the provider or recipient's perspective. In VDML value is a measurable characteristic of the product or service delivered to a recipient and includes economic value (price/cost), but also includes many other factors such as reliability, timeliness, appearance, and provider's reputation that will be evaluated from the perspective of the recipient.

B.4 e^{3}value

This modeling language for evaluation of the viability of e-commerce business models or value constellations, e^{3}value, as presented by Gordijn and Akkermans (2003, 2004), represents a group of economically independent entities, including market segments, that exchange transactions with economic value for mutual benefit. This seems straightforward, but in e-commerce the number of entities, their different interests and multiple exchanges can obscure the net value realized by the different participants. Each of the participants must have a sustainable business model for the overall exchange to be viable.

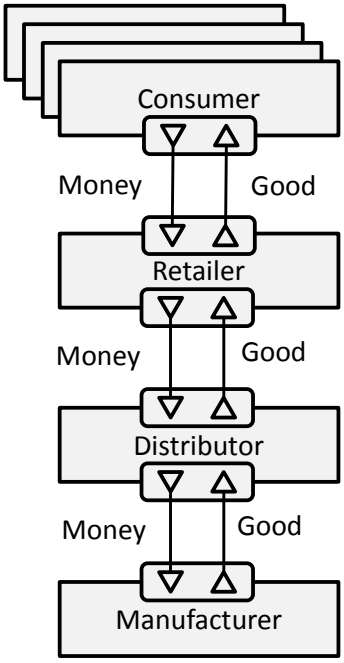

Figure B-3: Example of an e^{3}value Model

Figure B.3 illustrates an example e^{3}value model. Table B.3, below, aligns the more detailed e^{3}value concepts with VDML concepts.

Table B.3 - Mapping of e^{3}value Concepts to VDML Concepts

VDML Concept	e^{3}value Concept	Remarks
Actor/collaboration	Actor	In e^{3}value, an actor is restricted to an economically independent entity. An economically independent entity in VDML may be an actor or collaboration. The collaboration will generally be specialized as a community (representing a set of potential actors such as a market segment) or an organization unit such as a company.
Business network	Composite actor	A business network may have supporting business networks. A supporting business network consists of parties working together to participate in a parent business network.

Business network	Constellation	A collaboration/exchange of complementary transfers of value between independent business entities.
Deliverable flow	Dependency element	In e^{3}value, a flow within a business entity. In VDML, flows are between activities and roles (through activities) internally or externally.
Community	Market segment	A market or market segment is a community of potential parties in a business network or other collaboration.
Scenario	Scenario	Similar concept of applying different circumstances to evaluation of the viability of the model.
Activity	Value activity	Value activity is a collection of operational activities which can be assigned as a whole to actors.
Business network	Value interface	Not explicitly defined in VDML but is the aggregate of value propositions provided and received by one party in a business network. So a business network determines the scope of value interface of each party.
Business item	Value object	The thing that is provided or received, which is of economic value for at least one of the actors.
Value proposition	Value offering	May be a value proposition as well as a value proposition that represents the aggregation of value propositions provided or received.
Port	Value port	The point of departure or receipt of a value object/business item
Unit of production	Value transaction	The set of value objects and transfers that represents a complete cycle of exchanges between parties in a constellation such that their net gain/loss can be assessed.
Deliverable flow	Value transfer	Flow of value between business entities (actors in e^{3}value or parties in VDML)

B.5 Capability Maps

In recent years, considerable attention has focused on *capability mapping*. A capability map defines a hierarchy of capabilities required for the enterprise to deliver the desired results along with assessment of the importance and performance of these capabilities. The capability map is analyzed to identify those capabilities that require improvement—often called a capability "heat" map.

A capability map, as used in capability analysis, defines a hierarchy of capabilities required for the enterprise to deliver the desired results along with assessment of the importance and performance of these capabilities. The capability map is analyzed to identify those capabilities that require improvement—often called a capability "heat" map, an example of a part of which is shown in Figure B.4.

According to Krohn (2011), the capability map is the framework for defining scope and analyzing impact. A capability is "what" the business does. By focusing on the what, the map becomes very stable. "How" something is done changes frequently; with every system implementation or process improvement, it is altered. However, what is done remains relatively the same, year after year. The map organizes these capabilities into a hierarchy, with each capability level providing progressively more detail. The hierarchy enables to start with a broad discussion and then dive into more detail where needed. Creating a capability map, containing commonly used or usable definitions of capabilities, with their associated detail, establishes a common vocabulary across the business. This will enforce productivity in design or re-design of business models, and will facilitate discovery of opportunities to consolidate or outsource (or purposefully not doing so) capabilities.

The core concepts of capability mapping—the capability definitions and capability hierarchy— map directly to the VDML capability definition and capability library. Figure B.4 illustrates a typical capability heat map where critical capabilities are highlighted. There does not appear to be a generally accepted specification of additional detail to a capability map model, but VDML represents a number of related concepts that would generally be

expected to support the capability map: the organization(s) that have and offer the capability, the activities performed to deliver the capability, the capabilities/organizations that use the capability, the resources consumed and deliverables produced by the capability, and the values contributed (at the activity level) by the capability.

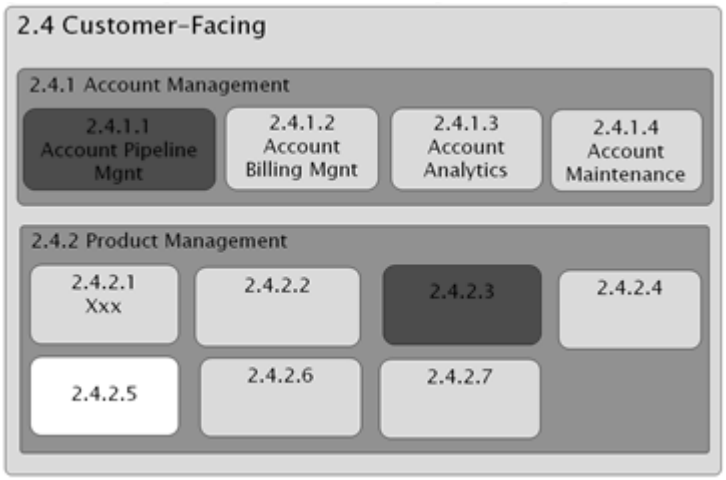

Figure B-4: Capability Heat Map

B.6 Value Stream

A value chain has been described as "a [disaggregation of] a firm into its strategically relevant activities in order to understand the behavior of costs and the existing potential sources of (competitive) differentiation" Porter (1985). A value stream has been described as: "an end-to-end collection of activities that create a result for a 'customer' who may be the ultimate customer or an internal 'end user' of the value stream" Martin (1995). The focus in both cases is on delivery of value to a customer. VDML addresses both of these by supporting top-down, and industry or ecosystem analysis and decomposition of activities and their contributions to cost and value, and by supporting end-to-end detail of the contributions of activities to create a result for a customer, internal or external, as well as the exchanges of value in the marketplace. The approach is up to the modeler.

Value stream mapping, as explained by Rother and Shook (1998), is a lean manufacturing technique used to analyze and design the flow of materials and information required to bring a product or service to a consumer. Customer value is the leading motivation, and focus is on improving value, by reducing waste. It combines material flow (product produced) and information flow (e.g., sales orders or forecasts that trigger production). Broader systems can be modeled via decoupling buffers or stores (called "supermarkets"). The focus is on improving operational performance via detection and elimination of non-value added (i.e., wasted) time.

VDML supports all of these approaches. There does not appear to be a generally accepted ontology for value stream modeling, but the concepts can be inferred. In VDML, a value stream can be identified within a VDML model as the network of capabilities and their activities that contribute to the values and deliverables identified in a value proposition. Essentially this is a backward trace from the value proposition and can extend to suppliers and outsourced capabilities/services.

B.7 Business Model

The following paragraphs discuss VDML alignment with both Lindgren's and Osterwalder"s business model frameworks; these both provide a high-level abstraction of what an organization does to achieve its purpose.

B.7.1 Lindgren

A business model describes how an organization creates, captures, delivers, and consumes value from the perspective of primary stakeholders. Peter Lindgren defines seven building blocks of a business model: value proposition, user and consumer, value chain, competencies, network, relations and value formula, Lindgren (2011). These are depicted in Figure B.5 (relationships are in the middle).

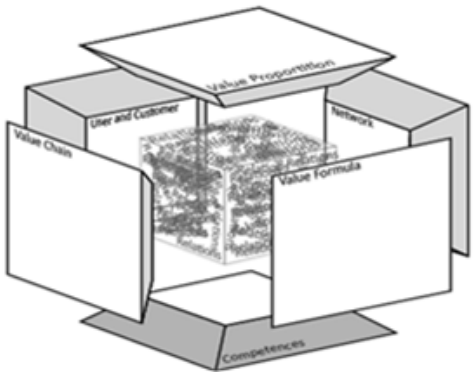

Figure B-5: The Business Model Cube (Lindgren)

All of these building blocks and their related components can be described in terms of more detailed VDML model elements focusing primarily on the business network level of value exchanges with business partners and customers, but supported by the value streams that identify capabilities, resources, costs and values. Table B.4, below, outlines the relationship of Lindgren's seven components to VDML concepts.

Table B.4 - Mapping of Business Model Cube Concepts to VDML Concepts

VDML Concept	Business Model Cube Concept	Remarks
Capabilities	Competencies	Competencies in BM may be more general, including resources and methods.
Business network	Network	Network in BM includes business partners where the relationships may not be restricted to particular business exchanges. In VDML, a business network can define a range of business relationships or it may be restricted to the parties involved in a particular set of related exchanges.
Deliverable flows	Relations	Relations in the BM Cube link internal activities and capabilities with the external BM components. VDML deliverable flows define these relations as well as external exchanges within a business network context.
Party	User and customer	In BM cube framework user would refer to parties that do not pay economically for the value proposition offered by the Business Model; customer would most often refer to a typical customer in a market or market segment. In VDML, a customer is a particular party in a business network and may be one of a community of potential customers. Users

		can be represented as other community(s), related to customer(s), possibly as a business network collaboration.
Value stream	Value chain	The concept of a value stream is not an explicit element in VDML, but is the network of activities and capabilities that contribute to the deliverable(s) and values of a value proposition.
Measure	Value formula (Profit formula)	In VDML, a measure defines how a measurement is determined. Here, a measure may be a formula that combines certain factors from the model to provide a profit measurement (the result of applying the formula).
Value proposition	Value proposition	Same concept but more detailed in VDML.

B.7.2 Osterwalder

Alex Osterwalder (2004, 2010) defines nine components: customer segments, customer relationships, distribution channels, revenue streams, value propositions, key activities, key resources, cost structure and key partners. These are depicted in Osterwalder's graphic in Figure B.6.

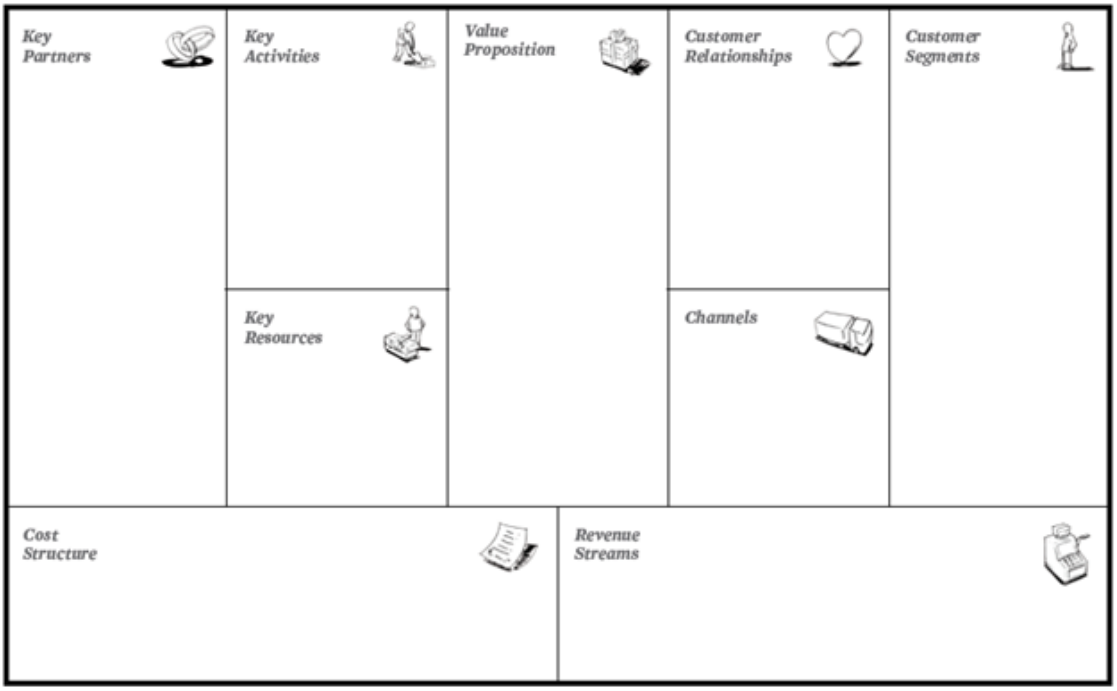

Figure B-6: The Business Model Canvas (Osterwalder)

All of these components can be described in terms of more detailed VDML model elements focusing primarily on the business network level of value exchanges with business partners and customers, but supported by the value streams that identify capabilities, resources, costs and values. Table B.5, below, outlines the relationship of Lindgren's seven components to VDML concepts.

Table B.5 - Mapping of Business Model Canvas Concepts to VDML Concepts

VDML Concept	Business Model Canvas Concept	Remarks
Channel	Channel	In VDML, a deliverable flow with channel attribute to define the mechanism of flow, e.g., telephone, email, postal service, etc. In BM Canvas, flows typically are to customers.
Value contribution component (cost)	Cost structure	In BM Canvas, cost structure describes all costs to operate the business. In VDML, cost per unit of production is one value that is captured for activities. The VDML model would support aggregation of costs for a capability by reference to the activities that use that capability. The cost to the enterprise would require multiplying the cost per unit of production by associated production volumes (by activity). This can be user defined, and could be computed for a value proposition component.
Collaboration	Customer relationships	In BM Canvas, customer relationships are approaches to engaging customers leading to business exchanges. In VDML, collaborations can be defined to represent different types of customer relationships including collaboration with automated and non automated services and collaboration within communities.
Community	Customer Segment	In VDML, different customer segments are represented by different communities where a business network will typically engage a typical member as a party in the network.
Capability Library	Key activities	In BM Canvas, key activities refer to the most important things a company must do. In VDML, the core things a company does are identified as capabilities in a capability library (taxonomy of capabilities). Key capabilities would highlight those of primary importance. A capability heat map might be used to highlight the key capabilities (i.e., activities). These would likely be capabilities at higher (broader) levels in the taxonomy.
Business network	Key partnerships	In BM Canvas, key partnerships refer to the network of suppliers and partners necessary for successful operation of the business. In VDML, a broad business network may represent relationships and key deliverable exchanges with multiple partners and suppliers. Sub-business networks can be used to represent different BM Canvas types of partnerships: Strategic, co-opetition, joint venture, and buyer-supplier.
Capability	Key resources	The focus of BM Canvas is resources required to perform including facilities, people, money, etc. In VDML, a capability includes key resources as well as the activities to apply the capability and produce value.
Business network	Revenue stream	In BM Canvas, revenue stream is a flow of income from a product or service minus the cost of the product or service. In VDML, the price/revenue and the cost are attributes of value propositions exchanged with other parties in a business network.
Value proposition	Value proposition	Same concept. BM Canvas refers to value propositions offered to the market, while VDML expands the concept to value propositions offered between roles in a collaboration.

B.8 Possession, Ownership, Availability (POA)

Possession, Ownership, Availability (POA), is a method, notation and ontology for modeling business processes

focusing on value delivery, which is suitable for model-driven design of ERP and enterprise information systems (Scheller, Hruby, 2009, 2011). The POA model defines specific roles in a business process, and describes value delivery as flows of possession, ownership and availability of resources between the roles, as well as deposits and withdrawals from the roles' repositories. The concepts of possession and ownership correspond to the same concepts in legal systems, and enable constructing the chart of accounts and balance sheet of an economic entity. Availability determines the production-possibility frontier of economic entities in the network.

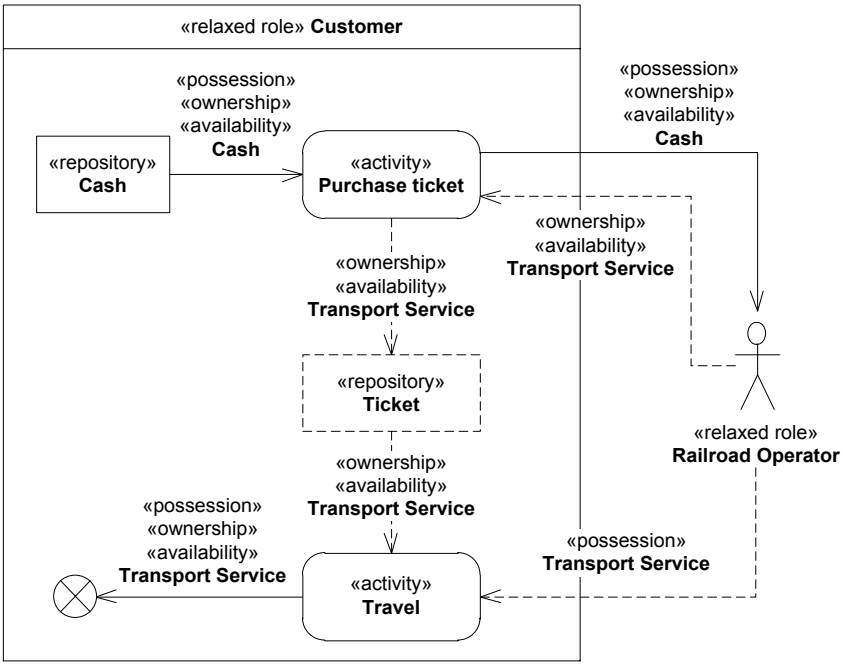

Figure B-7: Example of a POA Model

As the POA concepts can be mapped to the accounting concepts, the POA model can be used for verifying consistency between an existing accounting system and the business process model. The primary purpose of the POA model is specification of an executable, platform-independent model as described in the OMG model-driven architecture, from which accounting and enterprise information systems can be generated.

Figure B.7 depicts value delivery between a *Customer* and a *Railroad operator*, where *Ticket* represents a repository of availability and ownership of *Transport Service*. The *Purchase ticket* activity represents an exchange of *Cash* for availability and ownership of *Transport service*. The *Travel* activity represents receipt of possession of *Transport Service*, settlement of *Customer's* claim, and consumption of the service. Table B.6 outlines the relationships between the POA concepts and the VDML concepts.

Table B.6 - Mapping of POA Concepts to VDML Concepts

VDML Concept	POA Concept	Remarks
Activity	Activity	Definitions are compatible.
Value proposition	Availability	In POA, availability is defined as the conditional right to possess a resource. Depending on the condition, availability can represent anything from proposal to binding contract.
Collaboration	Business process	Definitions are compatible. In POA, a business process represents a description of how roles interact. Both in POA and in VDML the boundary of a network of roles and activities is a

		collaboration whereas an activity describes work of a single role within the collaboration.
Deliverable flow (Store)	Deposit and withdrawal	Definitions are compatible.
Deliverable flow (Role)	Flow	Definitions are compatible. In POA, flow represents transfer of possession, ownership or availability of resources from one role to another. In VDML, the deliverable, represented as a Business Item, can be defined as a transfer of possession, ownership or availability.
Intangible	Intangible	Definitions are the same.
Store	Repository	Definitions are the compatible. Like in VDML, repositories can be physical, such as warehouse, or abstract entities, such as debt. Repositories of availability and ownership (but not possession) may have negative value.
Role	Role	Definitions are compatible. In POA, a role can be marked as *relaxed*, meaning that the model does not have to specify how the role obtained the consumed and used resources. In VDML, roles are relaxed by default.
Tangible	Tangible	Definitions are the same. In POA, intangibility is a property of a resource.
Pool	Usage	In POA usage represents resources required by an activity, but without being consumed, such as usage of tools or usage of information. In VDML this is a Pool (a specialization of Store) managing a reusable resource.
Value	Value and Minibudget	Definitions are compatible.

B.9 VDML Support for BMM Strategic Planning

The OMG Business Motivation Model (BMM) defines a framework for the capture of strategic planning information. This framework reflects widely accepted strategic planning techniques. The resulting strategic plans define requirements for business changes, but there remains a significant gap between these requirements and the realities of implementation. VDML (Value Delivery Modeling Language) can help bridge this gap through a more rigorous specification of the current state of the business and the future, desired state.

In this article, I will begin with a brief overview of BMM and then discuss the application of VDML to further detail and refine the strategy and to support transformation planning and management. This discussion is not intended as a standard method, but illustrates how VDML can be used to improve the discipline and rigor of strategic planning and transformation.

B.9.1 Overview of BMM

The diagram, below, is an abstraction of the Business Motivation Model (BMM) taken from the OMG specification. We will briefly discuss the Means and Ends that represent a strategic plan. Influencers and Assessment are elements of the strategic planning process that provide input for development and refinement of the plan, but they are not, per se, elements of a strategic plan.

End

The End contains elements that define the desired future characteristics of the enterprise including the Vision and Desired Result. A strategic plan, at the most abstract level, is expressed as a Vision of what the enterprise wants to be and how it wants to be perceived. This is complementary to the Mission, below.

The desired result consists of Goals and Objectives. A goal is a long-term, qualitative result that the enterprise may already be pursuing or that may be advanced as a result of a business challenge or opportunity. It defines a purpose for the strategy. Objectives are specific, measurable results to be achieved by a strategy, and they support enterprise goals. While there may be many measures of performance, objectives focus on selected, key measurements that reflect progress toward the strategy and goals from a management and investor perspective.

Means

The Means contains elements that describe how the future state of the enterprise will be achieved based on the Mission, Course of Action and Directives. The Mission expresses why the enterprise exists—what it wants to accomplish. Successful pursuit of the Mission should support realization of the Vision.

A course of action is the approach to implementation of the Mission in pursuit of the Goals and Objectives. It consists of Strategy and Tactics. A strategy defines how the mission will be pursued and objectives will be achieved. In conventional strategic planning, it is an abstract description of how the enterprise will operate in the future. Typically, there will be multiple aspects to a strategy, potentially representing the integration of different ideas. Tactics are incremental changes to the state of the enterprise that lead to the desired future state required by the Strategy. The distinction between strategy and tactics is somewhat subjective. Tactics will focus on resolving particular problems and steps toward implementing related changes.

Directives are the business policies and business rules that are to be incorporated in the future state of the enterprise. Policies are statements of business operating requirements. Business rules define operating criteria or constraints in specific circumstances. Business rules implement business policies.

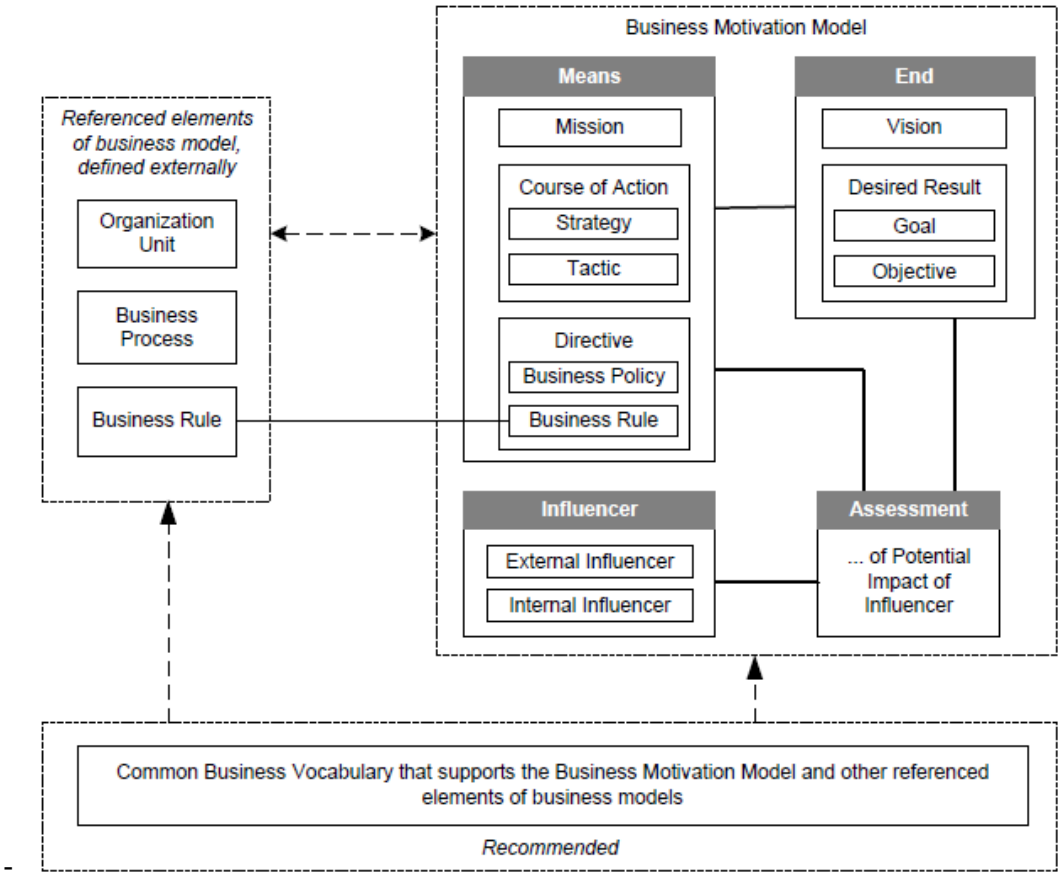

Figure B-8: Overview of BMM

B.9.2 Application of VDML

VDML does not address all aspects of a business transformation. I expect that VDML will be used to support strategic planning and transformation, program management for the operations and capabilities of the business. It will be complemented by related efforts such as development of policies, analysis and development of markets, design of incentives and development of contractual relationships.

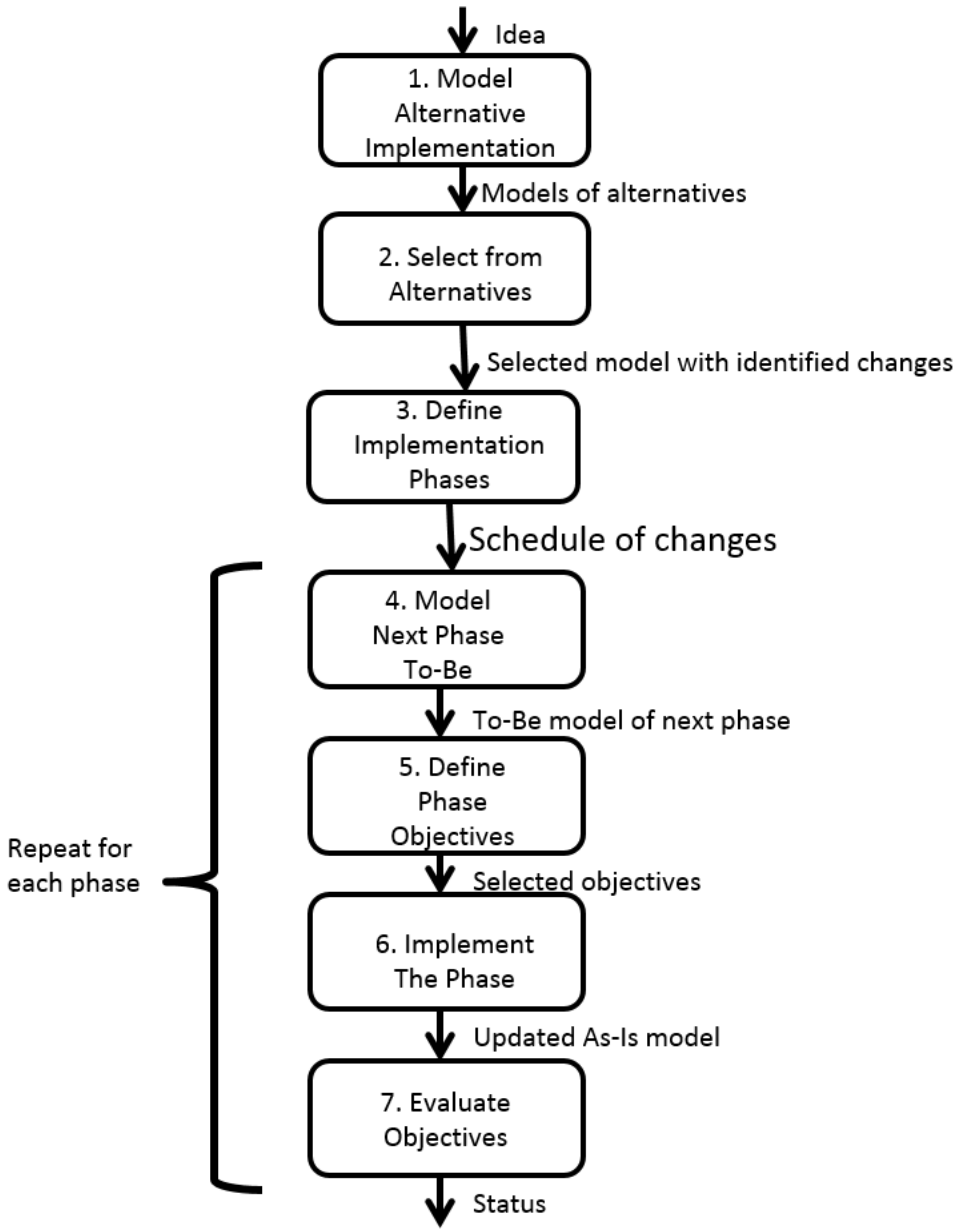

Figure B-9: Strategic planning process

The basis for consideration of changes is a VDML As-Is model. This represents the current state of the business and will contain measurements reflecting current business operations. It provides a context for understanding problems and assessing solutions.

An idea is a potential strategy at an inspirational stage of development. When that idea is refined, validated and accepted, it becomes a strategy (or a component of a more complex strategy). We start with consideration of one idea to improve the business.

The diagram, below, depicts steps of development of an idea through transformation of the business. We will discuss each of these steps in the paragraphs that follow.

Step 1 involves consideration of alternative approaches to implementation of an idea under consideration. Each alternative should be modeled as a VDML To-Be model. The VDML model will support analysis of the idea to

define a cohesive impact on the organization, capabilities, activities, resources, value contributions and value propositions, potentially including business partners.

Step 2 involves selection of the preferred alternative. The To-Be model provides the basis for estimating activity value contributions as a result of the changes along with their impact on value propositions. It also provides the basis for estimation of transformation costs and duration.

In Step 3, VDML will provide the basis for planning the work of transformation. The changes identified in step 2, above, must be organized into implementation phases. Each phase will have a package of changes to be implemented together. Some changes may be dependent on others—the VDML deliverable flows will help identify these dependencies. Each phase should achieve implementation of a stable business state that can be measured. If possible, each phase should yield business benefit.

Step 4 is applied prior to each phase. A To-Be model is developed or refined for the next phase. The To-Be model incorporates the activities, capabilities and resources to be developed in that phase. This, in turn, identifies the organization units responsible for the changed business operations as well as the organization units of related activities that will require collaboration and coordination.

In Step 5, objectives are developed for the next phase based on the To-Be model and the transformation plan. The phase To-Be model includes measurements for the expected value contributions of activities that are affected by the transformation. Changes to value contributions for each affected activity will be targets for performance by the organization unit that provides the supporting capability. The impact on aggregated values will be targets for the overall undertaking for the enterprise or the targeted line(s) of business.

Step 6 involves doing the work of transformation. Each phase may be managed as a separate project with a detailed plan. The project must address all aspects of the transformation for that phase including changes in products, technology, organization, personnel, facilities, information systems, and activity inputs and deliverables. Some aspects are beyond the scope of a VDML model, but the VDML model will help identify them.

At the end of each phase, an As-Is model is developed in Step 7 as a new baseline. This model should be very similar to the To-Be model for the phase, but the measurements are actual value measurements. These measurements are compared to the To-Be expected value measurements to evaluate success of the phase.

Continuous transformation

In the real world, enterprises do not focus on the implementation of one idea at a time. As some ideas are being implemented, other ideas will emerge. These new ideas will affect some of the same business elements undergoing transformation. Some ideas will be completed while work on other ideas remains to be done. The following paragraphs extend the above transformation steps to reflect on-going, continuous transformation for implementation of multiple ideas.

In Steps 1 and 2, a new idea may be evaluated and change requirements identified based on the current As-Is model as discussed, above, but the overall evaluation should also consider changes already planned for other ideas that affect some of the same business elements.

In step 3, the changes for the new idea must be reconciled with the To-Be models of pending phases of the current strategy, and the program plan must be updated to reflect these changes and align phases.

In Step 4, based on the alignment of new changes to pending changes, a To-Be model must be developed or adjusted for at least the next phase.

In Step 5, to the extent the next phase is expected to achieve different results with the newest idea, it will be necessary to reconsider which value measurements are appropriate objectives for that phase.

Step 6 involves completion of the current phase, potentially with changes for multiple ideas. In a continuous transformation environment, implementation of some ideas may be completed while work on other ideas continues.

Step 7 involves evaluation of phase objectives at the end of each phase as well as the end objectives for completed ideas.

Summary

The primary contributions of VDML to strategic planning involve support for Goals, Strategies, Objectives and Tactics

A VDML To-Be model provides the measurements and value propositions of an idea implementation for consideration of its contributions toward goals. The VDML model for a particular implementation becomes a detailed strategy. It provides detail for planning and evaluation of the transformation plan. Measurements of the To-Be model and the transformation plan become the basis for objectives for transformation phases and completion of strategies. Phases of the transformation plan may be considered Tactics.

Application of VDML will bring a change to the way strategic planning and business transformations are performed. Not only will it enable better planning and monitoring, but it will enable more effective management of complex and multi-faceted transformations needed to keep up with rapid changes of business and technology.

B.10 VDML Support for Balanced Scorecard and Strategy Map

The Balanced Scorecard (BSC) and Strategy Map (SM) are well-known, abstract models for structuring business transformation objectives. The analysis of objectives helps refine a strategic plan, and the resulting objectives provide a basis for continuing assessment of progress.

BSC/SM objectives represent desired changes to the state of the business—how changes to the design/operation of the business will achieve improvements. Thus the changes represented by BSC/SM objectives represent the difference between VDML As-Is and To-Be models.

The BSC defines four *perspectives* that classify objectives: (1) Learning and Growth, (2) Internal, (3) Customer, and (4) Financial. This classification drives a broader analysis starting with the development of capabilities (Learning and Growth), through development of internal methods and processes (Internal), to delivery of customer value (Customer), and finally to enterprise success and sustainability (Financial).

The Strategy Map introduces causal relationships—some objectives depend on achievement of other objectives. An internal process will not be successful without the capabilities that support it (e.g., people, machines, knowhow).

A VDML model represents the current or future state (sometimes the past) of the business and associated operating measurements. A model could involve multiple lines of business and multiple value streams. VDML models can extend the BSC/SM analysis and support more robust transformation planning and assessment. The diagram, below, depicts this.

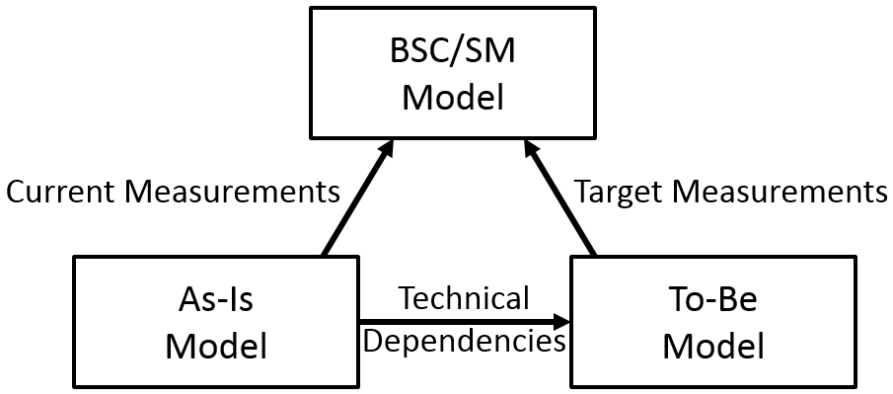

Figure B-10: VDML models for BSC/SM

So a VDML To-Be model can represent the improvements in performance and value creation if the strategy is successful. As the basis for objectives, these To-Be measurements must be compared to corresponding measurements in an As-Is (current state) model. The As-Is model will change as the transformation progresses. If the transformation goes as planned, the As-Is model will become the same as the To-Be model (transformation complete).

VDML can represent many of the measurements of interest for the BSC/SM objectives, so a BSC/SM objective may obtain its target measurement from the To-Be model and its current measurement from the As-Is model.

VDML can represent objectives of the BSC Financial Perspective as As-Is and To-Be value proposition measurements and market segment forecasts. However, VDML does not provide support for the development of these market forecasts. For example, profit is essentially price minus cost. VDML supports cost detail, but price is a management decision based on market analysis. Market share is a forecast based on price and other factors including future competition. VDML can capture those measurements, but does not provide support for market analysis.

Some measurements require more extensive analysis and mapping. For example, if a new capability is needed, the need is represented by the absence of the capability in the As-Is model. The objective measurement is essentially binary—it is either there or it isn't. That's not very useful for BSC/SM. We can refine this if there is an appropriate measure of progress. For example, if the capability requires skilled people, we might represent the need with a VDML store of people supporting the capability in the To-Be model. However, the current progress is then just a measurement reported by the training and hiring effort of the transformation. If we were to create a place for these measurements in the As-Is model, that would be outside the scope of the As-Is model since it would not represent the current operation of the business.

From a VDML perspective, the causal relationships of Strategy Maps (i.e., dependencies between objectives) are of six different types: (1) changes to enterprise capabilities (i.e., "capital" in Learning and Growth) that are necessary for implementation of Internal Perspective value stream changes, (2) changes to capabilities that increase customer value (Customer Perspective) such as adoption of an advanced technology, (3) changes to capabilities that improve investor value (Financial Perspective) such as development of intellectual property, (4) changes to a value stream (potentially involving multiple activity changes) that affect activity contributions to customer value (Customer Perspective), (5) changes to the value stream that improve investor value (Financial Perspective) such as cost reductions, and (6) changes in customer satisfaction levels that drive market changes reflected in investor value (Financial Perspective).

While these are all supported by VDML, they require two VDML models—an As-Is model and a To-Be model. Type (1) can be observed by a change in capabilities (new or modified) between the As-Is and To-Be models. Type (2) can be observed as value stream changes that improve desired customer value proposition measurements. Type (3) requires human recognition of the value investors will place on improved enterprise capabilities. Type (4) can be observed by tracing the sources of changes to the customer values in the To-Be model to the relevant customer value contributions that are improved from the As-Is model. Type 5 can be observed from changes in value contributions (such as cost reductions) that have a direct impact on investor value. Type 6 is based on changes in customer value propositions but it requires market insight and feedback regarding trends and competition to determine the impact on investor value. Many of these causal relationships may be based on VDML deliverable flows, but rather they represent differences between VDML models.

In a substantial transformation, there must be phases of implementation. Without objectives (or intermediate target objectives) for phases, management would have difficulty assessing progress on a long-duration project with multiple components. Each phase should be represented by a VDML model for the expected state of the business at the end of the phase. As the transformation progresses, a series of As-Is models would be created to represent the new, current state of the business. The As-Is model evolves toward the To-Be model so that measurements in the current state depict progress toward To-Be targets as depicted in the diagram, below.

The BSC/SM model then becomes a series of phase models with As-Is and To-Be supporting models to define intermediate targets—see the diagram, below. The To-Be VDML model of a phase becomes the expected As-Is model of the next phase, and the BSC/SM short-term, phase model represents the objectives of the current phase. The overall BSC/SM model can be derived from the initial As-Is model and the final To-Be model.

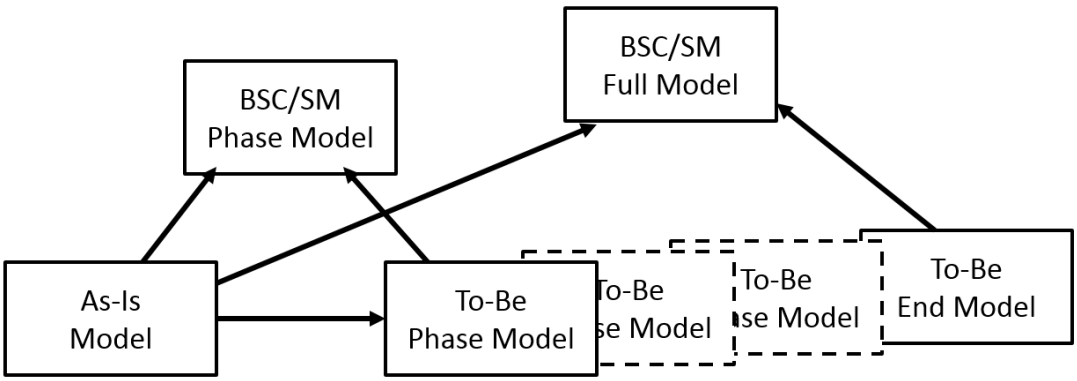

Figure B-11: Modeling transformation phases

While the full BSC/SM model can be stable (unless the end state of the business evolves), the current phase BSC/SM model will be incrementally based on new versions of the As-Is model as the transformation progresses, and it will be based on new To-Be models as new phases are started.

In summary, VDML can provide significant value in validating the strategy, setting and evaluating objectives that identify causal relationships and developing the transformation plan. However, the relationships between the BSC/SM objectives and the VDML models are diverse and require incremental alignment with new versions of VDML models. Development of needed modeling techniques is left for VDML implementers to explore.

B.11 VDML Relationship to BPMN

This sub-clause addresses the relationship of VDML to BPMN (Business Process Model and Notation). While VDML and BPMN represent some similar concepts, they address different business problems and areas of concern. They provide different viewpoints—they address the concerns of different stakeholders.

The intent of VDML is to address the needs of business leaders to define, manage and transform the design of the enterprise. This requires a broad perspective to incorporate multiple aspects of the enterprise. The focus of BPMN is defining and managing repeatable, reliable processes with an emphasis on automation. The focus is much more specific but much deeper in detail addressing many exceptions and variations. The community of concern is primarily managers, business analysts and systems developers closer to the business operations. In terms of the draft MDA Guide, VDML supports a business model and BPMN supports a logical system model.

The primary mode of application of VDML is "forward engineering" as described by the draft MDA Guide. This involves development of a business solution by starting at an abstract level of design and developing in stages of increasing detail. VDML enables business leaders to define and agree upon a high-level business design that can then be used to guide more detailed solutions that may be delegated to multiple business units. BPMN represents a primary tool for development of the next level of detail.

However, not all applications of VDML fit that pattern. The manner in which VDML is used and its relationship to BPMN will vary depending on the nature of the business problem being solved. A number of different modes are discussed in the following paragraphs. A paper that explores the relationship between VDML and BPMN in much more detail is available as OMG document number BMI/2013-11-01.

Business network analysis

VDML Business Networks models can be developed to represent relationships with customers, suppliers and others, typically in the context of certain collaborative activities such as a line of business, product development relationships, regulatory compliance relationships, etc. The collaboration structure and exchange of deliverables helps analysis focus on both tangible and intangible deliverables and the less obvious but important exchanges. This may lead to changes in business relationships, clarification of roles and deliverables, and development of improved internal business processes.

Customer value analysis

Customer value analysis will be based on value stream analysis. A basic value proposition must be defined to identify values of concern to a customer or customer community. Then the value proposition must be linked to collaboration and activity value contributions. In the absence of an existing VDML model, development of the value stream will typically be performed top-down, considering high-level activities or stages of the business and breaking these down into a hierarchy of activities and deliverable flows. The values of a customer value proposition are traced back up the value stream to identify sources of value and potential improvements. The level of detail will be driven by the level of confidence in the measurements and the need to drive down to specific capabilities and responsible organizations. This could start with BPMN processes, but typically this requires digging through too much detail. The analysis will lead to focused consideration of process improvements (BPMN).

The analysis of customer value will lead to identification of activities might be improved to improve customer value. This may also lead to a "capability map" that provides a visual breakdown of the capabilities required by the activities with highlights to identify those capabilities that need improvement.

Capability analysis

Capability analysis can go beyond the identification of critical capabilities in a value stream. In a large enterprise, multiple product lines or lines of business may require some of the same capabilities. Capability

analysis should identify similar capabilities, consider ways they can cooperate to improve their performance, and assess the potential to consolidate for economies of scale. This may drive process improvements or development of processes for a shared capability to meet the needs of multiple lines of business.

When capabilities are consolidated as shared services, there is a need to understand their performance in multiple contexts. Changing a capability for one line of business could adversely affect another. In addition, analysis of investments in improvement of capabilities should consider how each capability affects all of its uses to provide an enterprise-level perspective on allocation of investments. The implementation of improvements will likely involve process improvements based on the capability analysis.

New product/LOB analysis

Development of business operations for a new product or line of business should involve leveraging existing capabilities. VDML supports modeling the business at a level of abstraction that identifies capability requirements without requiring a lot of detailed process analysis. Capability requirements can be developed at an appropriate level of detail for strategic planning, to identify existing capabilities and any changes to their requirements along with requirements for new capabilities. This provides a framework for more detailed analysis, such as process modeling, to focus on the areas the need development. Business leaders can more quickly assess both the cost and time required to realize the needed production capability and make an informed business decision on the potential success of the product.

Merger or acquisition

Mergers and acquisitions, if they are to realize any synergy, require some consolidation of capabilities. VDML models can be used during due diligence to get a better understanding of the similarities and differences of the enterprises and the potential for sharing capabilities. While the enterprises may have similar products, that does not mean they have similar ways of doing business.

Here a transformation of BPMN to VDML, assuming processes are specified with BPMN, may be an expeditious way to develop VDML models for comparison. If transformation is not possible, or if starting at business processes is too big an undertaking, VDML can still support value stream models at levels of detail appropriate to confirm similarities or expose differences so that much of the modeling does not reach the business-process level of detail. These models will provide the basis for consideration of to-be models to realize expectations of synergy in the merger.

Strategic transformation

Strategic transformation potentially involves substantial change to the way the business operates. Starting with business processes will bias the result by making it difficult to see the forest for the trees. VDML supports a higher level view of the business where sufficient detail can be developed to realize a meaningful assessment of the scope and duration of the change, the performance expectations and competitive position as well as the framework for organization of initiatives and development of business processes and systems. Multiple VDML models can be used to define stages of transformation and define incremental change rather than one, long-term undertaking with no benefit (or failure) until the end.

Accountability

VDML provides the linkage of customer value to contributing activities, to use of capabilities and to responsible organizations so that organizations can be held accountable for poor results and recognized for important improvements. VDML will drive meaningful performance measurement, and support analysis that starts from a high level of abstraction and expands levels of detail to focus on a specific area for improvement. It will also help clarify expectations and responsibility for shared capabilities.

Annex C: Use Cases

(Informative)

Two substantial use cases were developed in support of the VDML development effort. These are independent documents that are available on the OMG server: Note that these documents are not normative. They were developed before the metamodel and notation were finalized and may include some inconsistencies with the final specification.

Manufacturing use case: http://www.omg.org/cgi-bin/doc?bmi/2012-11-10

Healthcare use case: http://www.omg.org/cgi-bin/doc?bmi/2012-11-11