EPRI

2025 TECHNICAL UPDATE

Common Information Model Primer

Eleventh Edition

Common Information Model Primer

Eleventh Edition

3002032389

Technical Update, September 2025

EPRI Project Manager
S. Crimmins

EPRI
3420 Hillview Avenue, Palo Alto, California 94304-1338 USA
800.313.3774 ▪ 650.855.2121 ▪ askepri@epri.com ▪ www.epri.com

DISCLAIMER OF WARRANTIES AND LIMITATION OF LIABILITIES

THIS DOCUMENT WAS PREPARED BY THE ORGANIZATION(S) NAMED BELOW AS AN ACCOUNT OF WORK SPONSORED OR COSPONSORED BY THE ELECTRIC POWER RESEARCH INSTITUTE, INC. (EPRI). NEITHER EPRI, ANY MEMBER OF EPRI, ANY COSPONSOR, THE ORGANIZATION(S) BELOW, NOR ANY PERSON ACTING ON BEHALF OF ANY OF THEM:

(A) MAKES ANY WARRANTY OR REPRESENTATION WHATSOEVER, EXPRESS OR IMPLIED, (I) WITH RESPECT TO THE USE OF ANY INFORMATION, APPARATUS, METHOD, PROCESS, OR SIMILAR ITEM DISCLOSED IN THIS DOCUMENT, INCLUDING MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, OR (II) THAT SUCH USE DOES NOT INFRINGE ON OR INTERFERE WITH PRIVATELY OWNED RIGHTS, INCLUDING ANY PARTY'S INTELLECTUAL PROPERTY, OR (III) THAT THIS DOCUMENT IS SUITABLE TO ANY PARTICULAR USER'S CIRCUMSTANCE; OR

(B) ASSUMES RESPONSIBILITY FOR ANY DAMAGES OR OTHER LIABILITY WHATSOEVER (INCLUDING ANY CONSEQUENTIAL DAMAGES, EVEN IF EPRI OR ANY EPRI REPRESENTATIVE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES) RESULTING FROM YOUR SELECTION OR USE OF THIS DOCUMENT OR ANY INFORMATION, APPARATUS, METHOD, PROCESS, OR SIMILAR ITEM DISCLOSED IN THIS DOCUMENT.

REFERENCE HEREIN TO ANY SPECIFIC COMMERCIAL PRODUCT, PROCESS, OR SERVICE BY ITS TRADE NAME, TRADEMARK, MANUFACTURER, OR OTHERWISE, DOES NOT NECESSARILY CONSTITUTE OR IMPLY ITS ENDORSEMENT, RECOMMENDATION, OR FAVORING BY EPRI.

EPRI PREPARED THIS REPORT.

This is an EPRI Technical Update report. A Technical Update report is intended as an informal report of continuing research, a meeting, or a topical study. It is not a final EPRI technical report.

NOTE

For further information about EPRI, call the EPRI Customer Assistance Center at 800.313.3774 or e-mail askepri@epri.com.

Together…Shaping the Future of Energy®

© 2025 Electric Power Research Institute (EPRI), Inc. All rights reserved. Electric Power Research Institute, EPRI, and TOGETHER…SHAPING THE FUTURE OF ENERGY are registered marks of the Electric Power Research Institute, Inc. in the U.S. and worldwide.

ACKNOWLEDGMENTS

EPRI prepared this report.

Principal Investigators
G. Gray
D. Lowe
R. Rhodes
J. Simmins

EPRI acknowledges the research work undertaken at the University of Strathclyde's Institute for Energy and Environment and various publications by A. W. McMorran, which have influenced various editions of this report.

EPRI acknowledges contributions by Alvaro Rodriquez and Chavdar Ivanov of ENTSO-E in related sections of this report.

This report describes research sponsored by EPRI.

This publication is a corporate document that should be cited in the literature in the following manner: *7BCommon Information Model Primer: Eleventh Edition.* EPRI, Palo Alto, CA: 2025. 3002032389.

ABSTRACT

The *Common Information Model Primer* helps readers understand how the Common Information Model (CIM) is used at electric utilities. The primer explains how the CIM originated and was developed through the years, with the support of working groups of Technical Committee 57 of the International Electrotechnical Commission.

The basics of Unified Modeling Language are explained for readers new to the language of the CIM. Building from examples of commonly understood objects, the concepts underlying the CIM are introduced in a step-by-step fashion. This conceptual understanding prepares the reader for an introduction to the complexities of electric power system modeling and data exchange approaches. The ninth edition of this report describes how to create enumerations to free string attributes without directly modifying the canonical CIM. This externalization of enumerations into local extensions allows for the enforcement of stricter data requirements while still being compatible with canonical CIM messages.

Learning aids include questions and answers to reinforce learning of key CIM concepts as well as case studies illustrating how utilities and vendors leverage the CIM for business benefit.

Keywords

Common Information Model (CIM)
International Electrotechnical Commission (IEC)
International standards
Semantic model
Unified Modeling Language (UML)
Application integration
Data integration

REPORT SUMMARY

The *Common Information Model Primer* explains the basics of the Common Information Model (CIM) for readers with limited background in utility power system models and standards. The primer describes how the CIM originated and grew through the various working groups within Technical Committee 57 of the International Electrotechnical Commission (IEC). The process of how an IEC standard is created is also outlined (the CIM consists of three IEC standards: 61970, 61968, and 62325).

Unified Modeling Language (UML) basics are introduced to help the reader understand the language of the CIM. Then, building on commonly understood objects (basic shapes), the primer introduces the concepts underlying the CIM. The reader is then brought into the world of power systems and the complexities of power equipment modeling.

Each edition of the primer has been updated with additional learning tools, including typical case studies, key questions and answers, and lists of tools that are helpful in modeling and integration.

Background

The CIM is a set of open standards for representing power system components originally developed by EPRI in North America and is now a series of standards under the IEC. The CIM began as part of the Control Center Application Programming Interface project at EPRI with the aim of developing a common definition for power system components for use in the Energy Management System (EMS) Application Programming Interface; the CIM is now maintained by IEC Technical Committee 57 Working Group 13 as IEC 61970-301. The format was adopted by EMS vendors to allow the exchange of data between their applications, independent of their internal software architecture or operating platform. IEC 61968-11 extends this model to cover other aspects of power system software data exchange such as asset tracking, work scheduling, and customer billing. The CIM for electricity markets then extended both of these models with IEC 62325-301 to cover the data exchanged between participants in electricity markets. These three standards—61970-301, 61968-11, and 62325-301—are collectively known as the CIM for power systems and currently have three primary uses: to facilitate the exchange of power system network data between organizations, to allow the exchange of data between applications within an organization, and to exchange market data between market entities.

Objectives

The objective of this work is to make the CIM understandable to individuals with little or no background with standards, UML models, or systems integration. This research is designed for line-of-business professionals who want a better understanding of integration technologies used in their operational technology (OT) environment as well as information technology professionals wanting a basic introduction to power system modeling within OT systems.

Approach

The material contained in this report has been developed over a period of years from many resource documents. These include short courses given on the subject, EPRI archives, careful reading and understanding of international standards, doctoral-level research publications, and feedback from practitioners who have used the CIM for systems integration.

Results

The resulting primer synthesizes an array of resources into easily grasped concepts, arranged to help the reader understand the complexities of modeling data exchanges among systems at electric power utilities.

EXECUTIVE SUMMARY

As a result of nearly three decades of industry collaboration, the utility Common Information Model (CIM) provides foundational benefit to the industry and ultimately, to utility customers. EPRI sponsored the initial research and continues to promote this collaboration. EPRI updates this primer regularly as a way of encouraging ongoing adoption and expansion of these industry standards.

This primer builds on common, understandable concepts and takes the reader into the application of Unified Modeling Language to power system modeling. It explains how the standards are used to support interapplication data exchange and how the CIM is extended to address new industry needs.

Readers who delve into this primer will come away with a deep understanding of the concepts surrounding the CIM. This foundational understanding of the CIM will help engineering and operations professionals understand how standard data models can facilitate information exchange among the operational technology applications they use. Information technology professionals at power utilities will likewise find this work to be useful in their understanding of how the CIM supports semantic modeling for enterprise-wide application integration and information management.

CONTENTS

1 Introduction 1

Learning Objectives 1

Additional Resources 2

Case Study 2

Section 1 Questions 3

2 General Background 4

Learning Objectives 4

Proprietary Power System Data Formats Present an Integration Challenge 4

Enterprise-Level Systems Integration Requires Extensible Data Exchange Formats 4

Information Modeling Supports Data Abstraction 6

Three Models for Data Management Must Be Considered 6

Information Models Can Be Defined for Existing Applications 7

Information Models Help Separate Structure from Format 7

CIM Standards Provide a Semantic Model 8

Case Study 8

Section 2 Questions 9

3 CIM Background 10

Learning Objectives 10

Origins of the Utility CIM 10

Technical Committee 57 Oversees Power Systems Modeling 10

Working Group 13 Focuses on Power System Models 11

Working Group 14 Focuses on Distribution Management and Enterprise Integration 11

Working Group 16 Focuses on Deregulated Energy Market Communications 11

Working Groups are Evolving with Grid Modernization Efforts 12

Standards Process 13

Step 1: New Work Item Proposal 13

Step 2: Working Draft and Committee Draft 13

Step 3: Committee Draft for Vote 13
Step 4: Final Draft International Standard 13
Standards Cycles Can Take Years of Time 14
Case Study 14
Section 3 Questions 15

4 Introducing UML 16
Learning Objectives 16
UML Makes Power System Modeling Possible 16
Classes 16
Inheritance 17
Association 18
Aggregation 19
Composition 20
Summary 21
Extensible Markup Language Facilitates CIM-Based Data Exchange 22
XML Syntax 22
Simple XML Example 23
XML Schema 23
Resource Description Framework Supports Further Structure for XML 26
RDF Syntax 26
Simple RDF Example 27
RDF Schema 28
Case Study 30
Section 4 Questions 31

5 UML for CIM 32
Learning Objectives 32
Overview 32
CIM Class Structure 32
Identity 32
Using Class Structure for Defining Components: Breaker Example 34

Subclasses of Switch 36
Defining Component Interconnections 38
Translating a Circuit into CIM 41
Identifying CIM Components 42
Transformers Prior to CIM v15 43
Transformers After CIM v15 46
Equipment Containment 51
Units and Language 52
CIM Representation 53
CIM Supports Both Asset and Functional Views 54
Modeling the Asset View 54
Benefits of Modeling an Asset View 55
CIM Supports Network Diagram Layouts 57
Separating Style from Layout 57
Diagram Layout in UML 59
Embedding Geographical Data 60
Modeling Unbalanced Multi-Phase Networks 61
Measurements 62
Three Phase Switches 62
Three Phase Transformers 62
Loads 62
Lines 63
UML for Unbalanced Network Components 63
CIM Packages Overview 64
IEC 61970-301 64
IEC 61968-11 64
IEC 62325-301 65
Case Study 65
Section 5 Questions 66

6 Contextual Modeling 67

Learning Objectives 67

Deriving Profiles 67
Moving from Information Modeling to Implementation 68
Profiles Can Be Combined Into Groups 70
Some CIM Profiles Are Standards 70
Common Power System Model 71
Common Distribution Power System Model 72
ENTSO-E 73
Case Study 74
Section 6 Questions 74

7 CIM RDF and XML 76
Learning Objectives 76
Mapping CIM to RDF 76
CIM RDF Serialization Examples 77
Object/Node Identification 80
Identifier Scope 81
Extensibility and Efficiency 81
Extensibility 81
Compression and Efficiency 82
CIM Difference Model 83
Difference Model RDF XML Format 83
CIM RDF Headers 87
RDF Schema Definition in CIMTool 89
Defining the Message Structure 89
Generating RDF Schema 98
RDFS Creation Summary 102
Case Study 102
Section 7 Questions 103

8 CIM XSD Messages 104
Learning Objectives 104
Mapping Application Interfaces to CIM 104

Enterprise Service Bus 105
Mapping CIM Profiles to XML Schema 106
Existing Application Data 106
XSD Definition in CIMTool 107
Defining the Message Structure 107
Generating XML Schema 115
XML Instance Data 118
XML Messaging Summary 119
Case Study 119
Section 8 Questions 120

9 Transforming Data to CIM 121
Learning Objectives 121
Bi-directional Transformation 121
Stateless vs. Stateful Transformation 121
Model-Driven Transformation 122
Example: CIM to PSS/E 124
Defining the PSS/E Meta-Model 125
Parsing/Writing PSS/E RAW 125
Transforming CIM to PSS/E 125
Benefits of a Model Driven Architecture 130
Case Study 130
Section 9 Questions 131

10 Extending the CIM 132
Learning Objectives 132
Accommodating Legacy Data 132
Adding a New Attribute 132
Adding New Classes 133
Introducing New Application Areas – CIM for Gas 135
Adding New Classes 135
Refining Existing Classes 136

Assigning a Gas Supply Point 136
Gas Containment 137
Creating Extensions in the CIM using Shadow Classes 138
Extensions Summary 140
Internal vs. Global Extensions 140
Adding a New Enumeration Through Class Extensions 141
Case Study 145
Section 10 Questions 145

11 Application Integration 147
Learning Objectives 147
Verbs 148
Nouns 148
Integration Patterns 149
IEC 61968-100 2nd Edition Updates 151
Case Study 152
Section 11 Questions 152

12 Compliance Testing 154
Learning Objectives 154
Identifying the Actionability Gap 154
Completing the First CIM Compliance Test 154
Case Study 155
Section 12 Questions 156
Additional Resources 156

13 Web Service Integration 157
Learning Objectives 157
RESTful and SOAP Web Services 157
JSON Versus XML 158
Extensible Markup Language 158
JavaScript Object Notation 159

Mapping CIM Message from XML to JSON 159
Security 162
Case Study 162
Additional Resources 162
Section 13 Questions 163

14 Creating Schemas from the CIM 164
Learning Objectives 164
Going Beyond the XSD 164
Schema Composer 165
How-To: Schema Composer 165
Schema Composer Summary 167
CIMConteXtor 167
How-To: CIMConteXtor 168
CIMConteXtor Summary 173
CIMTool 173
Summary 174
Case Study 174
Section 14 Questions 175

15 Implementing CIM in the Organization 176
Learning Objectives 176
The urge to customize and the tale of two standards: Green Button vs ODIN 181
CIM is the age of AGILE 182
Case Study 182
Section 15 Questions 184
Additional Resources 184

A CIM Success Stories 185
Success Stories: ESB Networks Use of CIM for Substation Configuration 185
The Utility 185
The Problem 186

The CIM Solution 186
The Impact 187
Success Stories – The Green Button Standard 187
The Problem 187
The CIM Solution 188
The Impact 189
Success Stories – Consumers Energy 191
The Problem 191
The CIM Solution 191
The Impact 192
Success Stories – Public Service Enterprise Group – Long Island (PSEG-LI) 192
The Problem 192
The Solution 193
The Impact 194
Success Stories – Idaho Power Company (IPC) 194
The Problem 194
The CIM Solution 195
The Impact 195
Success Stories – Oracle Utility Data Model (OUDM) 196
The Problem 196
The CIM Solution 196
The Impact 197
Success Stories – Sempra Energy 197
The Problem 198
The CIM Solution 198
The Impact 199
Success Stories – Southern California Edison Uses CIM for Green Button Support 199
Problem 199
Solution 199
Impact 200
Success Stories – PNNL Uses CIM for GridAPPS-D™ 200

The Problem 200
The Solution 200
The Impact 201

B Answers to Section Questions 203
Answers to Section Questions 203
Section 1 203
Section 2 203
Section 3 203
Section 4 203
Section 5 203
Section 6 204
Section 7 204
Section 8 204
Section 9 204
Section 10 204
Section 11 205
Section 12 205
Section 13 205
Section 14 205
Section 15 205

LIST OF FIGURES

Figure 4-1. The Circle Class 17
Figure 4-2. Class hierarchy for diagram shapes 18
Figure 4-3. Class hierarchy of diagram shapes with a Style 19
Figure 4-4. Aggregation Example with Layer and Shape 20
Figure 4-5. Composition Example of Shape and Anchor 20
Figure 4-6. Class Diagram for Shapes 21
Figure 4-7. Annotated simple XML Schema Example describing the data within a book 24
Figure 4-8. XMLSpy Style XML Schema View 25
Figure 5-1. IdentifiedObject and the Name Classes 33
Figure 5-2. Breaker Class Hierarchy in CIM 35
Figure 5-3. Switch Class with expanded subclass hierarchy 36
Figure 5-4. Switch class diagram with new types of Switch and Breaker 37
Figure 5-5. Connectivity Example Circuit 38
Figure 5-6. Connectivity Example Circuit with Direct Associations 39
Figure 5-7. Connectivity Example Circuit with Connectivity Node 39
Figure 5-8. Conducting Equipment and Connectivity class diagram 40
Figure 5-9. Connectivity Example Circuit with Connectivity Node and Terminals 41
Figure 5-10. Connectivity Example Circuit with Additional Breaker Terminal 41
Figure 5-11. Example Circuit as a Single Line Diagram 42
Figure 5-12. Example Circuit with Partial CIM Class Mappings 43
Figure 5-13. Transformer Model (pre-CIM v15) 44
Figure 5-14. CIM Transformer Instance Example 45
Figure 5-15. Transformer Class Diagram CIM15+ 47
Figure 5-16. Balanced Power Transformer Classes 48
Figure 5-17. CIM v15 Transformer Instance Example 48
Figure 5-18. CIM v15 Transformer with multiple tanks instance example 49
Figure 5-19. Tap Changer Class Diagram CIM v15 50
Figure 5-20. Complete CIM Representation 53
Figure 5-21. Excerpt of Functional Model with Asset Class 55
Figure 5-22. Breaker-Asset Instance Example 56
Figure 5-23. Breaker-Asset Change Example 56
Figure 5-24. CIM Diagram Layout Single Point Examples 57

Figure 5-25. CIM Diagram Layout Multiple Point Example 58
Figure 5-26. CIM Diagram Layout Class Diagram 59
Figure 5-27. CIM Location Class Diagram 61
Figure 5-28. Unbalanced three-phase network components 63
Figure 6-1. IEC61968-1 Interface Reference Model 67
Figure 6-2. IEC61968-1 Interface Reference Model – Current Revision and Work in Progress 68
Figure 6-3. Information, Contextual and Implementation Models 68
Figure 6-4. Example Information to Contextual Models 69
Figure 6-5. EMS Network Analysis Systems and Data Exchanges 71
Figure 7-1. Transformer Shown as four CIM Objects with attributes 78
Figure 7-2. CIM Header UML Model 87
Figure 8-1. Communications between enterprise applications 104
Figure 8-2. Enterprise Service Bus model for inter-application communication 105
Figure 8-3. CIM Interface Mapping 107
Figure 8-4. Creating a new CIMTool profile 108
Figure 8-5. Empty Profile 108
Figure 8-6. Selecting a class to be added to the profile 109
Figure 8-7. Profile with PowerTransformer class 109
Figure 8-8. Marking the PowerTransformer class as concrete 110
Figure 8-9. Add PowerTransformerEnd association for PowerTransformer class 110
Figure 8-10. Add PowerTransformerEnd class to message 111
Figure 8-11. Select associated class for PowerTransformerEnd association 111
Figure 8-12. PowerTransformerEnd Native Attributes 112
Figure 8-13. Add inherited attribute from TransformerEnd 113
Figure 8-14. Optional PowerTransformerEnd attributes 113
Figure 8-15. TransformerData Message Outline 114
Figure 8-16. Enable XSD Builder in CIMTool 115
Figure 8-17. TransformerData XSD Schema Element 115
Figure 8-18. PowerTransformer XML Schema Element 116
Figure 8-19. PowerTransformerEnd XML Schema Element 116
Figure 8-20. BaseVoltage XML Schema Element 116
Figure 8-21. Screenshot of raw XML Schema XML 117
Figure 8-22. XMLSpy style XML Schema view 117
Figure 9-1. Multiple Serialization Formats/Locations 122

Figure 9-2. Model Driven Transformation Process 123
Figure 10-1. Adding an extension to an existing class 133
Figure 10-2. Adding a new class to the model 134
Figure 10-3. Gas Transfer Equipment classes 135
Figure 10-4. Refinement of CIM Fossil Fuel class for Natural Gas 136
Figure 10-5. Connecting the Gas and Electrical models 137
Figure 10-6. Gas Pressure Level Containment Classes 138
Figure 10-7. Extending with shadow classes 139
Figure 10-8. Extended CIM UML 140
Figure 10-9. Future CIM with integrated extensions 141
Figure 10-10. Adding a custom extension to the CIM using Stereotypes 142
Figure 10-11. EnergyMarket class in the CIM 143
Figure 10-12. The new enumeration pointing to the datatype upon which it is based 144
Figure 10-13. Changing out the attribute for its enumeration at the profiling level 144
Figure 11-1. Basic request/response pattern 149
Figure 11-2. Basic request/response pattern using the Create verb and the EndDeviceEvent object 149
Figure 11-3. Example of message integration required for a remote disconnect of a smart meter 150
Figure 14-1. Adding a class to the new profile from the 61970 package 165
Figure 14-2. Schema Composer adding classes based on options chosen 166
Figure 14-3. Setting property restrictions in Schema Composer 166
Figure 14-4. Schema Composer schema selection options 167
Figure 14-5. First step in creating a profile using CIMConteXtor 168
Figure 14-6. Choosing which CIM packages will be used 168
Figure 14-7. Dialogue from CIMConteXtor to create copy of CIM class for its profile 169
Figure 14-8. Setting the attributes in a class using CIMConteXtor 169
Figure 14-9. Adding the association between two classes 170
Figure 14-10. Establishing the association between the added class and its parent 170
Figure 14-11. Selecting containment options for class associations 171
Figure 14-12. Sample completed profile view in CIMConteXtor 171
Figure 14-13. CimSyntaxGen JSON Schema example 172
Figure 14-14. Setting namespace, URI, and filename for schema 172
Figure 14-15. A list of builders CIMTool supports 173
Figure 15-1. Kotter's (modified) Eight-Step Change Process 177

Figure A-1. Data sources and stakeholders 185
Figure A-2. Data source, corresponding networks, and categories of data 186
Figure A-3. SERVO web-based application with automated data feeds 187

LIST OF TABLES

Table 11-1. CIM Verbs and their usage in naming patterns 148
Table 14-1. A comparison of tools used to generate schemas in the CIM..... 174

1. INTRODUCTION

Learning Objectives

- Understand the context of CIM development
- Understand the structure of each section: content, case study, and questions

Market regulatory changes and the growth in variable energy resources around the world requires power companies to exchange information on a regular basis. This information exchange supports reliability standards for the interconnected power networks owned and operated by many different power producers, transmission operators, vertically integrated utilities and market participants. Power companies use a wide range of systems and processes for managing and moving data across IT and OT systems. Along with this, new applications of data science and advanced analytics are creating more demand for higher-quality data across the utility enterprise and across power markets.

There is often a need to exchange information both internally between different applications as well as externally with applications at other companies. Using proprietary formats for sharing data between applications has typically required a myriad of translators to import and export data. This resulted in exponential growth in complexity as increasing numbers of applications exchanged data. The need to simplify integration complexity then drove adoption of a model-based data exchange approach in the electric power domain.

The CIM comprises three series of standards, each with a specific focus, but all are based on a single, unified model.

Standard	Modeling Focus
IEC 61970	Information exchange among systems directly involved with operation and planning of the overall interconnected electric grid which rely on power system network models to analyze the behavior of the entire interconnected grid at all voltage levels. This often involves interactions between systems at various different participants in the grid (that is, RTO, TSO, DSO, microgrid, generator, consumer).
IEC 61968	Information exchange among systems supporting business functions that support power system operations, maintenance and customer support. This includes major business functions such as asset management, work management, meter data management, customer information, geographic information systems and engineering design.
IEC 62325	Information exchange among systems directly involved with electricity market business processes such as transmission capacity allocation, forecasting, bidding, contracts, clearing and settlement.[1]

[1] In more modern models, these packages are now known as Grid, Support, and Market respectively.

These standards have three primary uses: to facilitate the exchange of power system network data between organizations; to allow the exchange of data between applications within an organization; and to exchange market data between organizations.

CIM development began primarily in North America and gained industry momentum when ERCOT implemented a CIM-based approach for managing the network model data required by its nodal market. Recent expanded CIM adoption was driven by the European Network of Transmission Systems Operators for Electricity (ENTSO-E) requiring CIM-based exchanges for compliance with grid codes. Expanded usage by a number of other European, Asian, and South American utilities has caused CIM adoption to grow well beyond its North American roots.

Additional Resources

EPRI has published several pieces of research on the subject of CIM based network model data management, including:

- *Network Model Manager and Repository: A Guide to Exploring the Potential of Centralized Network Model Management for the Interested Utility* (3002000609)
- *Network Model Manager Technical Market Requirements: The Transmission Perspective* (3002003053)
- *Example Request for Proposal for Transmission Network Model Manager Tool* (3002009552)
- *Using the CIM for Network Analysis Data Management* (3002002587)
- *Case Study on Network Model Management Solution Design* (3002009610)
- *Leading Practices for Network Model Management* (3002014082)

Case Study

The primer will follow a large utility, serving millions of customers, on its journey as it goes from using vendor proprietary interfaces and integration, to using the CIM as a core part of its enterprise semantic model, and using CIM-based adapters to translate from proprietary interfaces at the enterprise service bus, to create a uniform approach to back-office integration.

This case study will follow Jeff Kimble, manager of the integration effort at Electric Innovation Utility (EIU), as he learns that a model such as the CIM exists, how a semantic model is used, how objects are represented in the model, how the profiles are created to support messaging, and some techniques for extending the model when gaps are found.

Jeff is frustrated with the high cost and difficulty of integrating systems, and with vendor adapters that do not work as advertised, has recently learned that something called the “CIM” exists, and he attends an industry conference to find out more. Additionally, Electricity Innovation Utility has recently started a series of strategic projects to streamline operations, and charged with reducing head count by designing and implementing workflows that cross traditional utility interdepartmental boundaries. Jeff’s job is to facilitate these strategic initiatives by integrating back office applications.

Section 1 Questions

1. IEC 61970 covers the definition for:
    a. Power system data exchange
    b. Distribution systems
    c. Energy market information exchange
    d. Water and Gas systems
2. "CIM" stands for:
    a. Complete Information Management
    b. Common Industry Meaning
    c. Common Information Model
    d. Conditional Integrated Models
3. IEC 62325 covers the definition for:
    a. Power system data exchange
    b. Distribution systems
    c. Market information exchange
    d. Gas Systems
4. IEC 61968 cover the definitions for:
    a. Power system data exchange
    b. Exchange of electric operating data other than network models
    c. Energy market information exchange
    d. Gas Systems
5. The CIM is useful because it provides:
    a. A semantic model that describes the components of a power system
    b. A non-proprietary way to describe interfaces
    c. While describing a power system, it can also be used to define energy market exchanges
    d. All of the above

2. GENERAL BACKGROUND

Learning Objectives

- Understand the three basic types of models
- Understand the concept of separating structure from format

Power system engineers have utilized digital computing in a wide variety of applications, whether for performing complex analysis calculations in power system studies or for controlling real-time network operations. All power system applications require digital storage and exchange of data about the electric grid.

Proprietary Power System Data Formats Present an Integration Challenge

Large enterprise systems such as an Energy Management System (EMS), Enterprise Asset Management (EAM) system or Meter Data Management System (MDMS) typically use proprietary database schemas defining the structure of data storage within their software products. These schemas are customized to support the application's specific technical and functional requirements. Likewise, file formats for "offline" power system-related applications used in planning and operations (typically referred to as "OT systems" used in control centers) have proprietary formats. Those formats have been extended over many years as the products have matured.

These formats work perfectly when used only for exchanging the data required for that particular application; however, they are based on fixed formats and do not lend themselves very well to extensibility. The addition of extra items can potentially break any software trying to parse the data when it finds unknown entries. These formats typically lack metadata, which keeps files compact, but wider usability is compromised since the formats are not self-describing. Further, if the documentation is proprietary and not publicly available it is often difficult to determine the attributes and characteristics necessary to build an interface that leverages the file format.

Enterprise-Level Systems Integration Requires Extensible Data Exchange Formats

In modern utilities' IT infrastructures, large-scale applications such as the EMS and asset-management system communicate with each other, generally using a vendor's custom format based on the internal database schema. In the past this often required the user to purchase each piece of enterprise-level software from the same vendor to ensure compatibility when integrating them or to write an interface to translate between different vendors.

As the power industry moved toward deregulation, and more utilities moving away from writing their own custom software solutions, resulted in multiple utilities running software

from a number of different vendors and having to exchange large data sets on a regular basis. The use of proprietary, custom formats complicates this exchange, requiring complex translation between each of the custom formats.

Similarly, offline applications use a rigid, proprietary format containing only the data required by that particular version of the application. When subsequent versions of the program require additional details the file format is changed, resulting in multiple formats for a single application. This practice is common in the software industry, not just with power system applications. However, the impacts of these changes are minor since vendors typically provide import tools to convert prior file formats into the newest version format.

Larger challenges arise when data must be exchanged at an enterprise level. When companies need to exchange data between software applications from different vendors, or have multiple versions of the same software product in production use, organizations must consider the best strategy. Such a scenario typically presents the following options:

1. Maintain multiple copies of the same data in multiple formats.
2. Store the data in a format compatible with every piece of software, requiring the removal of application-specific data and a subsequent loss in precision.
3. Store the data in a single, highly-detailed format and create software to translate from this highly-detailed format to the desired application file formats.
4. Use a highly detailed format that is compatible with every application and whose standard format contains the basic data required to represent the power system while simultaneously allowing additional, detailed, application-specific data to be contained without invalidating the format.

The third option requires additional software engineering on the part of the company to create translation tools, but requires them to maintain only a single format containing all the data required.

The fourth option represents the ideal solution, allowing a company to maintain a single, highly detailed format that is compatible with any of their software. This option does, however, depend on three factors:

- A highly detailed model to describe the power system
- A file format that supports extensibility without affecting the core data
- Acceptance of this data model and format by power system software vendors and utility buyers (whether motivated by economic or regulatory reasons)

The Common Information Model (CIM) for Power Systems addresses the first factor, by supplying a highly detailed power system model which addresses design and operational differences across the globe. The model is expressed in Unified Modeling Language (UML), which is widely accepted as a software development standard. The second factor can be addressed with eXtensible Markup Language (XML), combined with the Resource Description Framework (RDF). The remaining requirements are more of a commercial and regulatory

challenge than a technical one. Universal acceptance of this format requires both utilities and vendors to acknowledge the benefits of adopting the standard. At present, many major power system application vendors are active participants in IEC working groups which extend these standards. Many have participated in CIM Interoperability tests for power system model exchange and enterprise-level message exchange.

Information Modeling Supports Data Abstraction

> "An information model is a representation of concepts, relationships, constraints, rules, and operations to specify data semantics for a chosen domain of discourse"
>
> *-- Y. Tina Lee, Information Modeling: From Design to Implementation, NIST, 1999*

Information modeling is an approach to dealing with and managing data that uses an abstract, implementation-agnostic model to describe the structure of the entities within a particular software domain. This approach is characteristic of Model-Driven Engineering (MDE), which promotes the use of conceptual models and abstract object representations. One instance of MDE is the Object Management Group's Model Driven Architectures (MDA)[2] within which file formats, database schemas, and internal application data structures can all be derived from a model.

Three Models for Data Management Must Be Considered

From the perspective of a software architect there are really three models that need to be taken into consideration when writing software or designing a system architecture:

- The External Model
- The Internal Model
- The Conceptual Model

The **External Model** describes the data that is exposed to the user or will be shared outside of the application. For example, this external model would define the data exposed as part of a user-interface or written to a file.

The **Internal Model** describes how the application or system stores the data internally, whether within in-memory data structures or a database schema. This data is available to the internal processes and algorithms.

The **Conceptual Model** is the abstract definition that integrates the internal and external model. Ideally the internal and external models are both derived from the conceptual model.

[2] For more information see the Object Management Group (OMG) website for MDA: https://www.omg.org/mda/.

In some cases, the three models will be the same, but often it is impossible to realize an internal or external model directly from the conceptual model in situations where the conceptual model uses concepts that cannot be directly mapped into the implementation technology. For example, as will be described in more detail later, UML supports *inheritance*, where a *class* inherits the properties of its parent class.

A normal database schema does not have a concept of inheritance for its table definitions, but there are multiple ways to *map* a class structure into a database and thus generate an *internal model*[3]. The internal model will in essence be the database schema, which differs from the conceptual model, but can be automatically derived from it.

Similarly, the external data that is exposed to the user or exported to other applications may only be a subset; of the overall conceptual model, which covers all data used by the application even if it is only used by the internal processes. The external model can thus be auto-generated from the conceptual model as a subset, still closely aligned but not identical.

Information Models Can Be Defined for Existing Applications

For situations where an application already exists and the conceptual model is not being created from scratch, there is often a need to extract an information model from the pre-existing data structures or file formats. This will make it easier to understand and interpret the application's data and also allow Model Driven Architecture (MDA) based technologies such as Model Driven Transformation to be used with the data.

In the simplest case an information model already exists for the data as the original architects for the software, system or standard used a model-driven methodology.

On other occasions a simple conceptual model can be automatically derived from existing data structures such as a database schema or XML Schema Definition (XSD). Such a model will mirror an internal or external model and so additional manual user manipulation may be required to refine the model and harmonize internal and external models into the conceptual model.

The most time-consuming situation is when there is no means to automatically derive a model and it must be manually created. This can be accomplished by interpreting documentation or reverse-engineering code to identify the data structures.

Information Models Help Separate Structure from Format

By defining an information model for an application or system, the structure of the data is decoupled from its serialization format. This has the benefit of making it easier to understand the data within the system as abstract entities rather than as a particular line or column of text within a file, or as one or more columns and tables in a database. This also allows the same data to be serialized in multiple different formats without impacting its structure and definition.

[3] Douglas Barry, Torsten Stanienda, "Solving the Java Object Storage Problem," Computer, vol. 31, no. 11, pp. 33-40, Nov. 1998, doi:10.1109/2.730734.

An information model can be used to derive database schemas, file formats, user-interfaces and often documentation and interfaces automatically. As such it allows a decoupling of the data from one particular format enabling it to be saved to or loaded from multiple locations and formats without any loss of data. From a software engineering perspective, the model should be defined before any software is written or from the perspective of a systems integrator, the model should be created before the interfaces are defined at any level of detail.

By having a well-defined data structure both users and developers will find it easier to interpret, understand and use the data without having to understand the added complexity of a particular serialization format or technology.

CIM Standards Provide a Semantic Model

The conceptual model discussed above is most often referred to in the CIM world as a semantic model. The CIM model acts as an ontology of a given business domain, or a vocabulary by which utility business objects and process can be precisely defined. The model contains business objects and the relationships between them with a clearly defined set of rules. UML is then used as the language for expressing the model with the vocabulary.

Because the CIM serves as a reference to which any interface can be mapped, it is sometimes referred to as a Canonical Data Model (CDM). Compared with one-to-one mapping methods for systems integration, mapping data sources to a common semantic model offers a much more scalable and maintainable way to manage and integrate enterprise data.

A semantic model is applied in the context of utility integration projects by organizing a discrete layer for use by adapters (that is, data converters/mapped interfaces). As the CIM is expanded and extended over time, it provides a trusted information exchange approach that is independent of individual system technologies, information architectures and applications.

Case Study

As an integration architect, Jeff Kimble is already familiar with the benefits of using XML to create self-describing file formats. But he is also familiar with having to deal with proprietary, legacy, interfaces that are typical in the utility IT landscape. In the traditional interface development lifecycle, changes tend to be “one-offs,” that is, each time a change is made it starts with the old interface specification. A copy is made of this interface, and then changes to that interface are adapted for the new specification. Finally, changes to the new interface are tested against all of the applications that used to old interface (if they are not missed in the test plan) against all of the applications the old interface was integrated with.

Jeff wants to learn more about the Model Driven Architecture approach and what is meant by a semantic model. Starting with a model of the data first, before designing the interface appears to add efficiencies in code design by front-loading the design work. Error reduction seems more likely when using a “model is king” approach, since the model can be used to generate database schemas as well as message payloads. This approach re-imagines the applications from the internal data outward, rather from the external interface inwards to the application.

Section 2 Questions

1. When using a Model Driven Architecture, three models need to be considered:
    a. Internal, external, conceptual
    b. External, conceptual, super
    c. Conceptual, perspective, integration
    d. Internal, conceptual, power system
2. The internal model describes:
    a. How the application or system stores the data
    b. The data that is exposed to the user
    c. The abstract definition of the data
    d. How the data is stored in a table
3. The external model describes:
    a. How the application or system stores the data
    b. The data that is exposed to the user
    c. The abstract definition of the data
    d. How the data is stored in a table
4. The conceptual model describes:
    a. How the application or system stores the data
    b. The data that is exposed to the user
    c. The abstract definition of the data
    d. How the data is stored in a table
5. Transforming data between different vendor solutions causes which challenges:
    a. Having multiple copies of the same data in different formats
    b. Using a highly detailed format that is compatible with every application
    c. Storing the data in a format compatible with every piece of software, requiring the removal of application-specific data and losing precision
    d. All of the above

3. CIM BACKGROUND

Learning Objectives

- Understand the history of the CIM
- Understand some of the basic components of US and European style markets
- Understand the IEC standards process and phases

Origins of the Utility CIM

Exchanging power systems data between utility companies has been problematic when proprietary formats are used. In the past a company would traditionally use a single software system, whether a custom in-house solution or whether purchased from a large software company, and there would be a single proprietary data standard and format used. With the deregulation of the power industry, the emergence of smarter grids, and the integration of consumer or third-party distributed energy resources (DERs), there is now a greater need to enable interoperability across a wide diversity of companies and systems.

The increase in choice provided by the number of power system software vendors and the different software packages and architectures available add to the challenge of data exchange. These issues point to a requirement for a single, open standard for describing electric utility data and to aid the interoperability between software packages and exchange of information both within one company and between companies.

The Common Information Model (CIM) was developed as an open standard for representing power system components. CIM was originally developed by the Electric Power Research Institute (EPRI) in North America and is now a series of standards under the auspices of the International Electrotechnical Commission (IEC). The standard was started as part of the Control Center Application Programming Interface (CCAPI) project at EPRI with the aim of defining a common definition for the components in power systems for use the Energy Management System (EMS) Application Programming Interface (API), now maintained by IEC Technical Committee 57 Working Group 13 as IEC 61970-301. The format has been adopted by the major EMS vendors to allow the exchange of data between their applications, independent of their internal software architecture or operating platform.

Technical Committee 57 Oversees Power Systems Modeling

IEC Technical Committee 57 was established in 1965 to develop standards for “Power systems management and associated information exchange”. The CIM standards are developed and maintained by three working groups under IEC TC57, working groups 13, 14, and 16. A further group, Working Group 19, is responsible for coordinating interoperability within TC 57 in the long term and harmonizes work across the TC57 working groups. This helps ensure there is no duplication of effort and promotes consistency and interoperability between the standards.

Working Group 13 Focuses on Power System Models

Working Group 13 maintains the core CIM model as language-independent UML model, defining the components of a power system as classes along with the relationships between these classes: inheritance, association and aggregation; and the parameters within each class are also defined. This provides the foundation for a generic model to represent all aspects of a power system, independent of any particular proprietary data standard or format.

This simplifies the interoperability between software applications, since there need only exist a translator to convert to and from the CIM based data, where previously there would have been the need for translators to convert to and from every other third party company's proprietary format.

The CIM as defined in WG13 covers the modeling of electrical networks from the perspective of the transmission system operator and so is focused on the definition of the electrical network and applications linked to online operations and offline analysis of the network.

Working Group 14 Focuses on Distribution Management and Enterprise Integration

Extensions to the CIM were subsequently added with the scope being that of system interfaces for distribution management systems. These extensions are maintained and developed by another working group under Technical Committee 57, Working Group 14. These extensions expanded the CIM beyond its roots in the transmission EMS world into the distribution network and the modeling of data exchanges between systems within the distribution utilities and distribution system operators (DSOs).

The original electrical model was expanded to include the modeling of unbalanced, low, and medium voltage distribution networks as well as DERs. Extensions addressed the needs of Network Operations, Records and Asset Management, Operational Planning and Optimization, Maintenance and Construction, Network Extension Planning, Customer Support, and

Meter Reading and Control. This significantly increased the scope of the CIM; although these extensions were originated for distribution utilities, there is a growing recognition that many are also applicable in the transmission domain.

This part of the CIM is then used to derive interfaces primarily for systems integration within the utility. How these messages are derived will be discussed later.

Working Group 16 Focuses on Deregulated Energy Market Communications

The CIM for Markets Extensions (CME) was created to expand the use of the CIM into the area of deregulated energy markets. These extensions cover the data required for electricity market operations including bidding, clearing and settlement. These extensions model the data that is communicated between market participants, not a model of the structure of the market itself.

There are two primary sub teams within the working group, one creating extensions for *European-style* markets, the other for *US-style* markets. These two styles have different characteristics:

European Style Markets:

- Ahead Markets: Bilateral
- Intra-Day Markets
- Balancing markets
- Collaboration with ENTSO-E

US Style Markets:

- Day Ahead Markets with Security Constraint Unit Commitment (SCUC)
- Hour Ahead Markets
- Real Time Markets with Security Constraint Economic Dispatch (SCED)
- Collaboration with ISO/RTD Council and ISO projects

Electricity markets often vary between implementations and so CME is seen as a starting point on which extensions can be added to support a particular market implementation.

Working Groups are Evolving with Grid Modernization Efforts

In recent years WG 13 and WG 14 participants have recognized that the respective roles of the working groups have been evolving in concert with the changes underway with grid modernization. These working groups were originally structured to follow the common utility separation of transmission and distribution organizations, business processes, and computing systems. Today, while there are still differences, that distinction is proving to be more of a hindrance than a benefit.

For example, rapid DER adoption in some markets drives the increased need for network model data sharing in the distribution domain and across the boundary with transmission. Likewise, DER aggregators will participate in markets, competing with independent power producers and vertically integrated utilities to provide grid services to transmission operators. This will shift traditional distinctions between generation, transmission, and distribution domains. DER providers will grow in complexity, in some cases managing energy storage systems, EV recharging stations, home energy systems and many other types of resources – all of which will require some level of modeling.

Around 2016, a decision was made that WG 13 would assume responsibility for distribution network model data. In the same time range, the value of the WG14 CIM asset model to data exchange in the transmission domain was recognized. The working groups now divide modeling responsibility primarily by the type of data as opposed to the voltage level. In addition, WG 16 is currently sponsoring a DER modeling task force where working groups 10, 13, 14, and 16 are represented.

These changes are designed to align with a new Interface Reference Model that is currently under development and to more clearly identify work gaps and responsibilities. The evolution of these working groups will continue into the future years, guided by Working Group 19.

Standards Process

The International Electrotechnical Committee is based in Switzerland and, as a global standards authority, requires the CIM and all related standards to go through a formal standards process before publication. This process contains a number of steps:

Step 1: New Work Item Proposal

A new work item proposal is the first step in creating a new standard. The NWIP is submitted by a working group and is voted on by all member countries. A minimum of 5 countries must provide experts to work on the proposal and a majority must vote in favor for it to move forward in the standards process.

Step 2: Working Draft and Committee Draft

The working group then prepares a working draft of the standard, which may take months or even years to prepare. This is then submitted as a Committee Draft (CD) that is circulated to all national committee for comment. These comments are then compiled and sent to the working group to be addressed.

Step 3: Committee Draft for Vote

After addressing the comments received from national committees about the CD the working group then prepares an updated version of the standard that is issued as a Committee Draft for Vote (CDV). This is circulated to member countries for a five-month voting period and is considered approved if two thirds of the votes cast are in favor and the number of negative votes does not exceed 25% of the votes cast. At this stage countries may still submit comments along with their vote.

Step 4: Final Draft International Standard

The working group, once again, addresses any comments that have been received and prepares a Final Draft International Standard (FDIS). This is submitted to the IEC Central Office and circulated to the national committees for a two-month voting period. At this stage a country may only make an explicit vote: positive, negative or abstention.

The FDIS is approved if two thirds of the votes cast are in favor and the number of negative votes cast does not exceed 25% of the votes cast. If approved, the document is published however if the conditions are not met it is referred back to the working group to be revised. Final publication is the responsibility of the IEC Central Office and leads to the publication of an international standard. This normally takes place within two months of the approval of the FDIS.

Standards Cycles Can Take Years of Time

The standards process can take a significant period of time:

- Committee Draft for vote: 5 months
- Final Draft International Standard vote: 2 months
- Addressing comments: up to 4 months
- Publication: up to 2 months

In addition, there is the time to prepare the NWIP, WD and CD with additional time in between all of these stages for experts to work on the drafts. As such the process can take several years from inception to the publication of the standard.

With such a long cycle time, some users choose to work with draft documents before final approval issued.; however, there is a risk that drafts may change before final publication. Any projects using draft documents should take this into account and plan accordingly.

Case Study

Jeff Kimble has mentioned to some of his colleagues that he is interested in CIM and learning how Electric Innovation Utility could leverage the model for their operations and integration. Unfortunately, the responses from his colleagues have run the gamut from a lack of awareness, to several negative impressions, such as:

- “It takes too long for the CIM to be updated.”
- “Nobody uses that.”
- “Don’t we have our own common model?”
- “The standard is always changing, how could we possibly deal with that?”

What would explain some of these perceptions and questions that Jeff’ colleagues have?

It does take a long time to update any given standards document within the IEC. The process is, by design, structured to give all participating countries an opportunity to comment on the content. While this makes the process take longer, the resulting standards have broader, international support, and are typically more robust.

A response that reflects that the speaker may not be aware of what standard the applications they use behind the scenes for its data model or integration.

Often, especially for larger utilities, they may have their own data dictionary. But often the common “vocabulary” is not used across the organization but may only be in a few silos, that is, data administration, or perhaps the engineering analysis teams. The use of the CIM not only facilitates cross-domain communication and semantic understanding, it also supports broader communication with other vendors and utilities.

It is interesting how often in the user group community desperate messages are heard; these may include “it takes too long” or “it changes too frequently”. CIM-based integration can support the change management capabilities of any vendor’s interface, using service level and XSD versioning, as well as supporting backwards compatibility.

Section 3 Questions

1. The Standards Development Organization responsible for CIM maintenance is:
    a. Organization for the Advancement of Structured Information Standards (OASIS)
    b. International Electrotechnical Committee (IEC)
    c. National Institute of Standards and Technology (NIST)
    d. North American Electric Standards Board (NAESB)

2. IEC 61970, maintained by TC57 WG13 is often referred to as the “core” because:
    a. Due to the acronym “Center for Object Related Extensions”
    b. Because the standard was approved with two-thirds vote
    c. Because it is for deregulated energy markets
    d. Because it defines the components of the power system
3. The time to go from New Work Item Proposal (NWIP) stage to a Final Draft International Standard (FDIS) is typically:
    a. More than a year
    b. Six months
    c. Five months
    d. Up to two months
4. The IEC 61968 extension to the original CIM added:
    a. Transmission network descriptions
    b. Energy market descriptions
    c. Unbalanced, low and medium distribution networks
    d. Unified Modeling Language
5. How many countries must vote to approve a New Work Item Proposal (NWIP)
    a. 5
    b. 10
    c. 4
    d. One from North America and one from Europe

4. INTRODUCING UML

Learning Objectives

- Understand the basics of UML class modeling
- Understand the basics of XML syntax
- Understand the basics of RDF syntax

UML Makes Power System Modeling Possible

The Unified Modeling Language (UML)[4] is a modeling and specification language that is used for modeling a wide variety of components within the software development lifecycle including data structures, system interactions, and use cases. The modeling is not tied to one particular implementation technology and can be realized on multiple platforms.

To understand the CIM, the user must understand UML class diagrams and the entities within it. A full description of UML is out of the scope of this publication and the reader is encouraged to read the resources available on the OMG UML website[5] if they wish to find out about the other aspects of UML.

Classes

Within a system, a class represents a specific type of object being modeled. A class hierarchy is an abstract model of a system defining every type of component within a system as a separate class.

In the CIM, for example, physical objects like transformers are modeled as classes, as well as more abstract concepts like “customer”.

Each class can have its own internal attributes and relationships with other classes. Each class can be instantiated into any number of separate instances, known as objects in the object-oriented programming paradigm, each containing the same number and type of attributes and relationships, but with their own internal values.

As a simple example the class *Circle* is used to describe the characteristics of a circle that is to be plotted on a diagram. Attributes of the circle that would need to be known if the circle is to be plotted and, assuming the diagram is a simple, two-dimensional drawing, requires an X and Y coordinate that represents the center of the circle[6]. The radius of the circle would also be

[4] For the most recent UML specifications, see https://www.omg.org/spec/UML/

[5] https://www.uml.org/

[6] The coordinate could instead represent the upper-left corner of the circle, which would be equally valid, but for the purposes of simplicity we will have our example refer to the center of the circle. This would be documented within the UML to make it clear to users what the coordinate represents.

required and so an additional attribute is added to store this value. The diagram may contain multiple circles on the screen at different locations with varying radii, but they can all be described in the same way using this *Circle* class shown in Figure 4-1.

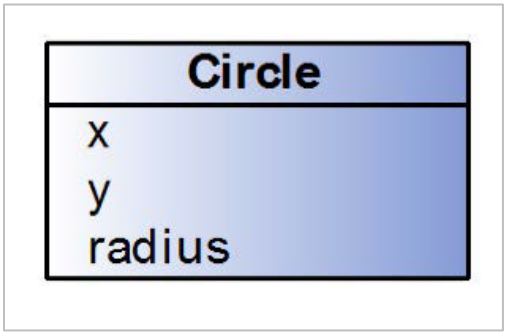


Figure 4-1. The Circle Class

If 100 circles were to be *instantiated* in the diagram the system would create 100 separate instances of circle, each containing an X and Y coordinate and a radius, but all are independent of each other. A change in one circle's radius will not affect any other.

Obviously, a diagram containing only circles is not particularly useful so it would be useful if it could contain rectangles, triangles, squares, lines, and so on. This will require the class structure to become more complex.

Inheritance

Inheritance (also known as Generalization) defines a class as being a sub-class of another class. As a sub-class, it inherits all the attributes of its parent, but can also contain its own attributes.

Classes can be **abstract** or **concrete**, depending on whether they are expected to be instantiated. If the class is in the class hierarchy to define an abstract class that represents a common parent for many other classes then it is considered abstract, but if it is something that may be instantiated then it is concrete.

An example of this is that circles, rectangles, triangles, squares, and so on, are all *shapes*. If a *Shape* class is added as a parent class, then a *Circle*, *Rectangle* and *Triangle* class can all be considered to be subclasses of *Shape*. In addition, a *Square* can be considered to be a subclass of *Rectangle*. Since the user will not be creating instances of *Shape* it is considered to be an *abstract* class while its children are all *concrete* classes as the diagram will contain *circles*, *rectangles*, *triangles*, *circles,* and so on.

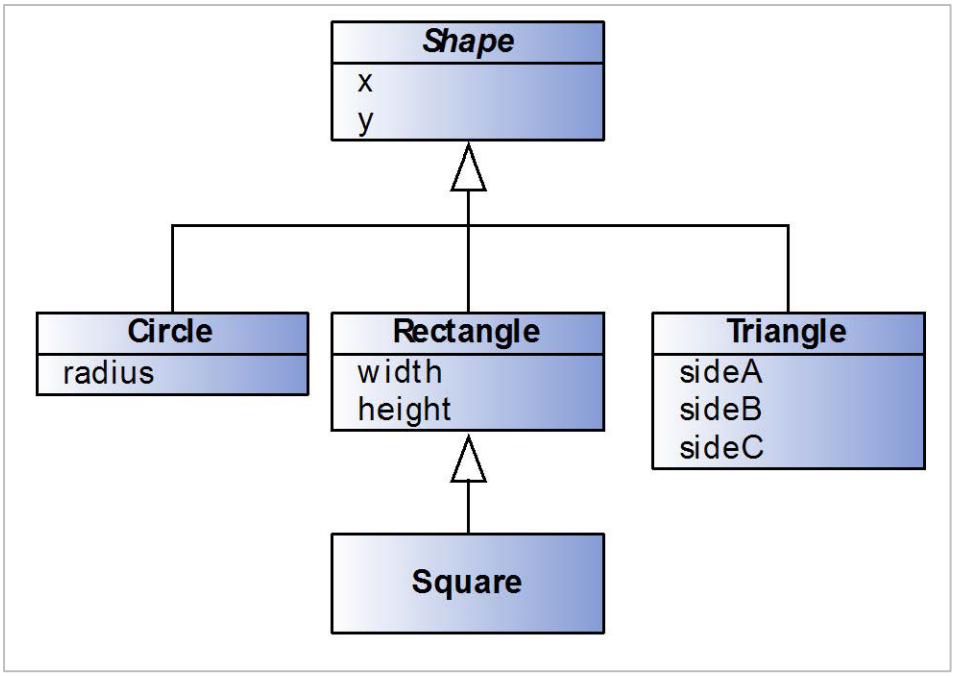


Figure 4-2. Class hierarchy for diagram shapes

In Figure 4-2 illustrates this class hierarchy with the abstract *Shape* class along with its child classes *Circle*, *Rectangle* and *Triangle* and *Square* as a subclass of *Rectangle*. Since every shape will have a location in the diagram the X and Y coordinates are moved from *Circle* to its parent *Shape* class and so is inherited by everything that inherits from Shape. The *radius* attribute remains in *Circle*.

The *Rectangle* class has additional attributes added to represent the *width* and *height* of the rectangle. Square inherits these attributes (along with *x* and *y* from *Shape*) and the implementation would ensure that *width* and *height* would be equal for a square. In the *Triangle* three attributes are added to explicitly define the length of the three sides.[7]

So, a *Square* is a *Rectangle* and a *Shape* but not all *Rectangles* are *Squares* and not all *Shapes* are *Rectangles*. A system would know that any *Shape* will have a coordinate, but it will only expect a *Circle* to have a *radius* and a *Triangle* to have *sideA*, *sideB,* and *sideC*.

Association

In Figure 4-2 the only relationship between any of the classes is that of parent-child. Classes can have other relationships defined that represent linkages between classes showing a relationship other than parent-child. As an example, given that the shapes will be drawn on a diagram, it would be beneficial if each shape could have a *style* associated with it representing

[7] There are multiple ways of defining a triangle but for this example we will simply define the three sides and the documentation would state that one side would represent the base and the angles could then be computed using the length of the other two sides.

the thickness of the outline, the outline color and the internal fill color. These styles may be used by multiple shapes and have a specific name associated with them.

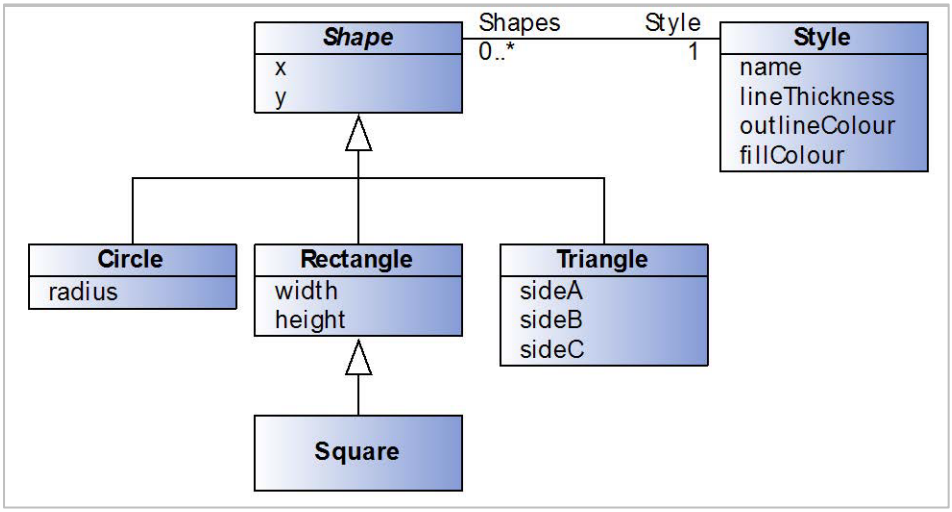


Figure 4-3. Class hierarchy of diagram shapes with a Style

In Figure 4-3 the *Style* class with its three attributes for *lineThickness*, *outlineColour* and *fillColour* and an additional *name* attribute to provide a human-readable name for reference is shown with an association to *Shape*. The association is shown as a line between *Shape* and *Style* and the association has roles on each end. This shows that a *Shape* has a role *Style* on the *Style* class with cardinality of 1. This means that a *Shape* must have an association to an instance of *Style*. On the opposite side, the *Style* class has an association to *Shape* with the role name *Shapes* with cardinality 0.*. This means that a *Style* may associate with 0 or more *Shapes*.

This means that any of the subclasses of *Shape* whether it is *Circle, Rectangle*, *Triangle* or *Square* must have a *Style* and they may share a common *Style* amongst multiple instances.

Aggregation

The Aggregation relationship defines a special kind of association between classes, indicating that one is a container class for the other. For the diagram shapes example, a *Layer* class is added that may contain multiple *Shapes*.

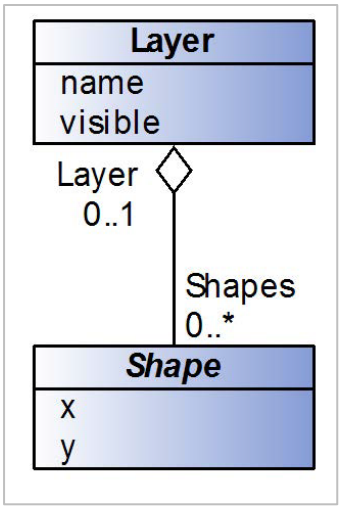


Figure 4-4. Aggregation Example with Layer and Shape

In Figure 4-4 an additional class *Layer* is added that can be used to group together *Shapes* into layers that can be turned on or off within the diagram. The *Layer* class has attributes to provide it a *name* and a flag to indicate whether it *visible*. The relationship to *Shape* is an aggregation relationship, shown with a diamond on the side *container* side of the relationship, with role names and cardinalities used in the same way as a normal association.

As such this diagram indicates that a *Layer* may *contain* 0 or more instances of *Shape* while a *Shape* will be in 0 or more *Layers* (it is assumed that *Layers* are optional). The clear diamond, however, indicates that the two are not completely inter-dependent, and that if the *Layer* was destroyed the *Shapes* would still exist.

Composition

Composition is a specialized form of aggregation where the *contained* object is a fundamental part of the *container* object. This relationship implies that if the *container* is destroyed then all the objects related to it via *composition* are similarly destroyed. In database terms this can be thought of as a *cascading delete*.

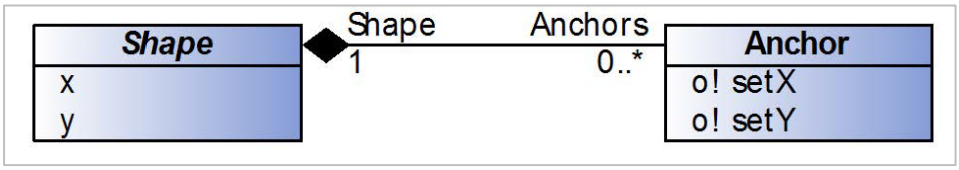


Figure 4-5. Composition Example of Shape and Anchor

In Figure 4-5 an additional class was added to represent an *Anchor* on a *Shape* that can be used show a point on the shape where a line can be *anchored*. The *Anchor's* location is defined as an offset from its parent *Shape* so has an *offsetX* and *offsetY* attributes to store this location. The composition relationship to *Shape* has a role named *Shape* with cardinality 1 indicating that it must have a parent *Shape*. The opposite role is *Anchors* with 0..* showing that a *Shape* may

have 0 or more *Anchors*. The filled diamond on the line shows that it is a composition relationship.

Any system that implements this design will know that if a *Shape* is destroyed then any *Anchor* instances that are contained within that particular instance will also be destroyed.

Summary

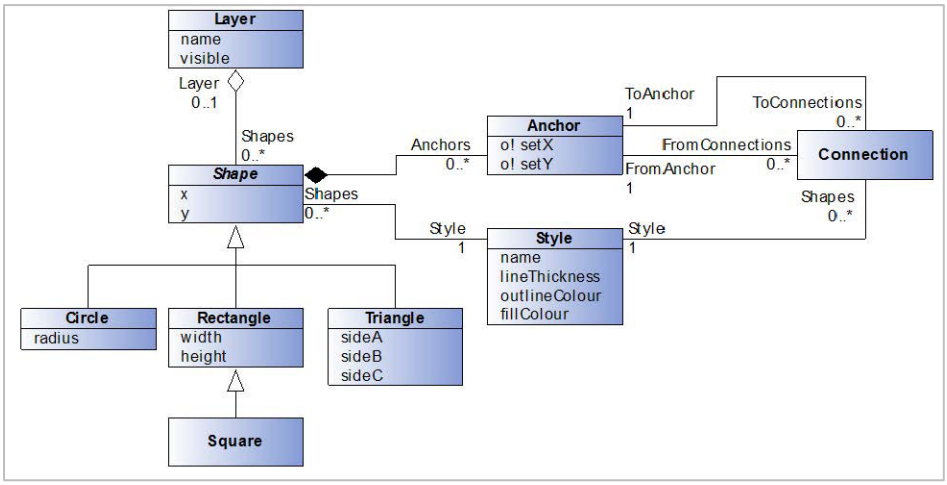


Figure 4-6. Class Diagram for Shapes

Combining the previous examples, a simple class diagram can be created describing how shapes are plotted on a diagram as shown in Figure 4-6.

As well as the previous examples a new class, *Connection* is added to represent the connection between two anchors. This class has two associations to the same class, *Anchor*, with different role names to differentiate between them. This allows an *Anchor* instance to have 0 or more *Connections* be associated with it as either *From* or *To* connections. A *Connection* must also have a *FromAnchor* and *ToAnchor* association from which it can calculate its start and end coordinates using the *Anchor's offsetX* and *offsetY* in combination with the parent *Shape* instance's *x* and *y* coordinates.[8] The *Style* class is reused to allow the *Connection* class to have styling information to define its color and line thickness.

The previous sections should have provided some basic understanding of what a class hierarchy is and how to interpret a class diagram.

[8] This is a simple example for the purposes of illustration so the problem of how to add bend-points to a connection to allow complex routing is deliberately omitted. How this can be achieved can be best seen in the CIM's *Diagram Layout* classes, which will be covered later.

Extensible Markup Language Facilitates CIM-Based Data Exchange

XML Syntax

XML, the eXtensible Markup Language, is a "universal format for structured documents and data"[9] which is quickly becoming the standard for storing machine-readable data in a structured, extensible format that is accessible over the Internet. XML is a meta-language[10] that facilitates the design of markup language to describe the structure of the data.

XML is a subset of SGML, the Standard Generalized Markup Language[11] designed for both on and offline storage and transfer of data. The data is encoded as plain text, thus allowing it to be both human and machine-readable and the use of standard encoding schemes makes it platform independent.

For a detailed understanding of XML syntax the reader should refer to the XML standard documentation however, for the purposes of interpreting and understanding data serialized[12] in XML, this document will try to explain the basic concepts.

XML uses *tags* to denote the elements within the document. Each element is either expressed as an open and close tag containing data of the form:

```
<tag>...Contained Data...</tag>
```

Or with as a single empty entry closed with a slash at the very end:

```
<tag/>
```

An entry may also contain its own attributes, which are expressed in the form:

```
<tag attributeOne="something" attributeTwo="somethingElse"/> or

<tag attributeOne="something" attributeTwo="somethingElse">...</tag>
```

When an element has a start and end tag, any elements contained within these two tags are classed as "children" of the parent element.

[9] W3C Recommendation, "Extensible Markup Language" Version 1.0, October 2000, available at https://www.w3.org/TR/REC-xml.

[10] A meta-language is a language used to describe a language (whether it be another language or itself).

[11] Information processing - Text and office systems - Standard Generalized Markup Language (SGML), ISO 8879.

[12] Serialization is the process of converting a data structure into a format that can be stored and then retrieved later whether it is in a memory buffer, file or transmitted across a network. A file format is an example of a serialization format, where data is written to a file in a set format that can then be read again later, recreating an identical in-memory data structure.

Simple XML Example

As an example, a simple XML tag-syntax to store a book will be created.

The contents and properties of the book can then be expressed as XML, using self-descriptive tags of the form:

```
<book title="Introduction to XML" author="Alan McMorran">
        <revision number="2">
                <year>2006</year>
                <month>January</month>
                <day>1</day>
        </revision>
        <chapter title="Preface">
                <paragraph>Welcome to <italic>this</italic> book…</paragraph>
                <paragraph>…</paragraph>
                …
                </paragraph>…and we shall continue</paragraph>
        </chapter>
        <chapter title="Introduction">
                <paragraph>To understand the uses…</paragraph>
                …
        </chapter>
</book>
```

Here the *book* element contains its own attribute to describe the *title* and *author*, with a child element to describe the *revision* of the book, plus several *chapter* elements. The *chapters* in turn contain elements for each *paragraph*, which themselves contain mixed data of other elements and text. Although to anybody with knowledge of the English language, the names of these tags make their semantics clear, the tag syntax and semantics must still be clearly defined if the data is to be interpreted correctly by an application.

XML Schema

While XML itself has no set tag-syntax or semantics, *schemas* can be defined for expressing almost any kind of data using XML notation. An application interpreting XML data must be given this knowledge of the syntax and semantics used; otherwise it will have trouble interpreting it. This requires the tag-syntax and semantics of the XML to be expressed as a schema, which provides constraints on the structure and contents of an XML document.

The most common formats for describing these schemas are in Document Type Definition (DTD)[13] format or the newer XML Schema Definition (XSD)[14]. The XSD defines:

- The elements and attributes that can appear in a document
- Which elements are child elements
- The number of allowed child elements for each element type
- Whether an element can include text (that is, is an empty element or within an open and close tags)
- The data types for elements and attributes
- Whether their values are fixed
- If there are any default values

Using the previous book example, a simple XSD can be created to describe the elements within the document and the restrictions placed on them. This example schema is shown, along with additional comments, in Figure 4-7.

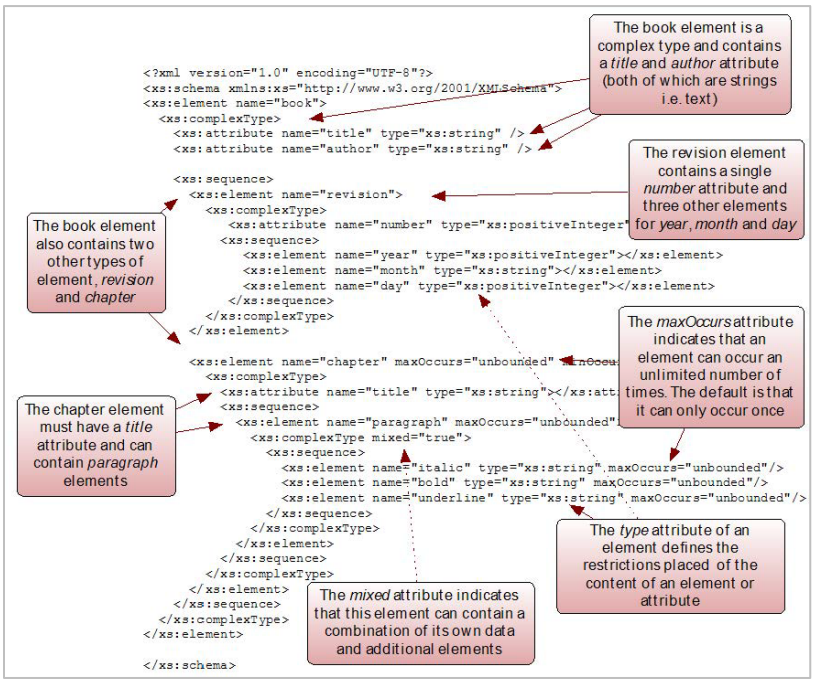


Figure 4-7. Annotated simple XML Schema Example describing the data within a book

[13] W3C Recommendation, "Extensible Markup Language: Prolog and Document Type Declaration" Version 1.0, October 200, available at https://www.w3.org/TR/REC-xml/#sec-prolog-dtd.

[14] W3C Recommendation, "XML Schema Part 0: Primer Second Edition", October 2004, available at https://www.w3.org/TR/xmlschema-0.

The same schema can be viewed in different tools and formats. Figure 4-8 below shows the same schema visualized in the format popular with applications such as XMLSpy[15].

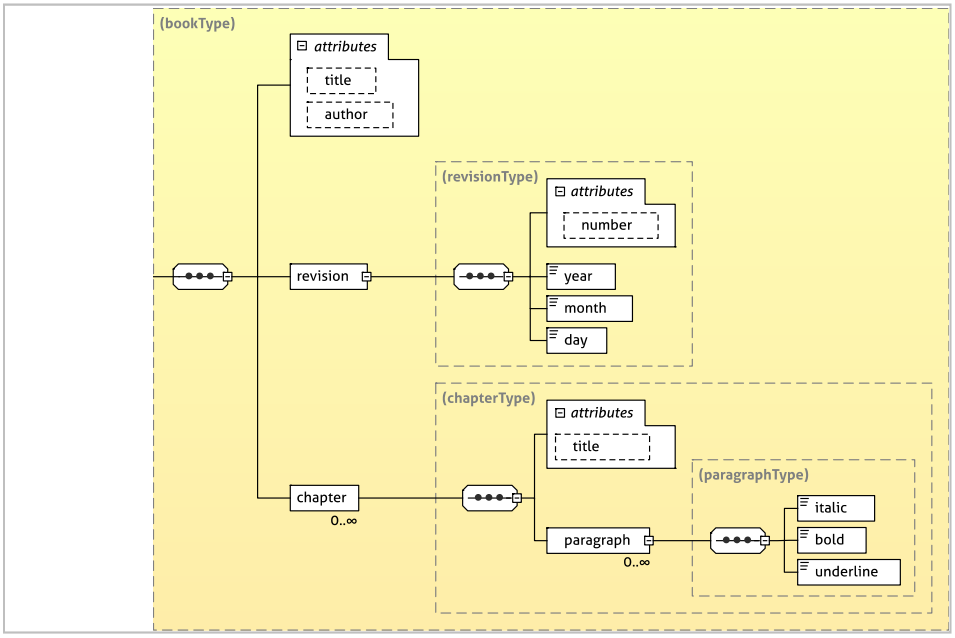


Figure 4-8. XMLSpy Style XML Schema View

The other notable feature of this document is the introduction of *namespaces*. In the example above, every element is prefixed by *xs:*.The document's root node contains an *xmlns:xs="http://www.w3.org/2001/XMLSchema"* attribute which indicates that every element prefixed with *xs* is an XML element that is part of the namespace identified by the Unique Resource Identifier (URI) *http://www.w3.org/2001/XMLSchema*[16]. An XML document can contain elements from multiple namespaces simultaneously, each of which denote a separate XSD with its own set of restrictions. For the previous example, the root node could become:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:ab="http://www.example.com/2011/AB-Schema"
xmlns:yz="http://www.example.com/2010/YZ-Schema">
```

Indicating that the XML document may contain elements from the http://www.example.com/2011/AB-Schema namespace, identified by an *ab* prefix, along with elements from http://www.example.com/2010/YZ-Schema namespace, identified by an *yz* prefix in addition to the original http://www.w3.org/2001/XMLSchema namespace.

This means that an XSD itself has an XSD that describes the elements that are allowed within the schema. The use of namespace makes the format very extensible, enabling the mixing of

[15] XMLSpy is a commercial XML Editor from Altova, https://www.altova.com/xmlspy-xml-editor.

[16] The W3C is the World Wide Web Consortium, the governing body for web standards. Their domain is w3.org and as such W3C standards such as XML Schema and RDF use this domain as part of their unique resource identifier.

data from different definitions in a single document without breaking importers that need only process the data defined by a schema it recognizes.

The XSD can provide an interface *contract* that allows two parties to agree on a well-defined structure for data that is to be exchanged and systems can then validate imported or exported data against the XSD.

Resource Description Framework Supports Further Structure for XML

The Resource Description Framework (RDF)[17] is an XML schema used to provide a framework for data in an XML format by allowing relationships to be defined between XML nodes. As with XML the reader is encouraged to explore the RDF specification for a detailed explanation of the format. This document will try to explain RDF in the context of how it is used with the CIM.

RDF Syntax

With a basic XML document there is no way to denote a link between two elements that are not a parent or a child. For instance, consider a library system containing entries for multiple books with information on their shelf position in the form:

```
<library name="Engineering Library">
 <book title="Power Systems, 1900-1950" author="J.R. McDonald">
  <position section="A" shelf="2"/>
 </book>
 <book title="A Brief History of Time" author="Stephen Hawking">
  <position section="E" shelf="4"/>
 </book>
 <book title="Power Systems, 1950-2000" author="J.R. McDonald">
  <position section="A" shelf="2"/>
 </book>
</library>
```

Each *book* element is contained within the library as an independent entry, but should the user wish to add a link between the *Power Systems, 1900-1950* and *Power Systems, 1950-2000* books to indicate that reader may wish to read the former book before the latter, there is no standard way to do this using the basic XML constructs[18].

Within an RDF document each element can be assigned a unique *ID* attribute under the RDF namespace http://www.w3.org/1999/02/22-rdf-syntax-ns# (which generally uses the *rdf*

[17] W3C Recommendation, "Resource Description Framework", February 1999, available at https://www.w3.org/TR/REC-rdf-syntax.

[18] There are a number of different ways this *could* be done, using customized attributes, XPath references etc. but XML Schema on its own does not define a standard for doing so.

prefix). Adding a *resource* attribute to an element allows references to be made between elements by having its value refer to another element's *ID*.

Simple RDF Example

For the library example above, assigning an *ID* under the RDF namespace to each book allows the addition of *sequel* and *sequelTo* elements. These elements contain only a single resource attribute that point to another element within the document by referencing their *ID*. The containment of books within the library element is replaced with a reference between each book and the library it is in.

To distinguish between the library elements and attributes, themselves governed by an XML Schema, and the RDF elements and attributes, an additional namespace http://www.example.com/libraries/2011/library-schema# is added with the prefix *lib*. An RDF root element is also added with *xmlns* attributes to denote the namespaces and prefixes. The new Library RDF XML representation is shown below:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:lib="http://www.example.com/libraries/2011/library-schema#">
 <lib:library rdf:ID="_lib0001">
  <lib:library.name>Engineering Library </lib:library.name>
 </lib:library>
 <lib:book rdf:ID="_entry0001">
  <lib:book.title>Power Systems, 1900-1950 </lib:book.title>
  <lib:book.author>J.R. McDonald </lib:book.author>
  <lib:book.position>
   <lib:position>
   <lib:position.section>A </lib:position.section>
   <lib:postion.shelf>2 </lib:position.shelf>
   <lib:position>
  </lib:book.position>
  <lib:book.sequel rdf:resource="#_entry0003"/>
  <lib:book.library rdf:resource="_lib0001"/>
 </lib:book>
 <lib:book rdf:ID="_entry0002">
  <lib:book.title>A Brief History of Time </lib:book.title>
  <lib:book.author>Stephen Hawking </lib:book.author>
  <lib:book.position>
   <lib:position>
   <lib:position.section>E </lib:position.section>
   <lib:postion.shelf>4 </lib:position.shelf>
```

```
   <lib:position>
  </lib:book.position>
  <lib:book.library rdf:resource="_lib0001"/>
 </lib:book>
 <lib:book rdf:ID="_entry0003">
  <lib:book.title>Power Systems, 1950-2000 </lib:book.title>
  <lib:book.author>J.R. McDonald </lib:book.author>
  <lib:book.position>
   <lib:position>
   <lib:position.section>A </lib:position.section>
   <lib:postion.shelf>2 </lib:position.shelf>
   <lib:position>
  </lib:book.position>
  <lib:book.sequelTo rdf:resource="#_entry0001"/>
  <lib:book.library rdf:resource="_lib0001"/>
 </lib:book>
</rdf:RDF>
```

As shown, the RDF provides a means of showing relationships between elements without the standard parent-child relationship previously shown with the XML example.

RDF schema contains additional elements that go beyond the simple *ID* and *resource* attribute but in the context of the CIM the concern is with the portions of RDF used within the IEC standards for serializing CIM as RDF XML so as to allow the reader to interpret and understand CIM RDF XML.

RDF Schema

While RDF provides a means of expressing simple statements about the relationship between resources, it does not define the vocabulary of these statements. The RDF Vocabulary Description Language, known as RDF Schema (RDFS)[19] provides the user with a means of describing specific kinds of resources or classes. RDFS does not provide a vocabulary for a specific application's classes like *lib:book.sequel* or *lib:book.sequelTo*, or properties like *lib:book.title* and *lib:book.author*. Instead, RDFS allows the user to describe these classes and properties themselves and indicate when they should be used together. For example, they may state that the property *lib:book.title* will be used in describing a *lib:book*, or that *lib:book.sequel* is an element of *lib:book* and should indicate a reference to another *lib:book* entry.

In essence, RDFS provides a type system for RDF. The RDF Schema type system is similar to that of object-oriented programming languages such as Java, .NET and C++. Amongst other things,

[19] W3C Recommendation, "RDF Vocabulary Description Language 1.0: RDF Schema", Version 1.0, February 2004, available at https://www.w3.org/TR/rdf-schema.

RDF Schema allows resources to be defined as instances of one or more classes and for these classes to be organized in a hierarchy.

For the previous example, the RDF Schema would, amongst others, contain entries to describe the *library* and *book* classes and the properties library, sequel and *sequelTo* in *book*.

```
<rdfs:Class rdf:ID="library>
        <rdfs:label xml:lang="en">library</rdfs:label>
        <rdfs:comment>The library catalogue</rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="_book>
        <rdfs:label xml:lang="en">book</rdfs:label>
         <rdfs:comment>A book contained within a library</rdfs:comment>
</rdfs:Class>

<rdf:Property rdf:ID="_book.library">
        <rdfs:label xml:lang="en">library</rdfs:label>
        <rdfs:comment>The library the book is in</rdfs:comment>
        <rdfs:domain rdf:resource="#_book"/>
        <rdfs:range rdf:resource="#_library"/>
</rdf:Property>

<rdf:Property rdf:ID="_sequel">
        <rdfs:label xml:lang="en">sequel</rdfs:label>
        <rdfs:comment>Indicates that the book has a sequel that is also within the library</rdfs:comment>
        <rdfs:domain rdf:resource="#_book"/>
        <rdfs:range rdf:resource="#_book"/>
</rdf:Property>

<rdf:Property rdf:ID="_sequelTo">
        <rdfs:label xml:lang="en">sequelTo</rdfs:label>
        <rdfs:comment>Indicates that the book is the sequel to another book also within the
library</rdfs:comment>
        <rdfs:domain rdf:resource="#_book"/>
        <rdfs:range rdf:resource="#_book"/>
</rdf:Property>
```

Here, the classes of *book* and *library* are defined, then the three properties *library*, sequel and *sequelTo* are defined. Each of these properties has their domain (the class the property is

within) referencing the *book* class, and for the *library* property the range (the class of element the property refers to) is the *library* class while the *sequel* and *sequelTo* properties have a range of *book*.

Should the library schema be extended so that instead of just having a book element, fictional novels could be differentiated with a separate *novel* element that, when modeled in UML, would be a simple sub-class of the existing *book* class. This can be represented in RDF Schema as:

```
<rdfs:Class rdf:ID="_novel>
        <rdfs:label xml:lang="en">novel</rdfs:label>
        <rdfs:comment>A fictional book</rdfs:comment>
        <rdfs:subClassOf rdf:resource="#_book"/>
</rdfs:Class>
```

The RDF, combined with RDF Schema, provides a mechanism for expressing a basic class hierarchy as an XML schema by specifying the basic relationship between classes and properties. This then allows a set of objects to be expressed as XML using a defined schema that retain their relationships and class hierarchy.

RDF and XML provide a means to express UML, in a syntactic representation that facilitates the use of model to define interfaces, data definition language (DDL) for database design, or for programming business logic.

Case Study

Jeff Kimble, as the manager of the integration effort at Electric Innovation Utility, has already been exposed to the “acronym soup” of syntactic representation. The integration effort EIU uses an enterprise service bus to facilitate integration and service reuse. The utility already has many services that use Java Messaging Services (JMS), web services, as well as more traditional file and database adapters that are used to connect applications. Jeff recognizes that the XML and XSDs are some of the same ways that the vendors uses to expose interfaces and so as EIU can take advantage of the CIM they will be able to leverage their existing investment in services infrastructure.

Questions to Ponder

Does your organization have one or more enterprise service buses?

Does your organization have a governance model for integration?

Does your organization promote the reuse of system interfaces?

Section 4 Questions

1. An XML Schema Definition (XSD), can define:
    a. The data types for elements and attributes
    b. Whether their values are fixed
    c. If there are any default values
    d. All of the above
2. If a class is a sub-class of another class it means that:
    a. The class inherits all the attributes of its parent, but can also contain its own attributes.
    b. The class inherits all the attributes of its parent, but cannot contain its own attributes.
    c. The class has certain attributes that complement the class it is associated with.
    d. The class is an aggregation of other classes.
3. In the examples used in this chapter, *Shape* was what type of class:
    a. Abstract
    b. Sub-class
    c. Generalization
    d. Concrete
4. One advantage of RDF over XSD is that:
    a. The elements have a start and an end tag.
    b. RDF can define relationships between classes and properties beyond parent-child.
    c. RDF provides a means to express UML syntactically.
    d. For the same set of classes and attributes, RDF is much less verbose.
5. If a class is associated with another class it means that:
    a. The class inherits all the attributes of its parent, but can also contain its own attributes.
    b. The class inherits all the attributes of its parent, but cannot contain its own attributes.
    c. The class has certain attributes that complement the class it is associated with.
    d. The class is an aggregation of other classes.

5. UML FOR CIM

Learning Objectives

- Understand how CIM handles unique identities
- Understand CIM inheritance and see examples of class hierarchy
- Understand the basics of converting a circuit diagram into a CIM model
- Understand how the CIM incorporates geospatial information into the model
- Understand the classes of the CIM diagramming layout meta-model
- Understand the overview of CIM packages

Overview

The CIM is an implementation-agnostic model, defining information used by electric utilities in UML as classes along with the relationships between these classes: inheritance, association and aggregation; and the parameters within each class are also defined. This provides the foundation for a generic model to represent information, independent of any particular proprietary data standard or format.

For some readers, the format of the Common Information Model (CIM) may at first appear confusing compared with traditional flat file formats. In Section 4, UML was examined with classes and their relationships explored. This section will cover how the CIM uses UML to describe the components of a power system and how this is then expanded to model data related to the operation and management of the power system.

CIM Class Structure

Identity

The CIM hierarchy currently has no official common super-class (that is, a class from which every component inherits). The majority of CIM classes, however, inherit from the *IdentifiedObject* class so for this section it can be considered the base class for the hierarchy.

The reason for having a super-class is to have common attributes for identification that can be used by any class that needs to be given a universally unique identifier (commonly referred to as a UUID or GUID) known in CIM as a Master Resource Identifier (MRID) and a human readable name.

With the CIM v15 release, a construct was also added to enable any class that inherits from *IdentifiedObject* to have 0 or more additional *Name* entries using a 0..* association between *IdentifiedObject* and *Name*.

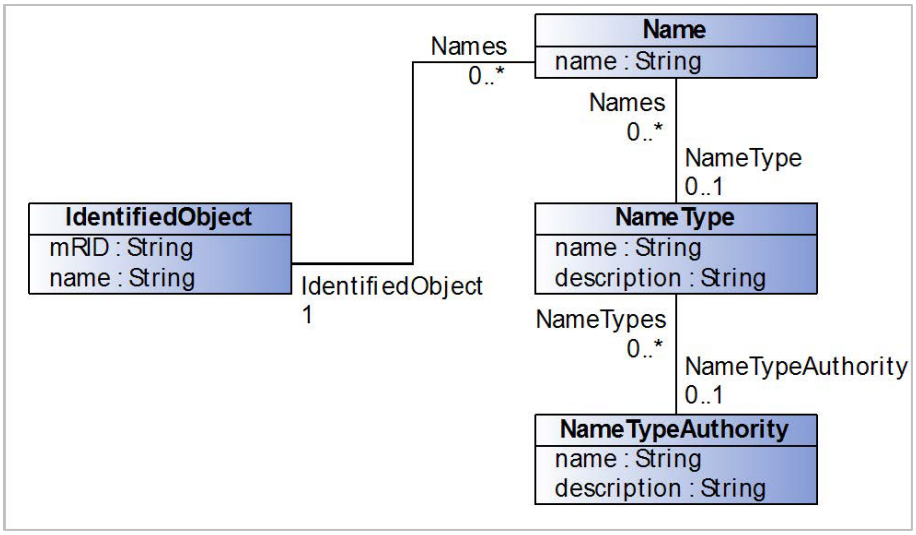


Figure 5-1. IdentifiedObject and the Name Classes

In Figure 5-1 the *IdentifiedObject* class is shown along with the *Name* class. In addition the *NameType* and *NameTypeAuthority* is also included. These latter classes are used to more accurately identify what an additional *Name* entries refers to. For example, a component may exist in multiple different systems with different identities depending on the application. This could be for a variety of reasons including:

- Naming limitations on the source system (that is, maximum of eight characters for a name)
- Historical divergence of data (two systems modeled the same network independently and now must be merged whilst maintaining the identity used on each system)
- To maintain local names from different utilities when network models are merged from neighboring utilities

The *NameType* class is used to identify what *type* of name is being provided. For example the *NameType* is *description* and the corresponding *Name* contains a verbose, human-readable description of the component. It may alternatively be *pathname* to indicate that the *Name* contains a value that represents a path for the component within the overall network hierarchy (that is, *country/region/city/locale/substation/voltage/bay/name*).

The *NameTypeAuthority* is used to optionally describe which authority issued this name. This could refer to an application for a systems integration environment or an organization if the data is coming from multiple organizations. For example a component may have a *Name* entry with a value on *name* that is of a *NameType* for name *localName* and has a *NameTypeAuthority* from *ENTSO-E* to indicate that it is the local name for the component as issued by ENTSO-E. It may simultaneously have another instance of *Name* assigned which also has a *NameType* with name *localName* but a *NameTypeAuthority* of *EDF* indicating that this is Électricité de France's (EdF's) local name for the component.

In versions of the CIM prior to v15 this *Name/NameType/NameTypeAuthority* construct did not exist and instead *IdentifiedObject* had additional attributes for *pathName*, *aliasName*,

localName and *description*. The issue with this approach is that it limits an exchange to having only one *alias* and a number of situations arose where there was a need to exchange more than one alias for a component. The new approach added with CIM v15 allows for *n* additional aliases to be included and adds an additional level of description by making it clear where the aliases came from and their type.

Using Class Structure for Defining Components: Breaker Example

The breaker provides a simple example that can be used to explain why it is advantageous to use a class structure for defining components instead of simply specifying attributes for every different type of component in the CIM as an independent entry.

A Breaker is one of the most common components in a power system described in the CIM as:

> "mechanical switching device capable of making, carrying, and breaking currents under normal circuit conditions and also making, carrying for a specified time, and breaking current under specified abnormal circuit conditions"

To understand how this fits into the CIM class hierarchy the Breaker can be thought of at different levels of abstraction.

At the most detailed level it is a *Breaker,* which itself is a type of switch that can be operated by *Protection Equipment*. At a more general level the breaker's most basic functionality is the ability to be open or closed; therefore, it can be described as a specialized type of *switch*. Within the power system a *switch* is part of the physical network that conducts electricity, and as such can be considered a piece of *conducting equipment*. Since the power system may contain equipment that does not conduct electricity directly, *conducting equipment* can be considered a type of generic *equipment.* A piece of equipment can similarly be considered as a being resource within the power system.

A Breaker can therefore be considered to be a *Power System Resource*; a type of *Equipment*; a type of *Conducting Equipment*; a type of *Switch*; and a type of *Protected Switch*. This corresponds to a class inheritance structure shown in Figure 5-2 below.

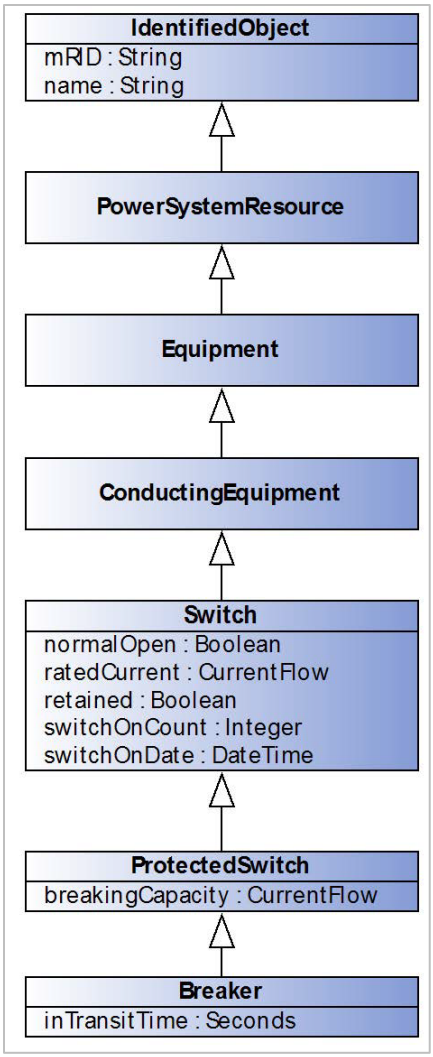


Figure 5-2. Breaker Class Hierarchy in CIM

The *IdentifiedObject* class is the root class for this particular branch of the CIM class hierarchy and other CIM classes in the Breaker hierarchy are:

- *PowerSystemResource*, used to describe any resource within the power system, whether it be a physical piece of equipment such as a *Switch* or an organizational entity such as a *ControlArea*.
- *Equipment*, which refers to any piece of the power system that is a physical device, whether it be electrical or mechanical.
- *ConductingEquipment,* used to define types of Equipment that are designed to carry current or that are conductively connected to the network.
- *Switch,* a generic class for any piece of conducting equipment that operates as a switch in the network and hence has an attribute to define whether the switch is normally open or closed; its rated current; whether it is retained or not when the network is converted to a

bus-branch representation[20]; and attributes to record the number of types the switch has operated and the last date it operated.

- *ProtectedSwitch*, a switch that can be operated by protection equipment and has a breaking capacity
- *Breaker*, a specific sub type of *ProtectedSwitch*, with an additional attribute to define the transit time.

As with the *Shape* examples in Section 2, all subclasses inherit the attributes from their parent class, and as such a Breaker will contain a *normalOpen*, from the *Switch* class, and the *name* attribute, from the *IdentifiedObject* class, as well as its own native attribute.

Subclasses of Switch

As well as Breaker, the CIM standard contains multiple subclasses of *Switch*, including *Jumper*, *Fuse*, *Disconnector*, *LoadBreakSwitch* and *GroundDisconnector*.

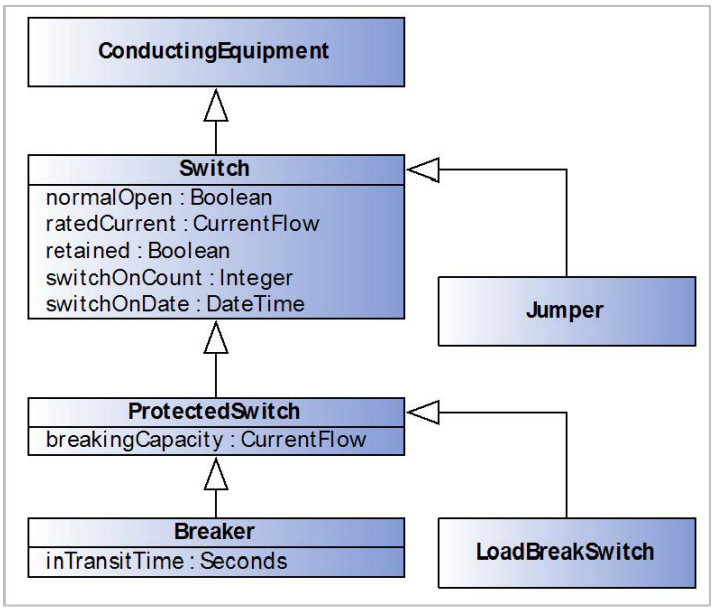


Figure 5-3. Switch Class with expanded subclass hierarchy

Figure 5-3 shows an example of how the *LoadBreakSwitch* class, a subclass of *ProtectedSwitch* fits into the class hierarchy along with *Jumper*, a subclass of *Switch*. Both *Breaker* and *LoadBreakSwitch* inherit from *ProtectedSwitch*, which in turns inherits from *Switch* along with *Jumper*. As such, *Jumper, Breaker* and *LoadBreakSwitch* all contain a *normalOpe*n attribute whilst maintaining their own internal attributes.

[20] For a description of how to convert CIM to bus-branch see the paper "Translating CIM XML Power System Data to a Proprietary Format for System Simulation", A.W. McMorran, G.W. Ault, I.M. Elders, C.E.T. Foote, G.M. Burt, J.R. McDonald, IEEE Transactions on Power Systems, February 2004, Volume 19, Number 1, pp. 229-235.

As well as dealing with them as their native class, the system can treat a *Breaker* or *LoadBreakSwitch* component as being a *ProtectedSwitch*, *Switch*, a piece of *Conducting Equipment*, a piece of *Equipment*, a *Power System Resource* or just an *IdentifiedObject*.

For example:

If a piece of software is performing a topological analysis on a power system network then it will need to know whether a switch is normally open or closed to determine the status of the network and calculate the buses (known as *Topological Nodes* in CIM). The software does not need to know whether the switch is a *Breaker*, a *LoadBreakSwitch* or any other subtype of *Switch* since the attribute it is concerned with, *normalOpen*, exists in all the classes that inherit from *Switch*. As the software traverses the network model, if the component it reaches is of the class Switch or any of its subclasses it extracts the value of *normalOpen* and continues accordingly.

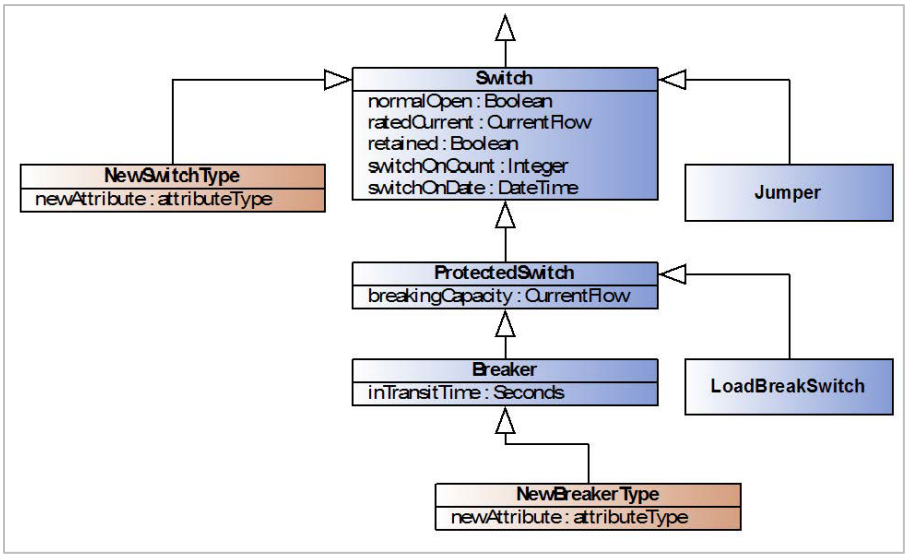


Figure 5-4. Switch class diagram with new types of Switch and Breaker

If a new type of *Switch*, *NewSwitchType* is added to the standard at a later date as shown in Figure 5-4, assuming the original *Switch* class is not modified, then the software will still be able to treat *NewSwitchType,* as if it were a *Switc*h when performing its analysis. Even though the class did not exist when the software was originally written it is looking for any components that are of a class that inherits from Switch.

Similarly, if a new subclass of *Breaker*, *NewBreakerType*, is added (as shown in Figure 5-4), it is still a type of *Switch* (since its parent class, *Breaker* is a subclass of *ProtectedSwitch,* which itself is a subclass of *Switch*) and can be treated as *Switch*, *ProtectedSwitch* or a *Breaker* by the software. In software engineering this is known as *polymorphism.*

As has been shown, this use of an inheritance hierarchy to define components allows classes within the system to be defined as specialized subclasses of a general parent class until the

desired level of detail has been reached, from the generic *PowerSystemResource* right down to the *Breaker* or *LoadBreakSwitch* class.

This use of a class hierarchy also allows extensions to be made to the standard by extending the existing classes instead of introducing completely new, independent entries. This approach, as shown, can allow existing software applications to interpret the new data, albeit at a higher level of abstraction, without necessarily requiring extensive modification.

Defining Component Interconnections

When defining how components within a power system network join together, rather than define direct connection between components, the CIM uses Terminals and Connectivity Nodes.

To understand why this approach is taken consider the very simple, circuit shown in Figure 5-5 below.

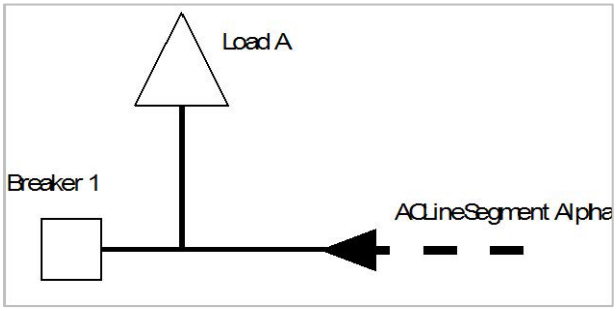


Figure 5-5. Connectivity Example Circuit

This circuit, containing a *Breaker*, *Load,* and *Line,* would require three CIM Objects to represent the pieces of physical conducting equipment: *An Energy Consumer* (to represent the load), a *Breaker,* and an *AC Line Segment* for the line.

The CIM does not model interconnections by associating each component with the other components it connects to, since having *Breaker 1* associate directly to *Load A* and *ACLineSegment Alpha*; *Load A* associate directly to *ACLineSegment Alpha* and *Breaker 1*; and *ACLineSegment Alpha* associate directly to *Breaker 1* and *Load A* would result in the interconnections being defined as shown in Figure 5-6.

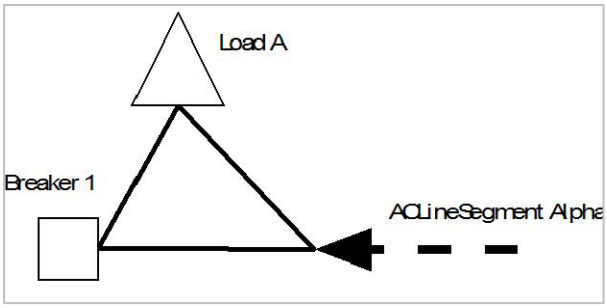


Figure 5-6. Connectivity Example Circuit with Direct Associations

Instead, the CIM uses a *Connectivity Node* to connect equipment, so that should three or more pieces of equipment meet at a T or Star point, the connectivity is accurately represented as shown in Figure 5-7.

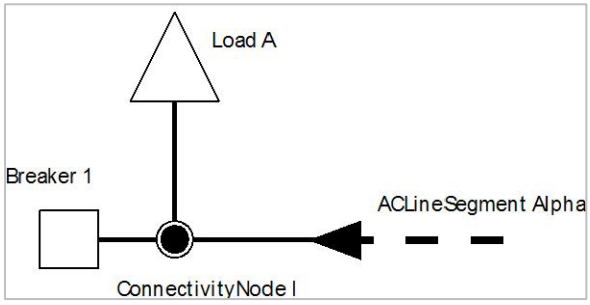


Figure 5-7. Connectivity Example Circuit with Connectivity Node

In CIM, however, pieces of conducting equipment are not directly associated with Connectivity Nodes. A piece of conducting equipment will have one or more Terminals associated with it, and these Terminals in turn are associated with a single Connectivity Node.

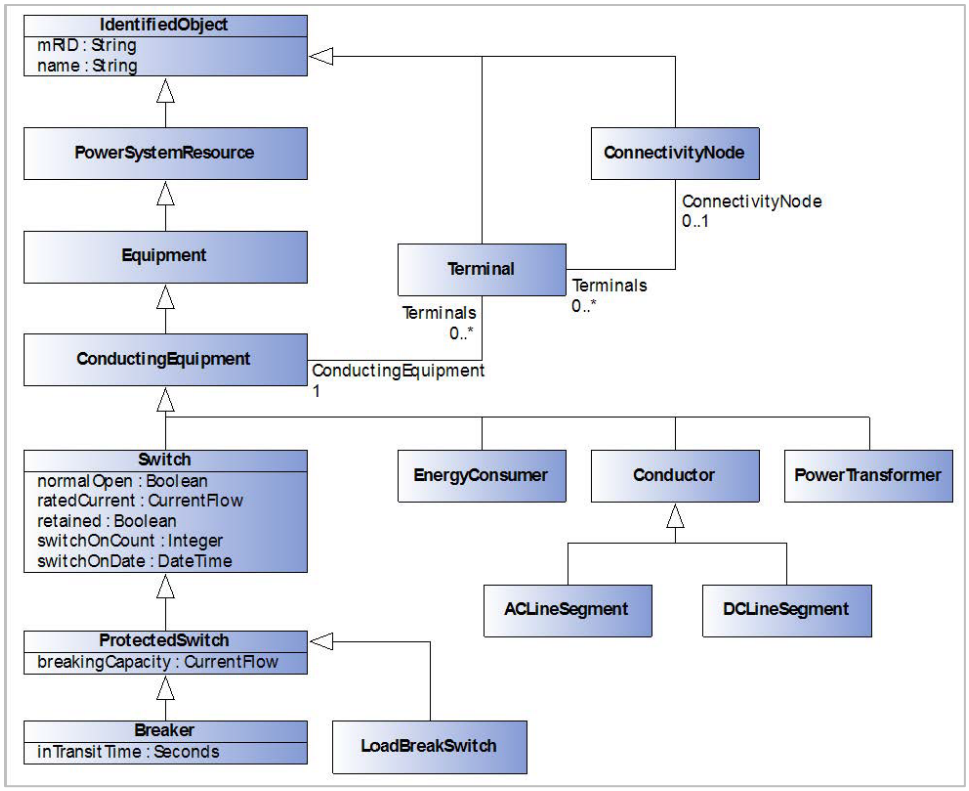


Figure 5-8. Conducting Equipment and Connectivity class diagram

The relationship between the *Terminal*, *ConnectivityNode* and *ConductingEquipment* classes is shown in Figure 5-8. Since only pieces of conducting equipment carry current on the network, the association to the *Terminal* class is from the *ConductingEquipment* class with cardinality 0..* meaning a piece of conducting equipment can have zero or more connections to the network.

The corresponding *Terminal* to *Conducting Equipment* relationship has cardinality 1 since a *Terminal* can only ever be associated with one piece of *Conducting Equipm*ent and cannot exist independently. Since the *Breaker* class (via its *Switch* class parent), *Energy Consumer* and *AC* or *DC Line Segment* (via the *Conductor* class) all inherit from *Conducting Equipment*, they too inherit the association relationship with the *Terminal* class.

The *Terminal* has an association to a single *ConnectivityNode*, and multiple *Terminals* may associate with a single instance of *ConnectivityNode*. By this mechanism electrical connectivity is defined as the *ConnectivityNode* is a zero impedance point of interconnection and all the *Terminals* that associate with a single *ConnectivityNode* are thus interconnected.

The connectivity relationship between the terminals, conducting equipment and connectivity nodes is illustrated in Figure 5-9 below.

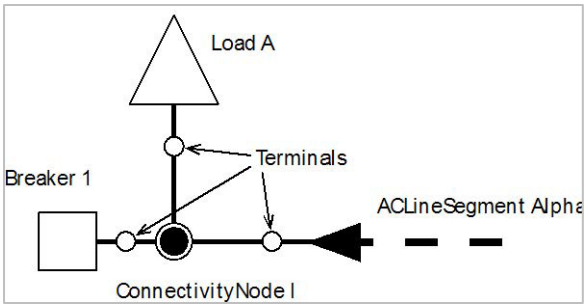


Figure 5-9. Connectivity Example Circuit with Connectivity Node and Terminals

The inclusion of the *Terminals* may initially seem unnecessary, but as well as defining connectivity, *Terminals* are also used for defining points of connectivity-related measurement in the network such as current flows and voltages.

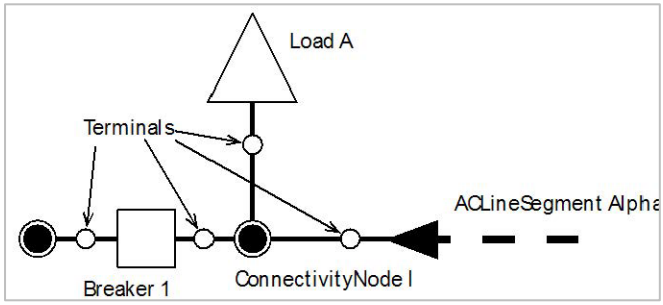


Figure 5-10. Connectivity Example Circuit with Additional Breaker Terminal

The importance of allowing the measurement point to be defined so exactly can be shown in Figure 5-10. In this diagram Breaker 1 has two Terminals associated with it to represent the two distinct network connection points it would have in a real-world power system network. If the Breaker is open then the measurement of voltage for the Breaker will be different at these two points where the Breaker connects to the network. This would result in an ambiguity if measurement were only defined as being on a particular component without specific information about which point of connection the measurement is to be made at.

Translating a Circuit into CIM

The previous sections have described a small section of the class hierarchy for describing CIM components and shown how Terminals and Connectivity Nodes are used to define the interconnection of components within the network. This section will use a more complex example to show how voltage levels, current transformers, power transformers, and generators are modeled by converting a standard line diagram into CIM objects.

Identifying CIM Components

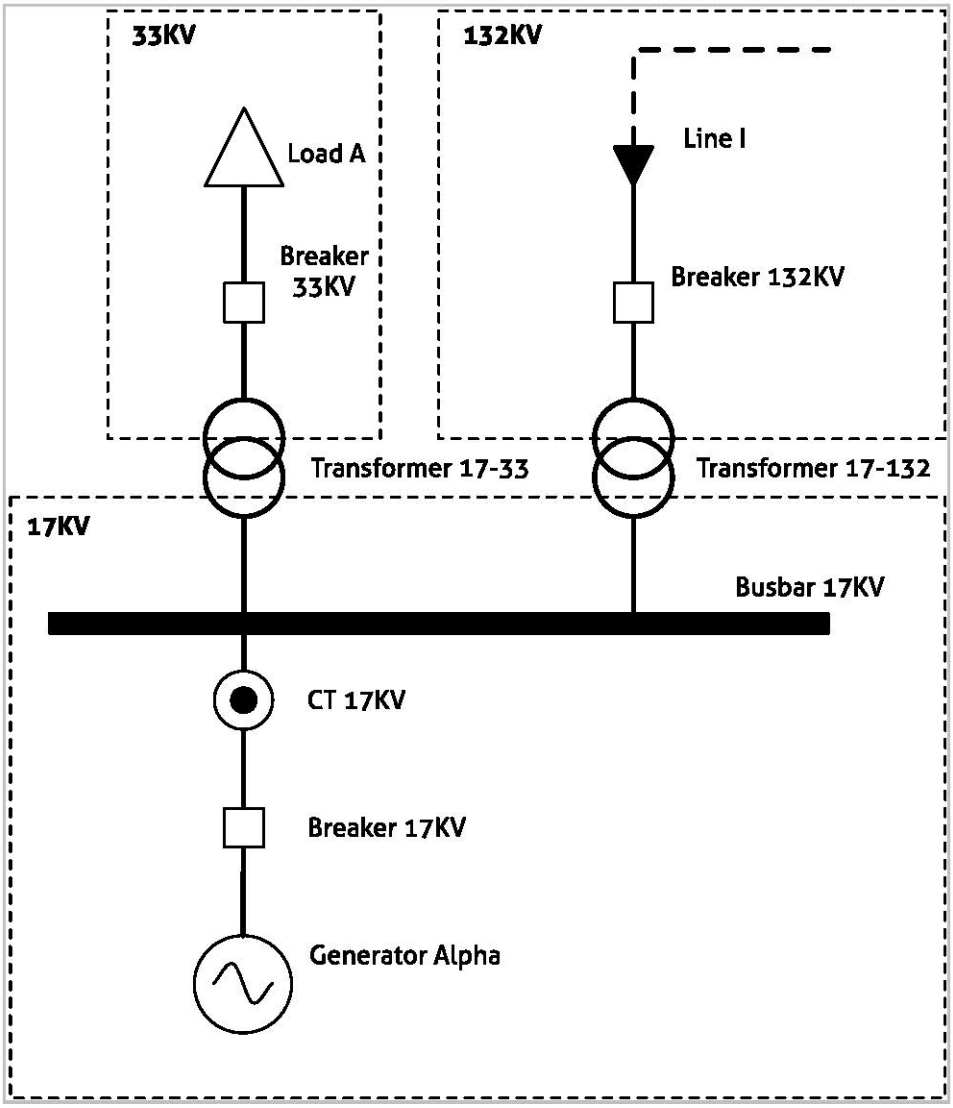


Figure 5-11. Example Circuit as a Single Line Diagram

The circuit shown in Figure 5-11 shows a circuit containing a single generating source, load, line, and busbar. The circuit also contains two power transformers resulting in three distinct voltage levels of 17kV, 33kV and 132kV.

The load, line, and breakers map to the CIM *EnergyConsumer*, *ACLineSegment,* and *Breaker* classes respectively while the busbar similarly maps to the *BusbarSection* class. *Generator Alpha* will map to a single piece of conducting equipment, the *SynchronousMachine*, an "electromechanical device that operates synchronously within the network" as defined in the CIM. When operating as a generator, the *SynchronousMachine* object must have an association with an instance of the *GeneratingUnit* class.

The *GeneratingUnit* class does not represent a piece of conducting equipment that physically connects to the network; instead, it represents "a single or set of synchronous machines for converting mechanical power into alternating-current" as defined in the CIM.

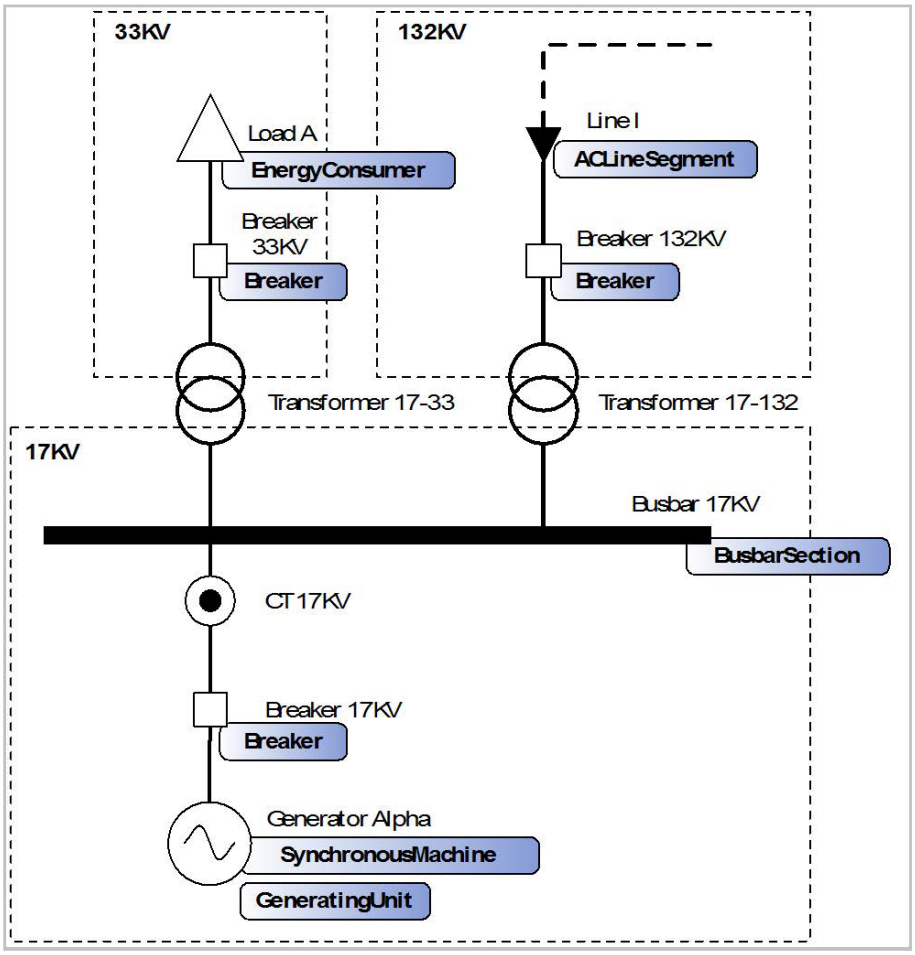


Figure 5-12. Example Circuit with Partial CIM Class Mappings

These mappings are shown in Figure 5-12, leaving only the two power transformers and current transformer to be mapped to CIM classes.

Transformers Prior to CIM v15

Prior to 2011 the CIM Transformer Model had been relatively unchanged since its inception. The model reflected the typical EMS representation of a power transformer which is not mapped to a single CIM class; instead, it is split down into a number of components with a single *PowerTransformer* class. A two-terminal power transformer becomes two *TransformerWinding* objects within a *PowerTransformer* container. If a tap changer is present to control one of the terminals then an instance of the *TapChanger* class is associated with the *TransformerWinding* object of that particular terminal while still being contained within the

PowerTransformer instance. The UML class diagram for the classes that form a transformer is shown in Figure 5-13 below.

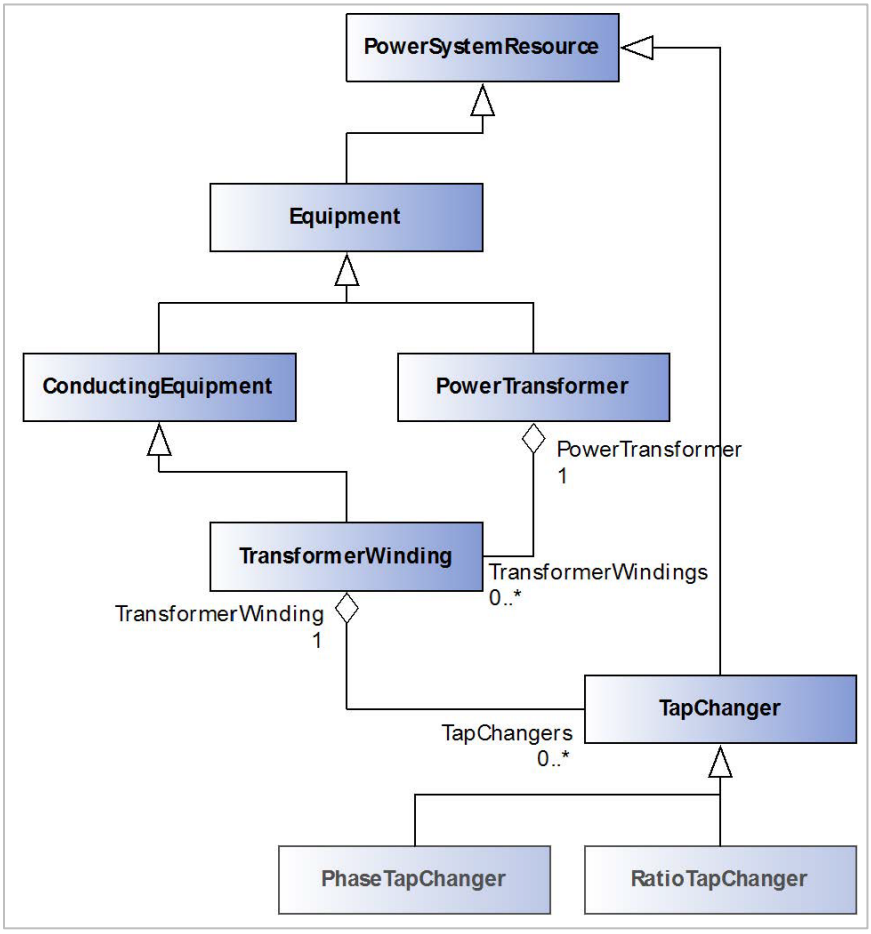


Figure 5-13. Transformer Model (pre-CIM v15)

Although a *PowerTransformer* is still a piece of *Equipment* in the system, in versions of the CIM prior to v15 the class itself does not inherit from *ConductingEquipment* but from its parent, *Equipment*. A *TransformerWinding*, however, does inherit from *ConductingEquipment* and it is through these classes that the *PowerTransformer*'s connectivity is defined. The *TapChanger* is part of the *TransformerWinding* and as such cannot be considered to be a separate piece of equipment in its own right and inherits from *PowerSystemResource*. In some later versions of the CIM, *TapChanger* was itself sub-classed into *PhaseTapChanger* and *RatioTapChanger* to represent the different tap changer types.

The *PowerTransformer* and *TransformerWinding* classes have an aggregation relationship[21], meaning that a *PowerTransformer* is made up on 1 or more *TransformerWindings* which in turn can be made up of zero or more *TapChangers*.

[21] Although it could be argued that this relationship is composition rather than aggregation the CIM class structure contains no composition relationships. This is due to the flexible design of the standard, where a composition

When considering a physical transformer sitting in a substation the *PowerTransformer* container can be thought of as the shell of the transformer. The shell itself does not conduct any of the electricity in the network, but instead holds the windings of the transformer, the insulating material, magnetic core, and all the other components that make up the transformer.

The connections from the transformer to the network are made with the windings themselves, a relationship that is mirrored in the CIM representation where it is the *TransformerWinding* class that inherits from *ConductingEquipment*.

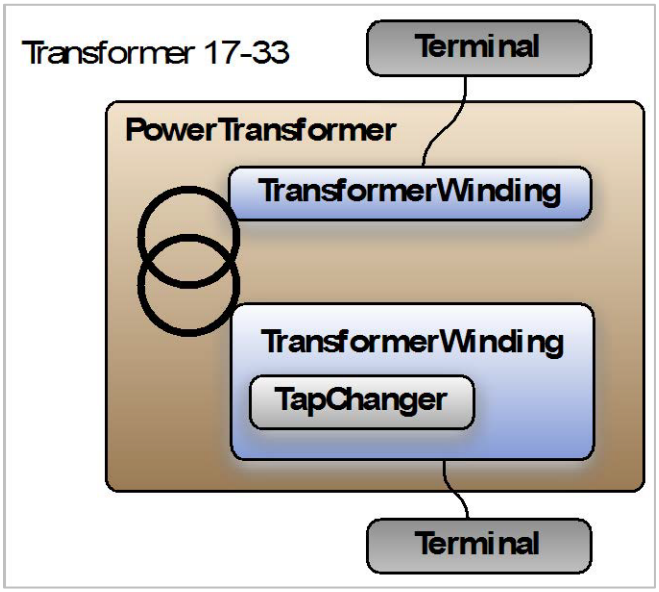


Figure 5-14. CIM Transformer Instance Example

Thus, Transformer 17-33 from Figure 5-11 can be represented as 4 CIM objects: two *TransformerWindings*, one *TapChanger* and one *PowerTransformer* as shown in Figure 5-14 along with the *Terminals* associated with each *TransformerWinding*.

Similarly, a transformer with a tertiary or quaternary winding can be represented as a single *PowerTransformer* containing three or four instances of the *TransformerWinding* class.

relationship would indicate a tighter relationship between classes than is necessary for a number of applications of the standard.

Transformers After CIM v15

As the CIM has moved beyond its original roots in modeling balanced transmission networks into the distribution world there was a desire to create a single, harmonized transformer model that is:

- Suitable for phase balanced or unbalanced networks
- Suitable for representing unbalanced construction commonly used for low voltage distribution networks
- Allows for simultaneous representation of network components as both balanced and unbalanced depending upon the level of detail desired (that is, when performing detailed phase unbalance studies on nominally balanced high voltage networks, or for modeling the transition between balanced and unbalanced representations)

A group of experts worked on an update to the CIM transformer model to reflect these requirements, which was then integrated into the draft of CIM v15.

Changes to Transformer Model

There are a number of significant changes compared with the previous model that was described in the previous section:

- *PowerTransformer* now inherits from *ConductingEquipment* rather than *Equipment* so the *Terminals* are now associated with the *PowerTransformer* rather than the windings as with the previous representation.
- *TransformerWinding* now renamed to *TransformerEnd*. This is because the term *winding* was felt to imply that the model represented the internal configuration of the transformer, but in reality the *TransformerWinding* was really describing the transformer's terminals. Since the term *Terminal* is already used in the CIM *TransformerTerminal* would have been confusing and ambiguous so the term *TransformerEnd* was used instead.
- Additional classes were added to model transformer *tanks*, typically used in distribution systems where a single transformer contains multiples tanks each with its own *Transformer*Tank*Ends*.
- The transformer impedance and admittance attributes as star or mesh explicitly. Previously these attributes appeared on the *TransformerWinding* class which meant that on a two-winding transformer the impedance between the primary and secondary windings would be put on the primary winding with zero values on the secondary winding.
- Impedances and admittances can be defined once and re-used across multiple instances, reflecting the concept of *catalogues* of common values that are used across multiple instances in distribution networks. The ability to explicitly define impedance and admittance values on the *TransformerEnd* is still maintained[22].

[22] The concepts of balanced/unbalanced, catalog/de-normalized, and impedance representation as star or mesh form are in theory completely orthogonal in the CIM information model. In practice the balanced and de-

- The *TapChanger* model is expanded to explicitly model not only *ratio* and *phase* tap changers explicitly but also specializations of *PhaseTapChanger* for *symmetrical* and *asymmetrical* models.

Balanced Transmission Transformer Class Model

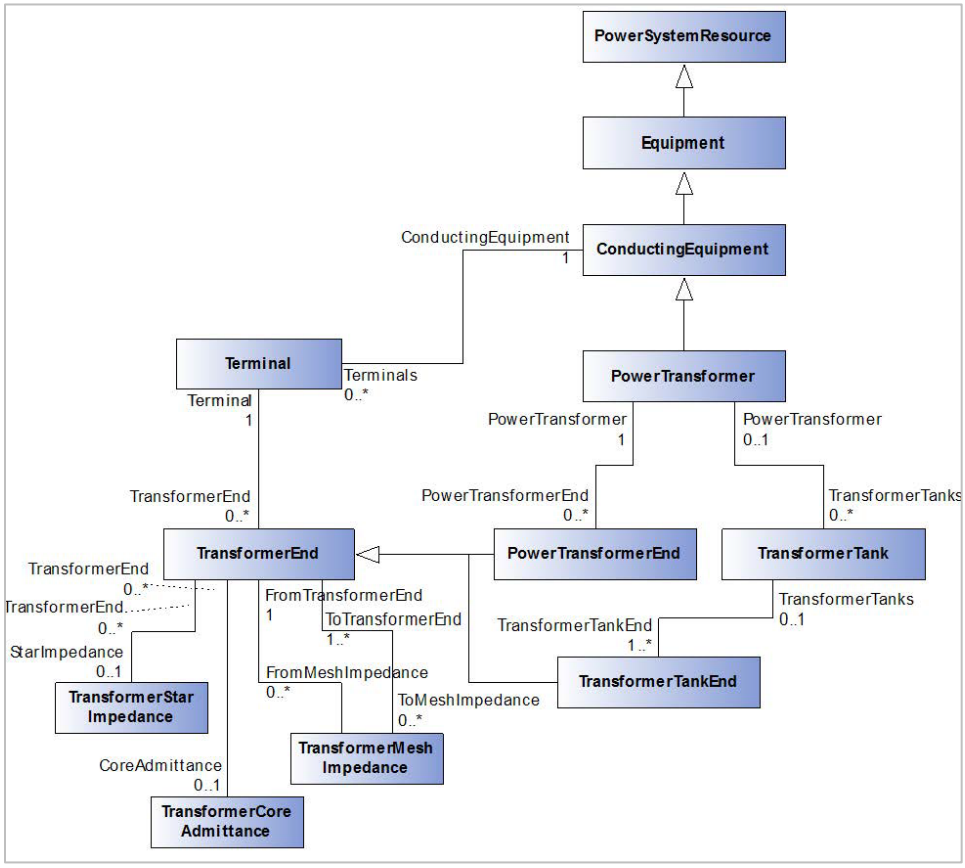


Figure 5-15. Transformer Class Diagram CIM15+

The complete class diagram for the new transformer model is shown in Figure 5-15 and even with the *Tap Changer* model omitted it is significantly more complex than Figure 5-13. This does not mean that representing the same transformer in CIM v14 and CIM v15 requires a corresponding increase in complexity. The diagram reflects the two different potential routes to model transformers depending on whether they are for a balanced transmission system or represent multi-phase transformers (that is, pole-mounted distribution transformers) and when modeling a balanced transmission level power transformer the changes are minor.

normalized star impedance are used together for transmission exchanges and the unbalanced and catalog features are used in distribution model exchanges.

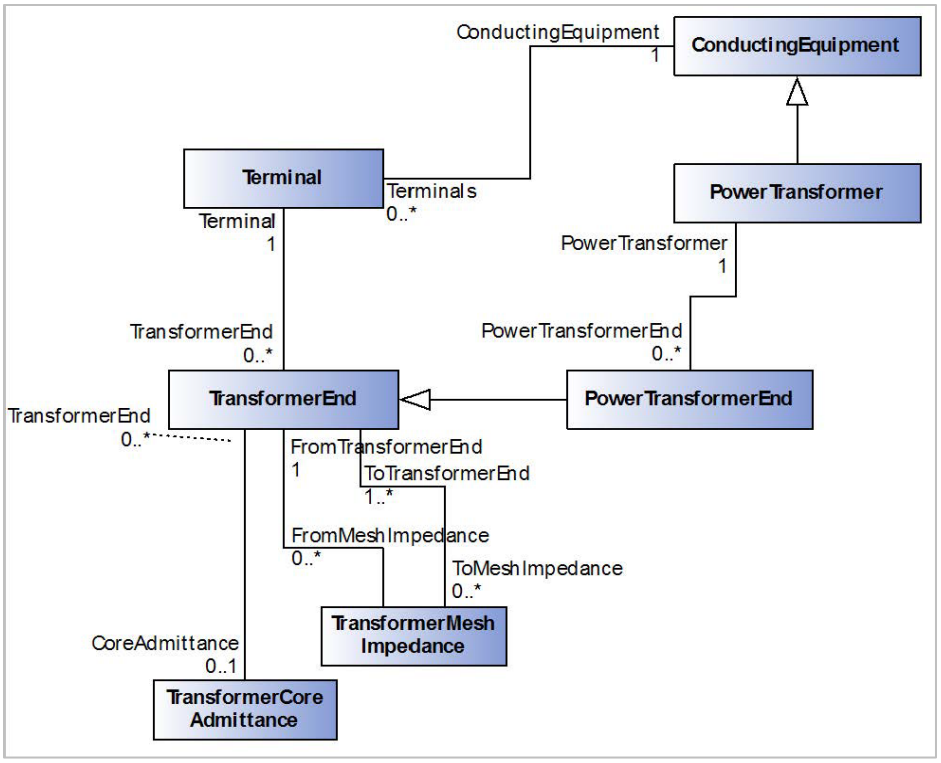


Figure 5-16. Balanced Power Transformer Classes

Figure 5-16 shows the classes required to represent a balanced transmission level power transformer in CIM v15. This includes the new *TransformerMeshImpedance* and *TransformerCoreAdmittance* classes, which could be considered optional as the legacy explicit attributes for impedance and admittance are still included in *TransformerEnd*.

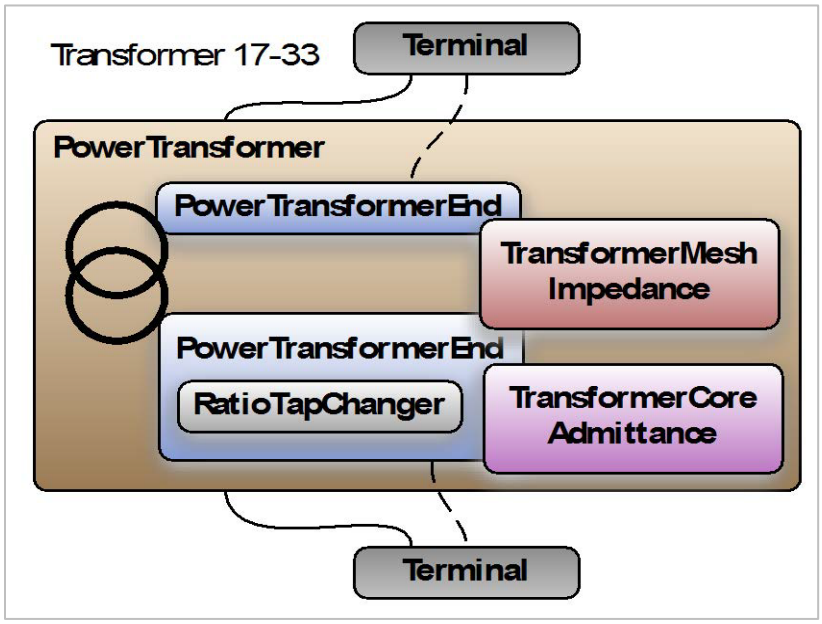


Figure 5-17. CIM v15 Transformer Instance Example

The corresponding instance example for CIM v15 is shown in Figure 5-17 where a *PowerTransformer*, two *PowerTransformerEnds*, a *TransformerMeshImpedance,* and a *TransformerCoreAdmittance* are modeled. In this diagram a *RatioTapChanger* has also been included, although this could also be a *PhaseTapChanger*, in CIM v15 they are explicitly defined as separate associations and there are multiple subclasses of *PhaseTapChanger* which are described later in this section.

The *Terminals* in this representation have an association to the *PowerTransformer* since it now inherits from *ConductingEquipment*, but there is also a direct association between *TransformerEnd* and *Terminal* (shown as the dotted line in the diagram).

At first glance the new transformer model appears far more complex than was present before CIM v15. However, the reader should see that representing the same data for a balanced transmission transformer requires only minor refactoring of the data elements and the inclusion of some new classes for more explicitly modeling impedance and admittance. These new classes for impedance and admittance are currently optional as the attributes for representing the impedance and admittance are still present in the new *TransformerEnd* class (however these are not used in the example above as they should be considered legacy or deprecated).

Unbalanced Distribution Transformer Model with Tanks

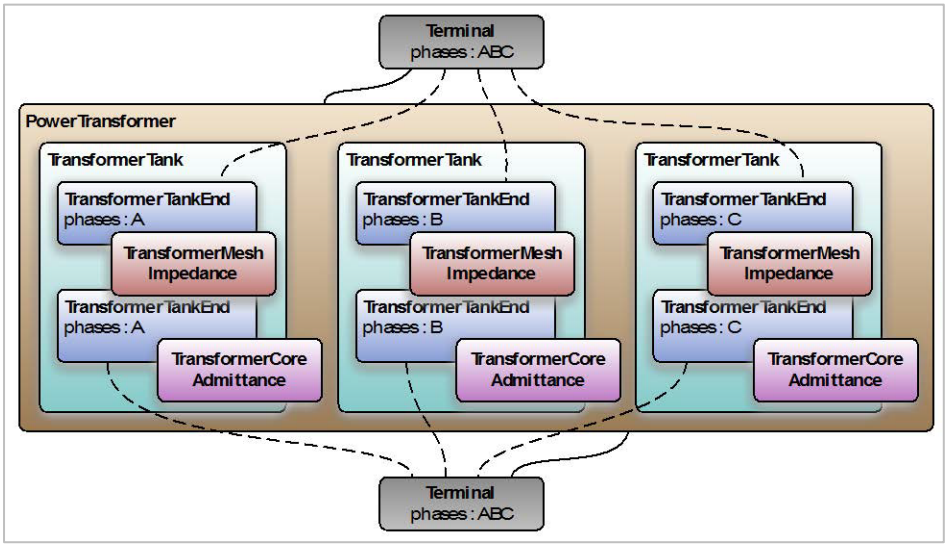


Figure 5-18. CIM v15 Transformer with multiple tanks instance example

When defining an unbalanced distribution transformer, the *PowerTranformer* will contain one or more *tanks*, each with its own phase and two or more *TransformerTankEnds*. An instance example of this is shown in Figure 5-19 with a single *PowerTransformer* with two *Terminals* that is part of a three-phase system, but with three *TransformerTank* instances, each containing two

TransformerTankEnds of a single phase. Both the *Terminals* have a *phase code* of ABC[23] and a *ConductingEquipment* association to the *PowerTransformer*, but each *Terminal* also has three additional associations to the individual *TransformerTanksEnds* on each side of the transformer.

The *TransformerTankEnds* and *PowerTransformerEnds* from the previous example share references to a common *Terminal* instance but specialize the abstract *TranformerEnd* class and thus reuse the same representation of impedance schema. This means that the two models share common representations for the key electrical properties, the major differences being the inclusion of multiple *tanks* within a single *PowerTransformer* instance when modeling the distribution network; and the *TransformerTankEnd* specifies the phases of the tank, which must be a subset of the phases of the associated *Terminal*.

Thus, the *Terminal* represents the external connection of the *PowerTransformer* and the *TransformerTank* and *TransformerTankEnd* represent the internal model of the transformer and how it connects to the external terminals.

Tap Changer Model

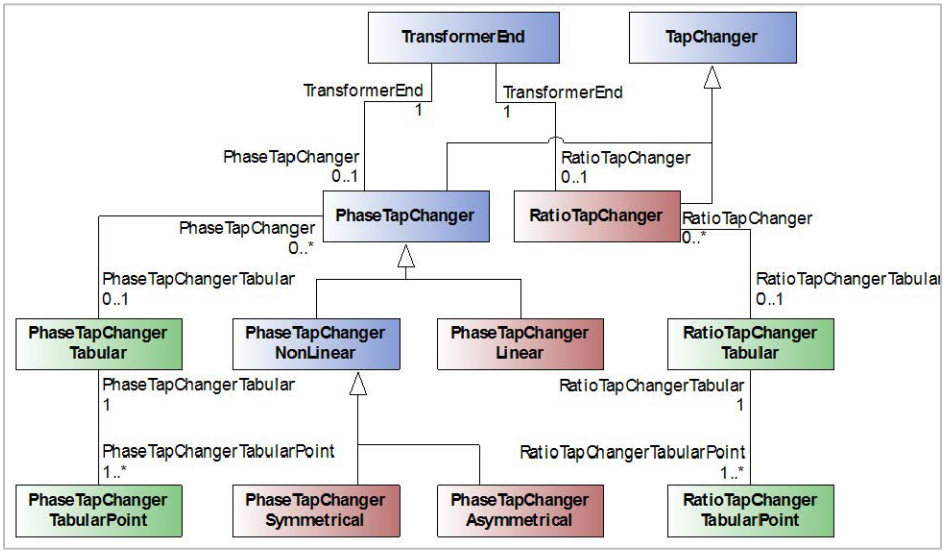


Figure 5-19. Tap Changer Class Diagram CIM v15

The *Tap Changer* model shown in Figure 5-19 at first appears to be very complex but this is primarily due to the class's hierarchy. The *concrete* class are represented in red and the reader can see that the different types of *Tap Changer* remain as *ratio* and *phase* but with the addition of explicit *Phase Tap Changer* specializations for a *linear* tap changer and for the two types of *non-linear* tap changers: *symmetrical* and *asymmetrical*.

In addition, both *phase* and *ratio* tap changers may have tabular entries with multiple points. This allows the change in *phase* and/or *ratio* to be described as an explicit curve rather than as

[23] They could also be ABCN to denote that the neutral is included.

a linear change (or in the case of the *non-linear* tap changers, using standard formulae for tap changer behavior). These classes, shown in green above, allow both *ratio* and *phase* tap changers (including all subclasses) to have a single *<Ratio/Phase>TapChangeTabular* entry and multiple *TapChangerTabularPoint* entries. Each entry contains attributes for the impedance, admittance and step with the *ratio* or *phase angle* depending on the type.

Equipment Containment

As well as having component interconnections defined using the *ConductingEquipment-Terminal-ConnectivityNode* associations, the CIM has an *EquipmentContainer* class that provides a means of grouping pieces of *Equipment* together to represent both electrical and non-electrical containment.

Voltage Levels

Pieces of conducting equipment generally do not have a voltage attribute to define the voltage as a specific value, instead they are associated with a *VoltageLevel*, a subclass of *EquipmentContainer*. Each instance of the *VoltageLevel* class has an association to an instance of *BaseVoltage* that contains a single attribute to define the nominal voltage of that particular group of components. A single *BaseVoltage* instance exists for all of the standard nominal voltages used within the data. As such they may be associated with more than one *VoltageLevel* since standard voltage levels (that is, 33, 132, 275, 400kV) will exist throughout the network. Each *VoltageLevel* instance, however, contains only the interconnected pieces of equipment at the same voltage level. This is an example of using a subclass of *EquipmentContainer* to represent electrical containment.

Substations

The Substation class is a subclass of *EquipmentContainer* that can contain multiple *VoltageLevels* and is used to define a collection of equipment “through which electric energy in bulk is passed for the purposes of switching or modifying its characteristics” in the CIM.

In the example network shown in Figure 5-11 the three different voltage levels identified by the dashed bounding boxes are mapped to three instances of the *VoltageLevel* and contained within a single *Substation* instance. Each *VoltageLevel* object also has an associated *BaseVoltage* object with a nominal voltage of 17, 33 and 132kV.

The *Substation* class, being a subclass of *EquipmentContainer* can also contain other instances of *Equipment*, such as *PowerTransformer*, which as previously explained, was itself a container, not a piece of conducting equipment in CIM prior to version 15. The *Substation* class is an example of a subclass of *EquipmentContainer* to represent non-electrical containment since it will contain pieces of equipment that are physically grouped, but not necessarily electrically connected.

Lines

The *ACLineSegment* is not contained within a *VoltageLevel*. Instead it is contained within an instance of the *Line* class. The Line class in CIM is used to define a "component part of a system extending between adjacent substations or from a substation to an adjacent interconnection point". A *Line* may contain multiple line segments of either the AC or DC variety, but does not itself represent a piece of physical conducting equipment.

Since a line segment is used to represent "a wire or combination of wires … used to carry alternating [or direct] current between points in the power system" so it would be inaccurate to define it as being inside a specific voltage level within a substation. As such, the *AC* and *DCLineSegmen*t classes contain a direct association to the *BaseVoltage* class to define their nominal voltage level.

In transmission models the *Line* class generally contains only *ACLineSegments* and *ConnectivityNodes* however when used in the distribution networks it may contain all equipment considered to be within a feeder such as switches and transformers.

Units and Language

Data Type Units

As an IEC standard the CIM requires that all units must be defined as SI or SI-derived. This means that in the CIM values are not expressed as per-unit which is very common for power system formats. All distance and volume measurements are metric and temperature in Celsius[24], and so on.

The CIM defines data types for all units used and are defined in the UML with a *stereotype* of *CIMDataType*. For numeric units there is also the option of defining a standard multiplier along with the default unit. This means that values that are commonly in the thousands or millions can have a default multiplier of *kilo* or *mega* added thus allowing voltages to be *kilovolts* or power as *megawatts* reflecting the common representations within the industry without breaking the rules requiring SI units.

Enumerations

A number of attributes are defined as having an *enumerated* type. An enumeration is a list of pre-defined values and the value of these attributes must be one of these values to be valid. Examples of enumerations include the *phase code* (A, B, C, AB, AC, ABC, and so on) or the operating mode of a *Synchronous Machine* (generator or condenser).

[24] Although it could be argued of course that Kelvin would be the *purest* definition of temperature.

Language

As an international standard the CIM is written in English, but even though it originated in North America, the IEC require the use of British English. As such all CIM terms are defined in British English terms, resulting in terms that, to American users, may appear misspelled.

CIM Representation

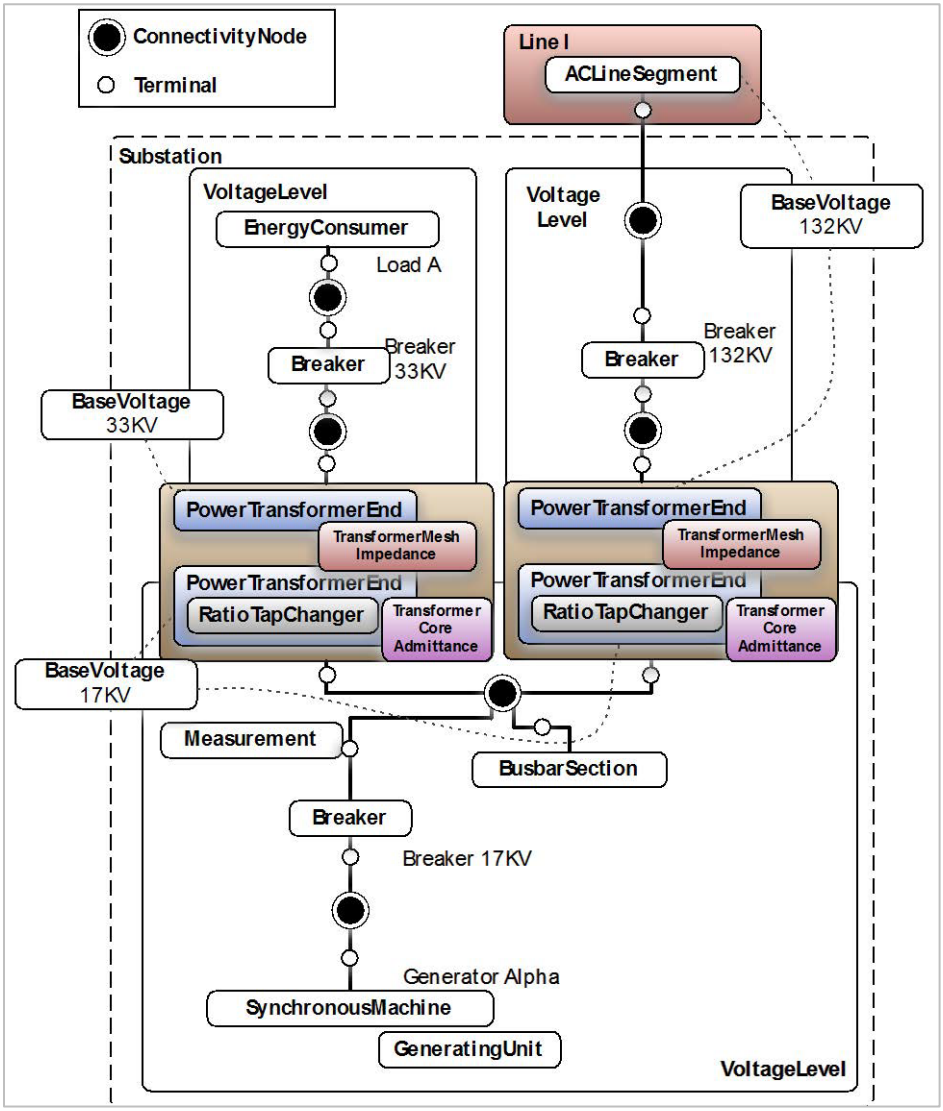


Figure 5-20. Complete CIM Representation

Figure 5-20 shows the CIM representation for the circuit shown in Figure 5-11. The *BusbarSection*'s position may at first seem erroneous, but in the CIM the *ConnectivityNodes* are used to define the point of interconnection for pieces of equipment. As such, the *BusbarSection* object is used primarily to provide a point of association (via its *Terminal*) for *Measurement* objects measuring the voltage at that particular bus-bar in the system. This reflects the

positioning of equipment in the physical system, since a voltage transformer will often measure voltages at the bus-bars within a substation.

The *current transformer* (CT) in the original diagram does not map to a piece of *ConductingEquipment* in the CIM. Instead a *Measurement* instance, representing a SCADA measurement from the CT is associated with the *Terminal* on the *Breaker*. Since the CT does not impact on the electrical characteristics of the network and affect the results of any analysis, it is not explicitly modeled in the same way as a breaker, generator, load, and so on. There may be a need to know about the physical piece of equipment that is in this place in the network, but this is the *Asset view* of the system which will be discussed in the next section.

This representation of the example network could be extended further with the addition of objects to represent control areas, equipment owners, measurement units and generation/load curves, but for now it is enough to understand how an existing network representation can be mapped to CIM objects.

CIM Supports Both Asset and Functional Views

Until this point the focus has been on the CIM's representation of the functional electrical network as seen by an EMS or DMS and applications that analyze the electrical network. This is independent of the physical assets that fulfilling these functions on the network since a power-flow application is concerned with the function a piece of equipment provides; not its make, model and serial number.

As the CIM expanded from its roots in EMS there was a need to include such information, which is of vital importance to systems concerned with Asset Management, Work Management, Procurement, and so on. There is a need to both represent the physical asset and link to the function (or functions) it is performing from an electrical network perspective.

Modeling the Asset View

In the CIM there is separation of asset from role but with a linkage between the two perspectives. The *asset view* is concerned with how a piece of equipment is tracked, monitored, maintained, and so on from the perspective of procurement, maintenance, accounting, and so on. This view of the network includes the manufacturer, model, purchase date, installation date, maintenance history, and so on, but is generally unconcerned with how the asset fits into the electrical network beyond knowing its physical location.

Assets will also exist without being connected to the functional network view of the enterprise. They may be because some assets are not electrical equipment (that is, trucks, buildings, tools, and so on) or because they are not in commission and sitting in a warehouse or service yard. The asset management system must still have a record of these assets and track them from the point of ordering through their entire life within the organization.

The asset may also move its position in the network, performing the same function but at a different location. The asset serial number does not change, but it is now linked to a different

functional location in the network, which itself will have a unique identifier independent of the asset fulfilling the role.

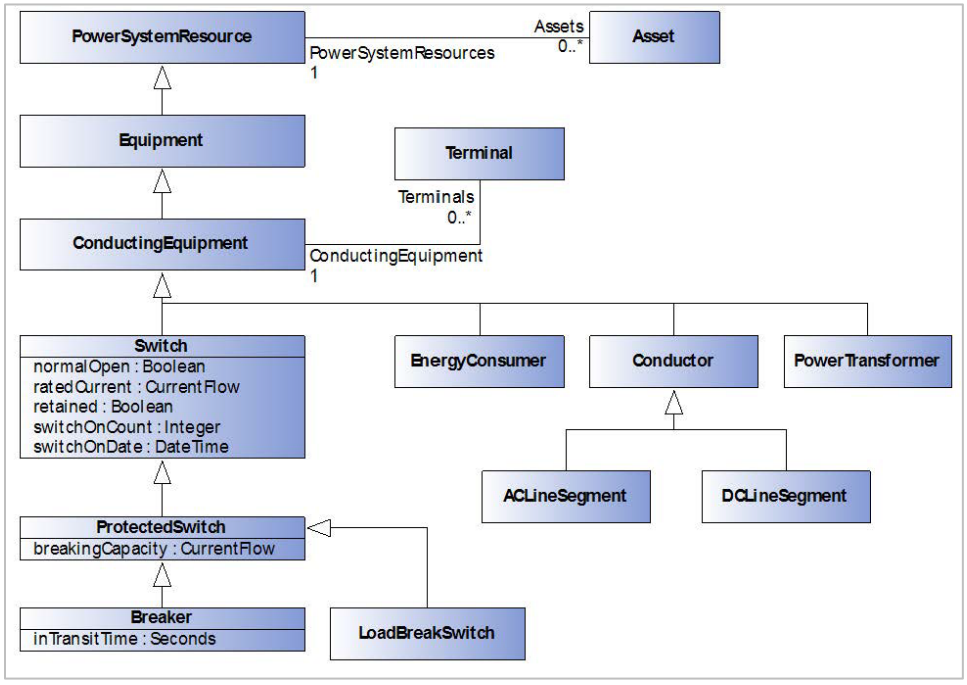


Figure 5-21. Excerpt of Functional Model with Asset Class

The link between the CIM's *Asset* class and the functional network is done via an association between *PowerSystemResource* and *Asset* as shown in Figure 5-21. When this association is fulfilled, the user can navigate from an instance of *PowerSystemResource* such as a *PowerTransformer* or *Breaker* to the *Asset* record that is fulfilling that role.

Benefits of Modeling an Asset View

A major benefit in this approach is that the replacement of a physical asset does not require the functional equipment entry to be altered. For example, take the situation where a breaker is to be replaced in the field. From the point of view of the operators of the network there will be a period of time where that part of the network is de-energized while the work is carried out and then it will be reconnected. From the perspective of the DMS or EMS, nothing fundamental has changed; there is still a breaker at that point in the network.

In some companies there has been cases where there is no distinction between functional and asset views in the system that maintains the network model. When a breaker is replaced the user deletes the previous breaker, inserts a new one, fills in the appropriate data, reconnects the terminals and the update is generated. This has caused problems where a system concerned with only the functional network view has a deletion and insertion of components that are, from a functional view, identical, but breaks linkages to other systems that were tied to the unique identifier of the previous breaker.

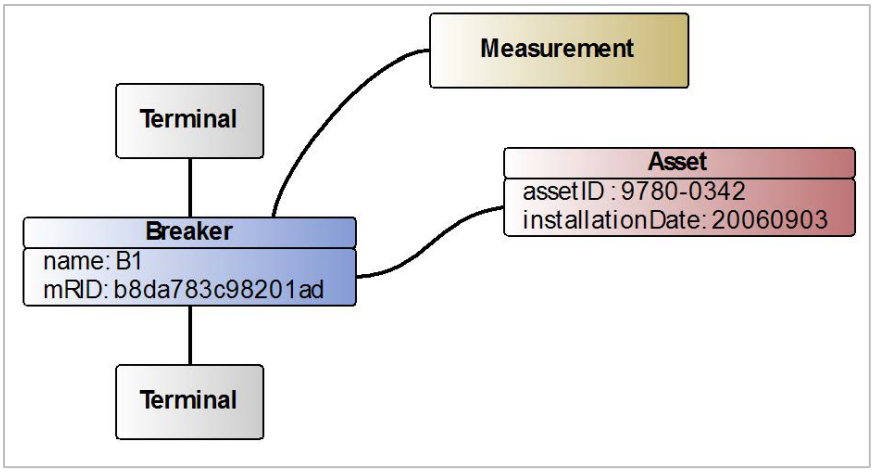


Figure 5-22. Breaker-Asset Instance Example

For example, looking at Figure 5-22 provides an example of a *Breaker* with an association to both of its *Terminals*, a *Measurement,* and the Asset with ID 9780-342 that has been fulfilling this role on the network since 3rd September 2006.

If the asset is to be swapped with a new one the linkages between the *Breakers,* its *Measurement,* and *Terminals* must not be lost. Additionally, its *name* or *mRID* should not be changed as this would also break the linkage. As such the functional *Breaker* instance is not altered, instead the association is changed to the instance of *Asset* that reflects the new piece of physical equipment.

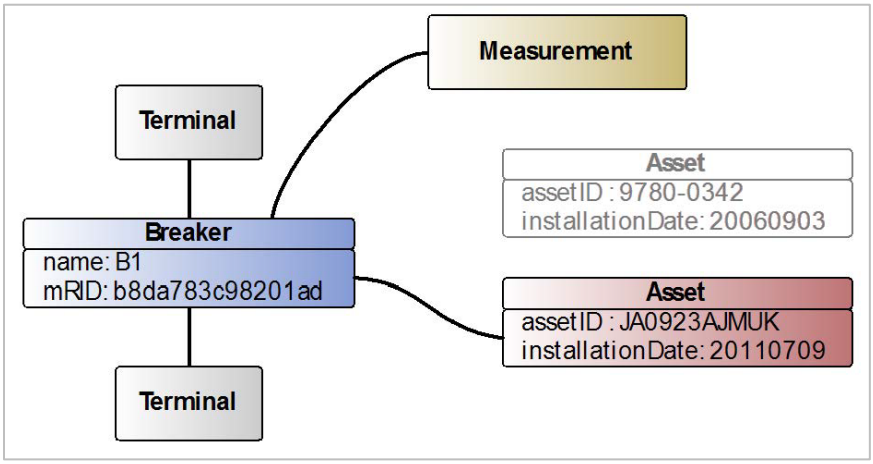


Figure 5-23. Breaker-Asset Change Example

The resulting associations, shown in Figure 5-23 have the *Breaker* instance associating with the new *Asset* instance with ID JA0923AJMUK and the previous association to the old *Asset* is removed. Those systems concerned only with the functional model that were linking to the *Breaker* and using its mRID are unaffected.

Understanding these CIM capabilities helps utilities work with system vendors and systems integration partners to ensure that asset management systems and operations systems can coexist and interoperate well. CIM offers a level of modeling that ensures, if model-driven integration is deployed, that enterprise applications will work well together. All necessary views of an asset are represented, throughout the life cycle of assets and in all operational scenarios.

CIM Supports Network Diagram Layouts

The ability to visualize network schematics can be critical for engineers when interpreting the status of an electrical network. As the number of systems utilizing electrical models grows and utilities move towards a single, common network model across applications, there is a need to ensure single line diagrams and other schematics are reproduced accurately across each system.

The diagram layout portion of the CIM provides a mechanism for describing the layout of electrical schematics. The diagram layout classes allow graphics and network data to be exchanged simultaneously, thus ensuring that schematic views are synchronized with the electrical model across multiple systems. This allows each system to recreate the layout of a schematic accurately while using their own styling for icons, colors, fonts and other graphical rendering attributes.

Separating Style from Layout

A key driver of the diagram layout model was that any receiving systems should be able to draw a schematic in the style that its users are familiar with. A SCADA system may use different icons, coloring and line styles than an EMS, but the operator will want to ensure that the layout of a substation, the position of bus-bars, switches and transformers is the same as in the EMS. With this use-case in mind, the diagram layout model is focused on the exchange of layout information allowing a graphic to be created, rather than the exchange of a graphic on which layout information must be derived.

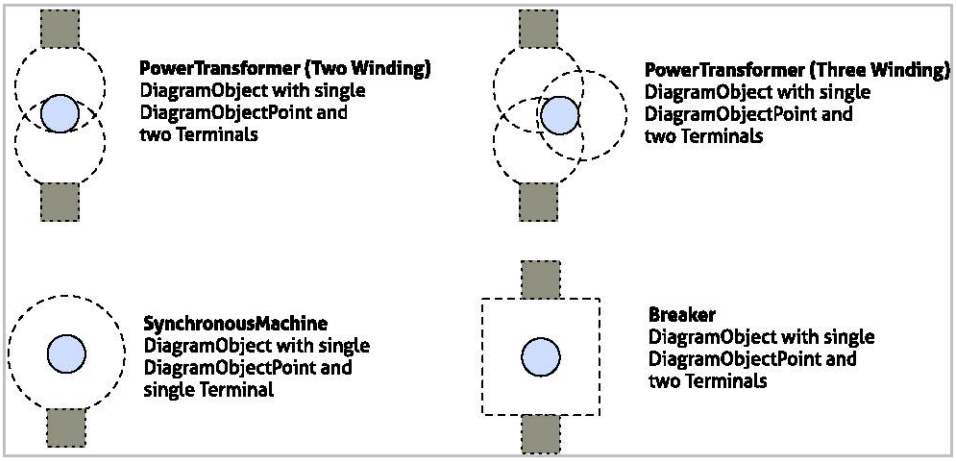


Figure 5-24. CIM Diagram Layout Single Point Examples

As such each element that is to be rendered will have a corresponding Diagram Object, which can have one or more points to describe its location on the schematic. For many elements this is a single point defined as being the center of the object's icon, as shown in Figure 5-24. Here each element itself has only a single point, with another Diagram Object and point used for each Terminal if it must be rendered on the screen.

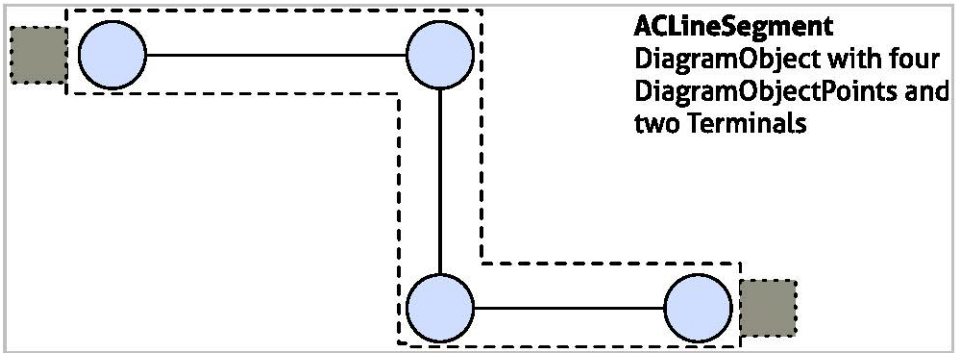


Figure 5-25. CIM Diagram Layout Multiple Point Example

More complex layout information, such as line routing, is accomplished with multiple points for a single *Diagram Object* as shown in Figure 5-25. As with the single point objects, each Terminal may have its own Diagram Object if required.

It is up to the receiving system to determine what icons they wish to use, what thickness a line should be and what color to use to ensure the graphic produced is familiar to the user.

Functional Model Integration

Extending the CIM to support Graphics Exchange allows the diagrams to be tied directly into the electrical model and can be stored and exchanged using existing CIM-compliant applications and formats such as CIM RDF XML which will be explored in Section 7. There are two major benefits to this approach:

1. Synchronization

   By directly tying the schematic data to the electrical model data using the CIM, the exchange of schematic and network models is closely synchronized. This avoids the situation where something is changed in one system without a corresponding change appearing in the other due to there being no direct link between the data in each system.

   By modeling the data in CIM, existing data exchange formats can be leveraged, allowing the network model exchanges to include the schematic data directly. By ensuring a network update always includes the corresponding schematic update, all systems can ensure their displays are kept synchronized with each other and the electrical model.

2. Duplication of Data

   By having a direct link between a schematic Diagram Object and the corresponding electrical model component the amount of schematic data that needs to be exchanged can be drastically reduced. For a receiving system to render a schematic the icons, coloring, line thicknesses and other styling information is based on the electrical properties of the element, so can be derived from the underlying electrical model.

   For example, the schematic data for a *Breaker* need only include a single *Diagram Object* and point since its icon will be dependent on its current position, normal position, voltage, and type, all of which can be derived from the electrical model and the Breaker element the Diagram Object is associated with.

   By deriving information from a single source, the duplication of data is avoided between systems while helping the graphics derived from the schematic to be synchronized with the underlying network model.

Diagram Layout in UML

The diagram layout model is defined in UML and since CIM v15, has been part of the CIM UML. The class model is shown in Figure 5-26 with the previously discussed *DiagramObject* class in the center, and *DiagramObjectPoint* providing one or more spatial points to define an object's location.

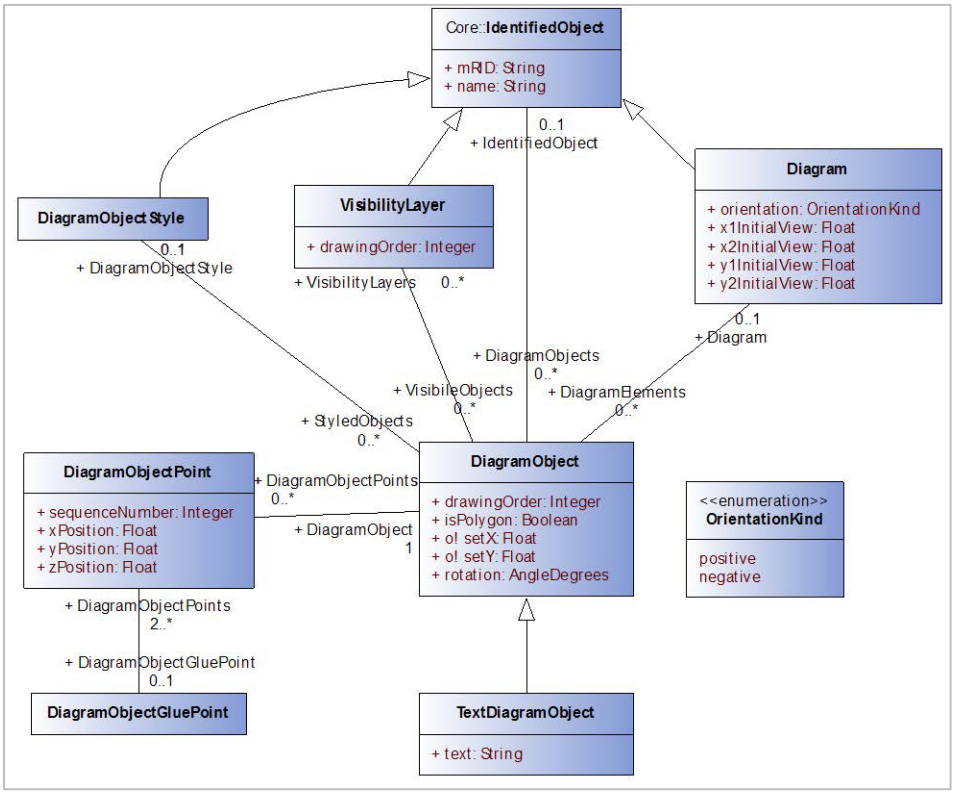


Figure 5-26. CIM Diagram Layout Class Diagram

The other classes allow more complex layouts to be exchanged including layers, differing orientations with ascending or descending vertical axes, and to allow the sender to mark specific points, from multiple elements as glued points where a change in one may affect the others (that is, at points of interconnection).

While the standard is not intended to define a new standard for exchanging styling information when existing standards such as Cascading Style Sheets (CSS) can accomplish this, it does allow the style of a *Diagram Object* to be referenced. This is to allow two or more parties to agree on styling references using unique identifiers should they wish to.

The model allows for a single domain object (anything that is a subclass of the CIM's IdentifiedObject class) to have multiple Diagram Objects. This is to allow the same element to exist on multiple diagrams, but to also allow for a single element to have multiple Diagram Objects within a single diagram (that is, an icon and a text label).

Having the ability to integrate the schematic layout data with the electrical network data will help users ensure their diagrams are synchronized with their network model, while enabling them to utilize existing exchange formats and technologies. The diagram layout model offers a lightweight solution to this problem, providing a common, vendor-neutral format that will allow systems to re-draw a diagram while maintaining its layout.

Embedding Geographical Data

The CIM also contains classes to define the geographical location of functional equipment or assets via the *Location* class that is very similar to that of the CIM Diagram Layout. The classes, shown in Figure 5-27 provide a construct similar to the diagram model, but with the addition of an explicit *CoordinateSystem* class with an attribute to denote the *coordinate reference system*. This uniquely identifies the coordinate system being used (that is, Latitude Longitude with Decimal Degrees, Lambert I-IV, GB Ordnance Survey Grid Codes, and so on).

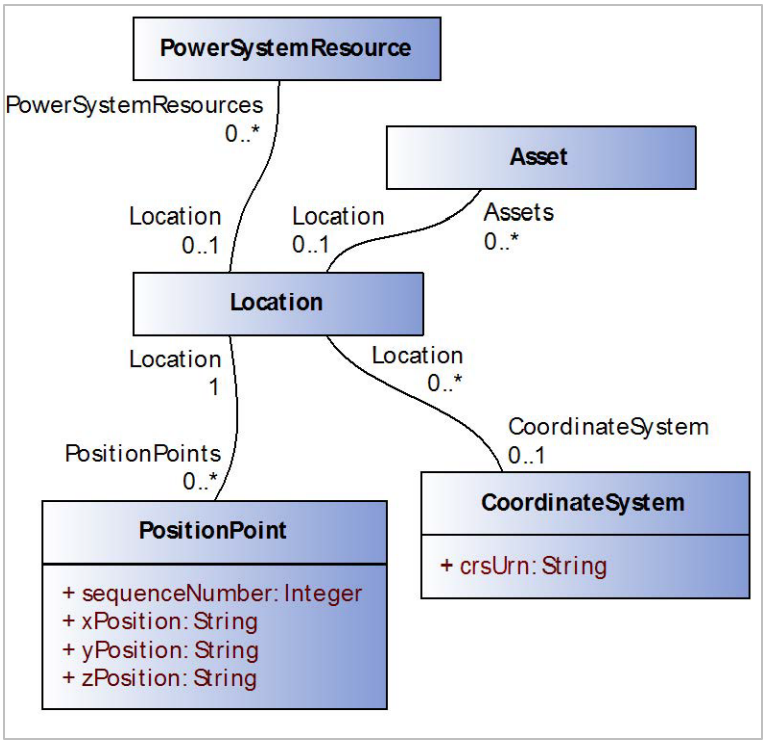


Figure 5-27. CIM Location Class Diagram

Each *Location* can have one or more *PositionPoints*, which contains an X, Y, and optional Z coordinate to define a point in space. The *Location* class has associations to both *Asset* and *PowerSystemResource* and so can be associated with either.

The reason to associate the class with *Asset* is quite obvious since a physical asset is going to have a geographical location, whether it be in a substation or sitting in a warehouse. There are two primary reasons for associating it with *PowerSystemResource* as well:

1. In some exchanges of network data, especially with external organizations there is a need to send the geographical layout of the network, which may be in a simplified equivalent form, but the receiving party is not authorized to view the company's asset data.
2. During the design phase for a network extension the functional network will be built with geographical routing data which will be needed to calculate costs, rights of way, and so on. At this stage there is no asset with which to associate the *Location*.

The CIM Location classes have been successfully tested at both transmission and distribution level and provide a simple mechanism for exchanging the geographical location both from a functional and asset view.

Modeling Unbalanced Multi-Phase Networks

The CIM network model described in this section has focused on balanced, three-phase systems which are exchanged at the transmission level. For low-voltage distribution systems, however

the balanced representation is often insufficient as there are many single phase loads which results in an unbalanced power-flow on three-phase devices.

There were a number of requirements that drove the CIM to be extended to more accurately model unbalanced three-phase (or two-phase) systems.

Measurements

A key requirement for modeling the individual phases is to be able to associate SCADA measurements with the individual phase devices. This allows the individual measurements (that is, current, voltage, reactive power, tap position, switch status, and so on) to each phase.

When there is a single device in the CIM with a single measurement it implies the device is balanced and all three phases have the same value (although in some cases the SCADA system may only measure one phase).

Three Phase Switches

Although normally modeled as a single device in the CIM, a three-phase switch (including subclasses like breaker and fuse) is actually composed of three separate physical switches, one for each phase. In the majority of cases each switch has identical physical properties, but it may be possible that each phase may have different characteristics.

As an example, in the case of a fuse there may be different ratings placed on the fuses for different attributes. There may also be changes in the response time and thus real-time status of switch statues as a result of these physical differences.

For gang operated switches the physical design of the switch enforces the synchronized state changes, but for un-ganged switches there may be situations where the normal and/or real-time operational state for each phase could be different.

In normal circumstances an operator wants to see a single logical switch with all three phases the same, but under certain conditions there would be a need to view each phase independently so they could be operated or monitored more precisely.

For asset information there is a need to tie each phase's device to the physical asset that fulfills the role when that asset has attributes that impact on the operation of the device.

Three Phase Transformers

As was seen in the previous section, the requirements for modeling transformers in unbalanced systems has resulted in a detailed, complex transformer model that supports the modeling of a transformer's internal configuration in three-phase, unbalanced systems.

Loads

There is a need to specify individual single-phase loads and three-phase loads where the individual phases do not have the same characteristics.

Lines

For unbalanced three-phase systems there is a requirement to accurately define the individual conductor parameters for each phase and their spacing. This is to allow unbalanced power-flow engines to create the impedance matrices for the unbalanced lines. In distribution systems it is common to define catalogues of values that are re-used across hundreds and thousands of instances, thus allowing a single *type* of conductor to have its parameters defined once (on a *per length* basis) and then applied to every line that uses that conductor.

In some cases, the conductor types may differ for each phase on a three-phase line so there is a need to be able to identify the phases on the line; associate them with the different conductor types, and define the spacing geometry. This does not require individual *Terminals* and *ConnectivityNodes* as the overall connectivity and topology of the network is common for each phase.

UML for Unbalanced Network Components

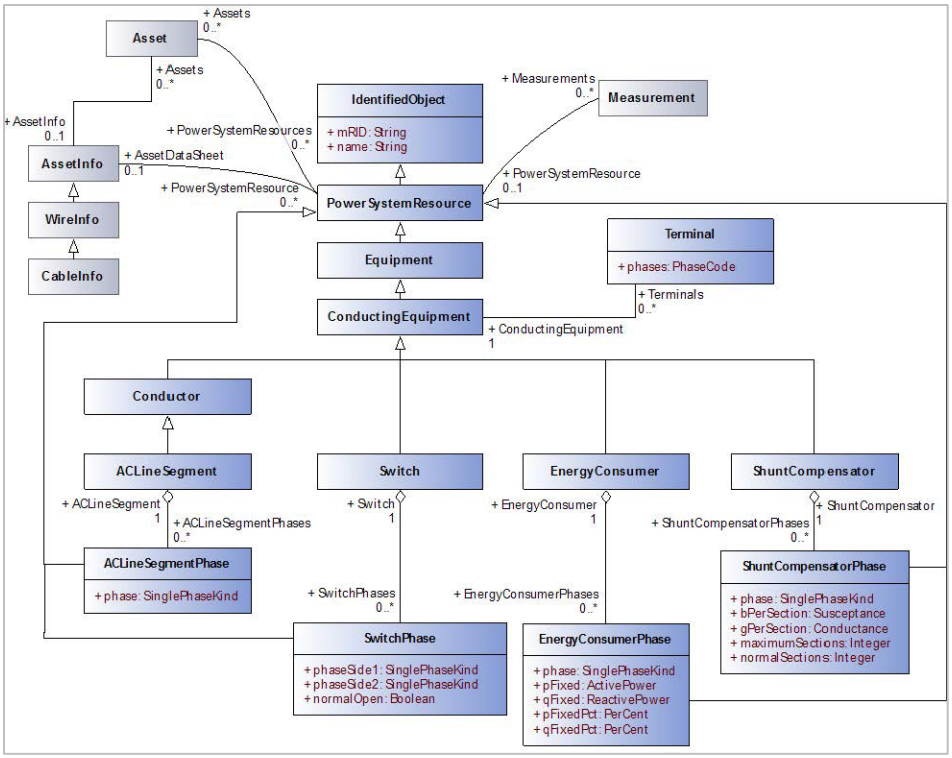


Figure 5-28. Unbalanced three-phase network components

Figure 5-28 shows the resulting CIM UML for describing the individual phase components for AC Line Segments, Switches, Energy Consumers (Loads) and Shunt Compensators. Each follows a common structure where the *Phase* class inherits from *PowerSystemResource* and so inherits all of its associations, including the association to *Asset*, *AssetInfo* (and thus *WireInfo* and *CableInfo*) and to the *Measurement* class (which represents a SCADA measurement point).

Each of the *Phase* classes has one or more attributes to define the individual *phase* the instance represents, and for the *SwitchPhase* class there are two attributes to allow a switch to span between two phases. Additional attributes are then added for the other classes to describe the per-phase values as appropriate.

These classes may be expanded in the future if there is a requirement to model other CIM electrical devices for unbalanced three-phase networks.

CIM Packages Overview

As with any other complex class structure, classes in CIM are grouped together into packages[25] dependent on their role within the utility. For a detailed view of all the packages, their classes and relationships the reader is urged to view the CIM UML model available to members of the CIM Users Group as an Enterprise Architect file or in the relevant IEC standard publications. This section will provide a brief summary of the three different standards that constitute the CIM UML.

IEC 61970-301

The core IEC 61970-301 standard contains a number of main packages, along with a global domain package used for defining data types. The Core, Wires and Topology packages contain all the basic classes for defining the physical characteristics of a power network.

The Wires package defines classes that are required to represent the electrical components of a network, such as *Transformers*, *AC Line Segments* and *Switches*, while the *Core* and *Topology* packages define the interconnection and containment of components.

These three packages alone; however, do not fully describe a functioning power system, but provide only the basic electrical characteristics of the equipment and describe how they are connected. To provide a detailed description of a network at an operational level, other classes are required to define the data required for operation and control including: generation data, operational limits, SCADA measurement, state variables, and protection. Most recently IEC 61970-301 has seen the addition of the *Diagram Layout* classes for defining the layout of diagrams linked to the electrical network data.

IEC 61968-11

IEC 61968-11 extends the IEC 61970-301 model and introduces the classes aimed at the exchange of data within distribution companies. This includes extensions to the electrical network model to cover the requirements of distribution networks.

[25] A package is used to "group elements, and to provide a namespace for the grouped elements" (OMG Unified Modeling Language (OMG UML), Infrastructure, V2.1.1 p.158.) A package may contain other packages and provides a hierarchical organization of elements within UML.

The model also extends the CIM with the support for *Asset Management*, *Customer Management*, *Work Management*, *Meter Data* (including remote meter readings and meter data management) and *Geographical Data*.

IEC 62325-301

IEC 62325-301 builds on IEC 61970-301 and IEC 61968-11 and adds packages for defining the data exchanged between participants in electricity markets. These classes include the data exchanges at the different stages of market operations including: billing, clearing, and settlement.

The IEC 62325-301 classes do not themselves model a market, they are used to define the data exchanged between entities involved in the operation of the market so include classes to describe agreements, invoices, prices, bids, notifications, and so on.

Case Study

Jeff Kimble arrives at his first CIM Users Group meeting. One the first day he attends what is known as “CIM University”. This is a one day workshop where presenters hit attendees with what can only be described as “a firehose of information”. In one day attendees get an introduction to the CIM UML, the tool that is used to maintain it (Sparx Systems Enterprise Architect) so that they also learn how to search the model for items of interest, how to navigate the packages, and the many diagrams that are contained within the model that highlight certain relationships and ideas, for example, metering, or circuit models.

Jeff learns that there are hundreds of classes, thousands of attributes, and many relationships between the classes that is a very robust representation of the utility world. Each class also contains a description of what it is used for, so the model is also a form of a data dictionary, which Jeff learns is referred to as a “semantic model”. In addition to these many diagrams, Jeff learns that there is a thing called a “profile.” A profile is a “subset of the model, used to perform meaningful work, such as used to define a message”. These profiles then may describe a circuit diagram but may also be used to create a message that can be sent from one system to another.

Jeff learns that these profiles can be expressed syntactically as either XML or RDF. RDF tends to lend itself well to network model data exchange because the RDF intrinsically expresses data in graph form. He also learns that the XML profiles lend themselves very nicely for message definition; in fact there are dozens of profiles that can be used for messaging that have already been defined, and in many cases, have had interoperability testing performed by several vendors, some of which are used at IEU.

Jeff decides that he needs to learn which vendors support these CIM interfaces and how he can accelerate the migration to a CIM-based semantic model that is used at the enterprise service bus.

Section 5 Questions

1. The IdentifiedObject class is used in the CIM to:
    a. Determine which class that has been identified
    b. Determine which object has been identified
    c. Provide a globally unique master resource identifier
    d. Provide a globally unique master class identifier
2. The NameTypeAuthority is used to:
    a. Provide the authority to identify classes
    b. Determine the types of names that can be used for a class
    c. Describes the type of authority for identifying classes
    d. Used to identify the organization that creates Names for the Names class as an alternative to using IdentifiedObject
3. If the relationship between ConductingEquipment and Terminal is denoted as 1..0* respectively, this indicates that:
    a. For each conducting equipment there are 1 or more terminals
    b. For each conducting equipment there are 0 or more terminals
    c. For each terminal there are many ConductingEquipment
    d. For each terminal there are 0 ConductingEquipment
4. The CIM has a mechanism for storing where assets are located via the _____ class.
    a. GPS
    b. Location
    c. CoordinateSystem
    d. StreetAddress
5. In the CIM hierarchy, a breaker is:
    a. A PowerSystemResource
    b. A ProtectedSwitch
    c. ConductingEquipment
    d. All of the above

6. CONTEXTUAL MODELING

Learning Objectives

- Understand the CIM Interface Reference Model (IRM)
- Understand how to derive CIM profiles
- Understand the different between the information, contextual, and information models
- Understand the main components of the Common Distribution Power System Model (CDPSM)

Deriving Profiles

The CIM is by definition intended to be a single, "common" model. One of the key goals of the CIM is to prevent duplication of data definitions but still define all the data exchanged between the systems within organizations supporting electric grid systems. The IEC 61968 standard includes an Interface Reference Model (IRM, shown in Figure 6-1). This diagram illustrates the breadth of the model itself. The CIM has grown from an initial core set of less than 100 classes to describe a balanced electrical model for EMS, to a model with over 1,000 classes and thousands of associations and attributes.

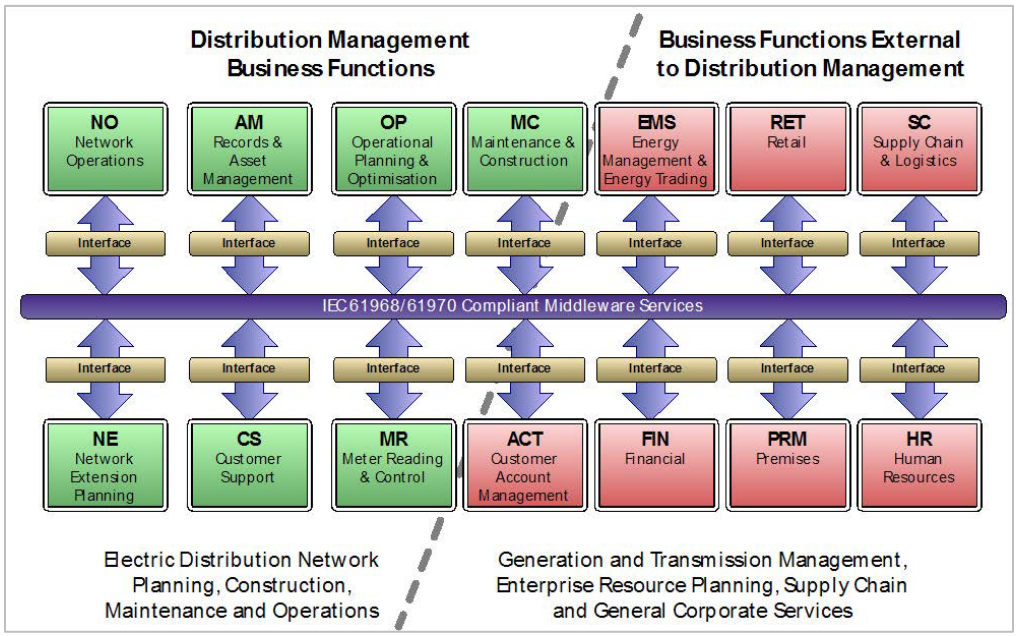


Figure 6-1. IEC61968-1 Interface Reference Model

Currently the IRM is being revised to combine both 61968 and 61970 business functions. The current work in progress is shown in Figure 6-2.

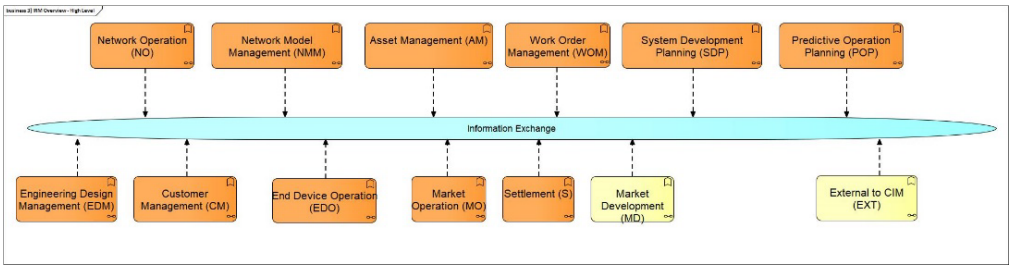


Figure 6-2. IEC61968-1 Interface Reference Model – Current Revision and Work in Progress

The scope of the CIM can be discovered in several ways:

- Reading the documents containing the IEC standards
- Viewing the UML model itself with Sparx Enterprise Architect or another tool
- Viewing the model in XML Meta Interchange (XMI) format with an XML viewer

Regardless of how a first-time visitor approaches the CIM, the model can appear unmanageably complex or even impractical, because every class and element is included. It's important to understand that contextual modeling can simplify the CIM and make it extremely practical and manageable as project teams move from design on into implementation.

Moving from Information Modeling to Implementation

The CIM must be viewed as an information model, not an implementation model. The relationships in the CIM are generalized and are 0..1 or 0..n, and all attributes are considered optional. This can result in multiple ways of expressing data in CIM, all valid, but potentially resulting in incompatible implementations downstream in the project delivery lifecycle. As such, the reader must understand the role of profiles.

To effectively adapt the CIM model for a specific context, CIM users can apply a *profile*. A profile is defined as a subset of the full model that serves to define the data exchanges required for target interfaces. Each profile is a collection of classes, attributes and references along with additional restrictions such as making attributes mandatory or restricting the cardinalities on associations.

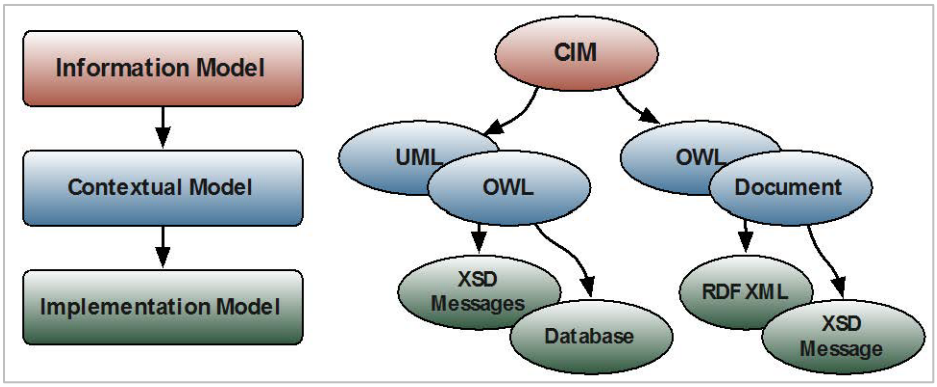


Figure 6-3. Information, Contextual and Implementation Models

This contextual model is then used to derive implementation models which define the serialization structure for the data. This could be a database schema, an RDF Schema or an XML Schema. There are different formats for defining the contextual model depending on the tools being used, but the overall approach is common to all.

Contextual Models Simplify without Definition Compromise

When viewed as its profile, a CIM interface is no longer a complex model of hundreds of classes, but is instead a small, select number of classes with the subset of their attributes and associations. When multiple interfaces are to be implemented by a single system, the practice of using profiles derived from a single overall model ensures that data definitions are re-used (and duplication is prevented). For example, a *GeneratingUnit* is defined once in the CIM but can be used in any number of profiles whether it is interfaces for transmission, distribution, market systems, or distributed energy resources.

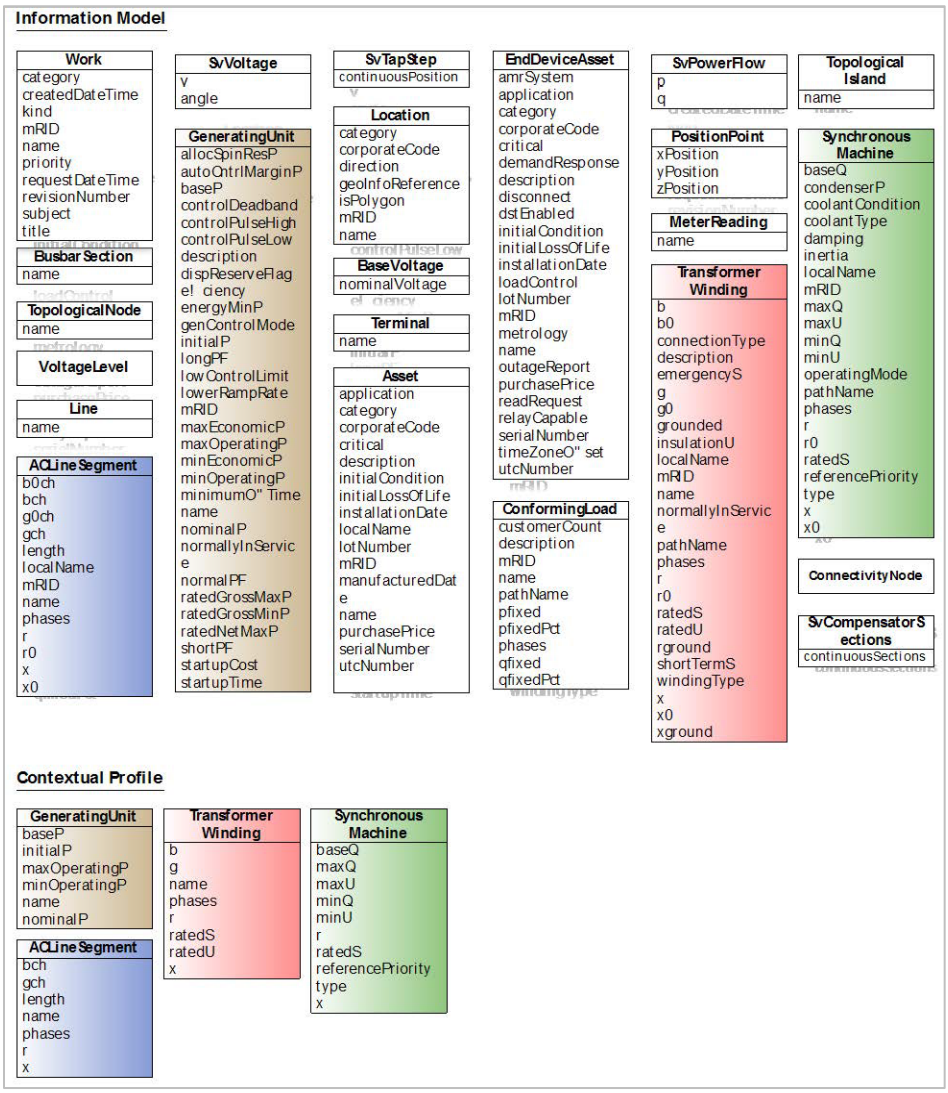


Figure 6-4. Example Information to Contextual Models

As shown in Figure 6-3, the addition of this *context* (that is, *perspective* or *view* of the model) makes it easier for a user to understand which parts of the CIM need to be implemented to support an interface. Supporting an interface of a few classes and attributes is far simpler than looking at 600+ classes and trying to determine how to map an existing interface.

Profiles Can Be Combined Into Groups

Even with profiles there is scope for modularity since many profiles cover specific exchanges that are of relevance to multiple use cases. For example, distribution and transmission system network profiles will have a lot in common, but there will always be differences since transmission system models do not, in general, need to model unbalanced power flow or single-phase, low-voltage transformers.

However, when it comes to concepts like the exchange of topology information and state variable solutions from power flow, the classes required are identical between the use cases. The approach here is to keep profiles as specific as possible so that multiple profiles can be combined together into a single *Profile Group* for a particular exchange.

With this approach a single data exchange becomes the exchange of multiple profiles, some of which may be shared with other groups in other exchanges. This provides additional modularity by enabling profiles to be re-used; keeps profiles smaller and focused, and prevents the duplication of profiles with minor changes for different use-cases.

Some CIM Profiles Are Standards

Since the CIM is so extensive and expansive, no organization would be expected to leverage or implement CIM in its entirety. But because there are common implementation patterns, profiles can become standards themselves. Examples are IEC standard documents such as 61970-452[26] (Common Power System Model), 61968-13[27] (Common Distribution Power System Model), and the Common Grid Model Exchange Standard (CGMES)[28]. These are each examples of groups of profiles derived from the overall CIM model for the purpose of facilitating exchange of electrical network models. For the exchange of messages between the enterprise and systems such as Metering, Work Management or Operations, a number of IEC 61968 standards profiles have been defined as well. Following is more detail on some sample profiles.

[26] "Energy management system application program interface (EMS-API) – 61970 Part 452: CIM static transmission network model profiles", IEC, Edition 3, 2016.

[27] "Application integration at electric utilities - System interfaces for distribution management – 61968 Part 13: CIM RDF Model exchange format for distribution", IEC, Edition 1.0, June 2010.

[28] "Energy management system application program interface (EMS-API) – Part 600-1: Common Grid Model Exchange Specification (CGMES) – Structure and rules", IEC 61970-600-1, Edition 1, 2021 and "Energy management system application program interface (EMS-API) – Part 600-2: Common Grid Model Exchange Specification (CGMES) – Exchange profiles specification", IEC 61970-600-2, Edition 1, 2021

Common Power System Model

IEC 61970-452 was the original CIM profile (now called a profile group). This is the Common Power System Model (CPSM) which is used for the exchange of electrical power system node-breaker models. Originally developed to define exchanges between EMSs, it has since grown to encompass the exchanges between multiple systems within the transmission environment. The key profiles within this group are Equipment, Short Circuit, and Operation profiles.

There are other profiles standardized in different IEC standards that define Topology, State Variables, Diagram Layout profile, etc. Together they all comprise the data exchanges between systems as part of the network analysis process (EMS or other systems) as shown in Figure 6-4.

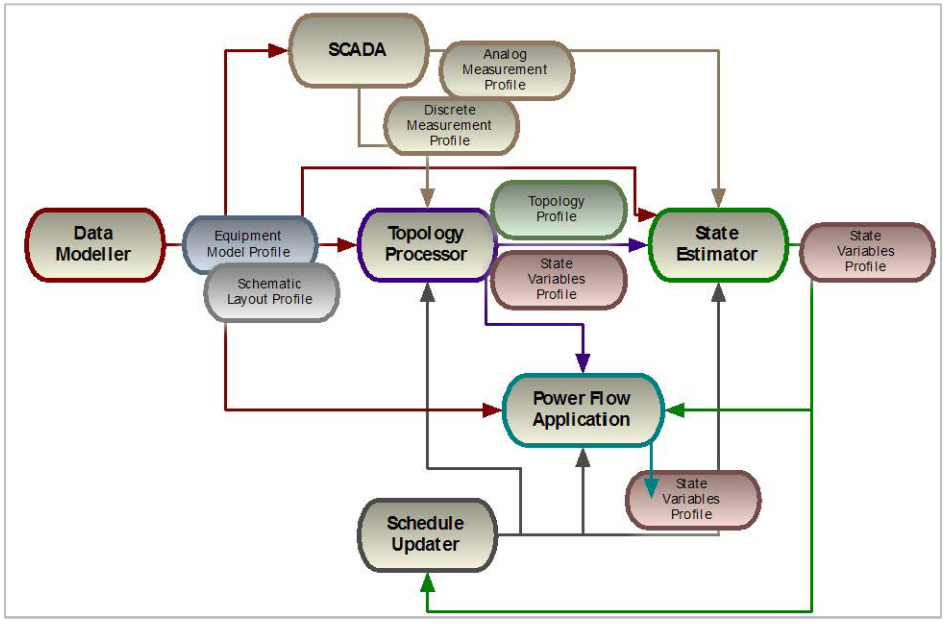


Figure 6-5. EMS Network Analysis Systems and Data Exchanges

Each instance of a profile (that is, when the profile is implemented and data is generated) is known as a dataset and there are often dependencies between each dataset allowing it to reference objects that exist in another dataset. For example:

- The *Equipment* dataset contains only internal references and so is not dependent on any other datasets.
- The *Topology* dataset is dependent on the *Equipment* dataset as it has references into objects that are created in *Equipment.*
- The *State Variables* dataset (or datasets) are dependent on both *Equipment* and/or *Topology* datasets as it may contain references to pieces of equipment and their *Terminal* objects and/or *TopologicalNode* objects that are in the *Equipment* and *Topology* datasets.
- The *Diagram Layout* dataset will reference the equipment and may also have references to *Topology* data.
- The *Measurement* datasets will reference the data in the *Equipment* dataset.

Common Distribution Power System Model

The Common Distribution Power System Model (CDPSM) is the distribution equivalent of CPSM, building on the standard balanced network model with additional data used to model low-voltage distribution networks.

The primary additions are:

- Classes to define the individual phase components for unbalanced network
- Asset information to tie the functional electrical model to the physical assets
- Catalogues of equipment parameters that are defined once then shared across multiple instances
- Geographical data to define the physical location of assets and/or functional components

Since distribution networks can specify the electrical properties of their equipment either explicitly for each instance (as is the case with the CPSM) or as catalogues, the *Equipment* profile from the CPSM is split into a core *Functional* profile that defines the basic electrical connectivity, voltages, and so on. This is combined with either the *Asset* profile which specifies catalogues of equipment properties or an *Electrical Properties* profile that adds the explicit properties to each instance.

The *Topology* and *State Variables* profiles from the CPSM are re-used, unaltered and the *Geographical* profile adds the geographical location data. Future versions of the CDPSM will also use the *Diagram Layout* profile and may also use the *Measurement* profiles.

- The core CDPSM profile group therefore becomes:
- Functional
- Electrical Properties
- Asset
- Geographical
- Topology
- State Variables

The re-use of *Topology* and *State Variables* from the CPSM is an example of how modular profiles are used in different groups that cover similar areas even when the standards are developed by different working groups[29].

The *Geographical* profile is also a generic profile that is not tied to a distribution representation. It associates to assets or functional components and with a loose association at the high level it can be used with high-voltage transmission systems as well as the low-voltage distribution network.

[29] CPSM is developed and maintained by IEC TC57 Working Group 13 while the CDPSM is from Working Group 14.

The target is to merge into a single family of profiles that will be used for both transmission and distribution.

ENTSO-E

The European Network of Transmission Systems Operators for Electricity (ENTSO-E) started an initiative in November 2007[30] to move from their existing UCTE Data Exchange Format (DEF) for load flow and three phase short circuit studies, a white space delimited ASCII file format, to a CIM XML based exchange. At that time, ENTSO-E exchanges involved high-voltage bus-branch planning models rather than the node-breaker models that the CPSM was developed for.

The CIM for Planning work was influenced by ENTSO-E's requirements and ENTSO-E also joined the CIM for Dynamics project to define a standard for exchanging dynamic models. ENTSO-E wanted to use standards wherever possible and since the project's inception the ENTSO-E profile has converged with that of the CPSM, using the *Topology* profile for the *Buses* in the system along with the *Equipment* and *State Variable* profiles.

In 2016 an interoperability test was held for CGMES Version 2.4. This profile group included:

- *Equipment* profile from IEC 61970-452
- *Topology* profile from IEC 61970-456
- *State Variables* profile from IEC 61970-456
- *Diagram Layout* profile from IEC 61970-453
- *Dynamics* profile from IEC 61970-457
- *Geographical* profile from IEC 61968-13

The ENTSO-E exchange specification was thus a profile group containing profiles from a number of IEC standards. All the profiles derive from the same CIM UML (in this case, CIM v16) and by modularizing the profiles and data in this way, applications need to access and import only the data required for their own internal processes. Later in 2017, ENTSO-E specification was transformed in IEC technical specifications, namely IEC TS 61970-600-1:2017 and IEC TS 61970-600-2:2017.

Following the requirements in the European Network Codes and the need to resolve issues in CIM16, in 2021 CGMES v3.0 was released as IEC standard – IEC 61970-600-1:2021 and IEC 61970-600-2:2021 and IEC technical specifications related to CGMES v2.4 were withdrawn. Nevertheless some processes are still using CGMES v2.4 but planning to transition to upper versions in order to satisfy business needs and improve on interoperability when exchanging detailed power system models.

[30] The project began under UCTE, the Union for the Coordination of the Transmission of Electricity which then became ENTSO-E in July 2009.

Case Study

With so many classes, attributes, and associations, when Jeff Kimble attended the CIM University training he was soon overwhelmed by the amount of information. It was difficult to even know where to start. Further, it turns out that the CIM isn't a single standards document, but "families" of documents apply to each of the standards that make up the CIM (IEC 61970, 61968, 62325). The power of the CIM is its reuse of definitions and the mapping of relationships between all of the entities, but the double-edged sword of this is that it can be difficult to understand where to begin.

Jeff did learn however that the CIM is maintained in a tool called "Enterprise Architect" (available from Sparx Systems). It uses standard UML to diagram the classes, associations, and attributes. He found that the three standards that make up the CIM (61968, 61970, and 62325) are organized in three high level packages – a convenient way to begin his CIM journey. He learned that by simply using the search function (Ctrl-F within the tool) he can search on a topic of interest whether that is an asset like a transformer or a more abstract concept like a customer. Once he finds a topic of interest he can learn more about its relationship with other concepts be finding where the topic (class) exists in the various diagrams. Very soon he becomes comfortable with navigating around the tool and learning how to find the information that he is interested in. Further, since the tool can export and import UML, he learns that the UML can be exported to any other tool that is UML compliant.

Section 6 Questions

1. In the CIM, the lowest level model (syntactic) is the:
    a. Information model
    b. Informative model
    c. Implementation model
    d. Contextual model
2. A CIM profile is:
    a. A subset of the model that may be more restrictive in its definition
    b. A subset of the model that is less restrictive in its definition
    c. A view of the model from the side
    d. A contextual view of the model
3. The Interface Reference Model (IRM) shows:
    a. All of the interfaces supported by the CIM
    b. The references used to develop the CIM
    c. A description of how to interface from two different systems
    d. A logical grouping and scoping of IEC 61968/IEC 61970 interfaces
4. The Common Distribution Power System Model (CDPSM) is the
    a. Standard for defining balanced power systems

b. Extends the standard for balanced power systems to model low voltage distribution systems
c. Model that defines the EMS
d. Model that defines ENTSO-E

5. Each instance of a profile is known as a:
    a. Dataset
    b. Model
    c. Profile
    d. Logical grouping

7. CIM RDF AND XML

Learning Objectives

- Understand how to map CIM into an RDF format
- Understand how difference files are used

Mapping CIM to RDF

As seen in Section 4, since the RDF and RDF Schema provide a means of mapping an object-oriented design to XML the CIM class structure can be mapped in a similar way. Existing tools can automatically generate an RDF Schema from the original CIM UML model.

This will directly map the CIM UML into RDF Schema and thus allow the CIM data objects to be mapped, one-to-one, into RDF instance data. RDF Schema supports a full class hierarchy and the subset used for CIM RDF XML[31] provides a logical serialization of CIM data into a standard XML format.

Using the previous example of the Transformer class hierarchy shown in Figure 5-13, the resulting RDF Schema takes the form.

```
<rdfs:Class rdf:ID="PowerSystemResource">
         <rdfs:label xml:lang="en">PowerSystemResource</rdfs:label>
        <rdfs:subClassOf rdf:resource="#IdentifiedObject"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Equipment">
        <rdfs:label xml:lang="en">Equipment</rdfs:label>
         <rdfs:subClassOf rdf:resource="#PowerSystemResource"/>
</rdfs:Class>

<rdfs:Class rdf:ID="ConductingEquipment">
        <rdfs:label xml:lang="en">ConductingEquipment</rdfs:label>
        <rdfs:subClassOf rdf:resource="#Equipment"/>
</rdfs:Class>
```

[31] IEC 61970-552 uses only a subset of the full W3C RDF standard. This allows a direct mapping between the CIM UML and RDF Schema without introducing additional choice or complexity. This simplifies parsing of CIM RDF XML data while maintaining compatibility with the standard. The result is that any RDF XML parser should be able to read CIM RDF XML but a parser written specifically for CIM RDF XML may not be able to cope with other RDF XML files that utilize the full specification.

```
<rdfs:Class rdf:ID="PowerTransformer">
        <rdfs:label xml:lang="en">PowerTransformer</rdfs:label>
        <rdfs:subClassOf rdf:resource="#Equipment"/>
</rdfs:Class>

<rdfs:Class rdf:ID="TransformerWinding">
        <rdfs:label xml:lang="en">TransformerWinding</rdfs:label>
        <rdfs:subClassOf rdf:resource="#ConductingEquipment"/>
</rdfs:Class>

<rdfs:Class rdf:ID="TapChanger">
        <rdfs:label xml:lang="en">TapChanger</rdfs:label>
        <rdfs:subClassOf rdf:resource="#PowerSystemResource"/></rdfs:Class>
</rdfs:Class>

<rdf:Property rdf:ID="TransformerWinding.PowerTransformer">
        <rdfs:label xml:lang="en">PowerTransformer</rdfs:label>
        <rdfs:domain rdf:resource="#TransformerWinding"/>
        <rdfs:range rdf:resource="#PowerTransformer"/>
</rdf:Property>

<rdf:Property rdf:ID="TapChanger.TransformerWinding">
        <rdfs:label xml:lang="en">TransformerWinding</rdfs:label>
        <rdfs:domain rdf:resource="#TapChanger"/>
        <rdfs:range rdf:resource="#TransformerWinding"/>
</rdf:Property>
```

Each CIM Class has a corresponding *rdfs:Class* entry, while the two aggregation relationships are expressed as RDF *Property* elements with the appropriate domains and ranges. The entire CIM Class structure can be expressed in this manner, and then this RDF Schema can be used to express a CIM power system model as RDF XML.

CIM RDF Serialization Examples

This class structure then defines how the instance data itself will look in an RDF format. As an example, the simple Transformer example from Figure 5-14 will be extended to include attributes for each object. This produces four objects with their own internal data as shown in Figure 7-1 below.

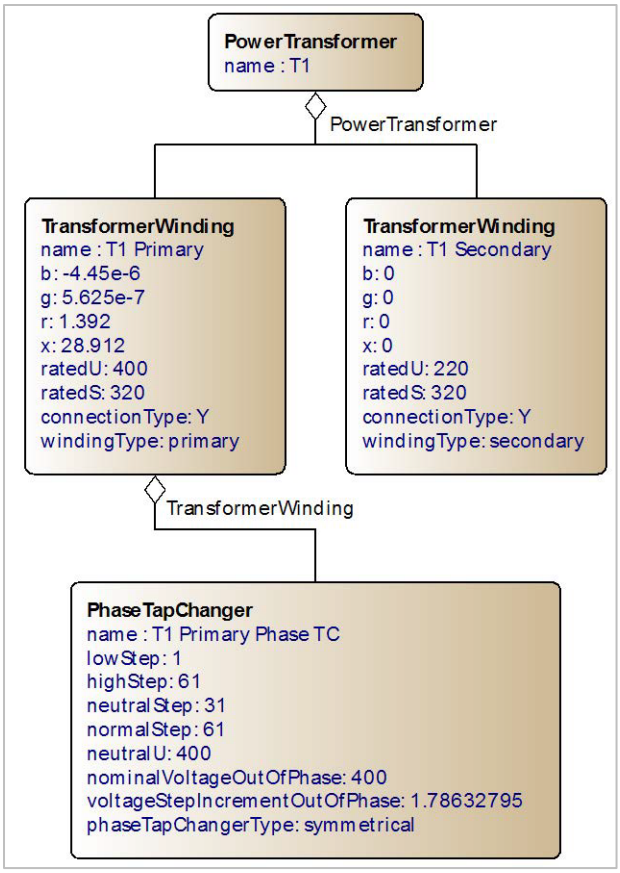


Figure 7-1. Transformer Shown as four CIM Objects with attributes

Each of these objects can then be expressed as an XML node using the CIM RDF Schema given the namespace http://iec.ch/TC57/2009/CIM-schema-cim14# and prefix *cim*: (the prefix itself can be set to any arbitrary alphanumeric string, but generally it is set as *cim* for CIM RDF XML exchanges). The resulting CIM RDF XML sample is shown below:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:cim="http://iec.ch/TC57/2009/CIM-schema-cim14#">

<cim:PowerTransformer rdf:ID="_1b8e9e856a5711dfa90800059a3c7800">
 <cim:IdentifiedObject.name>T1</cim:IdentifiedObject.name>
</cim:PowerTransformer>

<cim:TransformerWinding rdf:ID="_1b8e9e876a5711dfa90800059a3c7800">
 <cim:IdentifiedObject.name>T1 Primary</cim:IdentifiedObject.name>
 <cim:TransformerWinding.b>-4.45E-06</cim:TransformerWinding.b>
 <cim:TransformerWinding.g>5.625E-07</cim:TransformerWinding.g>
 <cim:TransformerWinding.r>1.392</cim:TransformerWinding.r>
 <cim:TransformerWinding.x>28.912</cim:TransformerWinding.x>
```

```
 <cim:TransformerWinding.ratedU>400</cim:TransformerWinding.ratedU>
 <cim:TransformerWinding.ratedS>320</cim:TransformerWinding.ratedS>
 <cim:TransformerWinding.connectionType
  rdf:resource="http://iec.ch/TC57/2009/CIM-schema-cim14#WindingConnection.Y"/>
 <cim:TransformerWinding.windingType
  rdf:resource="http://iec.ch/TC57/2009/CIM-schema-cim14#WindingType.primary"/>
 <cim:TransformerWinding.PowerTransformer
  rdf:resource="#_1b8e9e856a5711dfa90800059a3c7800"/>
</cim:TransformerWinding>

<cim:TransformerWinding rdf:ID="_1b8e9e906a5711dfa90800059a3c7800">
 <cim:IdentifiedObject.name>T1 Secondary </cim:IdentifiedObject.name>
 <cim:TransformerWinding.b>0</cim:TransformerWinding.b>
 <cim:TransformerWinding.g>0</cim:TransformerWinding.g>
 <cim:TransformerWinding.r>0</cim:TransformerWinding.r>
 <cim:TransformerWinding.x>0</cim:TransformerWinding.x>
 <cim:TransformerWinding.ratedU>220</cim:TransformerWinding.ratedU>
 <cim:TransformerWinding.ratedS>320</cim:TransformerWinding.ratedS>
 <cim:TransformerWinding.connectionType
  rdf:resource="http://iec.ch/TC57/2009/CIM-schema-cim14#WindingConnection.Y"/>
 <cim:TransformerWinding.windingType
 rdf:resource="http://iec.ch/TC57/2009/CIM-schema-cim14#WindingType.secondary"/>
 <cim:TransformerWinding.PowerTransformer
  rdf:resource="#_1b8e9e856a5711dfa90800059a3c7800"/>
</cim:TransformerWinding>

<cim:PhaseTapChanger rdf:ID="_1b8e9e8e6a5711dfa90800059a3c7800">
 <cim:IdentifiedObject.name>T1 Primary Phase TC</cim:IdentifiedObject.name>
 <cim:TapChanger.lowStep>1</cim:TapChanger.lowStep>
 <cim:TapChanger.highStep>61</cim:TapChanger.highStep>
 <cim:TapChanger.neutralStep>31</cim:TapChanger.neutralStep>
 <cim:TapChanger.normalStep>61</cim:TapChanger.normalStep>
 <cim:TapChanger.neutralU>400</cim:TapChanger.neutralU>
 <cim:PhaseTapChanger.nominalVoltageOutOfPhase>400
  </cim:PhaseTapChanger.nominalVoltageOutOfPhase>
 <cim:PhaseTapChanger.voltageStepIncrementOutOfPhase>1.78632795
  </cim:PhaseTapChanger.voltageStepIncrementOutOfPhase>
 <cim:PhaseTapChanger.phaseTapChangerType
```

```
  rdf:resource="http://iec.ch/TC57/2009/CIM-schema-
   cim14#PhaseTapChangerKind.symmetrical"/>
 <cim:PhaseTapChanger.TransformerWinding
  rdf:resource="#_1b8e9e876a5711dfa90800059a3c7800"/>
</cim:PhaseTapChanger>


</rdf:RDF>
```

The *TransformerWinding.connectionType*, *TransformerWinding.windingType* and *PhaseTapChanger.phaseTapChangerType* elements do not refer to other nodes within the document; instead their values are of an enumerated type. Enumerated types consist of a fixed set of legal values (that is, for a variable of type Days, the enumerated type would be: Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday and any variable of this type must have one of these values). Within the CIM there are certain class attributes that are also enumerated types and do not contain a node value but instead refer to an enumerated type within its RDF Schema.

This combination of the CIM, XML, RDF and RDF Schema allows an entire CIM power system model to be expressed in a standard, cross-platform plain-text format that is both human and machine readable and extensible.

Object/Node Identification

The CIM contains an attribute, *mRID* on the *IdentifiedObject* class that is the root for any class that is expected to be persisted beyond a single exchange[32]. In RDF every node has to have its own unique ID in the form of the *rdf:ID* (that is, an attribute *ID* under the *RDF* namespace, commonly denoted with the lowercase prefix *rdf*).

The convention with CIM RDF XML is that the *rdf:ID* will be the same as an object's *mRID* and thus the *mRID* omitted. However, every node within a CIM RDF XML file requires an *rdf:ID*, even those that do not inherit from *IdentifiedObject*. All RDF nodes will therefore have an *rdf:ID* and they should be persisted within an exchange between systems, that is, if A transmits CIM RDF XML data to B and B makes some updates or modifications, then the CIM RDF XML sent back from B to A should use the same *rdf:ID*s as the original exchanges.

The recommendation from the IEC working group is that these *rdf:IDs* should be Universally Unique Identifiers[33] (UUIDs) also known as Globally Unique Identifiers (GUID) when referring to Microsoft's implementation. The RDF standard specified that an ID may not start with a

[32] This means any object that is likely to be referenced or persisted within a system. An examples of a class that does not inherit from *IdentifiedObject* is *CurveData* which represents a point on a *Curve*. The *Curve* itself inherits from *IdentifiedObject* and so would be referred to by other objects and may exist in isolation. The *CurveData*, by contrast, is part of the curve (that is, a perfect example of *composition* as described in the background to UML in Section 4) so it does not need its own, persistent ID as it is the curve itself that is the persistent entity.

[33] See IETF RFC 4122 at https://www.ietf.org/rfc/rfc4122.txt.

number, only an underscore or character, so it is common to find all *rdf:ID*s in CIM RDF XML data prefixed with the underscore character.

Identifier Scope

Technically the true unique identity of a node is the concatenation of the local *rdf:ID* with the base Unique Resource Identifier (URI) of the file itself. As such a node of ID *123456* within the file with base URI *http://example.com/my/file* has the unique identity *http://example.com/my/file#_123456*. CIM RDF XML exchanges, by using the *mRID* as the *rdf:ID* do not necessarily follow this convention and the local node identifier is, itself, considered to be globally unique. As such any node across multiple files identified with the same *rdf:ID* is considered to be referring to the same, universally unique object. This means that when exchanges take place using multiple files the cross-file references use only the *rdf:ID*.

Extensibility and Efficiency

RDF XML is a verbose, self-descriptive format and so offers a number of benefits and perceived drawbacks. As the example above shows, with an RDF XML format every data element has descriptive tags. This allows the format to be extensible as additional elements can be added without 'breaking' parsers that rely on fixed formats.

Extensibility

By using the XML namespaces, additional elements are added under different namespaces. In the example below the namespace *http://example.com/extension#* is used to define an additional *type* element for the *PowerTransformer* class.

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:cim="http://iec.ch/TC57/2009/CIM-schema-cim14#"
 xmlns:ex="http://example.com/extension#">

<cim:PowerTransformer rdf:ID="_1b8e9e856a5711dfa90800059a3c7800">
 <cim:IdentifiedObject.name>T1</cim:IdentifiedObject.name>
 <ex:PowerTransformer.type>SuperGrid</ex:PowerTransformer.type>
</cim:PowerTransformer>

</rdf:RDF>
```

For those parsers that do not recognize or understand the additional namespace the extra elements are simply ignored. Multiple additional namespaces can be used, allowing additional, extended data to be integrated with the standard CIM data within a single file.

Compression and Efficiency

The nature of XML, its verbose format where the XML tag elements often take up more space than the actual instance data itself, results in large file sizes. This is often perceived as a weakness of CIM RDF XML over more traditional, fixed-width, comma or whitespace delimited formats.

This assumes that the CIM RDF XML will be transmitted as uncompressed data, ignoring the significant reduction in file size that can be accomplished with ZIP compression, an archive file format with near-ubiquitous support on modern computing environments. The format provides a means of compressing and archiving multiple files into a single entity using the DEFLATE[34] lossless data compression algorithm, itself based on LZ77[35] and Huffman coding.[36]

Other widely-used application formats including Sun Microsystem's Java Archive format (JAR), Microsoft's Office Open XML, Google's compressed Keyhole Markup Language (KMZ) and ISO/IEC 26300:2006 Open Document Format (ODF) all use ZIP to compress either a single file or multiple file into a single compressed archive file. This approach maintains the underlying data hierarchies and for ODF, Office Open XML and KMZ the underlying XML files, while significantly reducing the size of the resulting file.

Using the same approach with CIM RDF XML data file sizes can be reduced by a factor of over 20, thus reducing a 3Gb CIM RDF XML file to a ZCIM file of approximately 120Mb or less. The ZIP archive format compresses each individual file independently, and as such the files within the archive can be read independently without requiring the entire archive to be decompressed first. With modern languages such as .NET and Java having native support for reading and writing to ZIP archives and free libraries available for C, C++ and other common languages, applications can deal with these ZIP files and parse the individual XML files without having to ever decompress the entire file to disk.

While reading from and writing to ZIP archives does involve extra processor overhead to perform the compression and decompression, initial tests with large real-world CIM RDF XML files has found that there is no performance penalty incurred with import and export times. This can be attributed to the decrease in disk input/output required to read a smaller file and with disk input/output being significantly slower than memory access the increase in processing time for compression and decompression is offset by the decreased disk input/output required.

[34] P. Deutch. DEFLATE Compressed Data Format Specification version 1.3 [RFC 1951], https://www.ietf.org/rfc/rfc1951.txt, May 1996.

[35] Jacob Ziv and Abraham Lempel, "A Universal Algorithm for Sequential Data Compression," IEEE Transactions on Information Theory, May 1977, Volume 23, Number 3, pp. 337–34.3

[36] D.A. Huffman, "A Method for the Construction of Minimum-Redundancy Codes", Proceedings of the I.R.E., September 1952, pp. 1098–1102.

CIM Difference Model

Another manner of reducing the volume and complexity of the CIM RDF XML exchanges is to utilize the CIM Difference Model XML format that describes the changes between two CIM RDF XML files. This allows incremental exchanges of CIM data so that the full network model does not need to be exchanged.

The Difference Model format defines four statement sets:

- Reverse Differences
- Forward Differences
- Pre-Conditions
- Post-Conditions

(In reality the Pre and Post conditions are rarely used, and interoperability tests focus on the Forward and Reverse differences).

A difference model can add new objects, remove existing objects or update objects by changing, adding or removing attributes. The Reverse Differences define the removals while Forward Differences define the additions. A change (update) is defined as a pair of Forward and Reverse differences on an existing object.

The CIM DM XML format is RDF-based and uses a similar format to that of the full model CIM RDF XML described in the previous section.

Difference Model RDF XML Format

As with the full CIM RDF XML data, the Difference Model format begins with the namespace declarations under an RDF node:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:cim="http://iec.ch/TC57/2009/CIM-schema-cim14#"
 xmlns:dm="http://iec.ch/2002/schema/CIM_difference_model#">
```

In addition to the CIM and RDF namespaces that were present in the full model, the Difference Model namespace is included. This defines the additional elements that are used to contain the forward and reverse statements:

```
<dm:DifferenceModel>
 <dm:reverseDifferences parseType="Statements">
  ...
 </dm:reverseDifferences>
 <dm:forwardDifferences parseType="Statements">
  ...
 </dm:forwardDifferences>
</dm:DifferenceModel>
```

Inside these *reverseDifferences* and *forwardDifferences* nodes are placed the data elements that are to be added, removed or updated.

Addition

To add a CIM object the data structure is identical to that of the full CIM RDF XML format. The elements are placed with the *forwardDifferences* node and take the format shown below:

```
<dm:forwardDifferences parseType="Statements">
 <cim:ACLineSegment rdf:ID="_lang0182743">
 <cim:IdentifiedObject.name>A New Line</cim:IdentifiedObject.name>
 </cim:ACLineSegment>
</dm:forwardDifferences>
```

In the example above a new *ACLineSegment* has been added. The format and structure is identical to that of an *ACLineSegment* in a full RDF XML file, with the *rdf:ID* providing the unique identifier for the object and all attributes and associations included inline.

Deletion

The deletion of an existing object uses a very similar format, albeit under the *reverseDifferences* rather than the *forwardDifferences* node. For a deletion there are two methods for defining the node to be accomplished.

The first is a shorthand syntax that defines only the object's ID and none of the object's attributes or associations. For example, to delete the object that was added in the example above the shorthand syntax takes the form:

```
<dm:reverseDifferences parseType="Statements">
 <cim:ACLineSegment rdf:ID="_lang0182743"/>
</dm:reverseDifferences>
```

The other approach is a more verbose syntax that includes all of the object's attributes and associations:

```
<dm:reverseDifferences parseType="Statements">
 <cim:ACLineSegment rdf:ID="_lang0182743">
 <cim:IdentifiedObject.name>A New Line</cim:IdentifiedObject.name>
 </cim:ACLineSegment>
</dm:reverseDifferences>
```

The benefit of the shorthand syntax is that when removing a large number of objects it reduces the size of the incremental file and omits additional data that is, essentially, superfluous when the object is to be deleted anyway.

However, by using the longhand syntax, a difference file can be applied in reverse by changing all *forwardDifferences* to *reverseDifferences* and vice-versa. This allows a system to *undo* a

difference file using only the data contained within the difference file. By storing all changes as verbose, longhand syntax incremental difference model files previous systems states can be restored without requiring additional state information to be stored separately.

The longhand syntax also has the additional benefit of allowing a system to perform additional data verification by ensuring that the object's attributes and associations match those in the difference model. If a mismatch is identified this will alert the user that their system may be out of synchronization with whatever system generated the incremental.[37]

Updates

To change an existing object the syntax is slightly different. In this situation the RDF Description notation is used with the *rdf:about* attribute replacing *rdf:ID* as the identifier. This is used to indicate that the node is referencing a pre-existing object within the full model, rather than being the creation of a new object.

So, to remove an attribute or an association on an existing model, a Description element is added with the syntax:

```
<rdf:Description rdf:about="#_pqrs34567890">
...
</rdf:Description>
```

This refers to an existing object of ID *_pqrs34567890* that already exists within the system. Inside these *Description* tags are placed the attributes or associations that have to be added or removed on this object. For example, to remove the association from a *Terminal* with the ID *_pqrs34567890* to a piece of *ConductingEquipment* with the ID *_abcdef0123456789* the resulting DM XML is:

```
<dm:reverseDifferences parseType="Statements">
 <rdf:Description rdf:about="#_pqrs34567890">
  <cim:Terminal.ConductingEquipment rdf:resource="#_abcdef0123456789"/>
 </rdf:Description>
</dm:reverseDifferences>
```

The syntax inside the *Description* tags is identical to that of the full model RDF XML, the difference here is that we are indicating that his association is to be removed.

[37] This is not necessarily an unexpected situation. There are several scenarios where a difference file may be applied to a base model that doesn't exactly match that of the sending system (for example, where a difference file represents a potential network extension, and the user wants to apply it to a different planning model than the one it was created against). In this situation the user (or system) may choose to apply the differences even where there is an identified source/target mismatch.

The corresponding *forwardDifferences* statement is an identical format, but by being within the *forwardDifferences* element it is interpreted as an addition rather than a removal from the object:

```
<dm:forwardDifferences parseType="Statements">
 <rdf:Description rdf:about="#_pqrs34567890">
  <cim:Terminal.ConductingEquipment rdf:resource="#_lang0182743"/>
 </rdf:Description>
</dm:forwardDifferences>
```

The combination of a *forward* and *reverse* statement for the object therefore results in an update, changing the *ConductingEquipment* association in this case from *_abcdef0123456789* to *_lang0182743* (the ID of the *ACLineSegment* that was added in the first example).

Combining the addition and the updates together into a single DM XML file that adds a new *ACLineSegment* and changes an existing *Terminal* to associate with this new line we get:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:cim="http://iec.ch/TC57/2009/CIM-schema-cim14#"
 xmlns:dm="http://iec.ch/2002/schema/CIM_difference_model#">

 <dm:DifferenceModel>
  <dm:reverseDifferences parseType="Statements">
   <rdf:Description rdf:about="#_pqrs34567890">
   <cim:Terminal.ConductingEquipment rdf:resource="#_abcdef0123456789"/>
   </rdf:Description>
  </dm:reverseDifferences>

  <dm:forwardDifferences parseType="Statements">
   <cim:ACLineSegment rdf:ID="_lang0182743">
   <cim:IdentifiedObject.name>A New Line </cim:IdentifiedObject.name>
   </cim:ACLineSegment>
   <rdf:Description rdf:about="#_pqrs34567890">
   <cim:Terminal.ConductingEquipment rdf:resource="#_lang0182743"/>
   </rdf:Description>
  </dm:forwardDifferences>
 </dm:DifferenceModel>

</rdf:RDF>
```

CIM RDF Headers

The CIM RDF XML samples above contain no meta-data to denote versioning; the type of dataset it is; any profiles to which is conforms; any dependencies between datasets; or information about the creation date and author. To address this a standard *CIM Header* format has been created that will be inserted at the beginning of a CIM RDF XML file to contain this meta-data. As well as embedding the information required for the actual exchange, these headers provide useful data for importers that are to deal with the network model files.

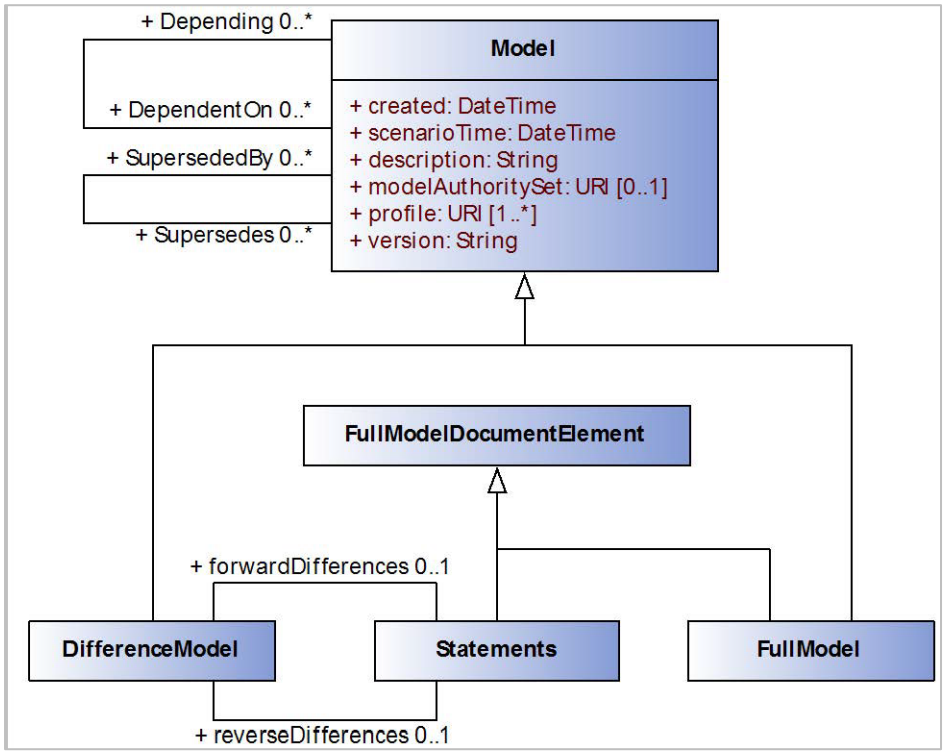


Figure 7-2. CIM Header UML Model

Figure 7-2 shows the UML for the CIM Header. While on first inspection the class diagram may appear complex, in reality the classes *FullModel* and *DifferenceModel* are the two concrete classes that are instantiated (sharing all the attributes and associations from the common parent class, *Model*). The additional classes are used so as to allow the header structure to be harmonized with that of the Difference Model format described in the previous section.

The *FullModelDocumentElement* class describes any of the elements that may appear in the full model document itself. It has two subtypes in the form of *Statements* and *FullModel*. A full CIM RDF XML file typically will contain a *FullModel* instance and one set of *Statement* elements.

The *Statement* class represents a series of RDF XML nodes that describe the instance data for the CIM objects (either as full nodes or the *rdf:Description* elements for a *Difference Model* update)

The *FullModel* class is the full model header used to provide the meta-data for a CIM RDF XML instance file

The *DifferenceModel* class represents both the Difference Model Header and its root element that was shown in the previous section as: <dm:DifferenceModel>

This element contains two sub-elements of *forwardDifferences* and *reverseDifferences* that are the statements defining the additions, removals and updates. Each of these will contain one set of statements[38].

The abstract *Model* class has the core attributes that is inherited by all *DifferenceModel* and *FullModel*. Unlike the RDF nodes described in the full CIM RDF XML instance data exchange, the header data uses the *rdf:about* syntax to define its identity rather than *rdf:ID*. This uniquely identifies the model not just the file and so reflects the *state* of the data. The same data may be stored in two different files with different names but the unique identity in the *rdf:about* will be the same.

The resulting instance data takes the form:

```
<rdf:RDF xmlns:cim="http://iec.ch/TC57/2010/CIM-schema-cim15#" xmlns:md="http://iec.ch/TC57/61970-552/ModelDefinition/1#" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
 <md:FullModel rdf:about="urn:uuid:2399cbd4-9a39-11e0-aa80-0800200c9a66">
  <md:Model.created>2011-06-28T10:00:00</md:Model.created>
  <md:Model.scenarioTime>2012-07-15T10:30:00</md:Model.scenarioTime>
  <md:Model.version>7</md:Model.version>
  <md:Model.DependentOn
   rdf:resource="urn:uuid:2399cbd2-9a39-11e0-aa80-0800200c9a66"/>
  <md:Model.DependentOn
   rdf:resource="urn:uuid:2399cbd1-9a39-11e0-aa80-0800200c9a66"/>
  <md:Model.description>Belgium topological node data</md:Model.description>
  <md:Model.modelingAuthoritySet>http://elia.be/Planning/ENTSOE/2
   </md:Model.modelingAuthoritySet>
  <md:Model.profile>http://iec.ch/TC57/61970-456/Topology/2</md:Model.profile>
 </md:FullModel>
```

The result of the CIM data then proceeds this header entry as described in the previous sections. The header has a UUID for its unique identity and is *Dependent On* two other models, referencing them with the *rdf:about* in their *FullModel* or *DifferenceModel* header entries (in this case the unique ID of these models is also a UUID). The dependencies allows and importer to ensure that all the prerequisite models exist (or can be found) prior to importing. Unresolved

[38] The Difference Model format precedes the Header definition and as such the Header was created to be backwards compatible with the pre-existing Difference Model format.

dependencies may indicate missing files and so the user can be appropriately informed (and may choose to proceed anyway or cancel and resolve the dependency issues).

The *modelingAuthoritySet* attribute defines the set to which this file belongs. In exchanges where multiple files constitute a single *Modeling Authority* then all the files will have the same *modelingAuthoritySet* ID. In the example above the attribute denotes that this file is part of a set for ELIA (the Belgian Transmission Grid Operator) in the ENTSO-E exchange. Other files for the *Equipment* and *State Variables* will have the same *modelingAuthoritySet* ID and so the collection of these files form ELIA's *Modeling Authority Set*.

The header may also denote one or more profiles that the data conforms to. This defines a minimum set of required data elements and restrictions but also allows an importer to quickly identify which of the collection of files it has received is relevant to its own particular data requirements. For example, a state-estimator has no need for geographical or schematic diagram layout data so it can quickly identify files that are superfluous and move on, ignoring their contents.

Similarly, the versioning information can be used to identify when a model has been updated from a previous import and so requiring the application to import and process the data or is the same version as the last import and thus allowing a model to be bypassed.

RDF Schema Definition in CIMTool

Defining the Message Structure

CIMTool is an open-source tool that supports the definition of CIM profiles. As of 2025, it can be used to generate JSON schemas, database schemas, XML schemas, RDF schemas, PlantUML, Java classes, and more. For this example, CIMTool v2.3.0 is used with CIM18.

Once a CIMTool project has been created using the CIM18 schema, a new profile can be created that will define the subset of the CIM that will be used for this grid model data management interface.

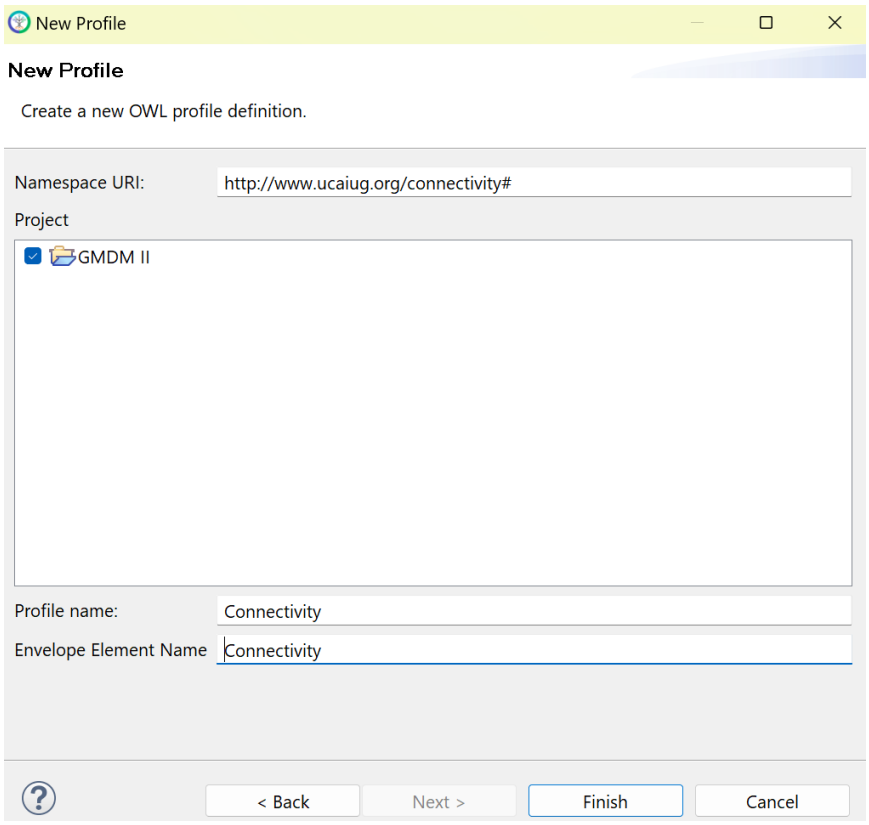


Figure 7-3. Creating a new CIMTool profile for RDFS.

In Figure 7-3 is an example of a New Profile dialog box. Each profile has a namespace Unique Resource Identifier (URI) which in this case has been set as http://www.ucaiug.org/connectivity#. The *Profile Name* is used to name the profile within the workspace and the *Envelope Element Name* typically defines the root of the message itself. However, since this is being used to generate RDFS instead of XML schema, this will not actually be used for that purpose.

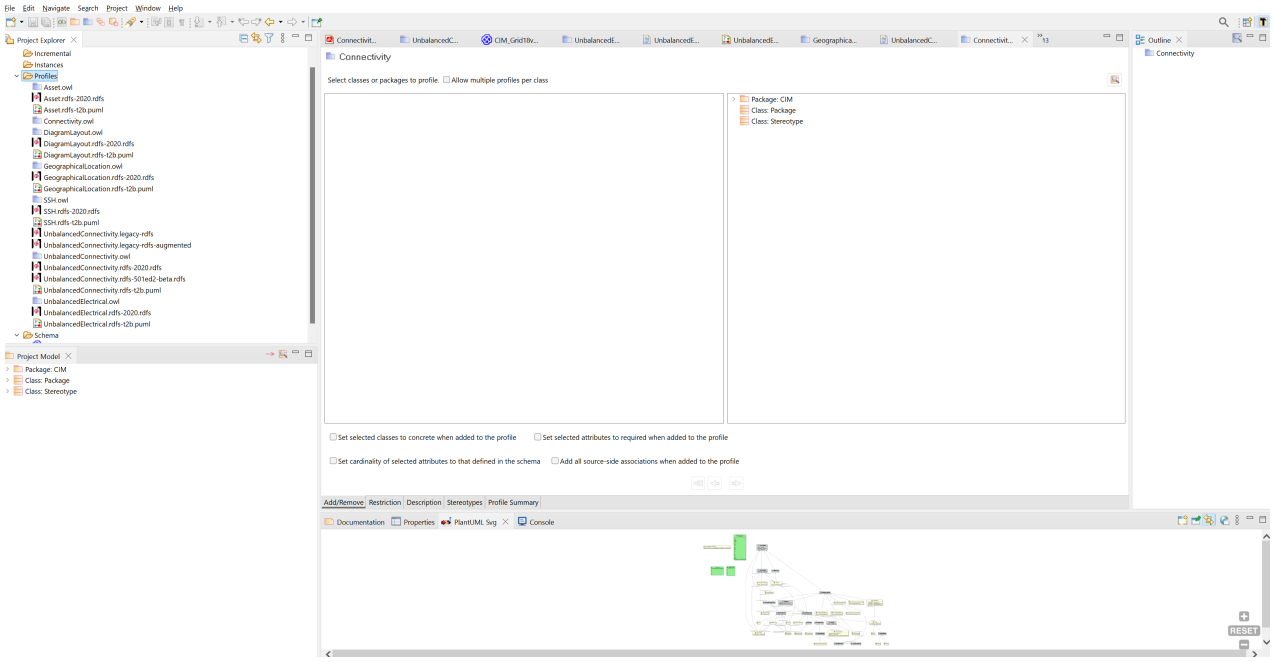


Figure 7-4. Empty Connectivity profile

This produces an empty profile called *Connectivity* as shown in Figure 7-4, which can be populated with classes taken from the full CIM UML model. To do this, there are two valid approaches to take assuming the schema has been added (this can be either as an XMI file or by bringing in the .eapx file directly). Either the CIM package hierarchy can be expanded and the proper classes manually added, or the search button can be used to look up the name of the class that is to be added or removed. For this example, the latter option will be utilized.

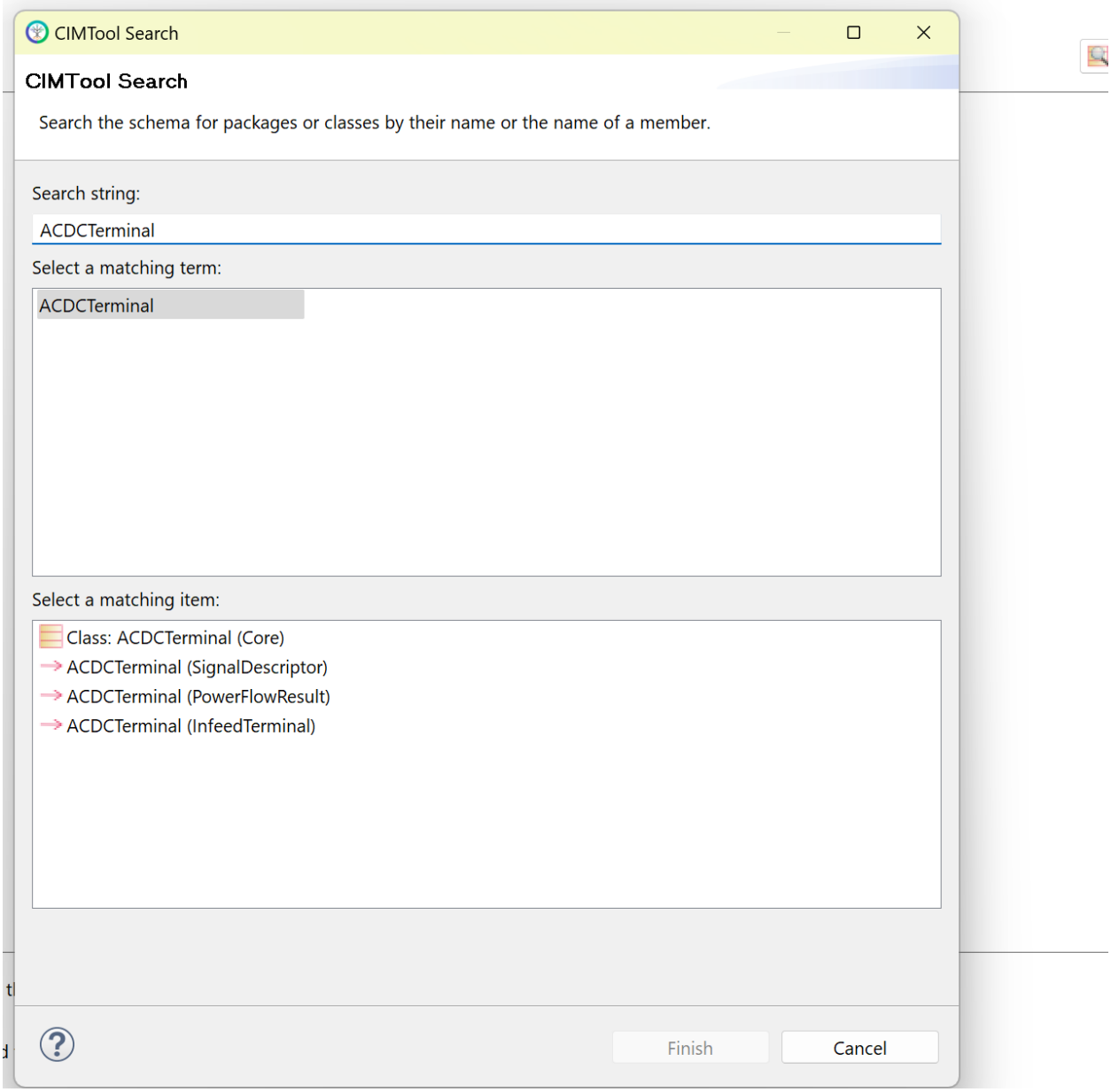


Figure 7-5. Using CIMTool's search to add ACDCTerminal

As shown in Figure 7-5, classes can be instantly searched and then dragged over into the profile. For this example, the class *ACDCTerminal* was searched and then dragged into the profile using the left arrow button. This class, however, is considered an Abstract class; that is, it is not to be directly referenced in RDF messages but is brought into the profile anyway so that other classes that inherit from it can be modeled. Because this class will only inherit from IdentifiedObject and the profile has no need of any other attributes, there is nothing else to be done with this class. So next, let's bring in the *Terminal* class that will be a Concrete class and will inherit from *ACDCTerminal*.

To add *Terminal*, the same steps as above will be performed. The class will be searched, the left arrow will be used to draw it into the profile, and from there things will look just slightly different. Since this is a concrete class, it must be set as such using the Restrictions tab in CIMTool.

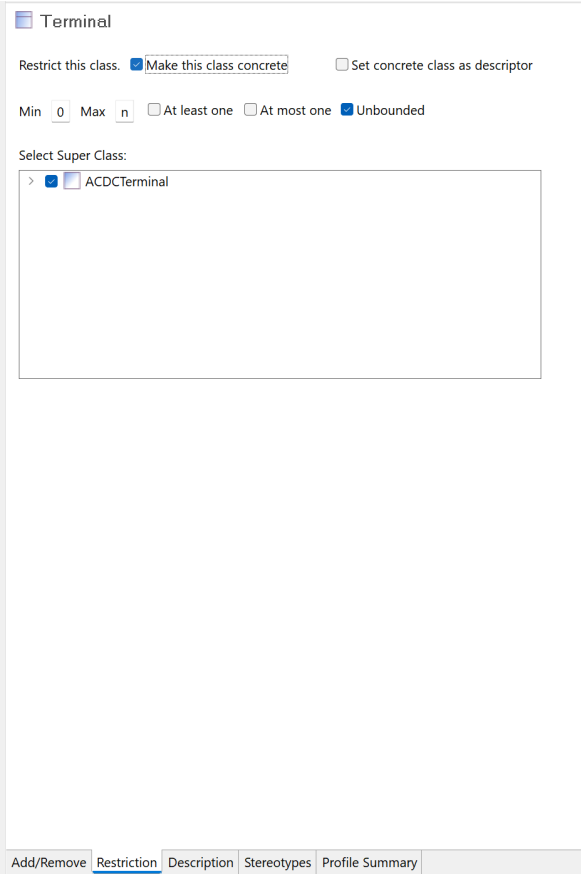


Figure 7-6. Adding Terminal as a Concrete class

As shown in Figure 7-6, *Terminal* was marked as a concrete class and a key visual for that is the appearance change of the icon next to *Terminal*. Now that *Terminal* is in the profile and is properly shown as a concrete class with *ACDCTerminal* as its Super Class (that is, the class from which it inherits its attributes), it is time to add some attributes and class relationships to it.

In this case, the only attribute that is being added to *Terminal* is **phases**. Since phases is an enumeration, adding it in CIMTool works slightly different. Enumerations in the CIM are modeled as classes, so not only does **phases** have to be brought in, but the *PhaseCode* class also must be added. To do this in the least amount of steps, simply use the double arrow icon to add this particular attribute.

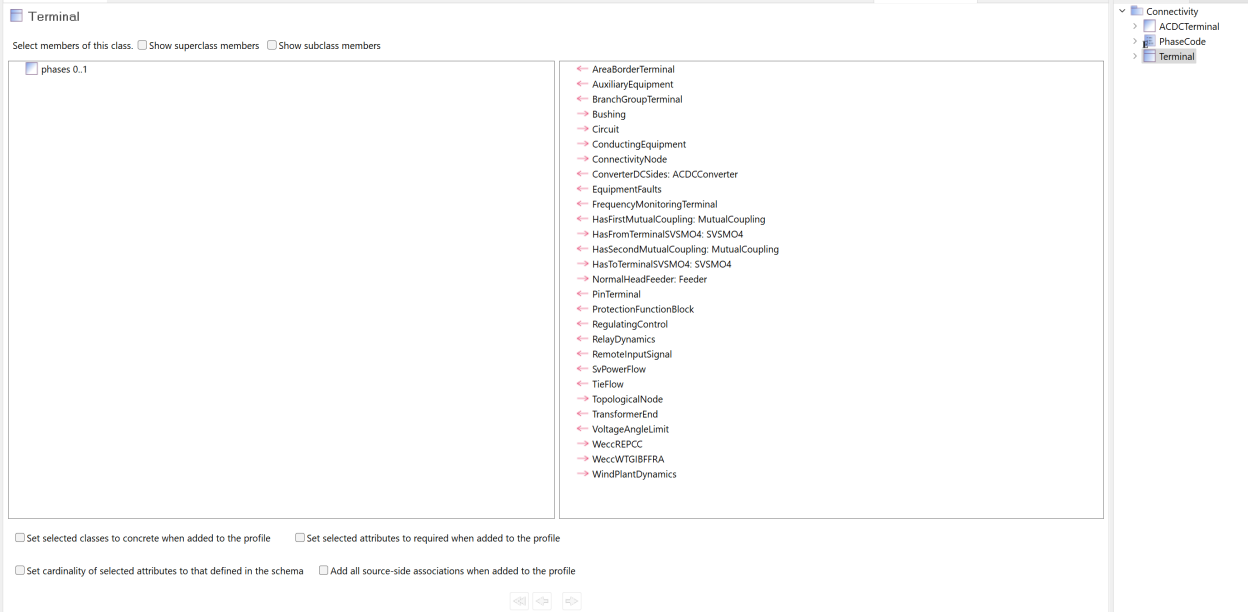


Figure 7-7. Adding enumerations in CIMTool.

Upon doing this, not only will this new attribute be added to Terminal, but as shown in Figure 7-7 there will be a new class added to the Outline for the profile as shown on the right side for *PhaseCode*.

In addition to adding attributes, there are also other classes associated with *Terminal* that can be added on this screen. In the CIM UML, *Terminal* has relationships to many classes such as *Feeder* or *ConnectivityNode*. Despite the fact that RDFS defines bidirectional relationships between these classes, when creating the RDF schema in CIMTool, only class relationships denoted by the right arrow should be defined within that class. In other words, *ConnectivityNode* should be added here, but not *Feeder*. Instead, *Feeder* would add its relationship to *Terminal*.

To add this class relationship, let's first bring ConnectivityNode into the profile as shown in Figure 7-8 below. This will be a concrete class.

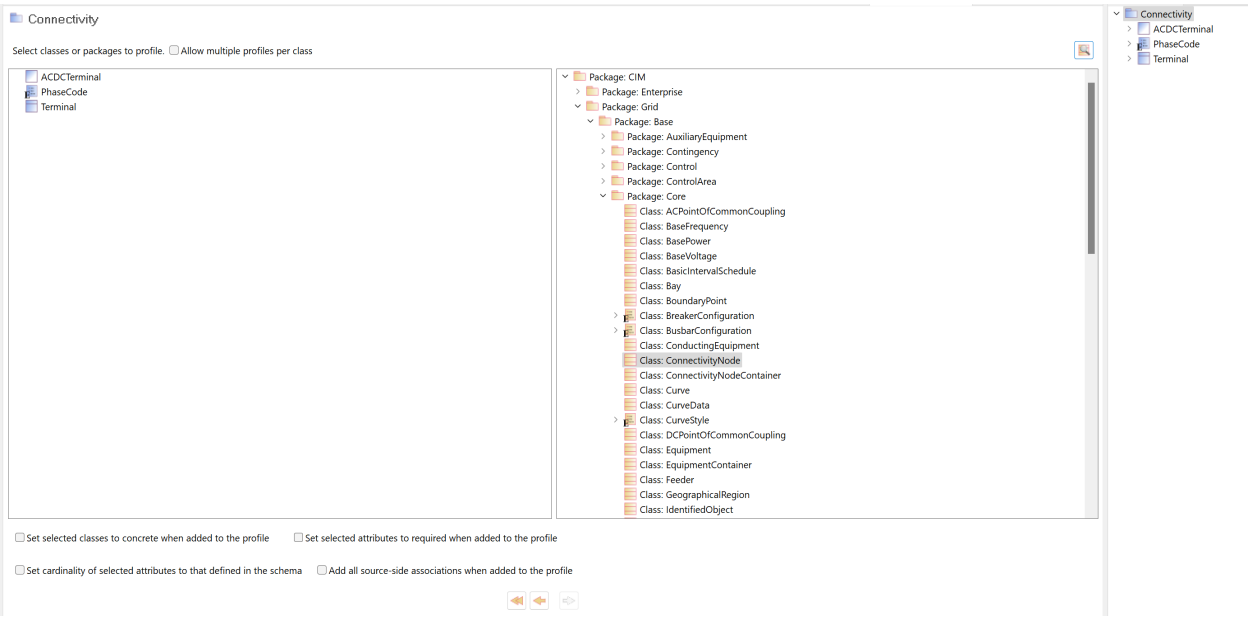


Figure 7-8. Adding ConnectivityNode to the profile.

With that out of the way, go back to Terminal and add ConnectivityNode and after doing that, double click the ConnectivityNode within its Terminal definition to add further restrictions. It should look like the page shown below.

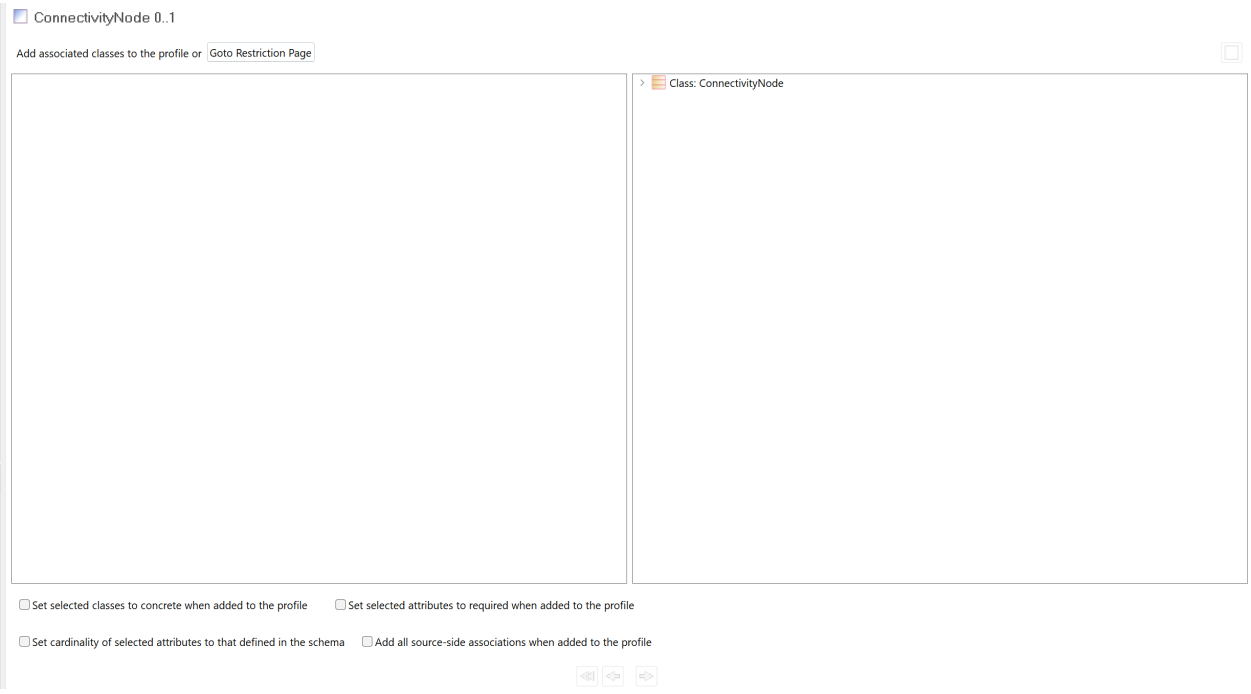


Figure 7-9. Adding restrictions to ConnectivityNode

It is important to do a few things here. One is to check the box under *Selected Associated Class or Classes*. Only *ConnectivityNode* should be there for this example, but it's possible for multiple classes to be here depending on the profile. Next, make sure to check the box saying that says *By Reference* to avoid any circular references in the schema.

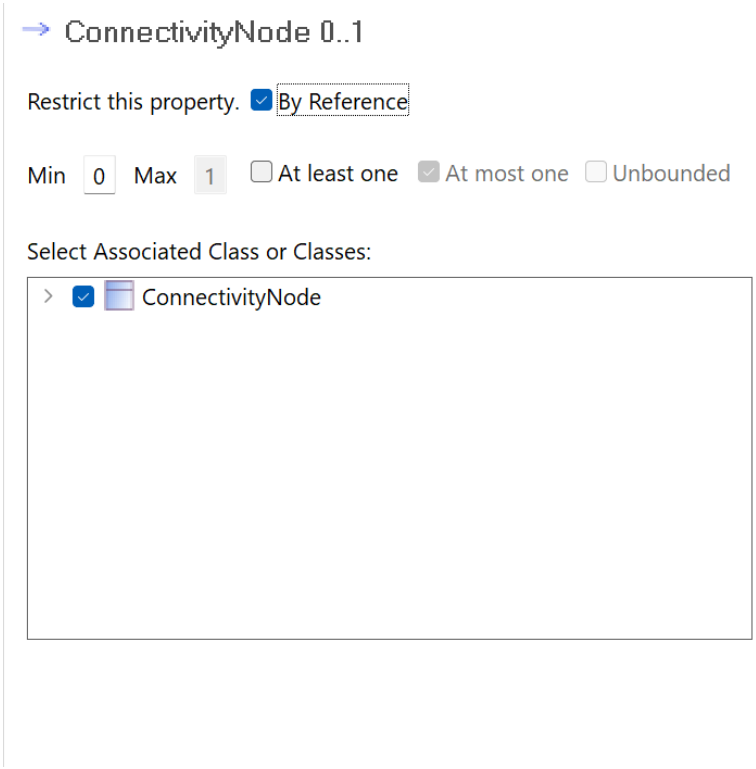


Figure 7-10. Associations by reference

But wait a minute. Is *IdentifiedObject* going to be defined? Of course, but the order of doing so does not matter and once it is added in, the proper classes will automatically inherit from it. To demonstrate, this will be added next so that *ACDCTerminal* can inherit its attributes.

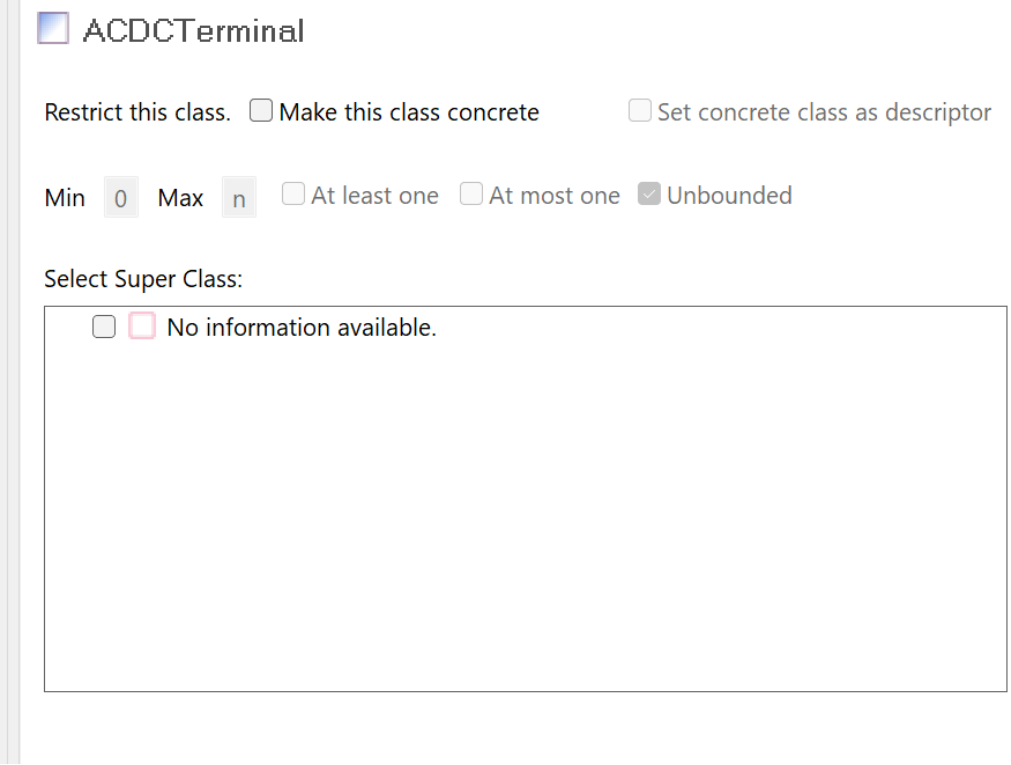


Figure 7-11. ACDCTerminal before adding IdentifiedObject

IdentifiedObject will be now be added following the same steps outlined earlier. It will have the attributes *mRID*, *description*, and *name*, with *mRID* being required. Upon being added, checking the restrictions page of ACDCTerminal confirms that it now inherits from IdentifiedObject and thus will have those same attributes.

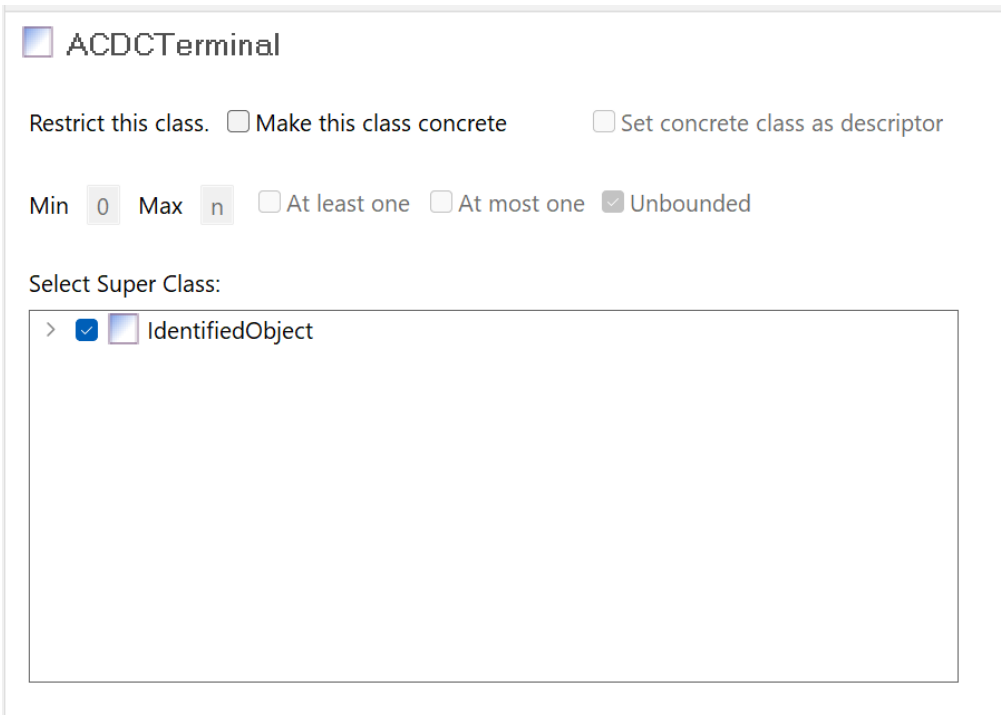


Figure 7-12. ACDCTerminal after adding IdentifiedObject

Although this is far from a full profile, all the pieces are in place to continue adding onto it as needed to create a full Connectivity profile.

Generating RDF Schema

Now that the profile has been defined and the message structure has set, CIMTool will generate the required artifacts. As an RDF schema is desired, one of the RDF builders will be used to create the resulting RDFS file. In addition to that, one of the PlantUML builders will be selected to get a UML view of the profile to give a visual and confirm that it looks as expected.

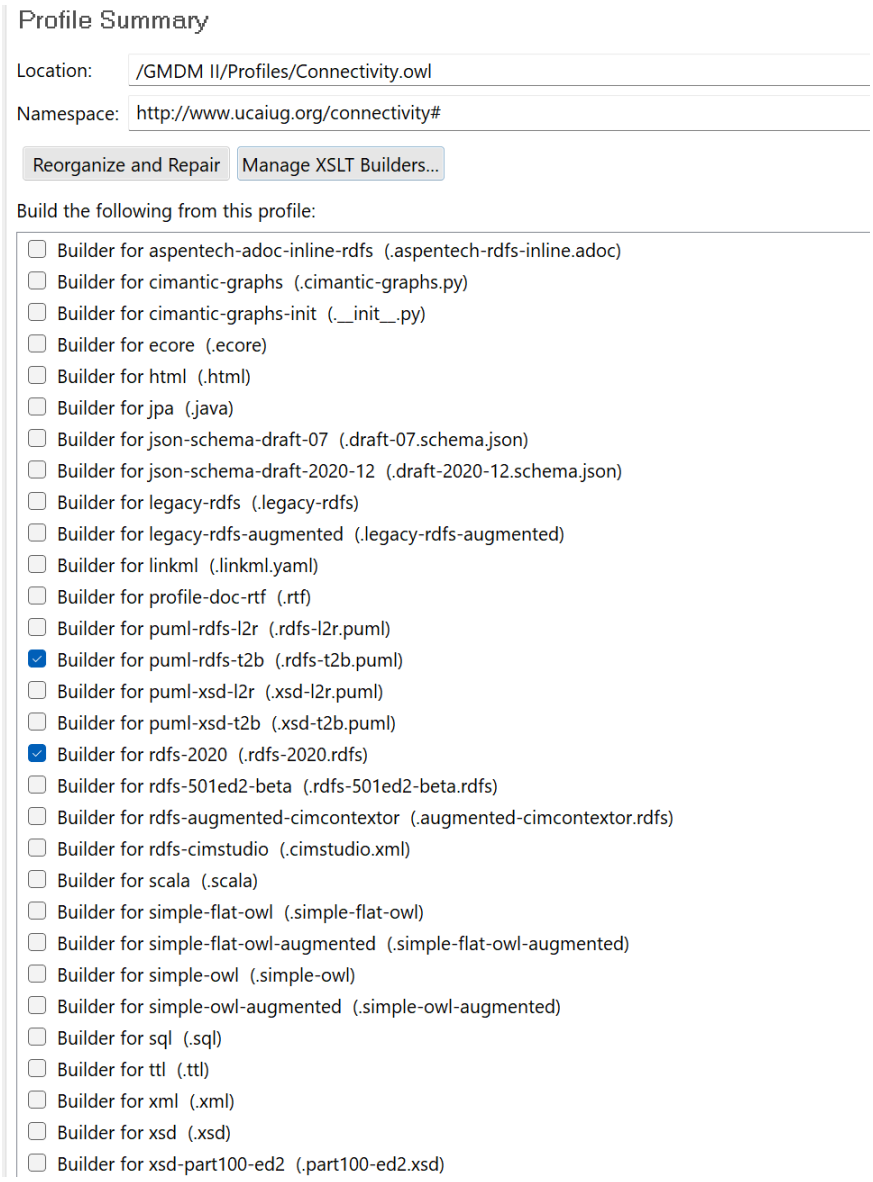


Figure 7-13. Selecting RDFS builders

As seen in Figure 7-13, there are many builders to choose from. For the RDFS itself, rdfs-2020 will give the most authentic modern version of CIM RDF. As for the PlantUML builder, RDFS and XSDs each have their own separate builders, with one marked l2r (right-to-left) and the other t2b (top-to-bottom). After selecting those and saving the profile, the artifacts will be created and shown in the Project Explorer. Before diving into the RDFS, first examine the profile's UML and verify everything looks as expected.

Right-click *Connectivity.rdfs-t2b.puml* and select the option to view it in the text editor. This should open a window with the text view of the UML, and the graphic view on the bottom of the screen.

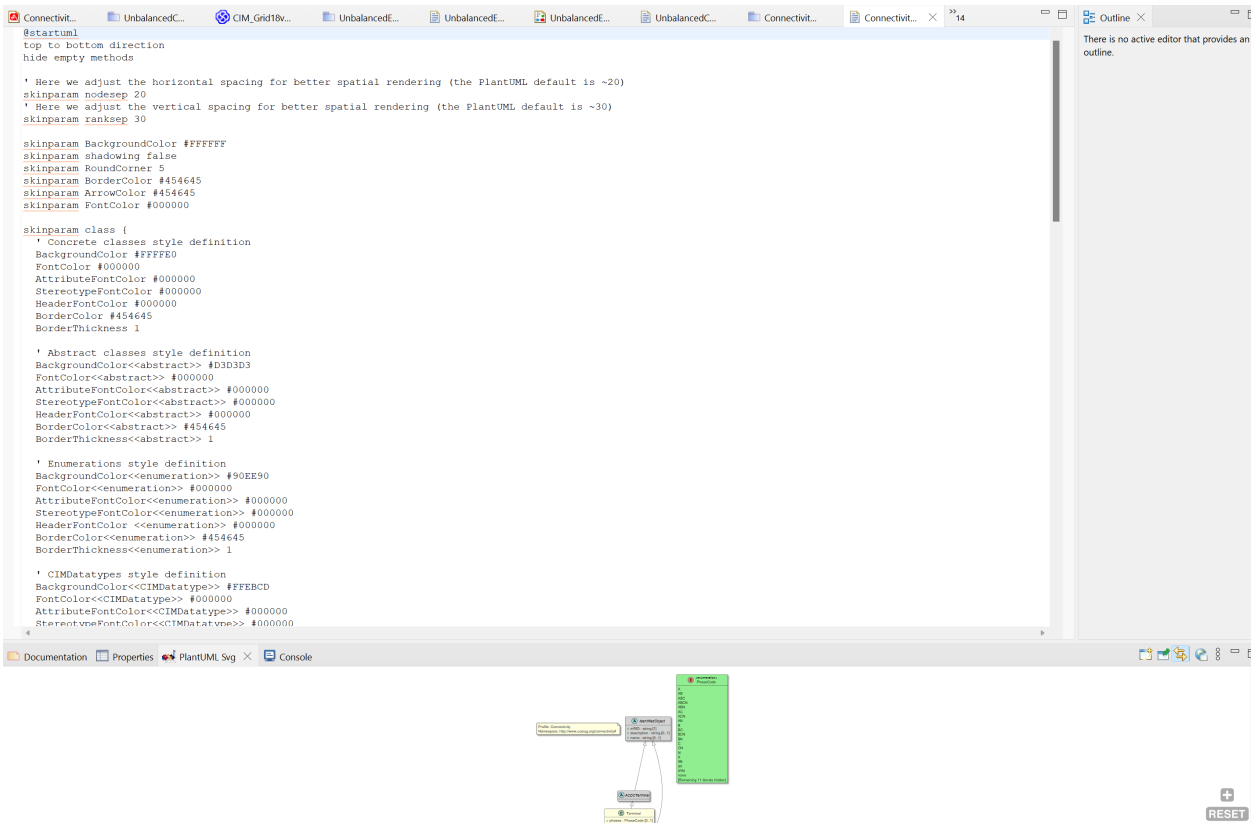


Figure 7-14. CIMTool's PlantUML editor

On the bottom right-hand side of the screen, select the globe icon to open this UML diagram in the web browser instead, and it should then open in the default web browser for the PC. The UML of the profile described earlier should be shown.

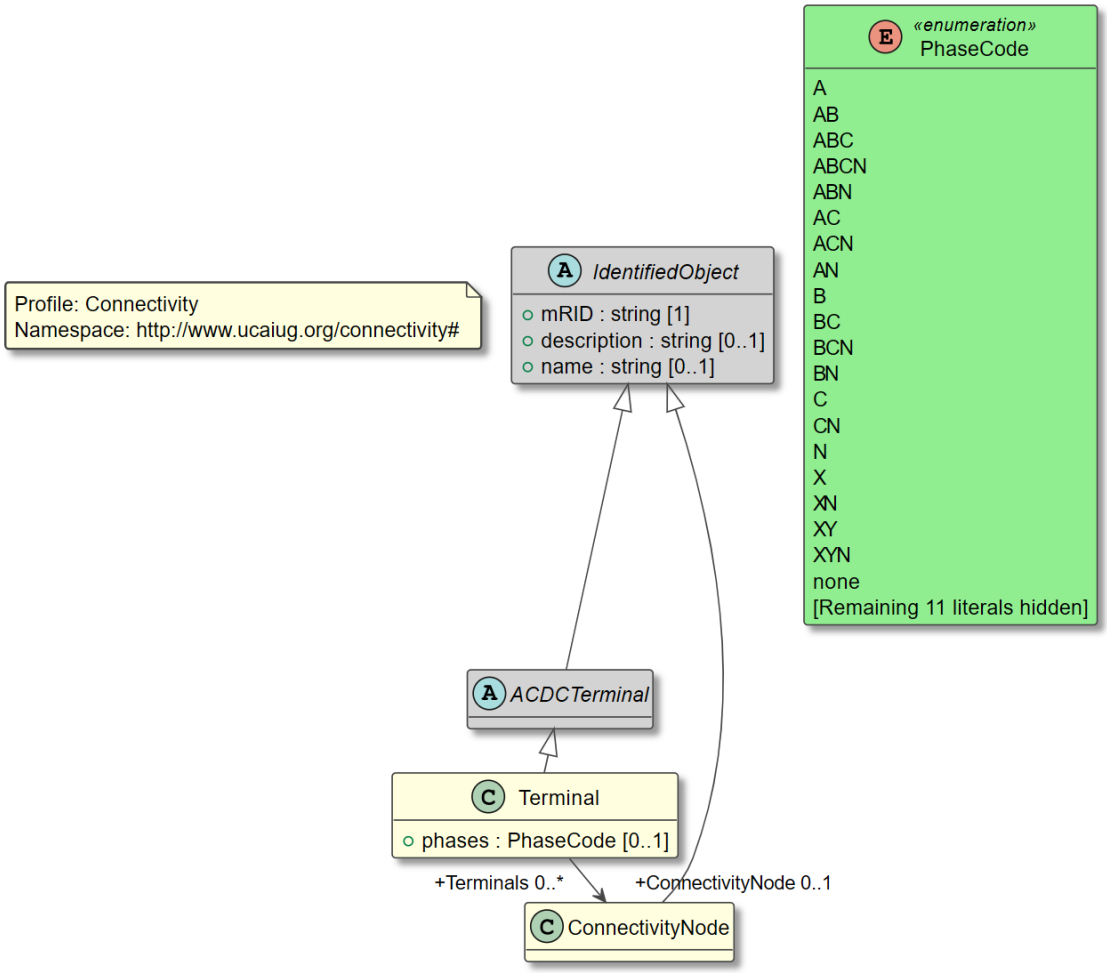


Figure 7-15. PlantUML in browser view

As seen in Figure 7-15, the correct classes are marked as concrete versus abstract and additionally, there are no red association lines, which would indicate an error in CIMTool. Everything looks great so far and now more classes can be added to the profile. With each change, simply save the profile to regenerate new PlantUML and RDFS, and repeat previous steps as needed until satisfied with the resulting profile.

To show a finished profile example, below is a zoomed-out picture of the UML for the *UnbalancedConnectivity* profile.

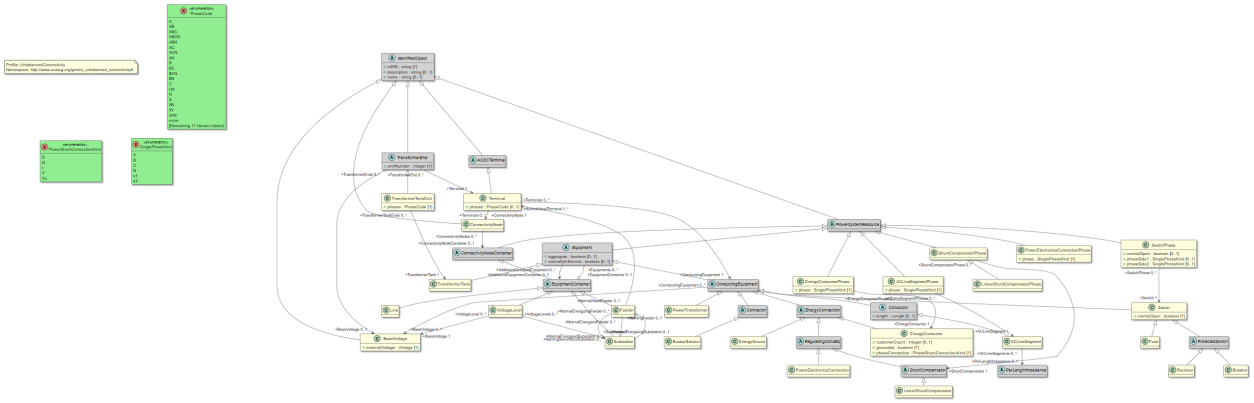


Figure 7-16. Zoomed out view of UnbalancedConnectivity profile

RDFS Creation Summary

This example has shown how to create an RDF schema typically used for grid modeling. Real-world examples of this can contain thousands of objects and instance data so large that FTP is the preferred mechanism for transport. The format of these RDF messages in the CIM typically follows an RDF schema, and these can be generated in CIMTool in a manner that has some overlap with XSD creation but is different in ways that warrant a dedicated section describing its implementation. The CIM here provides a common semantic model, which provides the consensus on the interpretation of each class and attribute, removing ambiguity and duplication of definitions.

Case Study

As part of the CIM University course, Jeff Kimble has learned the details about how the CIM can support network model exchange. This is important as a utility models the state of the network and how it is configured. While the location and types of assets may not change frequently, the settings and configuration of the network is critical to having a proper understanding of how the network will perform. Often this information is communicated between utilities and the entities that are responsible for managing the interoperation of different regions, such as an Independent System Operator (ISO) or Regional Transmission Operator (RTO). This information may also be important for energy market operations (as was seen in this section with ENTSO-E).

However, Jeff learned that while the RDF schema can be very verbose, it is very powerful, flexible, and extensible. However, the drawback with exchanging an entire network model is that the size of the model can be difficult to manage. Due to the challenges with model size, the CIM specifications also include guidance on how to send a *difference model* that contains only the information that describes the changes between prior and current versions of the model. This ability to send only the changes between model versions has a tremendous impact on the size of the file that may need to be exchanged.

Section 7 Questions

1. A simple way to think of serialization is that this is how data is represented in:
    a. On a serial port
    b. In a parallel file
    c. On the front of a Wheaties box
    d. In a file, or transmitted on the wire
2. The CIM difference model is used to:
    a. Describes the differences between two classes
    b. Describes the changes between two CIM RDF XML files
    c. Describes the changes between two CIM UML files
    d. Describe the difference between two message headers
3. While RDF can be very verbose, one of the benefits of this format is:
    a. The files are very large
    b. Additional elements can be added without 'breaking' parsers that rely on fixed formats
    c. The files are much smaller
    d. Unlike XML, RDF is self-describing
4. For model exchange the abstract *Model* class and attributes are important because:
    a. It uniquely identifies the model and reflects its state
    b. It uses long hand syntax
    c. The model class doesn't concern itself with trivial information such as versions
    d. The model class provides for compression
5. The difference model provides a mechanism to allow _____ to a network model.
    a. Additions only
    b. Deletions only
    c. Additions and Deletions
    d. Additions, Deletions, and Updates

8. CIM XSD MESSAGES

Learning Objectives

- Understand how to map application interfaces using the CIM
- Understand how to create an XSD-based profile from the CIM using CIMTool

Mapping Application Interfaces to CIM

As well as exchanging full power system model data as CIM RDF XML, the other main application of the CIM is as a common semantic model for enterprise application integration. Within utilities there are an increasing number of applications that must communicate with each other. At most utilities, this often results in a large number of point-to-point interfaces using custom formats and protocols to exchange data between software applications from a diverse collection of vendors. Adding a new application to the system requires additional interfaces to be defined and implemented, further increasing the complexity of the overall system. This results in an application ecosystem that is often referred to as "brittle". This is because a change to one system's interfaces may break the interfaces with another system. Additionally, the Total Cost of Ownership (TCO) of such a system is higher as each system has its own data definitions that must be mapped and managed for every interface to another system.

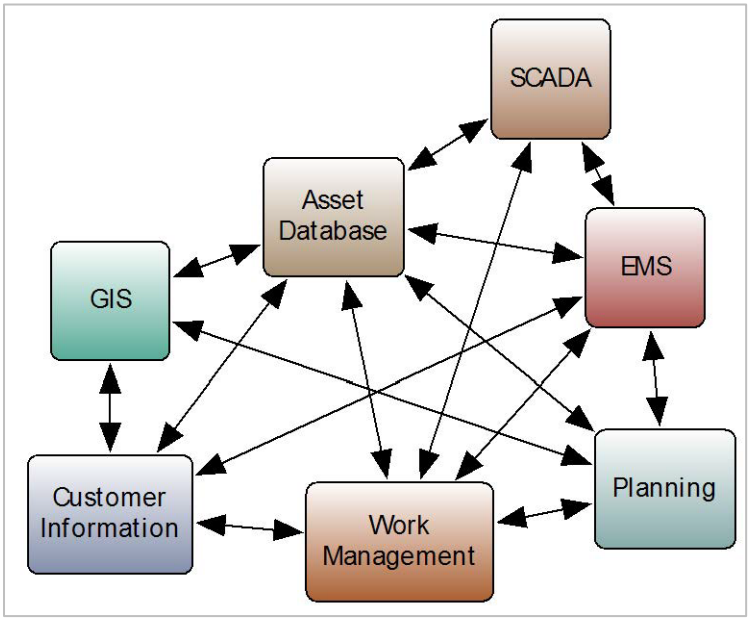


Figure 8-1. Communications between enterprise applications

As illustrated in Figure 8-1, even for a small section of the overall enterprise system, this can result in a large number of inter-application communication links. As companies expand their ICT (Information Communication Technology) infrastructure or replace existing applications with products from other vendors they must define new interfaces for each communication link, a process that is time consuming and expensive.

Enterprise Service Bus

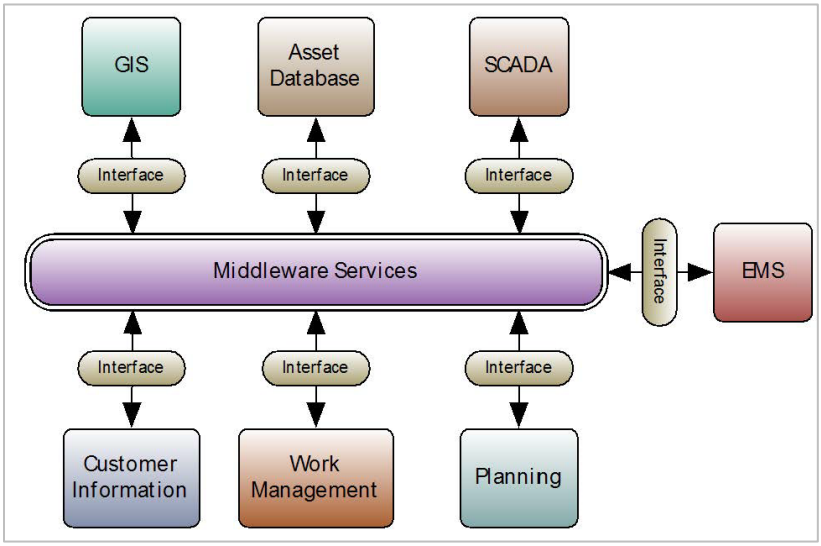


Figure 8-2. Enterprise Service Bus model for inter-application communication

As shown in Figure 8-2, middleware services replace these dedicated links with a single enterprise service bus (ESB). Using middleware services, this provides a mechanism for applications to communicate using a pre-defined message format and requires only a single interface to be written for each application.

For utilities, the CIM provides the common semantic model that can be used to construct messages for communication between the applications[39]. This requires each application to map its external interface to the CIM class structure allowing the inter-application messages to be defined in the CIM.

These messages, in XML format, use a restricted CIM XML Schema to define the payloads of the messages. This takes the standard CIM Schema, itself created from the CIM UML class structure and restricts the multiplicity of associations and required attributes and introduces a strict message hierarchy. Again, this is where the profile concept it useful; the profile is a subset of the model that is typically more restrictive, to facilitate application interoperability.

[39] For further information, see "The Root of All Data Exchange – Common Information Model," IEEE Power & Energy Magazine, Volume 14, Number 1, January/February 2016 (the entire issue is dedicated to CIM).

Mapping CIM Profiles to XML Schema

The IEC has issued the 62361-100 standard for *CIM profiles to XML schema mapping*[40] to ensure effective and consistent mapping of the CIM UML into XML Schema messages. These standards follow the United Nations/Center for Trade Facilitation and E-Business (UN/CEFACT) rules for deriving profiles from an information model.

In essence the 62361-100 standard defines how a UML element can be mapped into XML Schema, how references are defined (containment or by reference), when an attribute becomes a child element or XML attribute, and so on. This removes ambiguity for deriving schema and ensures multiple users and vendors will produce compatible schema. This international standard was issued in 2016 and the reader is urged to refer to the IEC document for more explicit details on how the CIM UML maps to XML Schema.

A number of free, open-source and commercial tools are available to automate the mapping of CIM profiles to XML. CIM tools most often mentioned in responses to a CIM survey in 2016 included the following:

- CIMTool by Langdale Consultants
- CIMSpy and CIMdesk by Power Info LLC
- CIMConteXtor by Zamiren
- Affirma, formerly CIM EA by Xtensible Solutions
- Cimphony by Open Grid Systems
- MODSARUS by EDF R&D

More information on these tools is provided later; but for purposes of illustration, CIMTool will be used here to provide an example of how to derive an XSD.

Existing Application Data

As a simple example, an EMS application's external interface may provide the user with access to data on transformers within the system. The EMS application's interface attributes for a transformer are:

- TRANS_NAME – The Transformer's name
- WINDINGA_R – The Transformer's primary winding resistance
- WINDINGA_X – The Transformer's primary winding reactance
- WINDINGB_R – The Transformer's secondary winding resistance
- WINDINGB_X – The Transformer's secondary winding reactance
- WINDINGA_V – The Transformer's primary winding voltage

[40] IEC 62361-100: Power systems management and associated information exchange – Interoperability in the long term – Part 100: CIM profiles to XML schema mapping.

- WINDINGB_V – The Transformer's secondary winding voltage

Each of these attributes can be mapped to a corresponding attribute within a CIM class, resulting in an interface to CIM mapping.

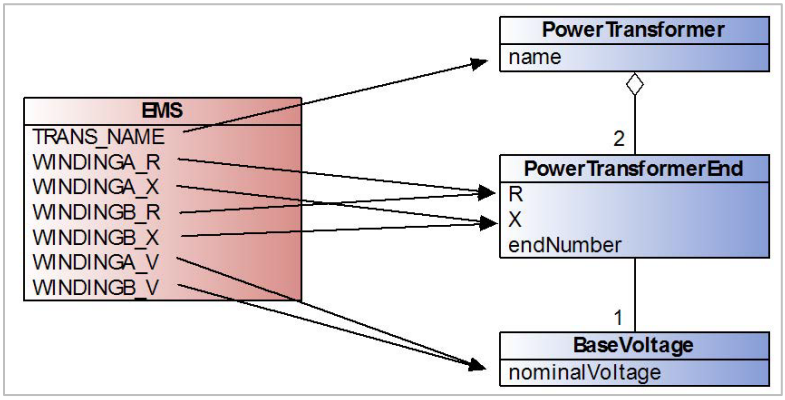


Figure 8-3. CIM Interface Mapping

This mapping, shown in Figure 8-3, highlights that although the two windings have separate names in the interface, they map to the same attributes within the CIM class structure. The aggregation relationship between the *PowerTransformer* and *PowerTranformerEnd* class has, however, been changed from a 0..n multiplicity to 2 (since in this example the EMS represents all transformers as having two windings). This means that there must be two instances of the *PowerTransformerEnd* class present in the message, with the *endNumber* attribute then used to differentiate between the primary and secondary windings.

The voltage for each winding is contained with the *nominalVoltage* attribute of the *BaseVoltage class*. The *BaseVoltage* instance is associated with the *PowerTransformerEnd* directly. The multiplicity is thus 1 on the *PowerTransformerEnd-BaseVoltage* association.

This small CIM message can now be converted into an XML Schema and used to define the interface.

XSD Definition in CIMTool

Defining the Message Structure

CIMTool is an open source[41] tool that supports the definition of CIM profiles and the resulting XML Schema and RDF Schema artifacts. It is available as a standalone application and as an Eclipse plug-in. For this example CIMTool v1.9.3 is used with CIM v15.

Once a CIMTool Project has been created using the CIM15 schema a new profile can be created that will define the subset of the CIM that will be used in the *Transformer Data* interface.

[41] Licensed under the GNU Lesser General Public License v2.1.

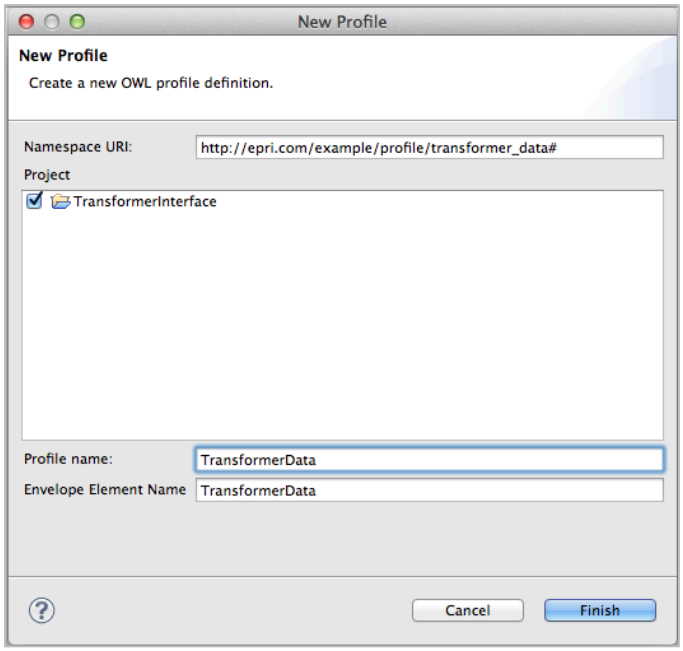


Figure 8-4. Creating a new CIMTool profile

In Figure 8-4 is an example of a New Profile dialog box. Each profile has a namespace Unique Resource Identifier which in this case we have set as *http://epri.com/example/profile/transformer_data#*. The *Profile Name* is used to name the profile within the workspace and the *Envelope Element Name* defines the root of the message itself.

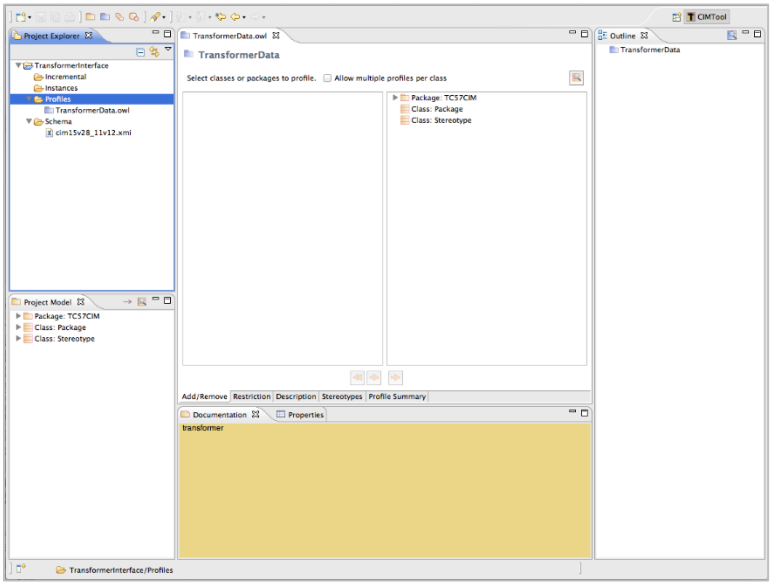


Figure 8-5. Empty Profile

This produces an empty profile called *TransformerData* as shown in Figure 8-5, which can be populated with classes taken from the full CIM UML model. To do this the CIM package hierarchy is expanded within the *Add/Remove* window's right-hand column and the class to be added (in this case *PowerTransformer* is located). Alternatively, CIMTool's search functionality can be used to search for the class by name.

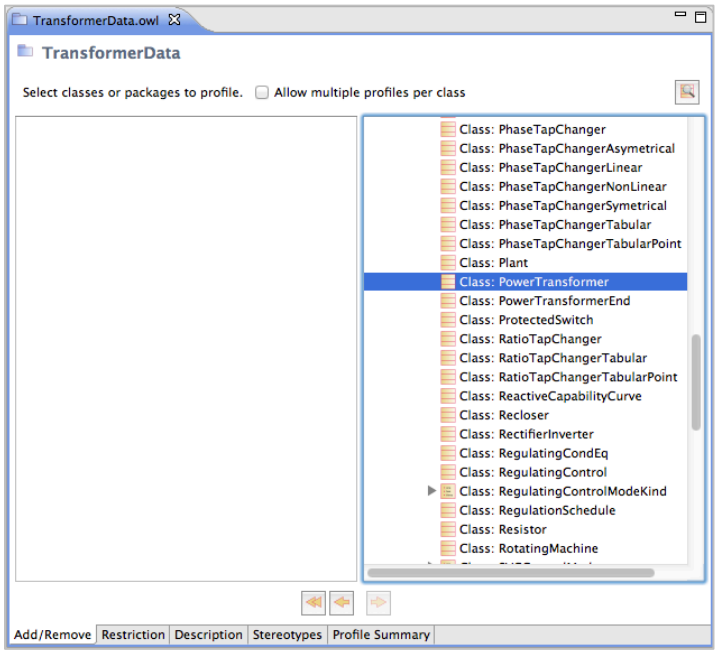


Figure 8-6. Selecting a class to be added to the profile

When the class is located it can be added by clicking the *left arrow* icon at the bottom of the window. This adds the class to the message as shown in Figure 8-7 below.

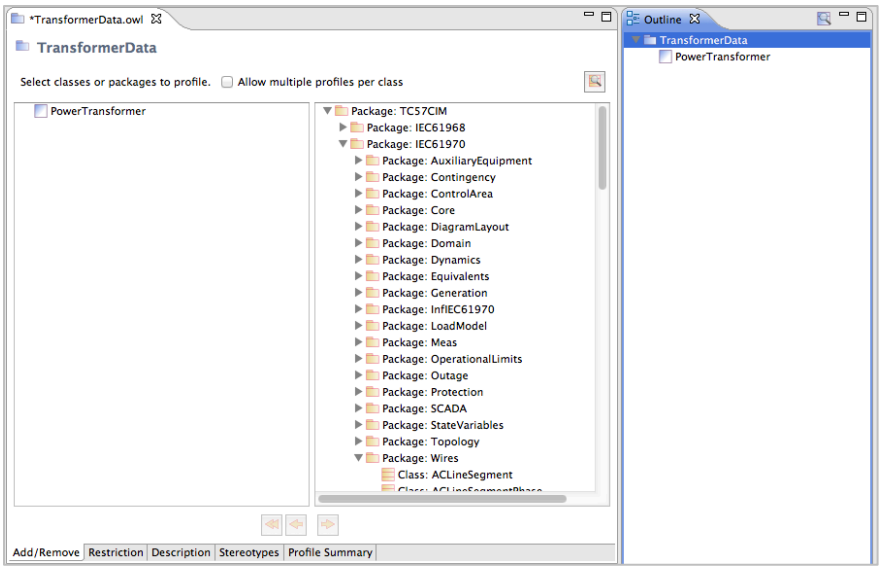


Figure 8-7. Profile with PowerTransformer class

The Outline view is now updated showing the single class within the message below the root *TransformerData* element. At this stage however the *PowerTransformer* is an abstract class so it must be marked as concrete.

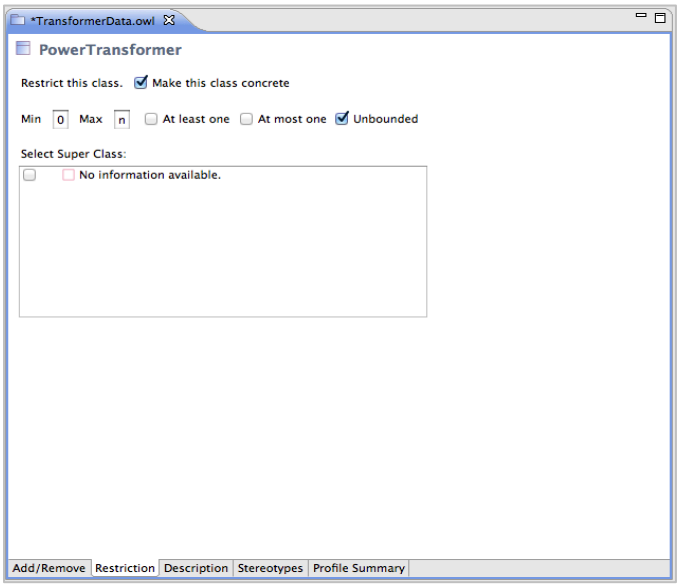


Figure 8-8. Marking the PowerTransformer class as concrete

As shown in Figure 8-8, this is accomplished by selecting the *Make this class concrete* option in the *Restriction* options page for the *PowerTransformer* class. Child classes can now be added for *PowerTransformer* which means that the *PowerTransformerEnd* association is to be included in *PowerTransformer* and then include the *PowerTransformerEnd* class itself in the message. These two steps are shown in Figure 8-9 and Figure 8-10 below.

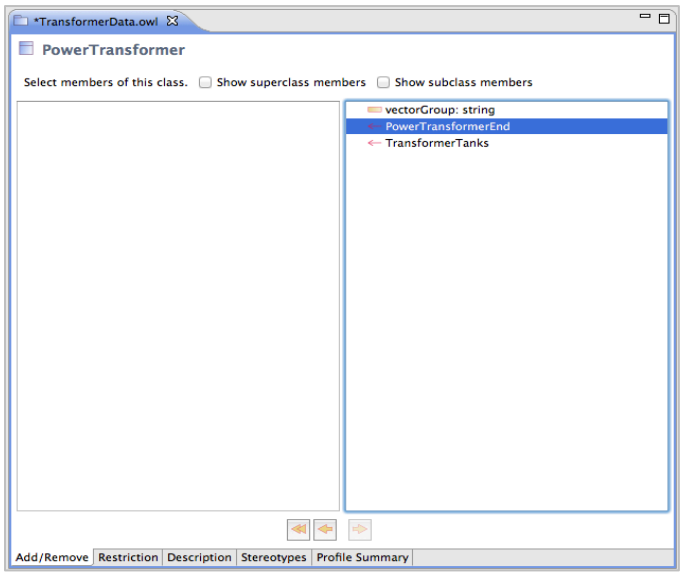


Figure 8-9. Add PowerTransformerEnd association for PowerTransformer class

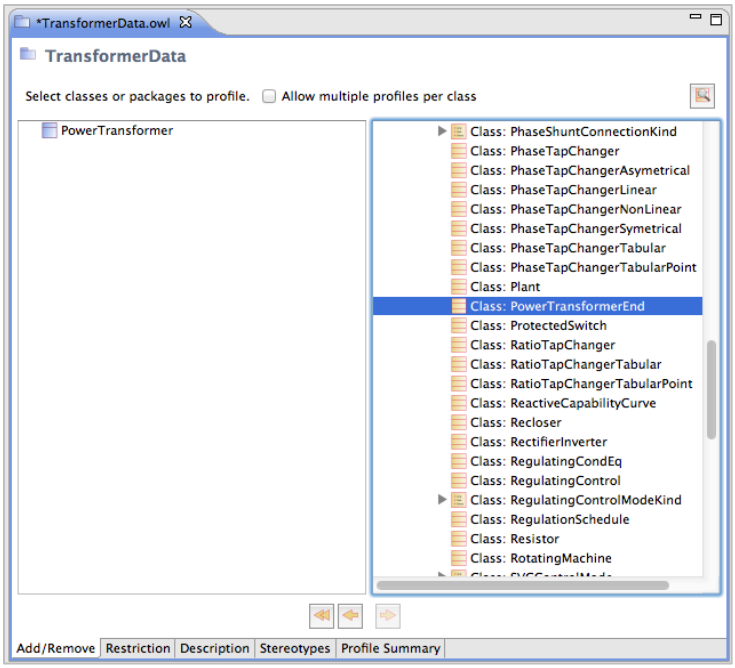


Figure 8-10. Add PowerTransformerEnd class to message

The *PowerTransformerEnd* association on the *PowerTransformer* class must now be told to use the *PowerTransformerEnd* class that has been added. This can be accomplished by selecting the *PowerTransformerEnd* association and marking the *Associated Class* to be the *PowerTransformerEnd*. This is done via the *Restriction* option page as shown below in Figure 8-11.

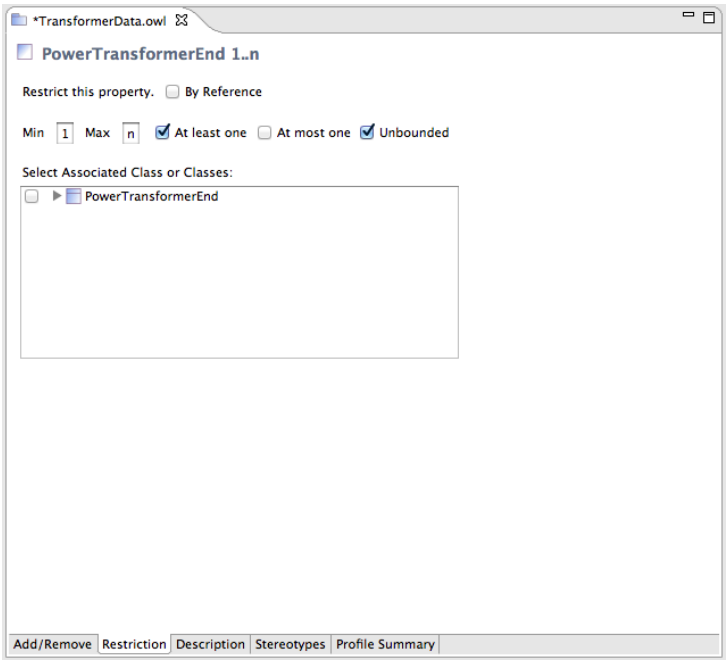


Figure 8-11. Select associated class for PowerTransformerEnd association

The *PowerTransformerEnd* class itself has attributes and so the native attributes are selected that are to be added, in this the *r* and *x* for the resistance and reactance, and some additional attributes for will also be added for susceptance, conductance, the winding connection type and rated voltage.

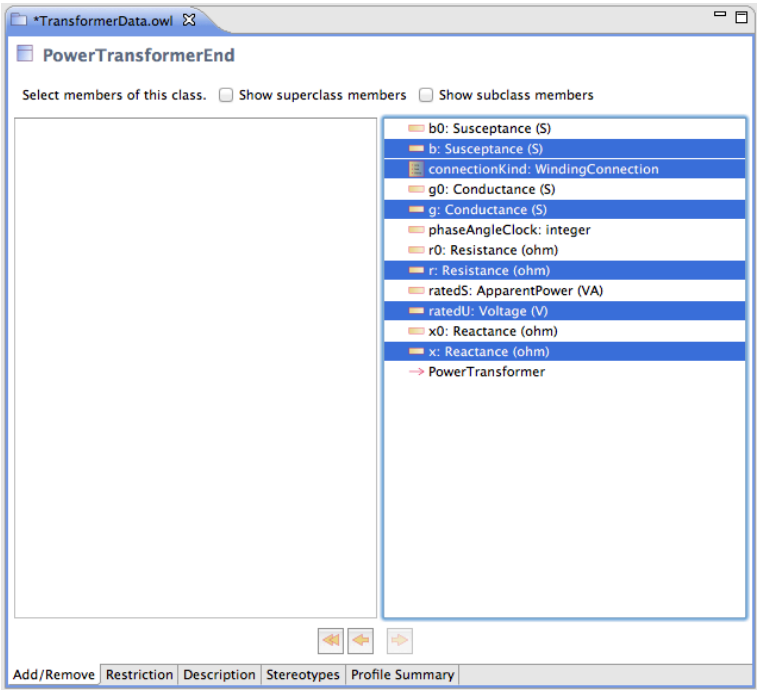


Figure 8-12. PowerTransformerEnd Native Attributes

In addition to these native attributes the profile requires the *endNumber* attribute that is within the *PowerTransformerEnd*'s superclass, *TransformerEnd*. To add this the *Show superclass members* option is selected and then add in the *endNumber* attribute as shown in Figure 8-13.

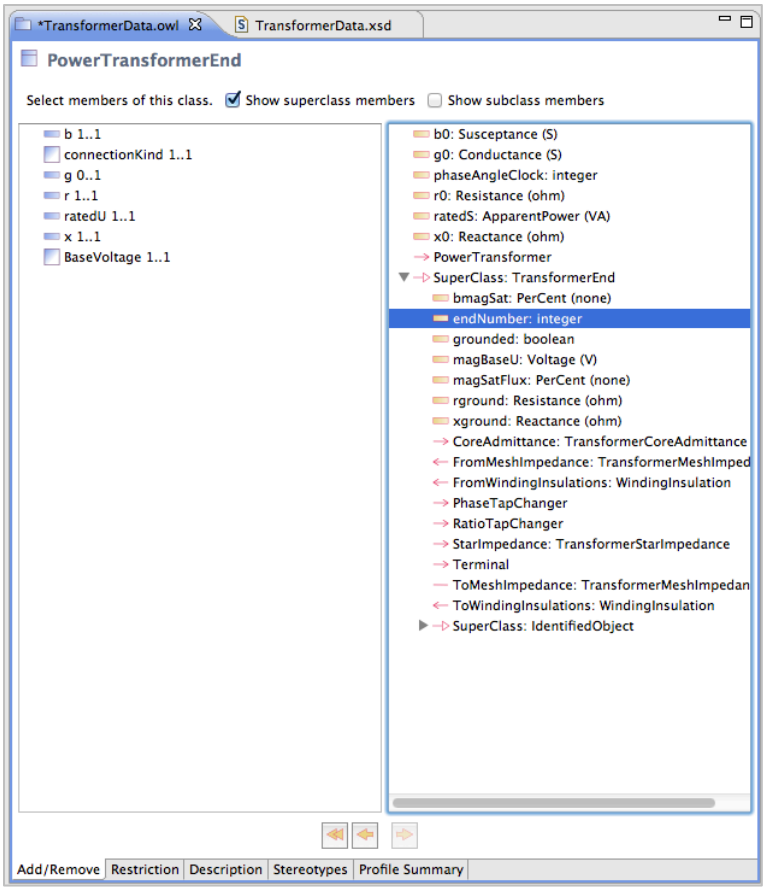


Figure 8-13. Add inherited attribute from TransformerEnd

Not all of the attributes that have been added are mandatory, so some attributes such as the conductance may be marked as being optional by changing their cardinality from 1..1 to 0..1 by deselecting the *At least one* option as shown in Figure 8-14. This allows all attributes for the *r* and *x* in the *PowerTransformerEnd* class to be marked optional.

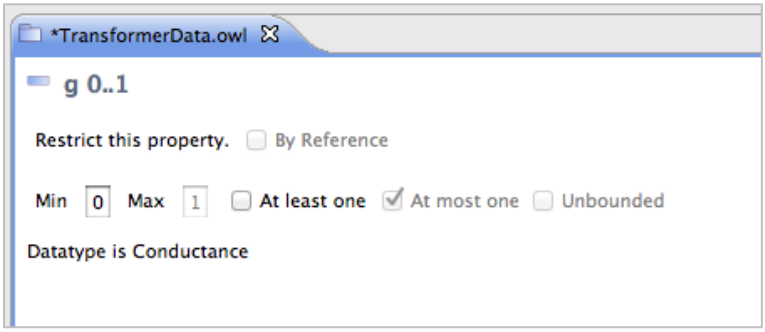


Figure 8-14. Optional PowerTransformerEnd attributes

As well as the *PowerTranformer* and *PowerTransformer* class, the last class required is the *BaseVoltage.* This is added using the same process as the *PowerTransformer* and *PowerTransformerEnd*, locating it within the CIM UML and adding it to the message. For the *BaseVoltage* the only attribute required is the *nominalVoltage* which is marked as mandatory.

In addition, to complete the message also add the *name* and *mRID* attributes to *PowerTransformer* which are inherited from its *IdentifiedObject* parent class and the cardinality on the *PowerTransformerEnd* association is changed from 1..n to 2..2, thus restricting it to having exactly two child entries. The *WindingConnection* enumeration is included to allow the type of winding connection (Delta, Wye, and so on) to be specified. The resulting message outline is shown in Figure 8-15 below.

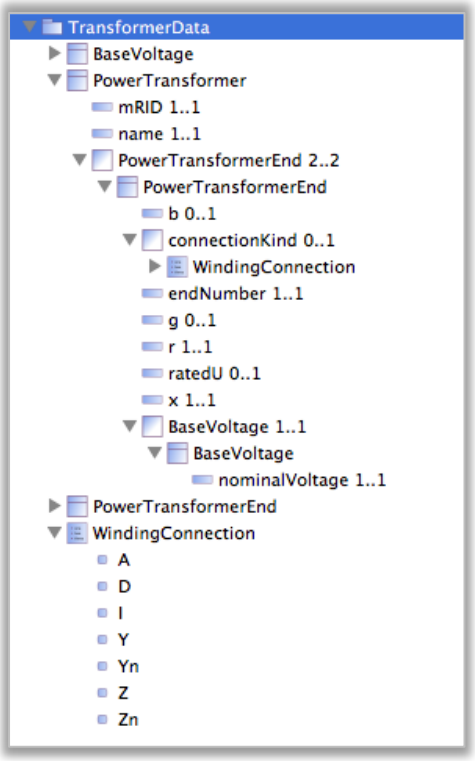


Figure 8-15. TransformerData Message Outline

Generating XML Schema

Now the profile has been defined and the message structure has been defined, CIMTool will generate the required artifacts. As an XML Schema is desired, the XSD builder is enabled in the profile options as shown in Figure 8-16, which will create an XSD for the defined message.

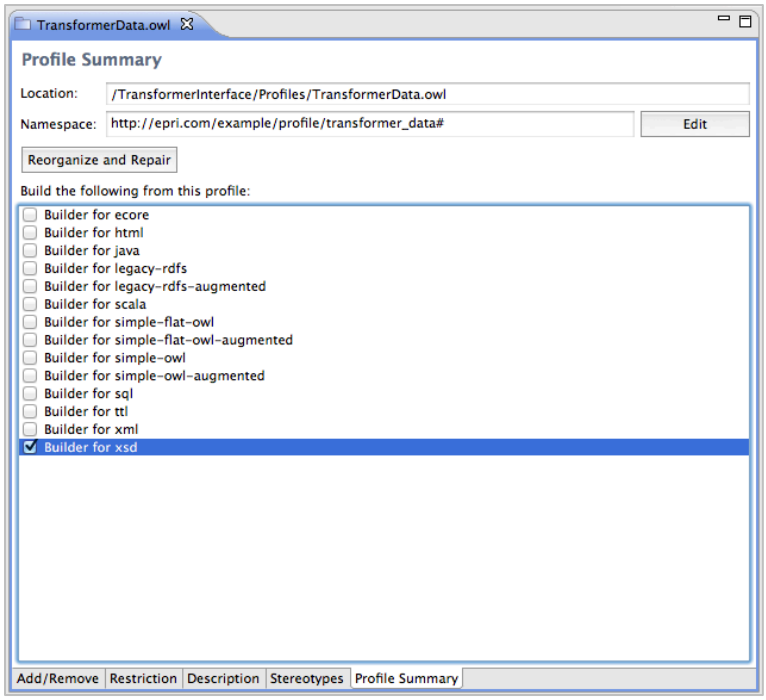


Figure 8-16. Enable XSD Builder in CIMTool

The generated XSD can be viewed both in its raw XML form or using the Eclipse XML Schema viewer. The resulting XSD when viewed in the schema viewer is shown in Figures 8-17 and 8-20 below:

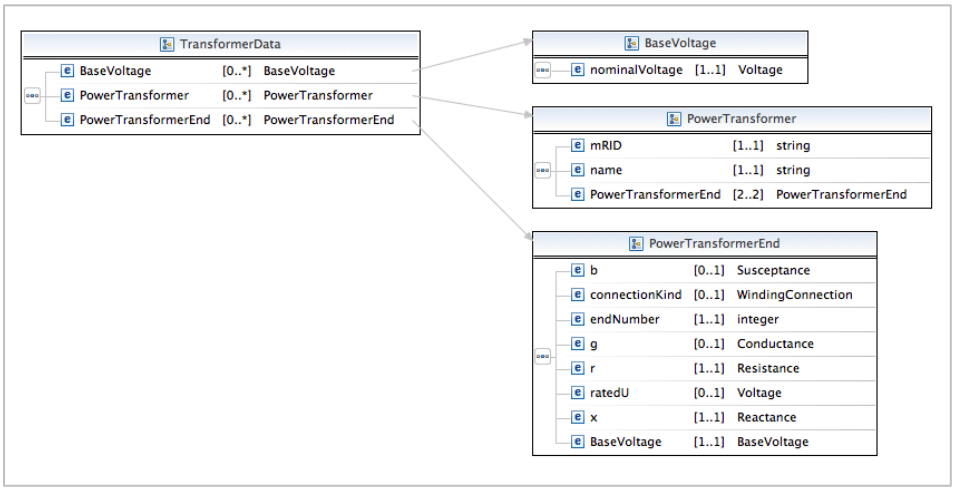


Figure 8-17. TransformerData XSD Schema Element

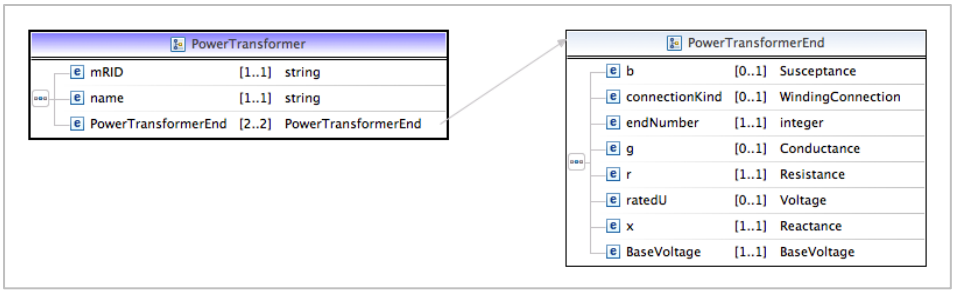


Figure 8-18. PowerTransformer XML Schema Element

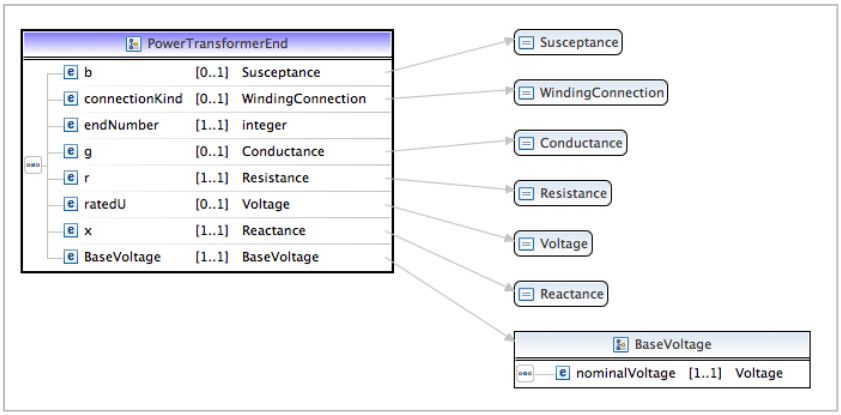


Figure 8-19. PowerTransformerEnd XML Schema Element

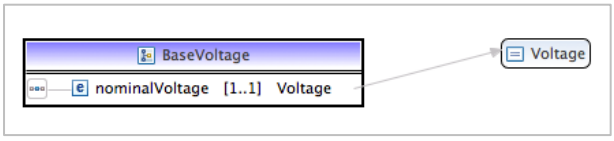


Figure 8-20. BaseVoltage XML Schema Element

The XML Schema itself is also viewable as shown in Figure 8-21. This conforms to the XML Schema standard and includes the documentation from the UML that is *pulled* through into the XML Schema documentation itself, allowing complete traceability from the CIM UML to the XML Schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:a="http://langdale.com.au/2005/Message#" xmlns:sawsdl="http://www.w3
<xs:annotation/>
<xs:element name="TransformerData" type="m:TransformerData"/>
<xs:complexType name="TransformerData">
<xs:sequence>
<xs:element name="BaseVoltage" type="m:BaseVoltage" minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="PowerTransformer" type="m:PowerTransformer" minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="PowerTransformerEnd" type="m:PowerTransformerEnd" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="BaseVoltage" sawsdl:modelReference="http://iec.ch/TC57/2010/CIM-schema-cim15#BaseVoltage">
<xs:annotation>
<xs:documentation>Defines a system base voltage which is referenced.</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:element name="nominalVoltage" minOccurs="1" maxOccurs="1" type="m:Voltage" sawsdl:modelReference="http://iec.ch/TC57/2010/CIM
<xs:annotation>
<xs:documentation>The PowerSystemResource's base voltage.</xs:documentation>
</xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>
<xs:complexType name="PowerTransformer" sawsdl:modelReference="http://iec.ch/TC57/2010/CIM-schema-cim15#PowerTransformer">
<xs:annotation>
<xs:documentation>An electrical device consisting of  two or more coupled windings, with or without a magnetic core, for introduc
<xs:documentation>A power transformer may be composed of separate transformer tanks that need not be identical.</xs:documentation
<xs:documentation>A power transformer can be modelled with or without tanks and is intended for use in both balanced and unbalanc
</xs:annotation>
<xs:sequence>
<xs:element name="mRID" minOccurs="1" maxOccurs="1" type="xs:string" sawsdl:modelReference="http://iec.ch/TC57/2010/CIM-schema-ci
<xs:annotation>
<xs:documentation>A Model Authority issues mRIDs. Given that each Model Authority has a unique id and this id is part of the mRID
<xs:documentation>Global uniqeness is easily achived by using a UUID for the mRID. It is strongly recommended to do this.</xs:doc
<xs:documentation>For CIMXML data files the mRID is mapped to rdf:ID or rdf:about attributes that identifies CIM object elements.
</xs:annotation>
</xs:element>
<xs:element name="name" minOccurs="1" maxOccurs="1" type="xs:string" sawsdl:modelReference="http://iec.ch/TC57/2010/CIM-schema-ci
<xs:annotation>
<xs:documentation>The name is any free human readable and possibly non unique text naming the object.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="PowerTransformerEnd" minOccurs="2" maxOccurs="2" type="m:PowerTransformerEnd" sawsdl:modelReference="http://iec
<xs:annotation>
```

Figure 8-21. Screenshot of raw XML Schema XML

The schema can be used by a number of applications, frameworks and tools. Users familiar with XMLSpy can view the XSD in a view similar to that shown in Figure 8-22 below.

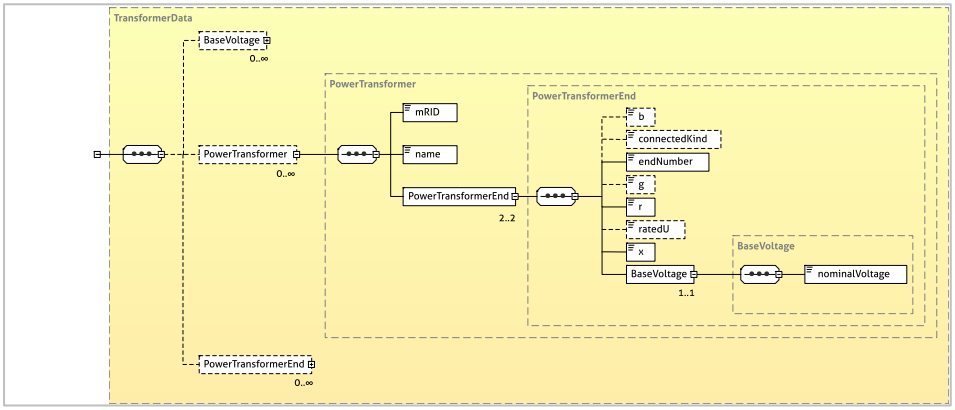


Figure 8-22. XMLSpy style XML Schema view

XML Instance Data

This schema now defines the structure of the data for this small, simple interface. This can be used to define the contents of a service contract (that is, Web Service Description Language – WSDL) so that the sending and receiving systems know the format and structure of the data that is to be exchanged.

Using the schema defined above, the resulting XML instance data would take the form shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<cim:TransformerData
  xmlns:cim="http://epri.com/example/profile/transformer_data#">
 <cim:PowerTransformer>
  <cim:name>Open Grid Transformer</cim:name>
  <cim:mRID>_836f6720-fde1-11e0-a6f7-e80688cf8a29</cim:mRID>
  <cim:PowerTransformerEnd>
   <cim:b>0.02</cim:b>
   <cim:connectionKind>D</cim:connectionKind>
   <cim:r>0.23</cim:r>
   <cim:ratedU>420.0</cim:ratedU>
   <cim:x>0.78</cim:x>
   <cim:BaseVoltage>
   <cim:nominalVoltage>400.0 </cim:nominalVoltage>
   </cim:BaseVoltage>
  </cim:PowerTransformerEnd>
  <cim:PowerTransformerEnd>
   <cim:b>0.03</cim:b>
   <cim:connectionKind>D</cim:connectionKind>
   <cim:r>0.46</cim:r>
   <cim:ratedU>290.0</cim:ratedU>
   <cim:x>0.87</cim:x>
   <cim:BaseVoltage>
   <cim:nominalVoltage>275.0</cim:nominalVoltage>
   </cim:BaseVoltage>
  </cim:PowerTransformerEnd>
 </cim:PowerTransformer>
</cim:TransformerData>
```

The namespace is identical to that of the namespace defined for the original profile and while the structure is similar to that of the CIM RDF XML examples, the most noticeable difference is that the data is hierarchical in nature. For this example only the *PowerTransformer* was given a unique identifier in the form of its *mRID* attribute (in this case a UUID) but the same attribute could have been added to the *PowerTransformerEnd* should there be a need to identify them via their *mRID*.

For the scope of the message, this means that the *PowerTransformer* element contains the entire required data making it simpler to then transform the element into another format if that is required. These messages are aimed at smaller, message-based exchanges rather than the large, bulk data exchange using CIM RDF XML.

XML Messaging Summary

This example has shown how a simple portion of an application's interface can be mapped to the CIM class structure and then used to construct a simple XML message payload using CIMTool. Real-world examples often use tens or even hundreds of elements to construct a message payload. The benefit of this approach is that when every application within the system is mapped to this common model it becomes far simpler to enable applications to communicate. The CIM provides a common semantic model, which provides consensus on the interpretation of each class and attribute, removing ambiguity and duplication of definitions.

Case Study

Jeff Kimble has learned that the CIM model can also be restricted via a profile for enterprise application integration. This is helpful because when he first started it seemed like the CIM had to be used in its entirety which would seem to be unwieldy. It turns out that the CIM is a *reference model*, much like a dictionary is used to lookup words, and the CIM can therefore be used to look up standard meanings and relationships among the classes.

This ability to use the CIM as a reference model has considerable appeal to Jeff. As the head of the integration team he has seen numerous naming conventions for the services and adapters that have both been provided by vendors or developed by his own team or when his company has worked with systems integrators. The amount of time it takes for someone to get up to speed on all the variations is tremendous. Jeff realizes that if they use the CIM as a reference model, as new adapters are adding to the enterprise service bus, this will cut down on the amount of mapping that is required from one interface to another, drastically lowering the maintenance costs. Further, instead of using disparate definitions for meanings on attributes, his team, and the other teams that they work on the behalf of, will all have a single place to go to reference these meanings. In short, the organization will use a common vocabulary which will have a positive impact on requirements gathering when new interfaces are designed.

Section 8 Questions

1. System interfaces are referred to as “brittle” when:
    a. There are consistent standards for how interfaces are designed
    b. There are many interfaces; inconsistently applied, where a change in one could impact a change in another
    c. The interface is made with a less flexible material
    d. The interfaces have been harmonized to eliminate duplication
2. An example of an artifact in the way the CIM refers to it is:
    a. An XSD that can be used for messaging
    b. A file that is used by archeologists
    c. A file that is leftover by the profile definition process
    d. A profile
3. The specification that guides how the CIM UML is mapped into XML Schema messages is:
    a. IEC specification 62361-100
    b. the schema specification
    c. the CIM
    d. XML
4. One way that traceability is supported in the creation of profiles is that:
    a. A GPS locator is attached to the file
    b. Documentation from the CIM is passed to the documentation tag of the resulting XML
    c. A master resource identifier (mRID) is used
    d. A line is created in the resulting file that points to the source
5. A benefit of using a common reference model for developing is that:
    a. The interfaces are more expensive to maintain
    b. The developer has to look up an attribute every time that it is used
    c. It is common, so everyone understands it
    d. It reduces ambiguity, the amount of mapping that needs to be done, and removes the duplication of definition

9. TRANSFORMING DATA TO CIM

Learning Objectives

- Understand how the concept of bi-directional transformation is applied
- Understand the difference between *stateless* and *stateful*

Bi-directional Transformation

Section 8 provided a simple example of how existing internal data structures can be mapped to the CIM. Given that CIM-derived profiles define *interface* standards, not internal data structures, it is very common for existing applications to map their internal data to and from the CIM.

The transformation is one-way when the requirement is only to either import or export data. But for many applications there is a need to both import and export to CIM and to ensure that there is consistency both ways. Consistency checking will ensure that a native data element within the application is always mapped to the same CIM element and vice-versa.

Such bi-directional transformation is essential when an application is to participate in an enterprise environment where it is communicating with other applications only via CIM-based messaging. There are two primary means of implementing such a transformation depending upon the structure and format of the source data: stateless or stateful.

Stateless vs. Stateful Transformation

In a stateless transformation all the information required to go from A to B and from B to A is contained within the data itself or can be automatically calculated /extracted without any external input. For applications with internal data structures that are derived from the CIM, it is likely that transformations to and from the CIM will be stateless as there will be one-to-one mappings between the CIM and internal application data structures.

In a stateful transformation either the source or target does not contain sufficient data to enable a bi-directional transformation without loss of data. In such circumstances any transformation must produce, along with the output, additional *state* data that will allow any subsequent reverse transformation to correctly map into the original source data.

When defining CIM transformations the most common issue that arises in creating a stateless transformation is ensuring that the CIM *mRID* attribute can be mapped into the application's source data. In many legacy systems the unique identity of a piece of equipment or other data element is a simple database table entry key. When mapping a CIM element that is uniquely identified by a 128 bit UUID there may be no field within the application's internal data structure to store this identifier. As such any subsequent re-export of the data back into CIM would not have this same unique identity.

The solution is to store additional mapping *state* data that maps the original identifier to the identifier in the system. Transforms therefore need to know both the source data and any *state* data about previous transformations. This same approach may also be required when multiple components in the CIM maps to a single entry in the application data (or vice-versa), ensuring that the same components are mapped in all transformations.

To the end-user this process should be seamless, but for developers and vendors there is an order of magnitude increase in complexity when writing and maintaining stateful transformations rather than stateless transforms.

Model-Driven Transformation

In Section 2 the concept of distinguishing between structure and format was briefly mentioned, and it is this concept that enables Model Driven Engineering (MDE) and Model Driven Architectures (MDA) to be applied with Model Driven Transformation (MDT). With MDT the transformations are written at the structural level rather than format-to-format.

For example, if the source data is a database and the target data is an XML file with MDT the source and target serialization formats are not known within the transform, it instead defines itself against the *structure* of the data. This enables the same data, defined by the same structure to be serialized in a number of formats, whether it be a database, flat file, XML, network data stream, and so on, as shown in Figure 9-1.

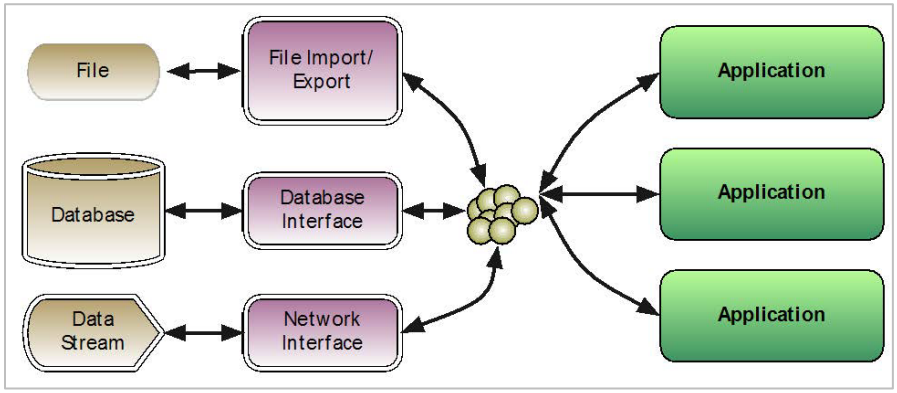


Figure 9-1. Multiple Serialization Formats/Locations

The format-agnostic data *objects* are identical in structure no matter what the source of the data is and so the transform is written to operate against these data objects rather than to read/write to database tables or parse file formats. Applications are written to process the data objects rather than requiring multiple interfaces to deal with the different possible sources. This approach also modularizes the software, separating the parsing of data from a file or reading from a database from the application logic itself.



Figure 9-2. Model Driven Transformation Process

This means the *transformation* is just one part of the overall process with multiple stages as shown in Figure 9-2. The data objects may be defined by a complex inheritance structure like the CIM or be a simple flat structure derived from a file format.

The other advantage of this approach is that data transformation itself is written against the structural elements making its syntax clearer, more concise and easier to read than code design to convert both format and structure simultaneously. An example of a MDT language, QVTO (Query/View/Transform Operational)[42] is shown below:

```
mapping CIM::IEC61970::Wires::ACLineSegment:: aclToBranch() : PF::Branch{
        var bus1 : PF::Bus := self.Terminals->asSequence()->first().
        TopologicalNode.map nodeToBus();
        var bus2 : PF::Bus := self.Terminals->asSequence()->last().
        TopologicalNode.map nodeToBus();
        if (bus1.number < bus2.number) then{
                FromBus := bus1;
                ToBus := bus2;
                MeteredBus := bus1;
        }else{
                FromBus := bus2;
                ToBus := bus1;
                MeteredBus := bus2;
        }endif;

        circuit := self.name.substring(0,2);
        R := self.r.toPerUnit(self.BaseVoltage.nominalVoltage);
        X := self.x.toPerUnit(self.BaseVoltage.nominalVoltage);
        B := self.bch.toPerUnit(self.BaseVoltage.nominalVoltage);
        length := self.length;
}
```

[42] QVT (Operational) is an Object Management Group specification for model to model transformation with an open-source Eclipse implementation available that runs on top of Java. The OMG specification can be found at https://www.omg.org/spec/QVT/.

This example code is for converting CIM *ACLineSegment* to a proprietary power-flow application's *Branch*. The mapping function defines its input using the full CIM package structure and the class for its output is *Branch* under the *PF* package. The attributes *R*, *X*, and *B* in the power-flow format are assigned by taking the corresponding CIM values *r*, *x*, and *bch* attributes then passing them through a helper function that converts them to per-unit using their value and that of the line segment's voltage (obtained via the segment's *BaseVoltage* association).

Mapping functions may call other mapping functions so that, in this example, the *From* and *To* buses are assigned by taking the first and last *Terminal* on the *ACLineSegment*, finding their *TopologicalNode*'s then mapping them to the power-flow format's *Bus* using another mapping function, *nodeToBus().*

What should be taken from this example is that the code is mapping the data based on its structure and relationships. At this level it is not known if the CIM data is coming from RDF XML, a CIM XSD Message, a database or even a custom CIM binary format. Additionally, nothing is known about the power-flow application's file format or whether the resulting data is intended to be written to a file, sent across a network or be used with an API.

Example: CIM to PSS/E

An example of how MDT can be used is taking a real-world example of translating CIM RDF XML data into PSS/E RAW[43] format[44] and looks at the high-level process involved in creating a CIM to PSS/E transform using MDT. PSS/E RAW is a space/comma delimited ASCII file format for bus-branch, balanced network data and, unlike the CIM, is not derived from an overall UML model (or if it is, the UML is not publicly available). As such, to create a CIM to PSS/E converter using MDT a number of steps are involved:

1. The PSS/E data structure must be derived to create a PSS/E meta-model
2. A parser for PSS/E RAW files must be written to create the format-agnostic data objects
3. A transform between the CIM classes and the PSS/E meta-model elements
4. A CIM to RDF XML serialization interface must be written to create the CIM RDF XML from the CIM data objects.

It is assumed that the CIM meta-model will be the UML (or a subset of the UML from the appropriate profile group).

[43] PSS/E is a power system analysis tool from Siemens PTI that is widely used in the power industry. The PSS/E RAW format is its native file format.

[44] A.W. McMorran, G.W. Ault, I.M. Elders, C.E.T. Foote, G.M. Burt, J.R. McDonald, "Translating CIM XML Power System Data to a Proprietary Format for System Simulation", IEEE Transactions on Power Systems, February 2004, Volume 19, Number 1, pp. 229-235.

Defining the PSS/E Meta-Model

Defining the PSS/E meta-model involves a reverse engineering of the format to determine the elements, attribute and associations. While some formats may provide a mechanism to automatically determine this meta-data (that is, extracting a database schema) with PSS/E the definition is primarily a manual task based on the comments and meta-data within the instance files (and if available, any documentation). With PSS/E there are a number of complex elements including: Bus, Branch, Transformer, Load, Generator and Shunt, as well as containment elements Area and Zone and the Owner element.

Unlike CIM there is no explicit inheritance so the direct mirror of the file format is to have a non-hierarchical class structure. Often during the creation of a meta-model, however, patterns emerge between classes (that is, multiple classes have a single association to a *Bus* or contain a *name* or *ID* attribute) and so a class hierarchy may be introduced to make the meta-model more elegant and remove duplication.

Associations in PSS/E are defined as index references to other classes (that is, a *Load* has a *Bus* index 1001 which corresponds to a *Bus* entry with *ID* 1001). In the meta-model these are instead defined as associations so that when the data is read into the format-agnostic objects the index references are resolved and become direct, in-memory associations (that is, pointers).

Parsing/Writing PSS/E RAW

The parsing of the files into memory is a case of identifying the format of the files then writing software that will read this format, creating the appropriate in-memory data object instances for each element from the processed data. Since PSS/E is an ASCII format a parser can be written in most programming languages.

The processing is generally a two-step process, creating all the objects and their attributes then resolving all the index references (since some elements refer to elements that are at the end of the file). Processing even large network models in this format can be accomplished very quickly.

Writing PSS/E data involves the processing of the in-memory data objects and serializing them to a file in the correct order and format. When the structure is derived from the file format serialization can be a relatively straightforward process; looping through each element and outputting the attributes in order.

More complex serialization rules are required to deal with associations (that is, knowing the correct index attribute to refer to) or dealing with fixed enumerations where the meta-model has human-readable names such as *OutOfService* or *InService* where the file is expecting the enumeration value of 0 or 1.

Transforming CIM to PSS/E

With the PSS/E meta-model defined and the CIM meta-model taken from the UML the transform is written against these meta-models. In some cases, there is simple one-to-one

mapping, but in others more complex logic is required. Some examples of the transformation logic are shown below in QVTO (with well documented code to explain what is happening).

CIM EnergyConsumer to PSS/E Load

A relatively straightforward one-to-one mapping is that of the CIM's EnergyConsumer to PSS/E Load as shown below.

```
Mapping CIM::IEC61970::Wires::EnergyConsumer:: getLoad() : PSSE::Load {
-- Set the Bus to be the first (and only)
 -- Terminal's Topological Node mapped to a PSSE:Bus
 result.Bus := self.Terminals->first().TopologicalNode.map getBus();
 result.id := self.name;

 var p : Real := 0.0;
 var q : Real := 0.0;

 -- If the EnergyConsumer's Terminal has an SvPowerFlow associated
 -- with it then we use the values from it, otherwise we use the fixed
 -- real and reactive power values for the load
 if (self.Terminals->first().SvPowerFlow <> null) then{
  p := self.Terminals->first().SvPowerFlow.p;
  q := self.Terminals->first().SvPowerFlow.q;
 }else{
  if (self.isSet("pfixed")) then{ p := self.pfixed;}endif;
  if (self.isSet("qfixed")) then{ q := self.qfixed;}endif;
 }endif;

 -- If a LoadResponse is set then this load will have Constant, Current
 -- and Impedance components. The PSS/E values are thus set based on the
 -- total real/reactive power multiplied by the component
 -- (which should add to 1.0)
 if (self.LoadResponse <> null) then{
  if (not self.LoadResponse.exponentModel) then{
   Pmva := p * self.LoadResponse.pConstantPower;
   Qmva := q * self.LoadResponse.qConstantPower;
   Pcurrent := p * self.LoadResponse.pConstantCurrent;
   Qcurrent := q * self.LoadResponse.qConstantCurrent;
   Padmittance := p * self.LoadResponse.pConstantImpedance;
   Qadmittance := q * self.LoadResponse.qConstantImpedance;
```

```
  }else{
   -- PSS/E cannot deal with the exponent model so set all as MVA
   Pmva := p;
   Qmva := q;
  }endif;
 }else{
  -- If no LoadResponse is present then only the constant power is set
  Pmva := p;
  Qmva := q;
 }endif;


-- The PSS/E Area is mapped to the SubGeographicalRegion and the Zone to
 -- the GeographicalRegion
 result.Area := self.getSubGeographicalRegion().map getArea();
 result.Zone := self.getGeographicalRegion().map getZone();

 -- For Load's with an OperatingShare the Owner becomes its OperatingParticipant
 -- ignoring any but the first as PSS/E supports only a single owner
 result.Owner := self.OperatingShare->first().
          OperatingParticipant.map getOwner();

 -- The isConnected() helper function checks if the Terminal is connected
 -- and so sets the status 127oolean accordingly
 status := self.isConnected();
}
```

As can be seen in this example, a number of other mapping functions are called to convert *TopologicalNode* to *Bus* and *GeograpicalRegion* to *Zone* amongst others. The association paths can be followed with a simple syntax so that the *EnergyConsumer*'s (*self* in this code as it is the source of the mapping) *LoadResponseCharacteristic* can be accessed via its *LoadReponse* association with the simple syntax *self.LoadReponse* with the internal *LoadResponseCharacteristic* attributes then accessed by appending them as *self.LoadResponse.pConstantCurrent* or *self.LoadReponse.exponentModel*.

SynchronousMachine to Generator

The mapping of generation is more complex as it as the CIM's *SynchronousMachine* class may not be behaving as a generator so there are some initial conditions placed. This mapping is also a one-to-many as it may produce a PSS/E Generator and multiple PSS/E Owner Share elements.

```
Mapping CIM::IEC61970::Wires::SynchronousMachine:: getGen() : PSSE::Generator
 -- Only SynchronousMachines operating as a Generator with a GeneratingUnit
 -- or as a condenser without a GeneratingUnit are mapped to the PSS/E Generator
 when {
  (self.GeneratingUnit <> null and self.operatingMode =
   CIM::IEC61970::Wires::SynchronousMachineOperatingMode::generator) or
  (self.GeneratingUnit = null and self.operatingMode =
   CIM::IEC61970::Wires::SynchronousMachineOperatingMode::condenser)}{

 -- The Generator's Bus is mapped to the SynchronousMachine's Terminal's
 -- TopologicalNode
 result.Bus := self.Terminals->first().TopologicalNode.map getBus();
 -- The PSS/E ID is mapped to the name
 result.id := self.name;


-- If the SynchronousMachine has a RegulatingControl that controls Voltage
 -- The the regulating bus for the Generator must be set to the
 -- Bus the RegulatingControl is controlling (obtained via the TopologicalNode
 -- of the Terminal it is associated to) and the setpoint set to the
 -- RegulatingControl's regulated set-point
 if (self.RegulatingControl <> null and self.RegulatingControl.mode =
   CIM::IEC61970::Wires::RegulatingControlModeKind::voltage) then{
  var cBus := self.RegulatingControl.Terminal.TopologicalNode.map getBus();
  result.VregulatedSetpoint := self.RegulatingControl.targetValue/cBus.baseKV;
  if (cBus <> null and cBus <> result.Bus) then{
   result.Regulating := cBus;
  }endif;
 }endif;

 -- If a generator the Pgen, max and min values are taken from the GeneratingUnit
 if (self.GeneratingUnit <> null) then{
  Pgen := self.GeneratingUnit.nominalP;
  Pmax := self.GeneratingUnit.maxOperatingP;
  Pmin := self.GeneratingUnit.minOperatingP;
 }endif;

 -- If the ratedS is set that is taken as the base power for per-unit conversion
```

```
-- otherwise the system base is used
var mBase := sBase;
if (self.isSet("ratedS")) then{ mBase := self.ratedS;}endif;
MVAbase := mBase;
-- The reactive power gen, max and min are taken from the SynchronousMachine
Qgen := self.baseQ;
Qmax := self.maxQ;
Qmin := self.minQ;
Rmachine := zToPerUnit(self.r, result.Bus.baseKV, mBase);
Xmachine := zToPerUnit(self.x, result.Bus.baseKV, mBase);

-- If the Terminal has an SvPowerFlow this is taken as being the actual
-- power output of the machine. The values are multiplied by -1 as CIM's
-- representation of powerflow has a Load as positive and a Source as negative
-- (in terms of direction of flow into or from the bus)
if (self.Terminals->first().SvPowerFlow <> null) then{
 Pgen := self.Terminals->first().SvPowerFlow.p * -1;
 Qgen := self.Terminals->first().SvPowerFlow.q * -1;
}endif;

-- A PSS/E OwnerShare is created for each OperatingParticipant in the CIM.
-- As such this mapping produces not only a PSS/E Generator but potentially
-- multiple PSS/E OwnerShare elements
self.OperatingShare->forEach(op){
 var share := object PSSE::OwnerShare{};
 share.Owner := op.OperatingParticipant.map getOwner();
 share.fraction := op.percentage/100;
 result.Owners := share;
};

-- The isConnected() helper function checks if the Terminal is connected
-- and so sets the status 129oolean accordingly
result.status := self.isConnected();
}
```

This and the *EnergyConsumer-Load* mapping show that the mapping of the actual electrical parameters (load, generation) often form the minority of the transformation code. Dealing with the differences in structure can account for the majority of the effort and it is here that the

code is both easier to write and maintain when it is written against the structure rather than trying to map CIM XML directly to PSS/E RAW in a single function.

Benefits of a Model Driven Architecture

An MDA approach when dealing with CIM data offers a number of benefits. By ensuring that all data has a defined structure with a meta-model, users can understand and identify how it maps to the CIM. Model Driven Transformation makes it easier to write and maintain the mapping of data between systems even though, on the surface, the process appears more complex.

By separating the parsing of the data from the transformation itself the process is modularized, with the parsing module re-usable in a number of processes and simplifying the transformations so they do not have to deal with the added complexity of file formats or database access.

The example shown here contains parts of a CIM to PSS/E transform to show a real-world example of how MDT can be used with CIM data and how a meta-model can be derived from existing, proprietary formats to support this architecture.

Case Study

Jeff Kimble has learned that the CIM, because it is a model, can support model driven transformation. This is useful information because his organization was considering acquiring a work management system that stated that some of its internal structures were CIM-derived. One of the systems integrators that Electric Innovation Utility was considering using to integrate this system with other necessary applications such as GIS and CIS have been pushing to use their proprietary adapters for the job. Jeff realizes that since the work management application is CIM-derived, if they support CIM adapters at the enterprise service bus, they could potentially have a stateless bi-directional transformation. This will save a tremendous amount

of effort creating the adapters. Business logic will not need to be created to capture application state. Information will be preserved as it passes from system to system, without extra coding in the adapters.

Section 9 Questions

1. If the internal data of an application are derived from the CIM a bi-directional transformation is most likely to be:
    a. Stateless
    b. Stateful
    c. Both stateful and stateless because it is bi-directional
    d. Neither stateful nor stateless
2. Which response most closely describes a stateful transformation?
    a. A stateful transformation has a token that can be used to pass application borders.
    b. The source or target does not contain sufficient data to enable a bi-directional transformation without loss of data.
    c. The source or target contains enough information for bi-directional transformation without loss of data.
    d. A stateful transformation is not concerned with transformation.
3. In Model Driven Transformation (MDT) the code is:
    a. Based on the CIM or other UML.
    b. Based on XML.
    c. Based on a database.
    d. The data based on its structure and relationships, not on any particular model, database, or file.
4. Model Driven Transformation makes it easier to write and maintain the mapping of data between systems when:
    a. The data is based on the CIM.
    b. The data is unidirectional.
    c. The data has a defined structure with a meta-model.
    d. The data is bi-directional.
5. By keeping the data transformation modular there is the benefit that:
    a. It is re-usable in a number of processes.
    b. The transformation deals with the added complexity of database access and file formats.
    c. It isn't based on real world examples.
    d. The data is unidirectional.

10. EXTENDING THE CIM

Learning Objectives

- Understand how to extend the CIM to handle gaps or new requirements

The CIM is a large and extensive information model, but since its inception it has been extended and expanded. Such extensions may occur due to IEC-led efforts to expand the CIM's scope into new areas that are eventually integrated into the next revision of the standard; or utilities and vendors need to extend the CIM for their own projects to deal with new application areas or enhance the model to cover their own data requirements.

To better understand how and why a user may need to extend the CIM a number of simple examples will be used to illustrate how the model is extended. How these extensions are defined and stored is not standardized by the IEC or its working groups. For the purposes of these examples *multiple-inheritance*[45] will be used, but the reader should understand this is not the only way to extend the standard and different vendors and applications have different approaches for marking and tracking additions to the UML.

Accommodating Legacy Data

Adding a New Attribute

A simple example of adding an extension to the CIM is to use the situation of a utility migrating data from a legacy system to a new system that uses the CIM. The existing data structures have been mapped into the CIM, but there is a requirement to retain information about whether *Equipment* data originated in a legacy system or not.

This requires all instances of *Equipment* to have a flag *fromABCSystem* that is marked when the data originated in the legacy system (called *ABCSystem* in this example). This requirement extends to all subclasses of *Equipment* so the attribute must be added to the *Equipment* class and then inherited by the subclasses. The extended attribute must be marked as being an extension so that any exporter can either choose to ignore extended attributes or, if using XML, knows to put the extended attributes under their own namespace (so that an importer can correctly identify *standard* CIM data from the extended, *non-standard* data).

[45] Multiple inheritance is when a class may have more than one parent class and thus *inherits* attributes and associations from multiple parent classes. The CIM UML does not contain multiple inheritance and the IEC working groups deliberately avoid using multiple inheritance. There has been discussions and proposals for introducing it to the UML, but to date the modeling guidelines prohibit its use in the standard UML. However, it has been used for defining extensions to the model in a project perspective.

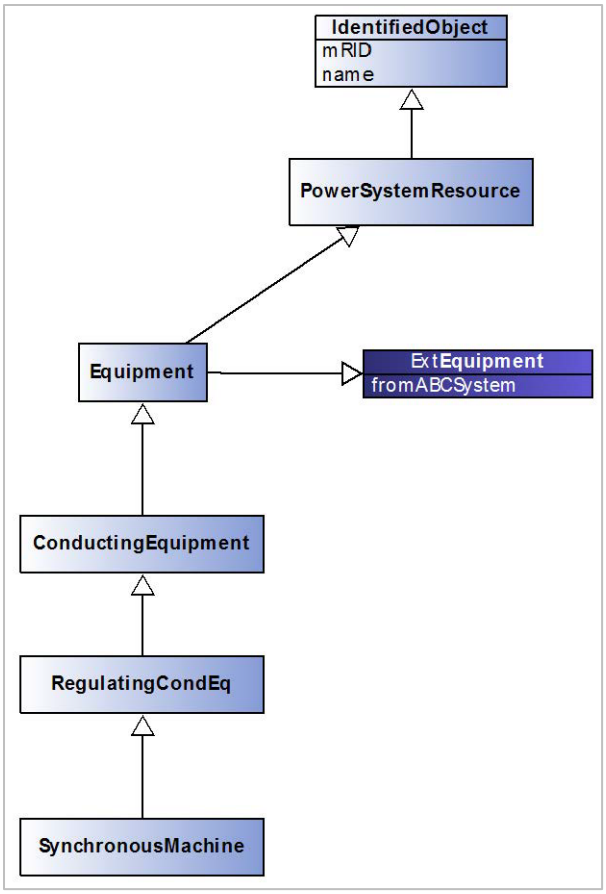


Figure 10-1. Adding an extension to an existing class

In Figure 10-1 the class *ExtEquipment* is added with the new attribute *fromABCSystem*. The existing, standard CIM class *Equipment* now inherits from its original parent, *PowerSystemResource* and from the *ExtEquipment* class. This means that all subclasses of *Equipment* inherit this new attribute, such as *SynchronousMachine* as shown in the example.

Adding New Classes

Extending this example, the scenario where two companies have merged and are integrating their systems into a single data model will be used. There may also be a requirement to retain a record of which system the data originated in. This could be added with another attribute on the class, however both the company and system the data originated in need to be modeled, as well as finding the source for an instance of *Equipment* needed to be able to navigate from the *OriginalSource* to all instances of *Equipment* that it originally provided.

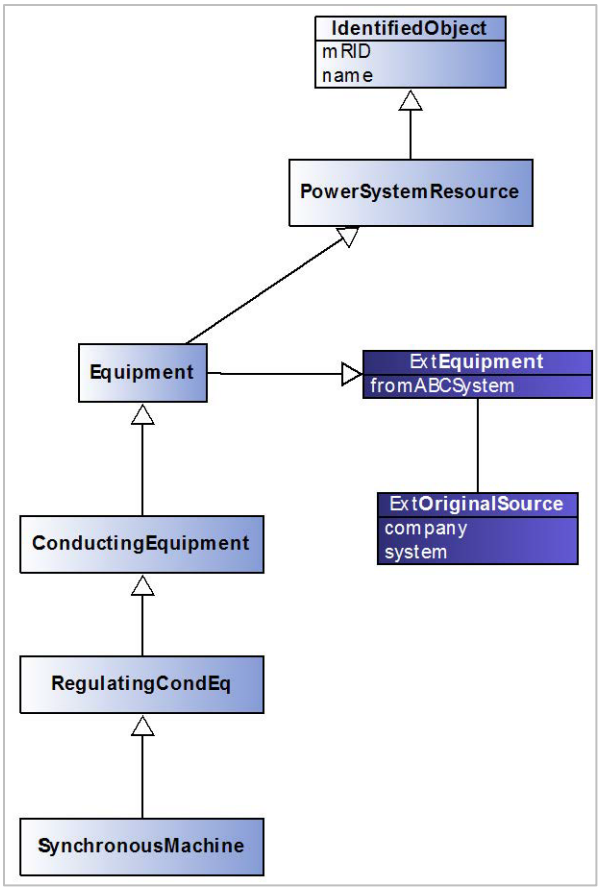


Figure 10-2. Adding a new class to the model

To accomplish this a new class, *ExtOriginalSource*[46] is added, with an association to the *ExtEquipment* as shown in Figure 10-2. This association is then also inherited by all subclasses of *Equipment* and by having a separate class it means that an instance of *ExtOriginalSource* need only be created once for each system that provided data and then shared between all instances of *Equipment*.

These extensions are examples of internal company/project specific extensions that are not intended to be shared outside of the company. They represent custom extensions that users would not expect to find in the IEC standard itself.

[46] For these examples all the new classes will be prefixed with *Ext* to make it easier to differentiate what is an extension and what is from the CIM. This is not a requirement of extensions although some tools would have issues with two classes having the same name, even if contained within different packages.

Introducing New Application Areas – CIM for Gas

Another scenario requiring extensions is when a new application area must be modeled with extensions built on the existing CIM concepts and structure that would be of relevance to other utilities. In this situation the extensions are done both to model the data requirements of the utility or project, but with the intention of eventually proposing them to the IEC working groups as an extension to the CIM that would be added to a later revision.

This example will use the scenario of a utility wishing to extend the CIM to model the gas network so that they can better integrate control and monitoring of the two networks. This will involve the modeling of the gas network itself and tying it into the existing CIM classes covering the electrical network where the two are interconnected (that is, at a gas-fired power station).

Adding New Classes

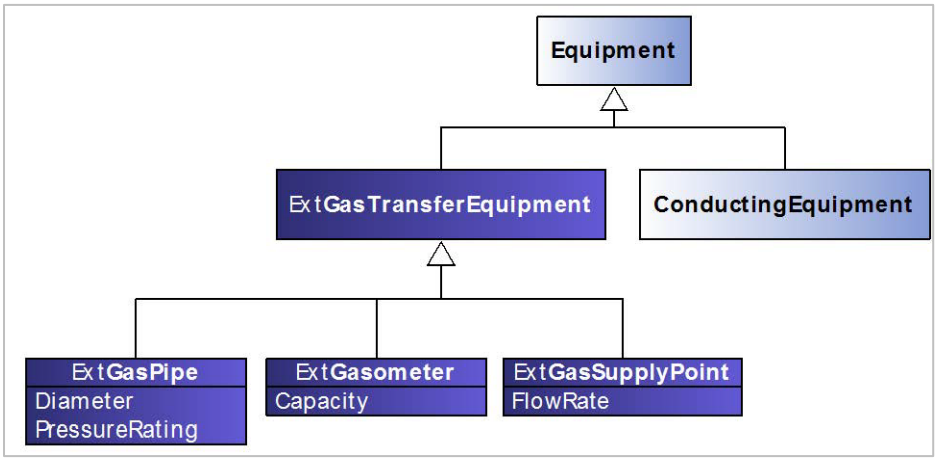


Figure 10-3. Gas Transfer Equipment classes

The CIM's existing *Equipment* class is taken as the starting point and where, for electrical conduction, it is specialized into *ConductingEquipment*, for the gas network a complimentary class *ExtGasTransferEquipment* is created to represent any equipment that can transfer gas.

This *ExtGasTransferEquipment* class is then sub-classed into:

- *ExtGasPipe,* with attributes to define its diameter and pressure
- *ExtGasometer* to represent a point of gas storage with an attribute to define its capacity
- *ExtGasSupplyPoint* to define a point on the network where gas is supplied

Refining Existing Classes

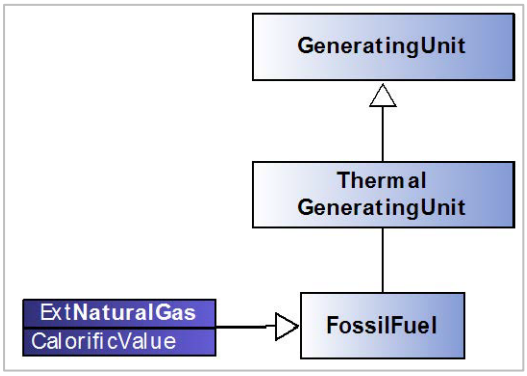


Figure 10-4. Refinement of CIM Fossil Fuel class for Natural Gas

For this example, a class that defines information about the gas being supplied will be added. The CIM already contains a *FossilFuel* class that is tied to *ThermalGeneratingUnit* and so this class will be extended to define natural gas with the *ExtNaturalGas* class as shown in Figure 10-4.

This re-uses an existing CIM class and extends it with the additional data that is needed for modeling the gas network, in this case the *calorific value* of the gas. By specializing the existing CIM class it inherits an association to the existing CIM *ThermalGeneratingUnit* class. Recall that specialization is a type of association between classes where the specialized class (also known as *subclass*) inherits all of the attributes and associations of the parent, but allows additional, specific attributes for the subclass to be added.

Assigning a Gas Supply Point

As part of this extension the connection points between the gas and electrical networks at a gas-fired power station will be explicitly modeled.

In Figure 10-3 the *ExtGasSupplyPoint* class was added and so now add an association between this and the existing CIM classes that represent electrical generators, primarily the *ThermalGeneratingUnit* class.

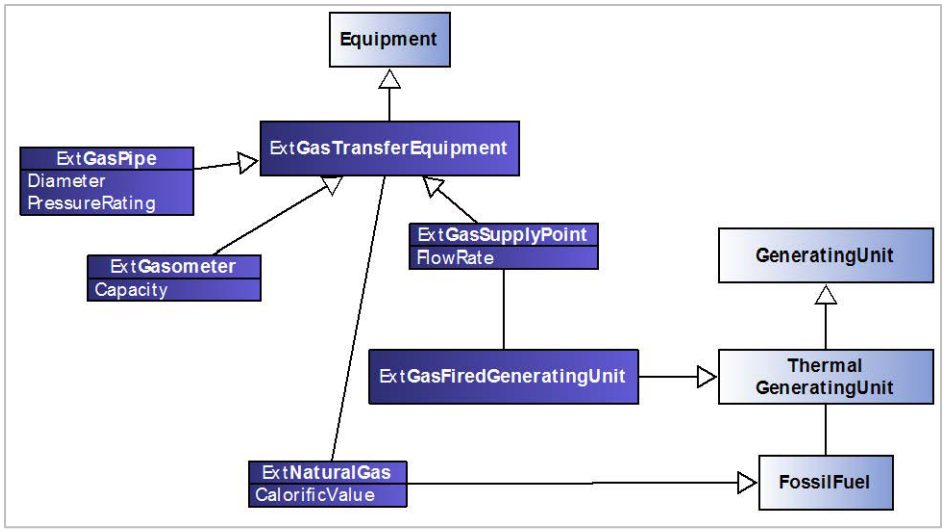


Figure 10-5. Connecting the Gas and Electrical models

ThermalGeneratingUnit however can refer to any power station that converts heat to electricity, whether it is fuelled by coal, gas, oil, or nuclear. As such the existing class can be refined and specialized into *ExtGasFiredGeneratingUnit* that has an association to the *ExtGasSupplyPoint* as shown in Figure 10-5. The *ExtGasTransferEquipment* class has an association added to the *ExtNaturalGas* class, which is then inherited by all subclasses of *ExtGasTransferEquipment*, including *ExtGasSupplyPoint*.

Gas Containment

The existing CIM containment structure has *Geographical Regions* and *Sub-Geographical Regions* containing *Substations*. This is suitable for this gas network example, but gas pipes and gasometers are not within *VoltageLevels* so an equivalent containment for gas is required.

A *ExtGasPressureLevel* class is analogous to *VoltageLevel* and will be used to contain gas equipment. This will still reside inside a *Substation* so the existing *Substation* class must be extended to include an aggregation association between it and the new *ExtGasPressureLevel* class to mirror the *Substation-VoltageLevel* relationship.

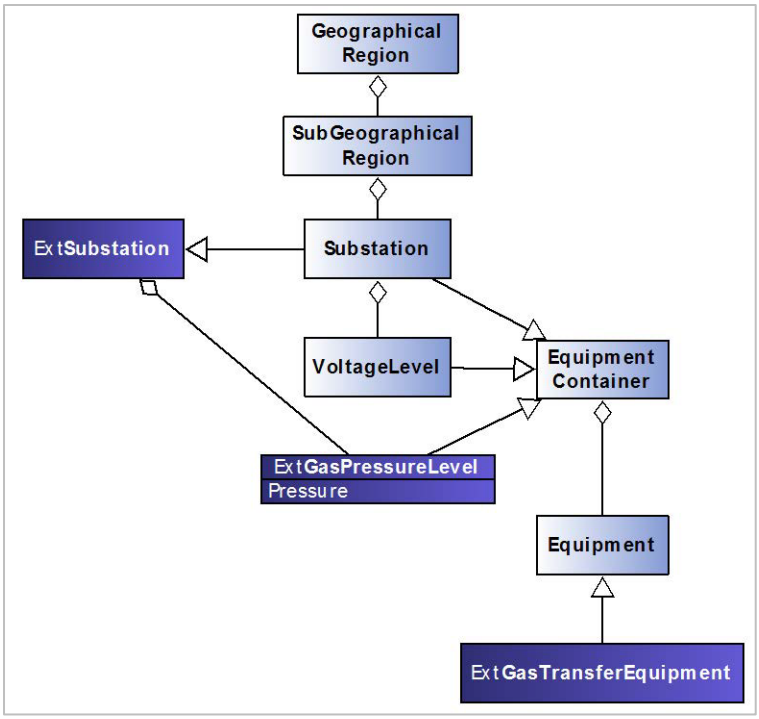


Figure 10-6. Gas Pressure Level Containment Classes

To add this new aggregation association to *Substation* an extension class, *ExtSubstation* is added which *Substation* now inherits from (as well as its standard inheritance from *EquipmentContainer*) as shown in Figure 10-6. When using this construct in a contextual model the *ExtGasPressureLevel* would have a restriction placed on the aggregation association it inherits from *EquipmentContainer* so that it may only contain instances of *Equipment* that are subclasses of *ExtGasTransferEquipment*. The alternative would be to have an explicit association between *ExtGasPressureLevel* and *ExtGasTransferEquipment* which may be considered more appropriate but for this example the existing CIM containment structure will be re-used.

Creating Extensions in the CIM using Shadow Classes

More recently, creating extensions in the CIM has taken a new approach, not too different from the Ext[Class Name] model shown in this chapter. This method involves the creation of "shadow" classes that are named exactly the same as the class being extended. What makes this extended class unique, however, is its stereotype. This added stereotype makes it clear that the class is an extension, while the extended class retains the same name so that when it is converted into instance data in an RDF, XML, or JSON file, it would look no different than the original class it is extending. Below is an example of a *TapChangerControl* class that has been extended to include new attributes.

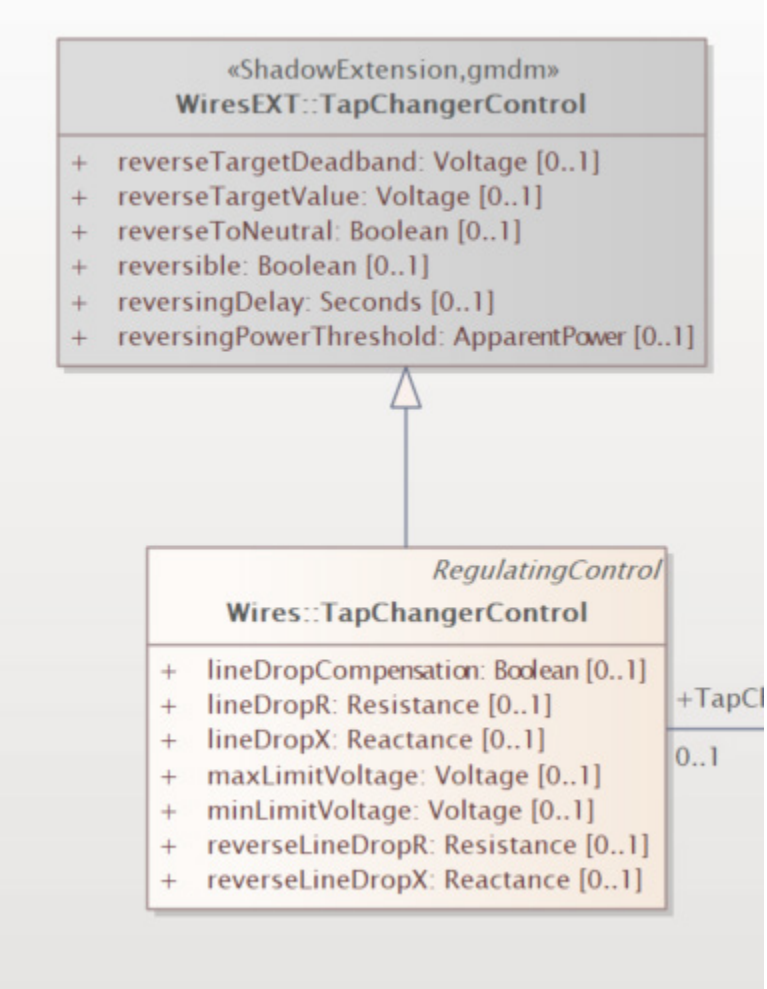


Figure 10-7. Extending with shadow classes

In the above figure, various “reverse” style attributes are now being tracked for a given *TapChangerControl* object. The extended class retains the same name as the class it is extending, and like the previous examples, the original class inherits from this shadow class. Profiling tools such as CIMTool or CimConteXtor can then pull this class’s new attributes in, and will even attribute them a different namespace in the schema itself to make it clear that some of its attributes belong to a class extension.

In the sample UML, this stereotype is also coded to make the class appear a different color in the UML so that those looking at the UML can see at a quick glance where these extensions have been made.

Extensions Summary

The CIM UML has now been extended with:

- Attributes and classes to record information about legacy data sources
- Refinement of existing CIM classes for fossil fuels and generating units
- A new hierarchy of classes for modeling a gas network
- Extended the CIM containment classes to provide a parallel gas containment structure

All the extensions to existing CIM classes have been accomplished using inheritance and no attributes or associations have been added to the existing CIM classes. All the extensions can be identified so systems implementing this extended model can still export *standard* CIM by discarded extended data (or converting it to a standard representation in the case of refined classes – that is, by exporting a *ExtGasFiredGeneratingUnit* as a *ThermalGeneratingUnit* and ignoring any attributes/associations introduced in the *ExtGasFiredGeneratingUnit* class).

Internal vs. Global Extensions

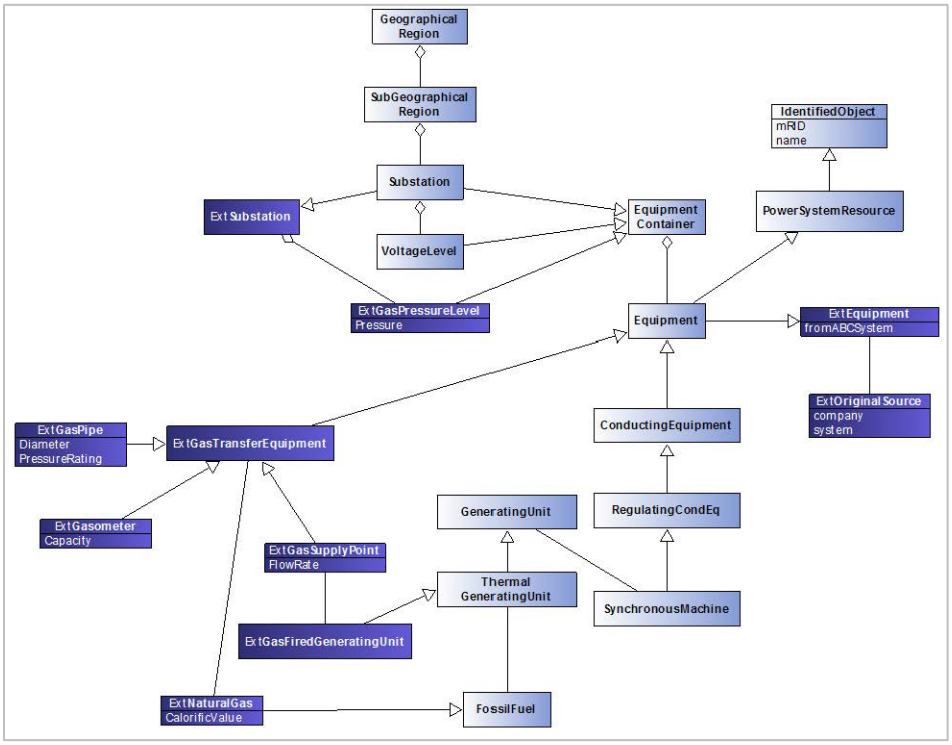


Figure 10-8. Extended CIM UML

All the extensions that have been introduced are shown in Figure 10-7. The extensions to *Equipment* to add the *fromABCSystem* flag and the *ExtOriginalSource* class can be considered internal extensions that are not of interest to other utilities not directly involved in the project and so are *internal extensions*. The other extensions, to add the gas network, could be

considered *global extensions* that would be of interest to other utilities that also operate gas networks.

These extensions may be considered candidates for addition to the IEC standard and so may be proposed to the working groups as an addition to the CIM UML. This would mean that in the future a utility would not be requesting support for a custom, internal data model from a vendor but would instead be asking for a standards-compliant interface.

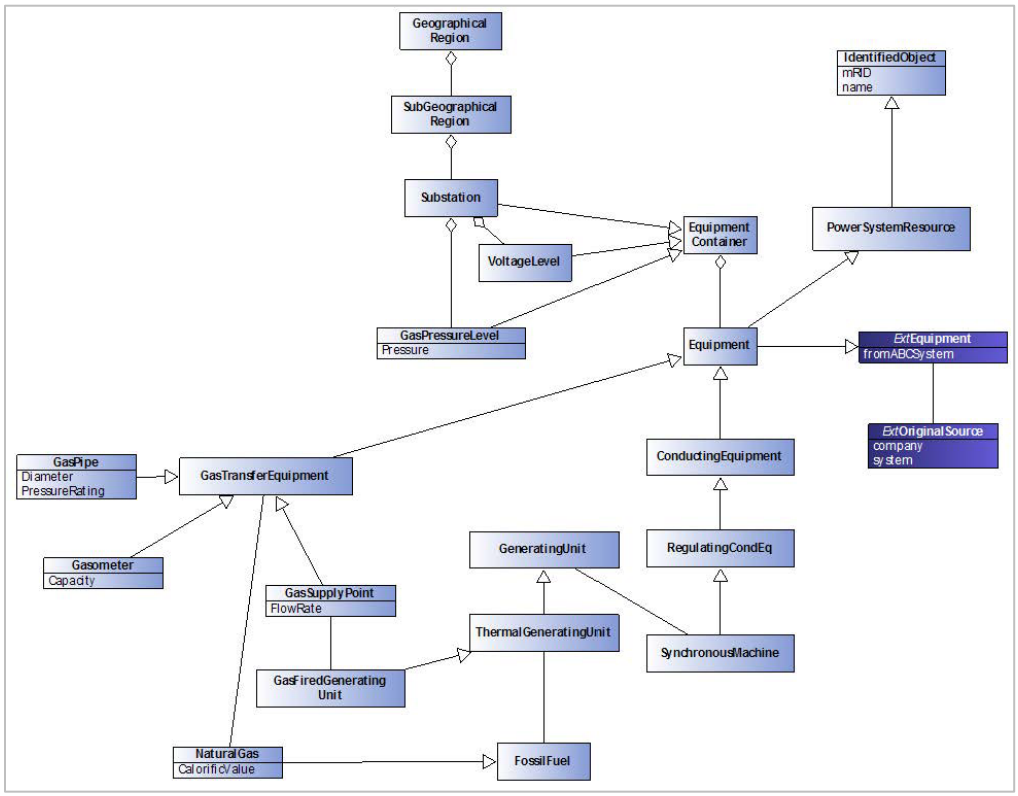


Figure 10-9. Future CIM with integrated extensions

The result would be that, as shown in Figure 10-8, the standard now includes all of the gas network classes and the only extensions that need to be maintained are for the legacy data. The gas network classes have been merged into the standard and so reducing the complexity, time and cost to the utility (and their vendors) in maintaining a custom data model.

Adding a New Enumeration Through Class Extensions

An enumeration is a datatype that represents a finite set of values. Enumerations are generally used to model domains in which the values are limited and known beforehand. For instance, the type of a windfarm could be either onshore or offshore. Therefore, it is possible to have an enumeration with two values for onshore and offshore. This enumeration already exists in the 61970 package.

In the example below the *WindGeneratingUnit* class is extended to accommodate a new enumeration for the turbine disposition which can be either horizontal or vertical axis. The first step is to introduce the extended enumeration class into the model. In the example below, it corresponds to *turbineDispositionKind* which contains the two values for *horizontalAxis* and *verticalAxis*. A good practice to help distinguish the extended enumerations, classes and attributes is to use stereotypes. In the example below, *Primer* is used as a stereotype.

After creating the *Primer* stereotype, the *WindGeneratingUnit* class needs to be extended. To do that, this class must inherit from a new *ExtWindGeneratingUnit* class. The extended class contains an attribute which is linked to the pre-existing enumeration *turbineDispositionKind*.

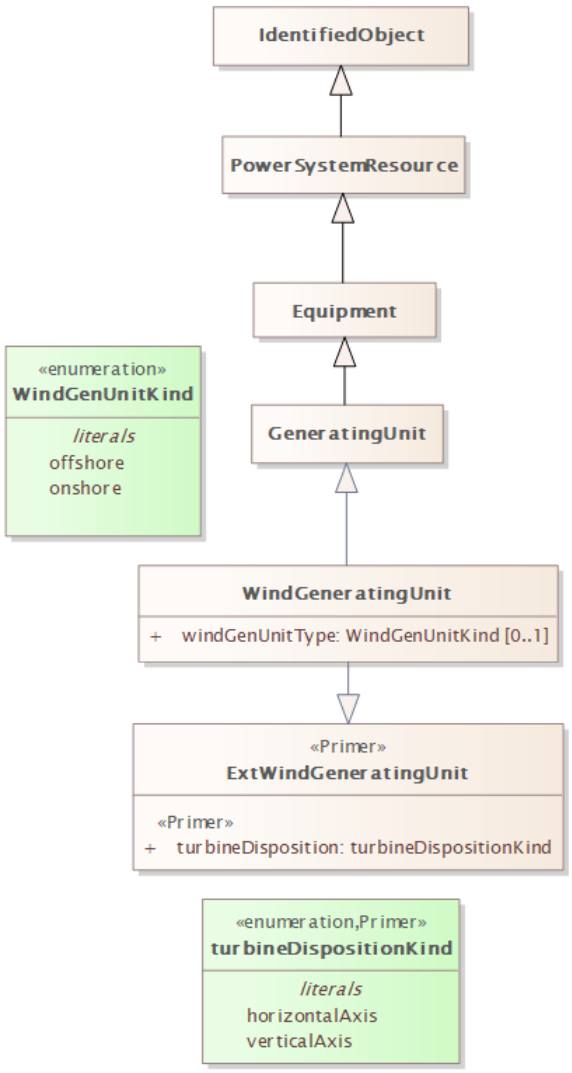


Figure 10-10. Adding a custom extension to the CIM using Stereotypes

Handling of code lists

The next area of focus in the enumeration space is the concept of code lists. A code list is just an enumeration defined at a profile level in which its values are assigned to a determined initial value (e.g. A01, A02, etc.). This approach gives a lot of flexibility because it allows the user to create or extend enumerations without having to modify the information model. As an example, ENTSO-E intensively uses code list for the ESMP (European Style Market Profile; IEC 62325-351).

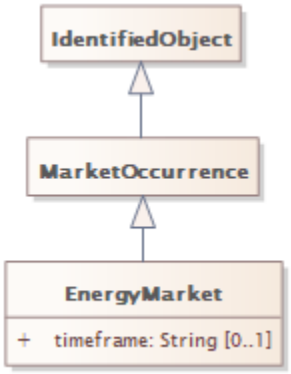


Figure 10-11. EnergyMarket class in the CIM

For a practical example, let's look at a use case that was recently handled by ENTSO-E. A process can be run in different timeframes like day-ahead or intraday for instance. In order to handle timeframes, the *EnergyMarket* class was recently introduced in the ESMP. In the snapshot below, reader can see how the *EnergyMarket* is currently modelled in the information model.

However, the *timeframe* attribute in EnergyMarket class is just a free string rather than an enumeration. As a result, it is necessary to do a restriction at the profile level. The first step shall be to create a new enumeration at the profile level with all the values that a determined business process requires. In the example below, it corresponds with the *TimeframeTypeList* enumeration. This enumeration is the same as the one described above; however, the only difference is the initial value. In the initial value, the user can define their preferred code. This enumeration needs to be based on the "String" datatype.

The second step shall be to create a datatype at the profile level. In the example below, it corresponds with the *Timeframe_String* datatype. Because the *timeframe* attribute in the information model level is a string, the datatype in the profile also needs to be based on a string, otherwise the dependency between profile and information model is broken. Therefore, an *is-based-on* association is required from the profile datatype to the one in the information model. This profile datatype shall have an attribute which points to the enumeration which was previously defined.

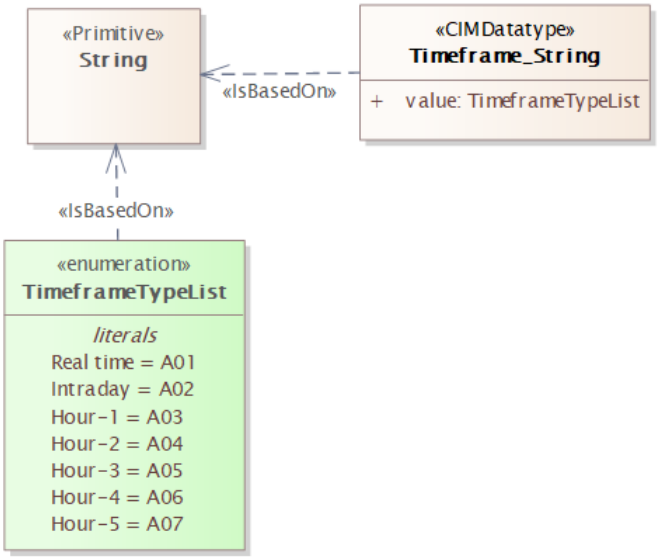


Figure 10-12. The new enumeration pointing to the datatype upon which it is based

The final step is to bring the class into the profile, in this case the *EnergyMarket* class. Once profiled, the *timeframe* attribute datatype which is a string and comes from the information model is replaced by the *Timeframe_String* datatype attribute defined in a profile level.

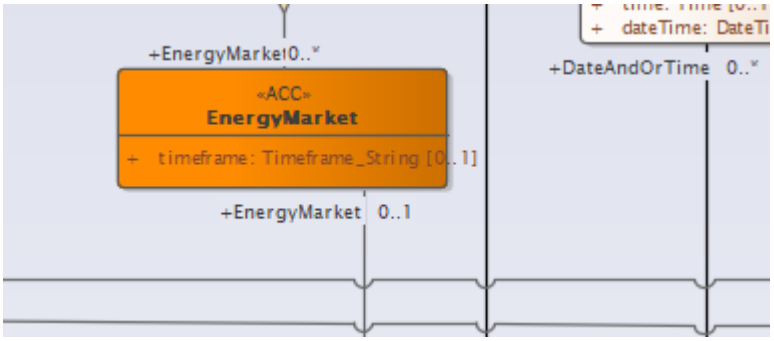


Figure 10-13. Changing out the attribute for its enumeration at the profiling level

ENTSO-E currently serializes all the code list enumerations in the ESMP CIM Package XSD[47] and the plan is also to move to new formats like JSON schema or Apache Avro in near future. This approach is also quite flexible because it allows the user to validate any of the XSD profiles against the XSD codelist. Serialization is currently done using CIMContextor and CIMSyntaxGen which are Enterprise Architect add-ins published under Apache 2 license. At this time, there is no support in CIMTool or Schema Composer for this style of profile generation.

[47] https://www.entsoe.eu/publications/electronic-data-interchange-edi-library/

ENTSO-E frequently publishes new versions of their code list in their webpage[48]. More detailed information on the ESMP profiling process and handling of code list can be found there.

Case Study

Jeff Kimble has learned that although the CIM is fairly mature, there are always new use cases being developed that can cause gaps between the current state of the CIM and what is needed for integration. The causes of these gaps are typically new technology and capabilities that require new attributes or classes to be added. Sometimes gaps are caused by a need to integrate two or more legacy systems, sometimes caused when utilities merge. Another source of model extensions is when CIM-like methods are applied to other domains, for example, water, or gas utilities (because the CIM is an electrical system model). Lastly, Jeff has learned that his organization can always externalize enumerations into their own profiles so that their stricter data requirements are still compatible with the canonical CIM.

By using a process to identify where extensions have been added to the model Jeff realizes that when updates are made to the CIM it will be easier to migrate those updates to artifacts that have been used for integration (assuming there is a business case for doing so). Since Electric Innovation Utility is considering acquiring a gas utility, Jeff finds these gas examples very well suited for the type of extensions he envisions for his organization's enterprise semantic model.

Section 10 Questions

1. Some reasons for extending the CIM include
    a. Support for legacy systems
    b. Support for capturing extensions that need to be shared with the IEC and Working Groups
    c. Support for other domains, that is, water, gas
    d. All of the above
2. Using an identifier as a prefix or suffix to an extension
    a. Allows the appropriate titles to be used with the extension
    b. Makes the extension required for the standard
    c. Facilitates identifying where extensions exist to avoid conflict with the standard
    d. Makes it harder to identify where an extension has occurred
3. To identify and extension
    a. *Ext* must be used
    b. *Zee* must be used
    c. *XYZ* must be used
    d. Any suffix or prefix could be used as long as it is used consistently

[48] https://www.entsoe.eu/publications/electronic-data-interchange-edi-library/

4. A utility wants to extend the CIM to support water operations. To extend the equipment class to support water, one class that might be added is…
    a. ExtGasEquipment
    b. Equipment
    c. ExtWaterEquipment
    d. ExtElectricalEquipment
5. A specialized class is a class that…
    a. Inherits all of the attributes and associations of the parent class
    b. Feels really good about itself
    c. Does not inherit any attributes from the parent because it is special
    d. Inherits some attributes, but ignores some other
6. What is the added value of using code lists
    a. Define multiple codes for an enumeration value
    b. Combining different enumerations with codes.
    c. Flexibility as it allows to create new enumerations and values without updating the canonical CIM
    d. No value at all

11. APPLICATION INTEGRATION

Learning Objectives

- Understand the key words used and the relationship between Verb/Noun pairs
- Gain familiarity with some of the standard integration patterns
- Understand where to go to get more information on using the CIM for application integration

One of the early criticisms of the CIM was that although it was a large, comprehensive model, users found very limited guidance on using it to integrate systems. The learning curve associated with CIM was steep. Even after profiles were introduced, it was difficult for practitioners to go from understanding the internals of a profile to applying it to the integration of two systems. CIM is intended to be used as a *reference*; it is not a *guide*. By comparison, a reader can find the meaning of words in a reference dictionary, but there is no guidance on how to write a sentence.

For these reasons the Working Group that supports CIM for distribution (IEC 61968) where application integration is a significant concern, took on the task of creating guidance for the use of the CIM in systems integration. This built on early developments made by the user community in forums such as the CIM Users Group and the Utility Communication Architecture International Users Group (UCAIUG). This sharing of information reflects the importance of the relationship between standards development organizations (SDO) and the supporting users groups that put standards into practice. The SDO, by producing a standard says in so many words, “thou shalt…” and the user community, as they attempt to put the standard into practice, finding gaps and creating extensions, feed their findings back to the SDO, in effect stating, “yes, and…”

To this end Working Group 14 created Implementation Profiles as a standard (see *IEC 61968-100: Application integration at electric utilities – System interfaces for distribution management – Part 100: Implementation Profiles*). This standard provides guidance on how to use IEC 61968 for utility application integration, specifically for Parts 3 through 9. The creation of profiles was already covered in previous sections, so this section will focus on the integration patterns and the naming convention for services.

The focus of IEC 61968-100 is web services and java messaging services (JMS); however, other integration technologies could be used, that is, Extensible Messaging and Presence Protocol (XMPP) or Representational State Transfer (REST). In fact, technologies such as XMPP and REST have been used with CIM profiles, it just so happens that the scope of the specification was limited to web services and JMS at the time of its writing.

Verbs

First it is important to understand the use of *verbs* and *nouns* in the service naming convention described by IEC 61968-100. Once the use of verbs and nouns in the naming convention is covered some of the basic integration patterns such as request/response will be explored.

Table 11-1. CIM Verbs and their usage in naming patterns

Request Verb	Reply Verb	Event Verb	Usage
Get	Reply	(None)	Query
Create	Reply	Created	Transaction
Change	Reply	Changed	Transaction
Cancel	Reply	Canceled	Transaction
Close	Reply	Closed	Transaction
Delete	Reply	Deleted	Transaction
Execute	Reply	Executed	Transaction

First it is important to be familiar with the list of verbs and when they are used.

Get – is used for queries.

Create – is used when creating an object that match the *noun* (more on this in a moment).

Delete – is used to destroy an object although an object in a business system might only be marked for deletion (keeping the record of the activity).

Close and *Cancel* – are used in relation to a business process such as closing a work order or perhaps canceling a request.

Change – is used to modify an object that was used in previously, for example, as part of a *Create* or *Created* action.

Execute – this is used in complex transactions and is used with an *OperationSet*, that is, a structure that may hold actions and contain more than one verb. This section will be limited to some simple patterns so OperationSet will not be explored here.

Update – new in the 2nd edition, update specifies a request to modify in part or in whole the data pertaining to one or more existing objects on the responding system.

Nouns

In IEC 61968-100, the use of *nouns* is used to convey or identify a payload structure. This often is equivalent to a profile, something that has been covered in prior section. A message payload is typically conveyed using an XML document that conforms to an XML schema, and can be validated against an XSD. For example, a standard profile from IEC 61968-9 is an

EndDeviceEvent profile. This profile is used to pass information about events that occur in the distribution network to other systems that may be interested. For example, if a smart meter sends a “last gasp” (the last message before it loses power) when this message is received by the Advanced Metering Infrastructure (AMI) Head-End System (HES), it might pass this information on to a Meter Data Management System (MDMS) or an Outage Management System (OMS) using an EndDeviceEvent message profile. But first, an understanding of some basic integration patterns will be in order.

Integration Patterns

The simplest and fairly common integration pattern is the request/response pattern. One system makes a request of another, and that system responds. In this generic pattern shown in the figure below, the client makes a request of the server, and the server responds. The request is represented as the verb(noun) shown on the top arrow.

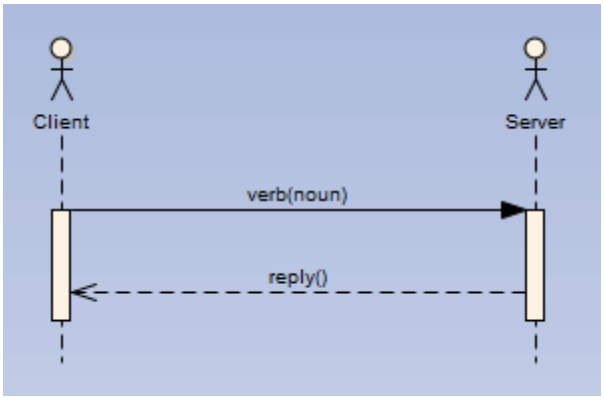


Figure 11-1. Basic request/response pattern

This basic pattern could be applied to the AMI HES to OMS integration that was described earlier. If the AMI HES wants to send an EndDeviceEvent message it will need to *create* the EndDeviceEvent object, so the basic message pattern looks like the figure below.

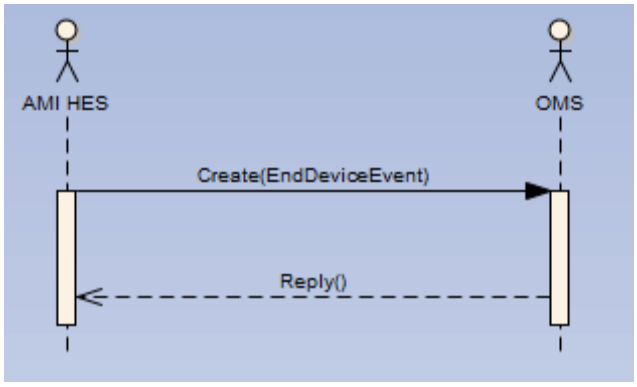


Figure 11-2. Basic request/response pattern using the Create verb and the EndDeviceEvent object

In this example the *Create* verb is used in conjunction with the *EndDeviceEvent* noun. Together this makes up the naming pattern of the web service. Once the OMS receives this message it responds with the *Reply*, to let the AMI HES know that it has received the EndDeviceEvent[49].

For developers, as they begin to get comfortable with the naming convention they very quickly begin to understand the types of interactions that are occurring, and the types of events and messaging that is being supported.

Now to move onto a slightly more complex example, suppose that a remote disconnect is going to be processed for a customer. There are several steps that will need to occur:

1. The CIS will need to pass a command to the AMI HES to initiate the disconnect (in this case via a MDMS).
2. The disconnect will be performed and the notice of the event will be passed back to the CIS.
3. A meter reading will be initiated to get the final read from the meter and pass this back to the CIS for billing purposes.

The sequence diagram to capture the follow and naming of the messages is shown below.

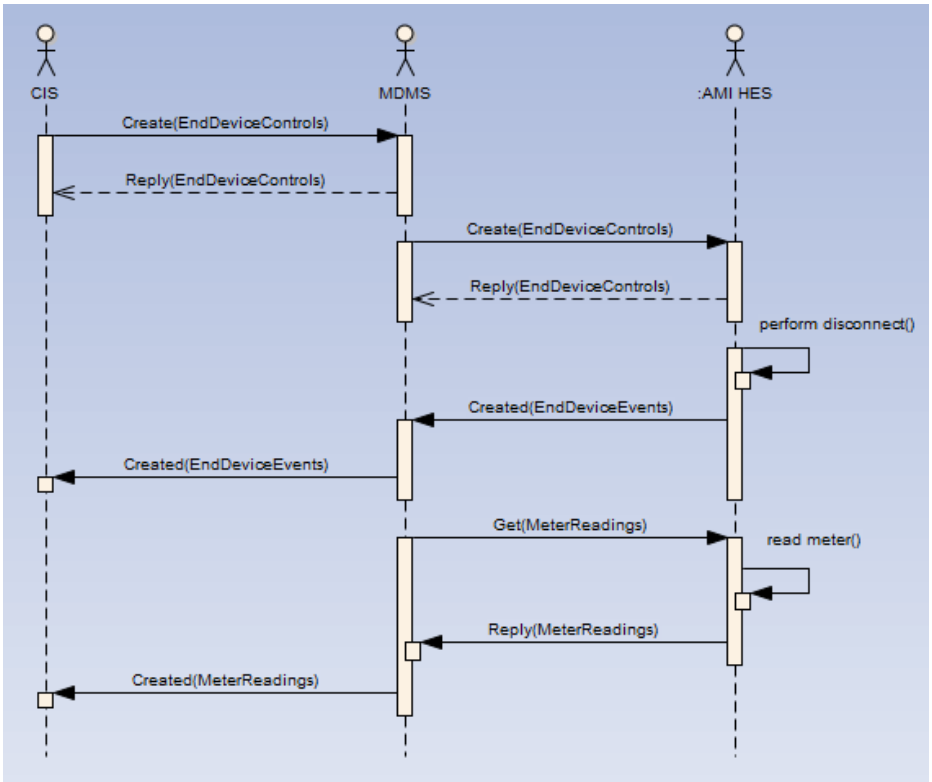


Figure 11-3. Example of message integration required for a remote disconnect of a smart meter

[49] For the specifics on message headers, the contents of the XML of the messages, and the structure of error handling, see IEC 61968-100 for more information.

As can be seen, the series of messages begins with a *Create* which uses an *EndDeviceControls* object. This is an object that is used for sending commands to end devices. Meters are considered "end devices" in the CIM. Rather than specify every type of possible end device with messages specific to each, the generic EndDeviceControls was created to handle commands to all the different types of devices.

The "create" is passed from the MDMS to the AMI HES. In turn each system responds with a Reply message so that the sending system knows that the message was received. When the remote disconnect is performed at the meter this will generate an *EndDeviceEvent*. This message is passed back up to the calling systems using the *Created* verb (as this is an event based on a prior action, the –ED is appended to the Create).

Finally, the MDMS initiates a meter reading to get the values from the smart meter post disconnect. This uses a *Get* with a noun of *MeterReadings*. The *Reply* returns the meter readings to the MDMS, which in turn, again, post event, uses *Created* to pass the meter readings to the CIS for billing.

At first glance this may appear to be confusing, but as developers become familiar with the patterns and the message profiles that are used, it soon becomes second nature. The added benefit is that developer *A* and developer *B* and developer *N* have not all come up with different naming conventions and patterns for the integration. It is this predictable pattern that allows developers to come up to speed very quickly on how to do system integration based on IEC 61968-100 guidance.

Only some basic integration patterns were used in this section to highlight the use of *Verb/Noun* pairing. IEC 61968-100 goes into more detail on the types of integration patterns, how to structure messages, error handling, use with JMS, and how to handle topics such as version control. This is out of scope for this section, but the reader is invited to explore this specification, especially if they are interested in standards-based systems integration.

IEC 61968-100 2nd Edition Updates

Although IEC 61968-100 is largely the same as the first edition, these versions are still fundamentally different and thus are not interoperable with each other. This means that a web service that was designed using the first edition will not be able to properly communicate with one using the 2nd edition. For those upgrading their systems, this is important to keep in mind. However, there are tools out there that exist to ease the transition between the two variations of the standard. For example, there is an option in CIMTool to generate part 100 2nd edition compliant schema during the profiling process when creating profiles. There is also a GitHub link in the 2nd edition that links to a Java program that will update existing 61968-100:2013 XSD profiles to a 2nd edition version. Thus, there are options to generate 2nd edition-compliant XSDs for both new and existing profiles.

Although using the approaches outlined above will create compliant schema, it is important to go through the new standard and see what changes have been made. For example, error codes have been significantly trimmed down in part 100 and have been left to other standards to

define where necessary. Many elements have slightly changed names, such as *Compressed* becoming *CompressedPayload*. A new verb has been added and an older one removed, so that could impact existing systems that relied upon the deprecated verb. Additionally, the *mrid* is now enforced as a UUID-type string. Any of these changes could break existing systems so it is important to be fully aware of what these tools are changing before upgrading to the second edition.

Case Study

Innovation is Electric Innovation Utility's "middle" name, but Jeff Kimble knows that his team also has to support legacy databases and JMS message integrations. However, as the CIM University training comes to a close he feels very comfortable going back to his organization armed with this CIM knowledge. He has learned that the CIM is very robust as a reference model, with hundreds of classes and attributes that describes the utility environment. He has learned that not only can the model be used to exchange network diagrams, but that the model can be pared down to create a profile to either describe a smaller section or to even be used for messaging. He has seen how the model can be used for bi-directional transformation, that the model can also be used to transform to and from database or other file schemas, which will help his team with its legacy integration work. Finally, he has learned that there is now specific guidance available for application integration, guidance that will replace the inconsistent naming conventions and patterns for integration using the enterprise service bus in his organization, with something that is consistent and repeatable, that supports both his legacy JMS integrations, but also his more recent integrations using web services. Finally, although his utility has recently been exploring the use of RESTful architecture and testing some XMPP protocols, the CIM profiles do not preclude using these types of integration technologies.

Section 11 Questions

1. The primary naming convention for CIM-based integration consists of a _____ pair
    a. Verb/Noun
    b. Command/Noun
    c. Noun/Object
    d. Integration/Object
2. The object used in a normal (non-event) response is a _____.
    a. Change
    b. Get
    c. Create
    d. Reply
3. The verb used for queries is…
    a. Change
    b. Get

c. Create
d. Reply

4. The verb used to alter an object once it has been created is…
    a. Change
    b. Get
    c. Create
    d. Reply
5. A CreateEndDeviceControls was used to initiate a command to a device. Later after some processing, some additional information needs to be passed to the calling system. This will most likely use the _____ verb.
    a. Get
    b. Change
    c. Created
    d. Create

12. COMPLIANCE TESTING

Learning Objectives

- Understand how a lack of compliance testing poses a challenge to CIM-based integration
- Get up to date with efforts to initiate and support compliance testing programs for CIM
- Understand the role of utilities and vendors in testing and measuring compliance

Identifying the Actionability Gap

The CIM users' community has recognized for many years the need to develop a compliance testing approach. CIM as a family of standards is well accepted due to its robustness, flexibility, and extensibility, but lack of means for certification and testing continues to present a major obstacle to wider CIM adoption. As mentioned earlier, CIM is a reference model; as such, implementation requires interpretation when applied to an integration project within a specific context.

The working groups that support the CIM strive for it to be both backward-compatible and to offer maximum flexibility. This is why, within the CIM, many of the attributes in the CIM are optional. This flexibility is a benefit, but for a more specific business process to be successfully completed, an attribute might be required.

Certification and testing programs are the solution needed to remove ambiguity and reduce confusion – particularly within less mature portions of standards that are open to interpretation or optional. End users (utility IT organizations, systems integrators, and project teams) need to verify that vendor claims are valid and to ensure that system integration costs will capture the benefits made available by standards-based integration. Vendor products that have been certified as compliant with CIM can help utilities reduce costs.

In early 2016 EPRI published *A Call to Action: Certification Testing for the Common Information Model*. EPRI noted that setting appropriate expectations and interpretations of a standard for implementers presents an "actionability" gap within the CIM standards development cycle. EPRI identified test scripts and certification as the missing links and challenged the industry to develop test scripts for key use cases. Test scripts specify the required information for any given interaction between systems to create the predictable and measurable behaviors that can be tested and measured for compliance.

Completing the First CIM Compliance Test

EPRI completed the first CIM compliance testing event in October 2017. EPRI worked with vendors and utility observers to test messages used to support software integration of distributed energy resource management systems (DERMS). This set of messages allows utilities and vendors to integrate in a standardized way, regardless of whether the DERMS is a standalone system, part of an Advanced Distribution Management System (ADMS), or a third-

party aggregator providing a hosted solution in the cloud. The use case was designed to test compliance with IEC 61968-5: Distributed Energy Optimization.

The EPRI test script includes WSDLs and XSDs that can be used to develop standards-compliant web services. The test script can be used further to test whatever software is developed using the WSDL and XSD artifacts, to ensure that it is in fact compliant with the standard.

For example, utilities can use it to verify vendor claims of standards-compliance when acquiring a Distributed Energy Resources Management System (DERMS) or a Distribution Management System (DMS) that also includes DERMS functionality, or systems that would interface with a DERMS.

At this writing EPRI and UCAIUG are working toward an expanded CIM compliance test in 2022.

Case Study

Jeff Kimble has made great progress toward his goal of implementing model-driven integration. With a CIM-based enterprise semantic model in place and a number of adapters completed for some quick-win business project, his customers are beginning to see the promise of faster and less costly system roll-outs. The organization's internal capabilities are maturing, and Jeff can envision the day when model-driven integration is well-understood – something practiced on every project, not just a nice concept.

One thing has been bothering Jeff though. As business leaders bring him into conversations with potential vendors, he often hears claims that software products are "CIM-compliant." He knows that CIM is a reference model, and that even often-used CIM profiles leave the integration developers with questions.

Jeff learned about an EPRI-sponsored CIM interoperability test and signed up as an observer. He quickly gained new perspectives while interacting with developers from different vendors and seeing the differences in their implementations of the same standard, for the selected use case. He was impressed with how quickly the vendors were able to identify the issues and move to clarification. The testing process methodically isolated ambiguous areas in the script as well as code bugs. Jeff was pleasantly surprised when at the conclusion of the test, the vendors were able to disconnect from the test harness and then connect with each other's systems and successfully exchange messages, using the compliance-tested CIM adapters.

Section 12 Questions

1. A lack of compliance testing poses a barrier to CIM adoption because:
    a. Vendors can't get their customers to pay for the extra costs of testing.
    b. The CIM model includes both normative and optional attributes, and project managers want answers, not options.
    c. Flexibility is required for standards to be widely useful, but without compliance testing to drive more specificity, CIM is just not actionable.
    d. Combinations of the above.
2. CIM interoperability testing programs will produce:
    a. Less vendor responsiveness, since they will expect the utilities to write their own adapters
    b. More influence for utility architects and project managers as they write specifications around script-tested standards
    c. More revenue for system integrators
    d. All the above
3. CIM Interoperability testing processes can be considered:
    a. Negligibly influential
    b. Likely to grow significantly in the future
    c. Critical for a transforming energy industry
    d. All the above

Additional Resources

These test scripts are informative only. These resources are EPRI interpretations of how a test script might be constructed that could indicate some level of conformance with the intention of the CIM standards. These scripts have been contributed to the UCAIUG under an Apache license. That being said, these test scripts can provide insight into what may be expected, and also contain artifacts such as XSDs, that can assist a developer and accelerating their understanding of how CIM-based data integration is accomplished. For the latest information regarding CIM testing, please see the CIM Users Group for more information.

1. *Test Script for International Electrotechnical Commission 61968-5 Distributed Energy Optimization, Third Edition* (3002014703)
2. *Program on Technology Innovation: Test Script for International Electrotechnical Committee 61968-6:FDIS Messages* (3002006862)
3. *Program on Technology Innovation: Test Script for IEC 61986-9 Meter Reading and Control* (3002014618)

13. WEB SERVICE INTEGRATION

Learning Objectives

- Understand at a high level the difference between Simple Object Access Protocol (SOAP) and REST for CIM messaging
- Understand at a high level the difference between XML and JavaScript Object Notation (JSON) for CIM messaging
- Understand the pros and cons of when to use SOAP/REST with XML/JSON payloads

RESTful and SOAP Web Services

The CIM is just an information model and can be implemented in multiple ways, many of which were described in Section 11. One of the most common implementation methods for exchanging data from machine to machine is a web service. Two main types of web services exist: SOAP and REST.

SOAP is a messaging protocol that uses XML for the machine-to-machine message exchange. SOAP web services are contract-based, making use of WSDL files and XSD files to define what messages can be exchanged. Desired for its security and language independence, SOAP can be used in both publish-subscribe and request-response messaging patterns.

The SOAP protocol can be more complex to both consume and create because of its required usage of WSDLs and XSDs. Many tools exist in multiple programming languages to assist with the development of these services, such as Microsoft's *svcutil* and the Apache CXF[50] framework.

For the purposes of exchanging messages using CIM, IEC 61968-100 primarily focuses on using SOAP as the exchange message between two systems. The IEC 61968-100 standard defines a Header element that contains a noun (for example, *MeterReading*) and verb (for example, GET, CREATE, or DELETE) to address SOAP's inability to use these Hypertext Transfer Protocol (HTTP) methods. This is because even though SOAP uses HTTP as one of its transports, SOAP is not limited to HTTP only.

REST is an architectural style for designing applications that communicate over HTTP. Web services that follow this style are referred to as *RESTful web services*. The style is distinguished by treating every object as a (web) resource with a Uniform Resource Identifier (URI) and limiting the web service actions to the HTTP methods—most frequently GET, PUT, POST, and DELETE. Working within this seemingly limited framework can produce interfaces that are uniform, scalable, and able to handle the inherent unreliability of network traffic. RESTful web services can exchange payloads in a variety of different formats, such as XML, JSON, plaintext, and Hypertext Markup Language (HTML). Most REST APIs use JSON for incoming requests and

[50] *Apache CXF is a registered trademark* of the *Apache* Software Foundation.

outgoing replies. As a result, RESTful web services are usually faster than SOAP web services because of the lightweight nature of JSON compared with XML.

REST has no contract such as SOAP's WSDL defining the web services' supported operations, nor is an XSD or equivalent required to define the information being exchanged. It is, however, common for the information being sent to be compliant to either an XSD or JSON schema.

REST makes full use of all HTTP verbs: GET to retrieve data, POST to create an object, PUT or PATCH to modify an object, and DELETE to delete an object.

REST web services are light and usually have lower overhead compared with SOAP web services. SOAP loses some speed because of its type checking against the contract, enforcing Web Services Security (WS-Security or WSS), if enabled, and then its markup of the response and header it generates in XML. With modern technology, however, the speed difference is negligible in real time.

JSON Versus XML

Extensible Markup Language

XML is a markup language, both machine-readable and human-readable, that describes data in a self-descriptive way. XML can be used for both the storage and transportation of data, using tags such as those used in HTML. Unlike HTML, XML is designed to convey rather than display data. Also, unlike HTML, XML contains no pre-defined tags: it is up to the author to define tags.

XML is both hardware- and software-independent and is used in SOAP web services. General guidance on the CIM uses XML examples because of IEC 61968-100's primary focus being on SOAP over HTTP or JMS.

XML's advantages include the following:

- **Its attributes**. XML uses attributes to display metadata about an object. JSON cannot do this without creating a new key-value pair object, which would then be read by the receiving program as another separate object.
- **Its namespaces**. Namespaces enable the reuse of the definitions—for example, data types from other documents. Namespaces can help prevent name conflicts from occurring, especially when multiple schema files are used together to define a payload.
- **Its XML schemas**. XML documents can be described using XSD. The schema definition defines elements and attributes that must, or can, be present in a document. In this way the XSD represents a contract that document consumers can use to validate the document and write code and processing logic.

JavaScript Object Notation

JSON is a lightweight, text-based data exchange format. It originated from the programming language after which it was named, JavaScript; however, JSON data can be written in any programming language. JSON data come in what is known as *name/value pairs*, with curly braces containing objects, commas separating different aspects of that data within the object, and square brackets containing arrays. Like XML, JSON is a programming language-independent data format, making it a popular choice to use in REST web services because of its lower overhead compared with XML.

JSON's advantages include the following:

- **Its verbosity**. JSON does not have closing tags like XML, making it more compact.
- **Its readability.** The lack of closing tags makes JSON more human-readable because of a lower amount of overall clutter. It also does not have nested attributes within a tag.
- **Its speed**. JSON is not only faster to process because of its smaller file size in comparison to XML but also because software libraries built to process JSON use less memory compared with those used to process XML. This advantage can be either great or negligible depending upon the application and the volume of data being transferred.

Like XML's XSD, JSON also has a vocabulary that allows for the annotation and validation of JSON documents. The JSON Schema, however, is still in its draft standard, which is Draft 2020-12 as of this writing. JSON Schemas, like XSDs, define elements (their type, their cardinality, their enumerated value, and so forth) that must be present in a message.

Mapping CIM Message from XML to JSON

The CIM is not constrained to a specific data format; however, it may become necessary to convert CIM messages from their XML format to JSON. The following sections detail how that can be approached for a few different scenarios. Note that a full conversion is not always necessary except to preserve the integrity of the original message, because some parts of XML, such as namespaces, serve no purpose in JSON.

Sample Header Message

The following is a sample header message in XML and then its equivalent in JSON:

```
<Header>
 <Verb>get</Verb>
 <Noun>MeterReadings</Noun>
 <Timestamp>2020-05-04T15:10:00Z</Timestamp>
</Header>
```

The equivalent in JSON would then be

```
"Header": {
  "Verb": "get",
  "Noun": "MeterReadings",
  "Timestamp": "2020-05-04T15:19:00Z"
 }
}
```

Arrays, Namespaces, and Attributes

Arrays in JSON use the bracket notation typically used for arrays in most programming languages. The following is an example of two meters in SOAP XML:

```
<Payload>
 <MeterReadingsxmlns="http://iec.ch/TC57/2011/MeterReadings#">
  <MeterReading>
   <Meter>
    <Names>
     <name>metername1</name>
     <name>metername2</name>
    </Names>
   </Meter>
   <Readings>
    <timeStamp>2020-10-04T14:00:00Z</timeStamp>
    <value>2.71828</value>
    <ReadingType ref = "0.0.0.1.1.1.12.0.0.0.0.0.0.0.0.3.72.0"/>
   </Readings>
  </MeterReading>
 </MeterReadings>
</Payload>
```

The JSON equivalent to this message would have to make sure to handle both the attribute and reference slightly differently, like so:

```
{
 "Payload": {
  "MeterReadings": {
   "@xmlns": "http://iec.ch/TC57/2011/MeterReadings#",
   "MeterReading": [
    {
     "Meter": {
      "Names": {
       "name": [
        "metername1",
        "metername2"
       ]
      }
     },
     "Readings": {
      "timeStamp": "2020-10-04T14:00:00Z",
      "value": 2.71828,
      "ReadingType": {
       "@ref": "0.0.0.1.1.1.12.0.0.0.0.0.0.0.0.3.72.0"
      }
     }
    }
   ]
  }
 }
}
```

These are direct mappings only. A true RESTful implementation would have no need for Part 100's Payload object and would have a Resource going straight to MeterReadings.

Security

With any web service that has a public-facing API, security should be a top concern. Although REST and SOAP web services support secure sockets layer (SSL), SOAP has an additional layer of support known as *Web Services Security*.

SOAP has additional protocols on top of its underlying transport layer for reliable and secure delivery of messages. SOAP's WS-Security protocol's main purpose is to provide XML encryption to messages being sent. Another protocol, WS-Addressing, is a standardized way of including message routing data in SOAP headers instead of relying on network-level transport to communicate routing information. WS-ReliableMessaging is a protocol that allows messages to be reliably transferred between systems, regardless of software, system, or network failures.

Although SSL is typically enough for most use cases, some can require an added layer of protection.

Case Study

At Electric Innovation Utility, Jeff Kimble knows that his team does not have to stay locked into using SOAP for web services even though it is still in use today by legacy systems. His team already uses REST for internal services, and applying that that to enterprise communications should be an achievable goal. He realizes now that there are valid reasons to use SOAP for certain services, whereas others might work better using a RESTful architecture. He has learned that the CIM model not only allows for the use of other data formats but that conversion between the two is certainly feasible. In fact, he thinks he saw a tool for converting his XSD schemas into JSON schemas to help speed along the process.

Being at a utility, however, Jeff realizes that security is of utmost importance. Thankfully, REST offers all the security that HTTP itself has to offer, including SSL protection for conveying sensitive information. Although Jeff is leaning toward embracing JSON for this new endeavor, he takes solace in the fact that XML is still a viable transport for his upcoming REST implementations.

Additional Resources

The EPRI Enterprise Architecture and Integration project set has published the following research on best practices for web services management, versioning, and development:

- *Program on Technology Innovation: Web Services Robustness, An Agile Integration Ecosystem* (3002017495)
- *Service Oriented Architecture – Versioning and Maintenance* (3002007876)
- *CIM Identities Web Services* (3002011464)

Section 13 Questions

1. Which is true when comparing SOAP and REST?
    a. REST is more secure than SOAP
    b. REST can only use JSON for its payload, whereas SOAP can only use XML
    c. REST and SOAP are both protocols for transporting data
    d. REST has less overhead and is much faster than SOAP
    e. REST typically requires more bandwidth
2. The most common payload for REST is:
    a. XML
    b. JSON
    c. Plain text
    d. HTML
    e. All the above
    f. None of the above
3. The primary payload for SOAP is:
    a. XML
    b. JSON
    c. Plain text
    d. HTML
    e. All the above
    f. None of the above
4. True or false: The CIM series of standards only supports SOAP and XML.
    a. True
    b. False
5. SOAP might be the best option when a user needs:
    a. More security
    b. The ability to use multiple payloads besides XML
    c. High scalability and performance
    d. Both b and c
    e. All the above
6. REST might be the best option when a user needs:
    a. More security
    b. The ability to use JSON or XML for message payloads
    c. High scalability and performance
    d. Both b and c
    e. All the above

14. CREATING SCHEMAS FROM THE CIM

Learning Objectives

- Understand the pros and cons of three different tools available for generating schemas (e.g. RDFS, XSD, JSON) from the CIM UML
- Learn how to generate schema or API definitions from the CIM UML

Going Beyond the XSD

Section 8 lists several free, open-source, and commercial tools available to automate the mapping of CIM profiles to XML. As the industry continues to move toward RESTful web services using JSON, however, tools that can support both XML and JSON schema formats will see more use from CIM developers. Analogous to the IEC 62361-100 standard for XSDs, the draft standard IEC 62361-104 is currently in development for JSON schema. Unlike the WSDL and XSD formats that have remained largely the same over the past decade, JSON Schema standards are still evolving, with new updates being released frequently with no guarantee of backwards compatibility.[51]

Three common tools for generating XML and JSON schemas are discussed in detail in this section:

1. Schema Composer by Sparx Systems Pty Ltd.
2. CIMConteXtor by Zamiren, Paris
3. CIMTool by CIM Users' Group

While this list is only a subset of the tools available for generating schemas from the CIM UML, an overview of these will provide a foundation for learning the intricacies of similar software. Each segment below lists the pros and cons of a tool and provides a short walkthrough on how to use it to generate whichever type of schema or other documentation is desired. (The section on CIMTool will have a minimal walkthrough of its use since this was mostly covered in Section 8).

Due to the regularity of JSON schema version updates, having tools that stay up-to-date to address fixes or updates can be a deciding factor for many developers. Schema Composer updates will occur on a fixed schedule alongside other Enterprise Architect updates. CIMConteXtor and CIMTool, due to their open-source nature, may receive updates as quickly as possible or never at all.

[51] See "Use of the *draft* designation in https://json-schema.org/. For more information about the specifications themselves, see https://json-schema.org/specification.html.

Schema Composer

Schema Composer is a tool built into the Corporate, Unified and Ultimate editions of Enterprise Architect from Sparx Systems. Unlike the other tools mentioned below, Schema Composer is not open-source. Schema Composer is also not specific to the CIM. The Sparx documentation lists other standards that it also supports. Because Schema Composer is built into Enterprise Architect, it does not require steps that other tools do, such as the exporting of XMI files, or installing additional plug-ins from third-party resources. This tool would also be ideal for those working in the other standards that Schema Composer supports.

Unfortunately, the schemas that Schema Composer generates do not follow every rule according to the IEC 62361-100 specification. For example, the IEC 62361-100 states that elements should be ordered with the mRID first, then other simple properties in alphabetical order, and then object and compound properties in alphabetical order. Although Schema Composer follows the first rule, it does not adhere to the others. As a result, Schema Composer in its current form requires some hand-editing of each schema after it has been generated to properly adhere to the IEC standards. However, the convenience of being able to use a built-in tool backed by professional support may potentially outweigh these shortcomings for some profile creators.

Schema Composer also supports JSON Schema and RDF Schema, although its support of JSON Schema as of July 2021 is Draft-04, four versions behind the most recent version listed at https://json-schema.org/specification.html.

How-To: Schema Composer

A class can be drag and dropped directly from the model's project browser into Schema Composer. Upon doing so, it will automatically populate a full list of its associations, attributes, and inherited classes.

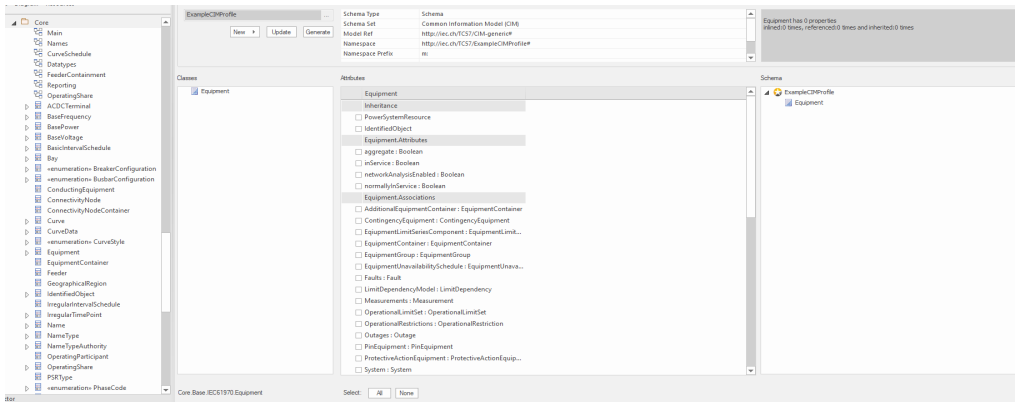


Figure 14-1. Adding a class to the new profile from the 61970 package

Checking each associated class or complex object will automatically add that class to the table view so that its appropriate attributes can be checked.

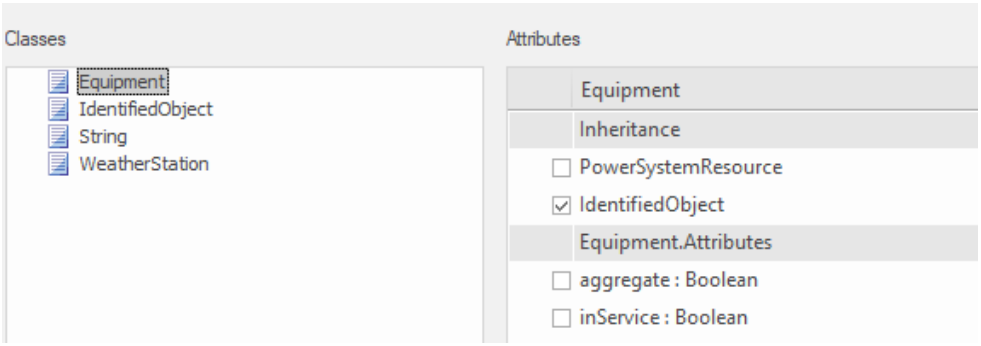


Figure 14-2. Schema Composer adding classes based on options chosen

Schema Composer allows the user to right-click and set cardinality and other constraints.

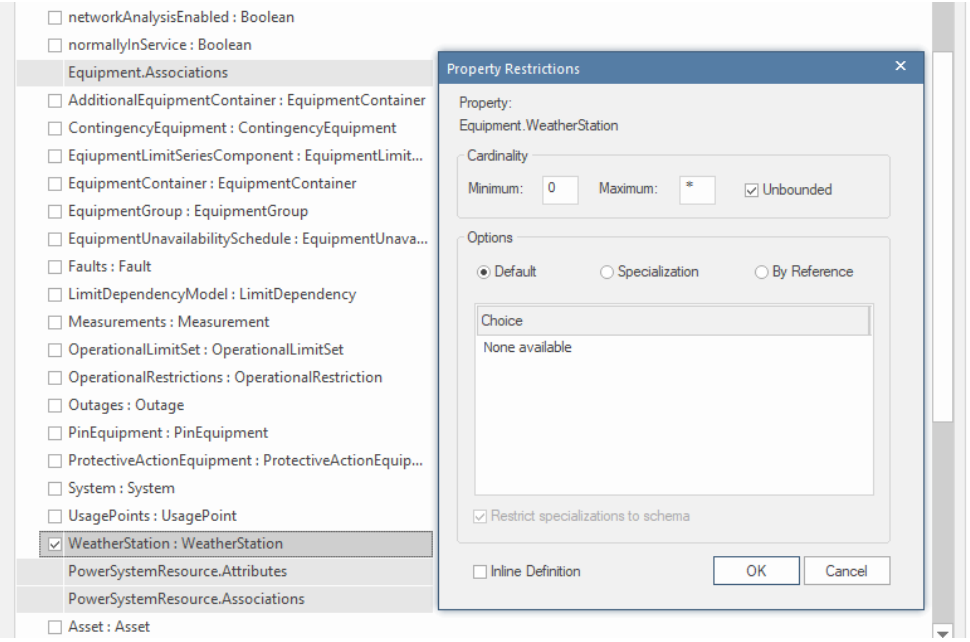


Figure 14-3. Setting property restrictions in Schema Composer

When generating a profile, options include RDFS, JSON Schema, and XSD. Currently this only supports Draft 04 of JSON Schema.

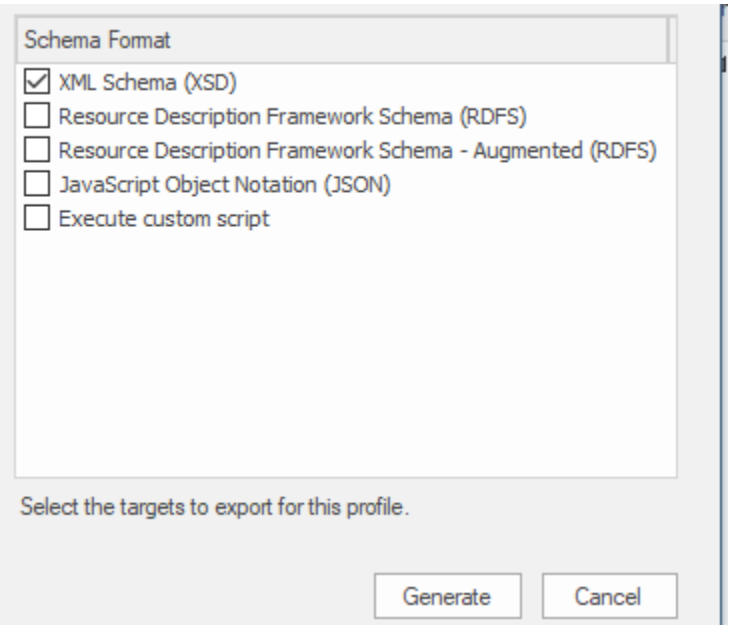


Figure 14-4. Schema Composer schema selection options

Schema Composer Summary

Although the schemas created with Schema Composer are not perfect, this tool has great potential if it continues to see support by Sparx and the IEC community. Presently, however, the schemas it creates are not 100% IEC compliant and would need to be hand-edited to ensure proper interoperability. This may or may not be a concern depending on the target application. For those who want to do everything within Enterprise Architect, Schema Composer conveniently allows for the creation and saving of profiles within Enterprise Architect.

CIMConteXtor

CIMConteXtor by Zamiren is a free, open-source plug-in for Enterprise Architect. It comes in two parts: CIMConteXtor which allows the creation and management of UML profiles, and CIMSyntaxGen, which generates the schema from those models according to IEC standards such as IEC 61970-501. The developers of CIMConteXtor have been working together with the developers of CIMTool to add JSON Schema support, following the draft standard IEC 62361-104. Unlike both Schema Composer and CIMTool, CIMConteXtor profile generation results in a graphical view like that of the UML model instead of the table view of the former two. This makes it easier to see the schema structure at a quick glance.

The free, open-source nature of CIMConteXtor means it must be downloaded from a site not affiliated with Enterprise Architect. It also requires the installation of Python and specific

libraries used for the plug-in. For people with strict security policies in place at their workplace, this may result in a barrier to getting started.

How-To: CIMConteXtor

After going through this short walkthrough, consider reading the software manuals provided directly on the CIMConteXtor website. These manuals discuss every nuance of the tools and modeling in Enterprise Architect itself. This how-to guide is meant to serve as a starting point for those who wish to use the tool for their own profile generation. For additional details on installing, manuals are provided on the downloads page of CIMConteXtor's website.

To start, create a new package at the project level in Enterprise Architect, right-click, select CimConteXtor, and CreateProfilePackage.

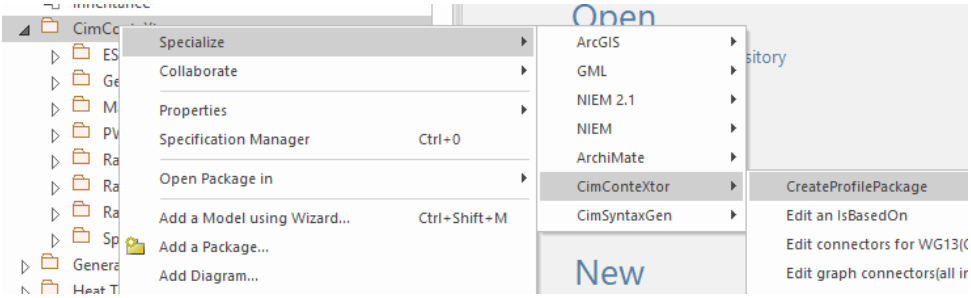


Figure 14-5. First step in creating a profile using CIMConteXtor

For this example, an extension to the CIM has been made that incorporates parts of IEC 61970 and IEC 61968, so in the next dialogue, those will be selected in addition to the new extensions package.

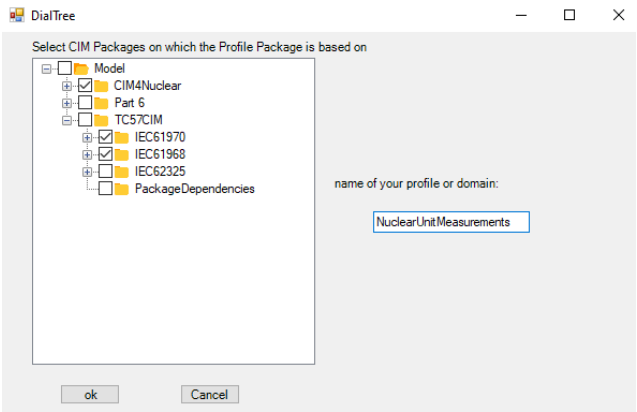


Figure 14-6. Choosing which CIM packages will be used

Like Schema Composer, classes can be dragged and dropped directly from the project browser. This time, instead of dropping them into a table, it is dropped onto the new empty diagram that was generated upon completing the last step. A message will pop up from CIMConteXtor. Click "Yes."

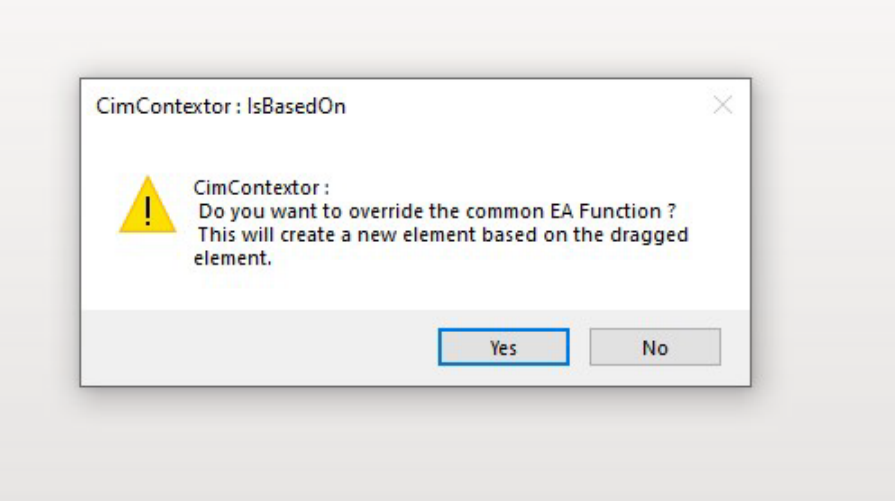


Figure 14-7. Dialogue from CIMConteXtor to create copy of CIM class for its profile

The next box that opens will have checkboxes for selecting desired attributes. This does not list any of the class associations which are set in a different step. If this is going to be the root of the XSD or JSON Schema, check the box on the right that says, “is root (active).” Click “Execute IsBasedOn” to finish generating the class in the profile.

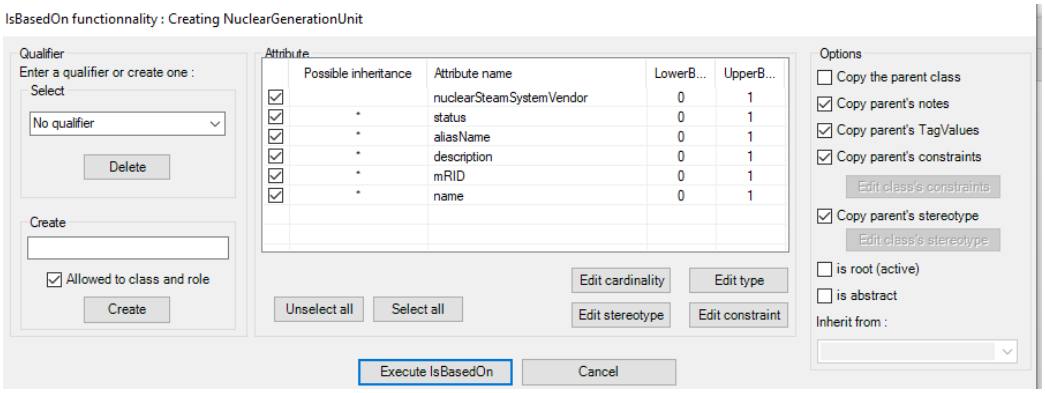


Figure 14-8. Setting the attributes in a class using CIMConteXtor

After this class has been created, repeat the previous steps and drag another class that is associated to the root onto the diagram. Once it has been placed, right-click the class that was just added and select “Edit hierarchical connectors” to establish the association.

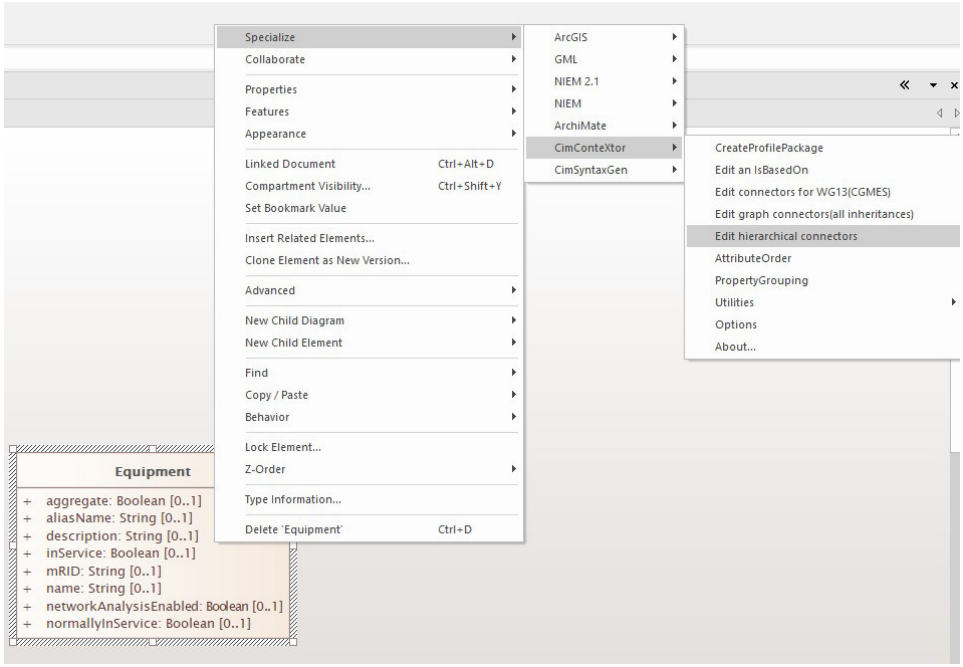


Figure 14-9. Adding the association between two classes

Pick the correct class to associate it with from the box that appears. Use the end role name to know for sure which version of the class should be picked.

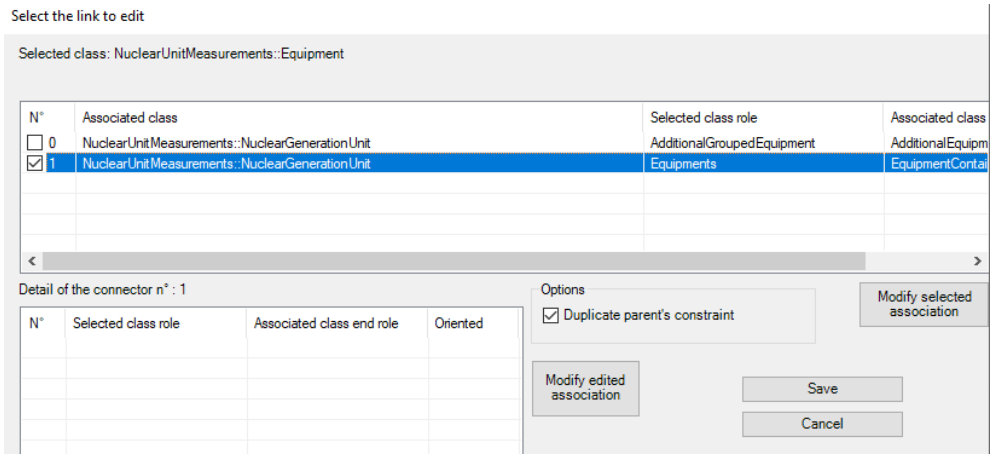


Figure 14-10. Establishing the association between the added class and its parent

With that row highlighted, select “Modify selected association.” This is where a decision must be made about whether the profile will be used to generate RDF Schema, or XSD/JSON Schema. If the former is going to be used, no change is necessary here. However, if XSD or JSON Schema is the desired end result, then check the box labeled “Containment” under the parent class to define a unidirectional association between the classes.

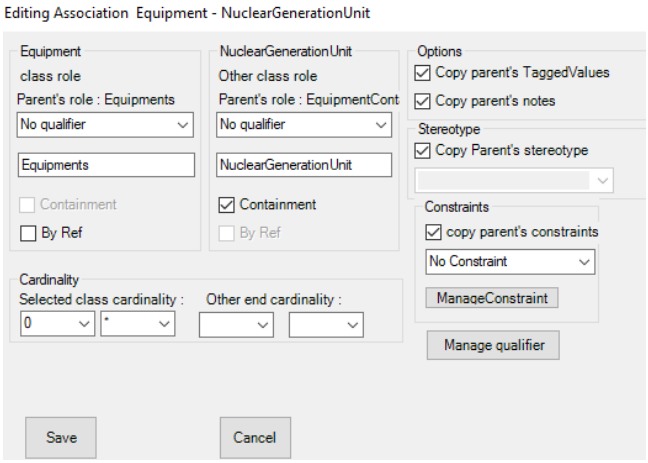


Figure 14-11. Selecting containment options for class associations

From this point on, the process repeats until all classes for the desired profile have been added. The final result will be a view not unlike that of the profiles in the information model packages themselves.

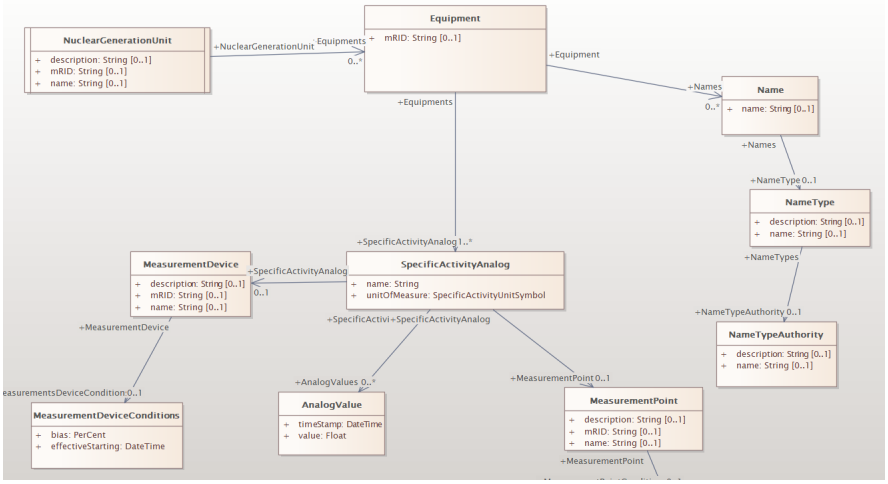


Figure 14-12. Sample completed profile view in CIMConteXtor

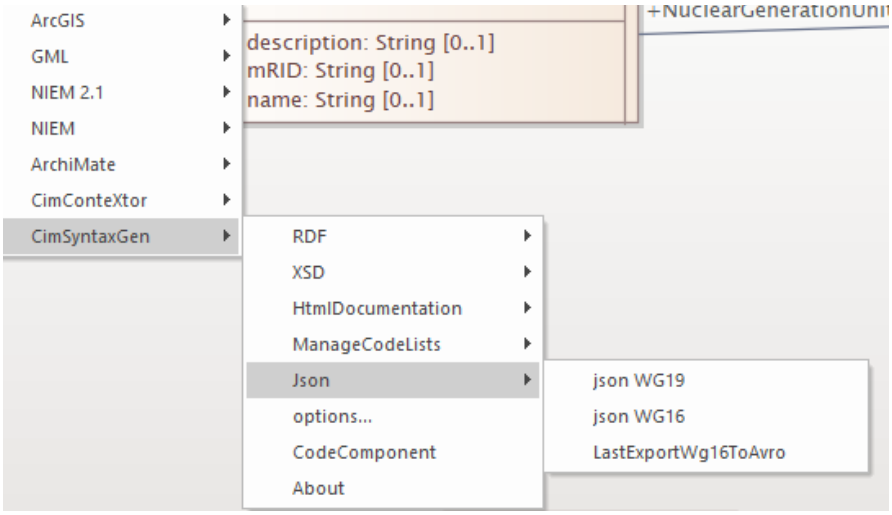


Figure 14-13. CimSyntaxGen JSON Schema example

For the next step of generating a schema, the CimSyntaxGen plug-in must be used. Like CIMConteXtor, this also has its own manual to goes into deeper detail. To generate a schema, right-click the profile package and select CimSyntaxGen, then pick your schema of choice.

The next dialogue will look slightly different depending on the format chosen, but in all cases there are text fields that must be filled out with information such as the root node name, the file name and where to save it, which version of JSON Schema should be used, and what the namespace and namespace prefix should be.

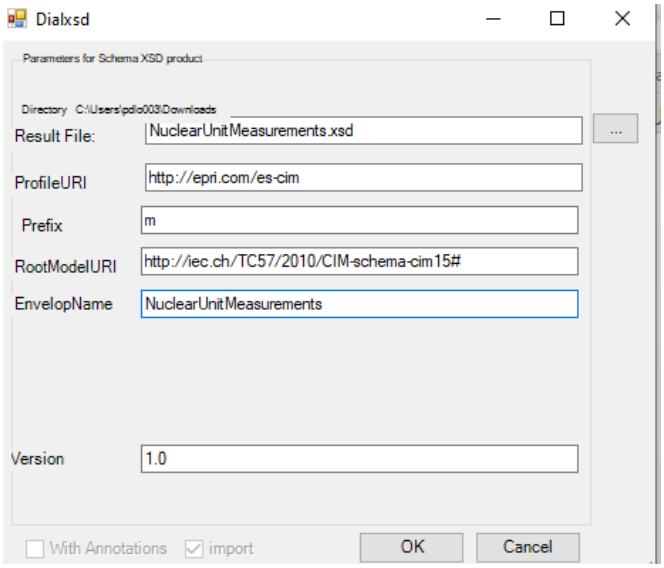


Figure 14-14. Setting namespace, URI, and filename for schema

The last thing the tool will ask is about the cardinality of the root node. Provide that and the schema will be complete.

CIMConteXtor Summary

CIMConteXtor is a good alternative to CIMTool for those who don't want to leave Enterprise Architect while creating their profiles. The schemas it creates conform to the IEC standards for XSDs and the IEC draft standard for JSON Schema. The developers of CIMConteXtor have been working together with the CIMTool team to ensure both tools remain in lockstep with the draft standard for JSON Schema. Its graphical profile view can be appealing to those who prefer a visual representation of the schema.

CIMConteXtor may not be a viable option if security policies prevent the installation of a third-party program that writes its data to the AppData folder in Windows. It also currently requires administrator access to generate XSDs so that it can access XSDexample.xsd within the Enterprise Architect directory in Program Files.

Like Schema Composer, CIMConteXtor allows for the creation and saving of profiles directly within Enterprise Architect, making it ideal for those who want to keep to one environment when creating CIM schemas.

CIMTool

Although CIMTool was discussed in detail in Section 8, it is worth mentioning again here due to recent changes. As of May 2020, a new version of CIMTool[52] was released that supports the IEC 62361-104 draft standard for JSON Schema. Development of CIMTool is currently managed as part of the UCA CIM Users' Group Open Source Initiative.

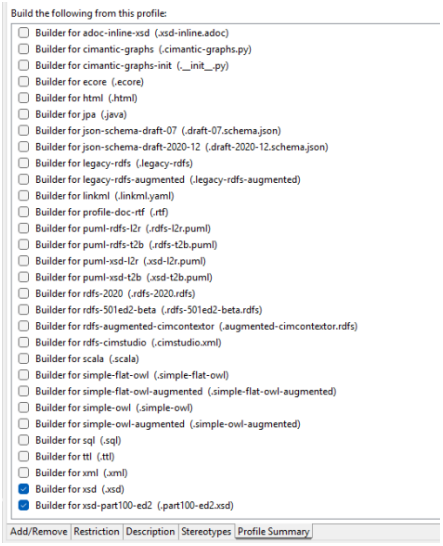


Figure 14-15. A list of builders CIMTool supports

[52] The latest version of CIMTool can be accessed on GitHub at https://github.com/cimug-org/CIMTool/releases/ , with its primary website accessible at https://cimtool.ucaiug.io/ .

CIMTool and Schema Composer share some similarities when it comes to the design of the schema using table views. The main difference is in the final form of the schema that is produced, with the one from CIMTool currently adhering to the IEC specification for all schema formats. Unlike Schema Composer where the schema is designed within Enterprise Architect and saved in its own package, CIMTool requires that the CIM model be first imported into CIMTool for use as the baseline from which to derive profile definitions used in generating RDFS, XSD, JSON and other schema-based artifacts. The base CIM model has historically been imported into CIMTool as an XMI file. Recent enhancements introduced in CIMTool 2.2.0 now include support for importing and directly using either native 32-bit Sparx EA CIM project files (.eap) or 64-bit EA project files (.qeap).

Summary

Each tool used to generate schema from the CIM UML can work depending on its intended use. Below is a table summarizing the capabilities of each to help determine which tool might work best.

Table 14-1. A comparison of tools used to generate schemas in the CIM

	Schema Composer	CIMConteXtor	CIMTool
Generates XSD	Yes	Yes	Yes
Generates JSON Schema	Yes (limited)	Yes	Yes
Generates RDFS	Yes	Yes	Yes
View	Table	Graphical	Table
Open-Source	No	Yes	Yes
IEC 62361 Compliant	No	Yes	Yes
Supports Other Non-CIM Standards	Yes	No	No
Profile Configuration Export	No	No	Yes
Profiles Saved to EA	Yes	Yes	No

Case Study

Jeff Kimble's enterprise application team has begun making the transition from SOAP to REST for many of their services. Jeff has just learned that many of the tools used to generate schema from the CIM information model now support JSON Schema. He was also happy to learn that these tools have added support for generating HTML documentation of their services, so that their public-facing webpage can provide details on the API.

As Jeff's team continues to extend the CIM for their own internal needs, he feels confident that they can do so for other formats besides XSD in a way that adheres to the IEC specifications.

Section 14 Questions

1. The specification that guides how the CIM UML is mapped into XML Schema messages is:
    a. IEC specification 62361-100
    b. IEC draft specification 62361-104
    c. The CIM
    d. XML
2. All schemas generated from the CIM UML are correct as long as the classes and attributes come from the CIM.
    a. True
    b. False
3. There is an official tool that must be used when creating schemas from the CIM UML.
    a. True
    b. False
4. Of the three tools described in this section, which one requires a separate profile definition for RDFS?
    a. Schema Composer
    b. CIMConteXtor
    c. CIMTool
    d. None of the above
5. Of the three tools described in this section, which one requires an export of the information model from Enterprise Architect?
    a. Schema Composer
    b. CIMConteXtor
    c. CIMTool
    d. None of the above
6. Only official CIM classes can be used with these tools, not extensions.
    a. True
    b. False

15. IMPLEMENTING CIM IN THE ORGANIZATION

Learning Objectives

- Understanding the pitfalls associated with CIM implementation on an individual organization basis
- Understanding the critical success factors that make the CIM a winning strategy over time

Rome wasn't burnt in a day[53]. A cautionary tale to keep in mind when implementing any change – even ones that have a clear track record of success elsewhere. Done incorrectly, such an implementation attempt can create frustration and not just doom this effort, but also future attempts to leverage standards or other change initiatives.

"People are resistant to change". This is often the mantra of people attempting to implement change – and they are correct - if a change initiative is done poorly. Luckily there are resources available to help guide a change effort. For example, John Kotter is the leading author on the topic of change management and his book, *Leading Change*[54], discusses an eight-step model for any change effort.

Kotter's modified eight-step model is shown in the figure below.

[53] The Great Fire of Rome occurred in July of AD 64. The fire was brought under control after six days but managed to reignite and burned for three more. When it was finally controlled only a third of Rome remained standing. https://www.nationalgeographic.org/thisday/jul19/great-fire-rome/

[54] Kotter, J. P. (2012). Leading Change. Harvard Business Review Press; 1R edition.

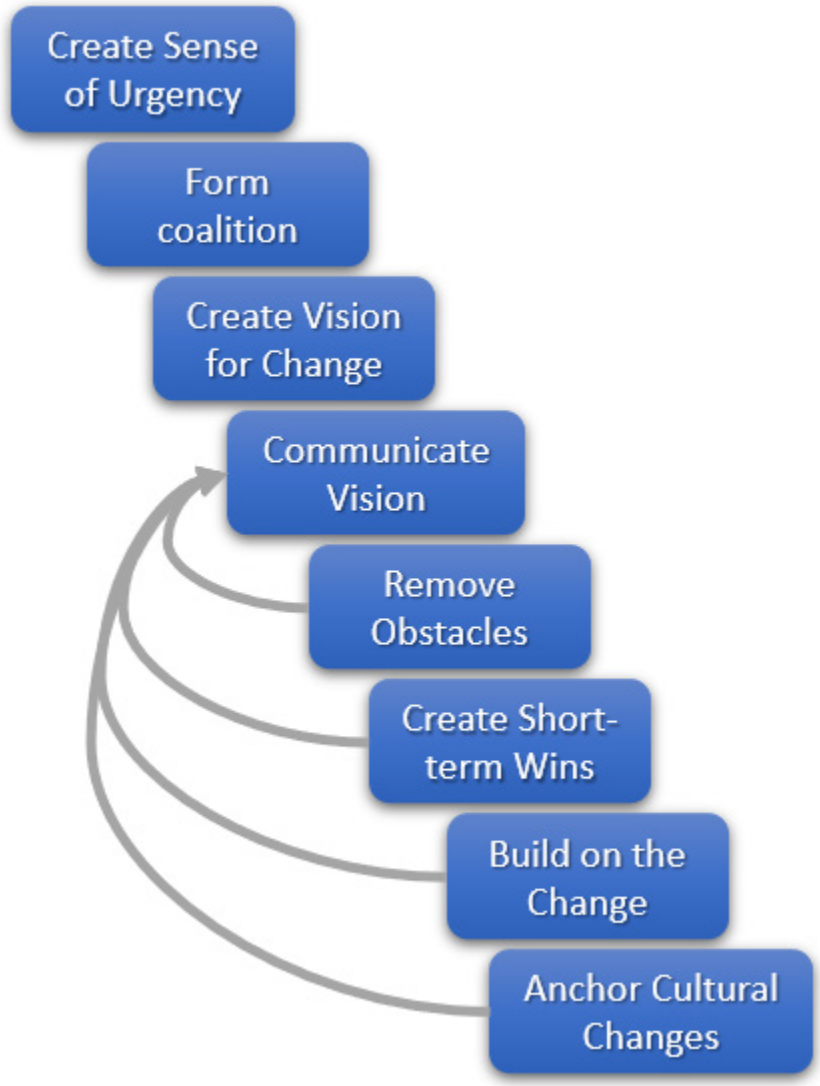


Figure 15-1. Kotter's (modified) Eight-Step Change Process

The process was modified to reflect that once the vision has been created, in each of the successive steps, that vision should be recommunicated. To truly get to step eight, where a standards-based approach has been anchored in the culture – the vision that describes why the organization undertook the change must be reemphasized continuously. Some people may have not been reached the first time around, some people may have resisted the effort if the change didn't directly impact them. It is always good to reinforce the value that the organization is receiving – especially when there exists an organizational impulse to act first and plan later.

1. Create a sense of urgency

   Avenue for Conversation: Dissatisfaction with the Status quo

   To create a sense of urgency, one needs to create a dissatisfaction with the status quo. Sometimes this can be difficult to do. People go about their day-to-day jobs and may not take the time to think about "is there a better way?" There is only the job and how they were trained to do it. But with a little prodding, with some well-placed questions, there are opportunities to raise awareness. For example, a utility might recognize that their GIS data could be better – that there are gaps; equipment that may be connected on phases that differ from the as-designed, assets that are mislocated, or assets that may differ from as-designed. One driver for this can be changes that were made during storm damage recovery that were accurately recorded. Other errors might be introduced when maps and other designs are moved from system to system or person to person within the enterprise. Sometimes merely asking, "Are these errors tracked?" or "Do we know what these errors cost us?" can be enough to raise awareness.

   Avenue for Conversation: Friction between IT/OT

   Another potential avenue for a conversation about the CIM, is to understand if there is friction between IT and OT related to data integration. Past friction may have been the result from systems being acquired without data integration costs being fully understood. "We paid <insert amount here> and it costs us <x times more> once it was fully installed." If there is dissatisfaction with such a scenario, the change agent may be able to take advantage of the situation to discuss the advantages associated with a standards-based approach to data integration.

   These kinds of conversations can bring up what is referred to as the "WiFi cost model" of data integration. Have you ever noticed that if you stay in a lower cost hotel room the WiFi is free? But in an expensive hotel room, you pay extra for WiFi? That's the WiFi model when it comes to data integration. If you talk to small utilities that don't have the development staff, often their expectation is that the data integration comes fully baked into the acquisition cost of a product. They have neither the time, nor the staff, nor the funding for big integration projects or customized code.

   Avenue for conversation: Never waste a good failure.

   Has there been an implementation project that may have missed its mark, with budget overruns, perhaps missed deadlines, or sacrificed functionality? This would be a good time to think about if any of these failures derived from underlying data integration or data definition issues that could be addressed by a standards-based approach.

2. Form a coalition

   If you're frustrated by the costs and complexity with integrating large enterprise systems, chances are you're not alone. There are other like-minded people in the utility, but they may work in other areas of the organization. The domain of data may differ from yours, but the same data integration challenges probably exist. There may even be people that are aware of other successful utility CIM projects (see CIM Success Stories in the appendix) who want to replicate that success in your utility. Find a champion in the senior management ranks and you may be on your way.

3. Create a vision for change

   Strategy and vision are things that can't be abdicated in favor of what your consultant thinks. For staff augmentation, consultants are a tremendous resource, but consultants tend to solve the last problem they worked on – and as the tongue in cheek quote says, "Consultants: If there's no solution, there's real money to be made in prolonging the problem". A bit facetious perhaps, but with the CIM there *is* a solution, but a solution that tends to reduce billable hours for consultants that specialize in data integration. Hence, all things being equal, such consultants may not be in favor of solutions that provide a roadmap for faster, more flexible integrations. This is where the organization needs to cast the vision, and then have the consultants (and the internal SMEs and stakeholders), execute upon it.

4. Communicate the vision

   Probably one of the biggest misconceptions is that communicating the vision is a "one and done" kind of deal. It is not. The vision must be constantly reinforced. It takes a while to change a culture and the easiest path is often to revert to the way things were done before – the known but broken. Thus, wins must be celebrated, communities of practice need to be engaged, and the guiding principles of standards-based integration reinforced. And in each of the successive steps described below, the communication of the vision must be made again and again.

5. Remove obstacles

   These might be the systems and tools that don't facilitate standards-based integration. These might be people and practices for example, naysayers[55]:

   "The CIM? Does anyone even use it?"

   "Yes, we're familiar with the CIM, but none of our customers ask for it."

   "We've heard of the CIM, but my understanding is that it hasn't been updated in a while."

   "Oh sure, we could use the CIM, but it changes all the time."

   "The CIM? Yes, we have one - but it's not the standard, we created our own."

   The common thread in these kinds of comments is that the people making them clearly didn't understand the state of the CIM and how it had been successfully used, or they have an agenda.

   Some examples:

   - Consultants might push for a product that they like because it's the product they know. This was seen at a utility where the consultants that were hired advocated for a vendor – which happened to be their prior employer. A reminder to ask your consultants about any challenges to their objectivity.

[55] All the following are quotes from real people

- Vendors that would rather lock you into their product via the integration. It is much harder and much more expensive to swap out to a competing vendor's product if you must redo all the interfaces. Is the data integration based on the whim of the vendor? Or are you setting the strategic vision?
- Vendors that either get an upsell on the data integration or their business model includes a partner that also profits on the data integration business.

As the CIM has matured as has the capability of the user community to provide data integration guidance and testing capabilities in addition to the CIM family of standards, less of this is seen – but the reader must be aware that these agendas may be out there.

6. Create Short-term Wins

One should keep in mind, the first iteration of implementing the CIM won't provide much value over that of using a customized solution – save usually for one big advantage:

> *No one must be locked into a room to debate what things are called, how they are defined, and how they are related.*

That difficult work has already been done. The users instead can spend that time understanding the model, what definitions already exist, and focus on small projects that the team can use to improve their familiarity of the CIM.

Also, just like with other disciplines related to software engineering, a little bit of training can go a long way. If you're reading this book, you may already be trying to develop your understanding. Participating in the "CIM University" (the first day of training when a CIM Users Group meeting is held), is another way to quickly be grounded in the CIM and its concepts.

The emphasis should be on creating a short-term win, some small data exchange – then build from there. As seen in the *Point-to-Point Standards Integration Cost Framework* report (see Additional Resources at the end of this chapter), the real gain of value occurs in each successive interface built using the CIM. With each new interface the developers gain knowledge of the underlying data model, and the formal design of the web service interfaces, accelerating their ability and being able to leverage prior interfaces using the CIM-based approach.

How to fail: Someone decides to "Big bang a CIM project". The thinking may be that "surely if a standards-based interface is good in one area, if semantic mapping is good for one – it must be good for everything". An initiative is then undertaken to convert all the interfaces of the organization to using the CIM.

Such a project is doomed to failure. Costs will be incurred with no incremental value. And often a large initiative like this will fail under its own weight.

Small, incremental changes, as new systems are brought online, is a better way to build maturity over time.

7. Build on the change

   This is an opportunity to celebrate the victory and once again, recommunicate the vision. Identify the next interface or the next use case where the CIM can be used. It is important to have tracked the costs of implementing the first change and be able to measure against a baseline because as these successive interfaces are built the incremental cost savings become part of the communication.

8. Anchor the changes in the culture

   Again, this comes down to communication: celebrating the victories and the people that made it happen. However, there can be another challenge with changing the corporate culture that can be described as overcoming the "run to failure mode of utility IT". Consider investor-owned utilities. They are incentivized to get a return on equity – capital investment. Thus, they desire to keep operations and maintenance costs as low as possible. As may be the case, there will be times where in a drive to keep these costs low, software support for enterprise systems may be cut. Or patches or other software updates are not kept up with. Over time these corporate systems incur architectural debt (see additional resources) that eventually must be addressed – often in the form of a large capital project that will be put into a rate base request.

   The challenge is that changing the corporate culture to a standards-based approach to data integration and/or semantic modeling may require an incremental increase in O&M – that may not rise to the level that meets the requirements of a capital investment. It may be the O&M "hump" that an organization cannot get over. Yes, standards-based data integration holds down O&M costs over time – but switching to that approach may cost more in the short-term. Hence, there can be incentive-based barriers in addition to the normal "that's not how we do it here" barriers one might see with other changes.

As this eight-step model is reviewed it is easy to see that it should not be wholly sequential – but that step four – communicating the vision – should be revisited as the new guiding principle of standards-based integration becomes ingrained in the culture of the organization, as shown in the figure below.

The urge to customize and the tale of two standards: Green Button vs ODIN

Green Button[56] (see the success story in the appendix) is a data exchange standard based on the CIM, specifically for metering data. The standard also has a robust test capability that can be used to verify the interfaces. In a conversation with some in the community, when the virtues of this standard were being extolled, a person expressed disappointment with the standard. Asked why, and it was revealed that, "It cost <utility> millions of dollars to implement Green Button and it took a lot longer than expected". Finding that outcome to be surprising, a little digging was required.

[56] https://www.energy.gov/data/green-button

It turned out that the time and expense was due to the desire of the utility to customize the interface, rather than deploying it as is. Which of course, is exactly counter to the benefits expected from a standards-based interface. In this case, it isn't a problem with the standard – it is a problem of execution.

Consider as an alternative, the Outage Data Initiative Nationwide (ODIN)[57]. ODIN is also a CIM-based standard based on the data model associated with exchanging outage data. This standard also has a test script in addition to the data integration specification. The experience for at least one utility, Seattle City Light (SCL) was completely different than the experience of the utility that had a challenge implementing Green Button. Why? No customization. The integration was so successful in fact, the SCL project manager stated, (paraphrasing), "Integrations that take minutes, not months". ODIN has now been widely adopted in the United States northwest and by CenterPoint Energy (based in Houston, TX) and the adoption of this standard continually expands.

CIM is the age of AGILE

> "We can't use the CIM because we use an agile approach to software development".

Great news – the CIM is especially useful if your organization uses an agile approach. To go fast, one must leverage the resources that already exist. When there is a readymade data model and a standard that provides message-exchange patterns and naming conventions, developers don't need to design these things; they can skip the user story "Waste a bunch of time by redesigning something that already exists".

Case Study

Jeff Kimble's team is ready. But is the rest of the organization? He approaches Bob Waring. Bob is leading a data quality effort for an application that is being re-platformed. They had a lot of data issues with the prior vendor and want to get ahead of them this time.

Jeff: Hi Bob, I heard about your new application and getting ahead of those data quality issues. I think my team can help.

Bob: How's that?

Jeff: There is a utility common information model that we can use to ensure that we're using industry standard definition, terms, and data relationships so we can limit customizations in the future.

Bob: That sounds nice. But the scope, budget, and schedule have already been set for this project. If we go back and revisit the timeline it's going to blow up my schedule. I'm going to have to pass.

[57] https://odin.ornl.gov/outages/pages/about.html

Disappointed but not surprised, Jeff looks for another opportunity. He hears that Jill Rodriguez is leading the new DERMS implementation and catches her in the hall.

Jeff: Hi Jill. How's the DERMS implementation going?

Jill: Hi Jeff. We're just getting started with identifying some of the interfaces that are going to be needed to other enterprise systems. The ADMS and CIS are high on that list.

Jeff: My team might be able to help you with that. The utility common information model has a related standard IEC 61968-5:2020 Distributed Energy Optimization and that standard already defines some standard data exchanges. And even better, there is a test script available for the standard with code examples that can greatly reduce implementation time.

Jill: This is perfect. My team uses SCRUM and I already didn't want them redesigning login, event handling, or system logs – if we can skip defining the interfaces to the ADMS and focus on implementation, this will really speed up the project.

Jeff: Terrific. Let's have our teams meet to level set on what resources are already available.

Jill: Sounds good to me!

Some time later…

Tom Hartman, IT Director, sees Jill talking with some colleagues in the hallway.

Tom: Hi Jill, I wanted to congratulate you on the DERMS implementation. The feedback I received from our partners in Operations indicated that they were delighted.

Jill: Thank you. Yes, it turned out our DERMS vendor already supported that CIM standard. The ADMS vendor did not, but my team was able to quickly make a CIM compliant adapter. Having access to example code and test scripts from the CIM User's Group made that a breeze. Connecting the DERMS to our old CIS was a little bit more work – that system is 30 years old, but we were able to leverage our experience with the DERMS/ADMS integration and were able to make a CIM adapter for that system as well. We finished the project early and under budget.

Tom: That was some great work. Say, could you help Bob with his data quality project? Apparently that project has gone off the rails and they're still arguing about what various data elements should be called.

Jill: Yes. I'd be happy to help. Let me coordinate with Jeff Kimble. He's our resident CIM guru and together we can probably help get that project back on track.

Section 15 Questions

1. The best way to implement the CIM is by making a corporate edict and creating a giant project that converts all existing interfaces to CIM-based interfaces.
    a. True
    b. False
2. The CIM is constantly being updated.
    a. True
    b. False
3. Once the vision for standards-based integration has been communicated, the team can focus on implementation.
    a. True
    b. False
4. The CIM, as both a data model and with its related standards can be used with agile software development approaches.
    a. True
    b. False
5. The CIM should be customized as much as possible to meet the organizations requirements.
    a. True
    b. False

Additional Resources

The EPRI Enterprise Architecture and Integration project set has published the following research on best practices for architecture debt:

1. *Governance and Enterprise Architecture Debt* (3002007898)
2. *Point-to-Point Standards Integration Cost Framework* (3002009981)

A. CIM SUCCESS STORIES

Success Stories: ESB Networks Use of CIM for Substation Configuration

The Utility

ESB Networks is the licensed operators of the electricity distribution system in the Republic of Ireland. They serve 2.3 million customers. They are responsible for constructing all the medium and low voltage electricity network infrastructure in the country as well as operating and maintaining this infrastructure. In their medium and high voltage network ESB Networks maintains 760 Substations, 100,000+ km of lines, and 10,000 power line sensors, of which ~99% of the operations information can be observed. In their low voltage network, they have ~99% unobserved operations, 252,276 Substations, 92,326 km of lines, and 100+ sensors.

For the "unobserved" portions of their network ESB has desires for future capabilities such as demand response and flexibility through electrified mobility, electrified heat, micro-generation, storage, enabling prosumers, and a mantra of "electrify everything!". But they also recognize that they have to cover the fundamentals of frequency, energy, power, and voltage while enabling these new capabilities.

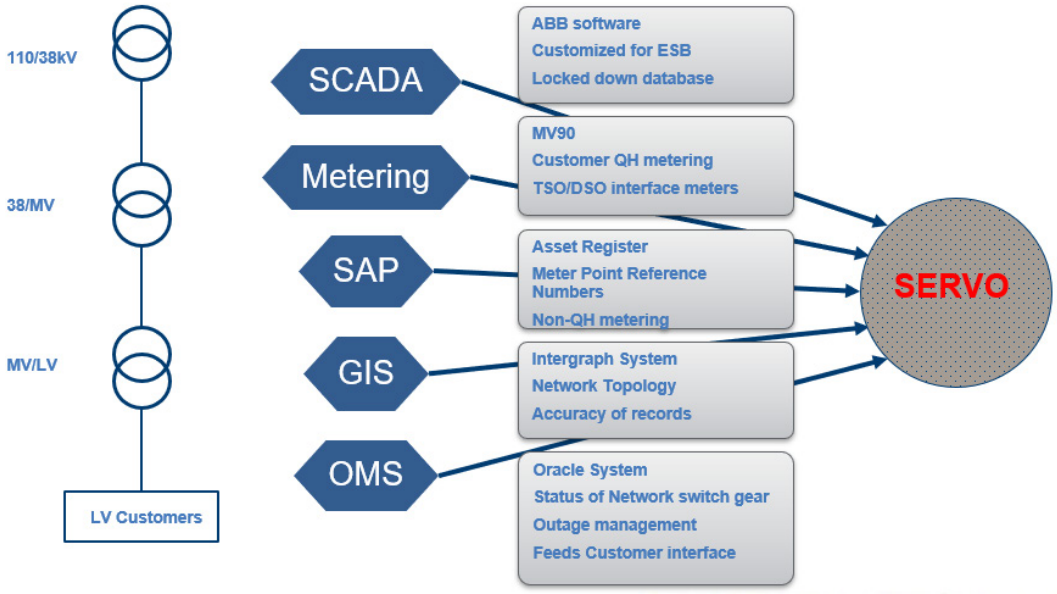


Figure A-1. Data sources and stakeholders

The Problem

What ESB Networks was trying to understand, fundamentally, was an answer to the question: "Can utilities deploy software in an effort to offset the investment in traditional assets?"

After 70 years of optimization they had seen a 300% increase in instantaneous demand, but this was also localized, so they also knew that they needed sensors and vision into what was occurring and needed to transform sensor data into knowledge. Then, share this knowledge with all actors to avoid congestion, control voltage, advise on frequency, procure services, all while avoiding traditional investment in infrastructure.

Additionally, ESB Networks was faced with the typical challenge of many data sources with many owners as can be seen in the figure below.

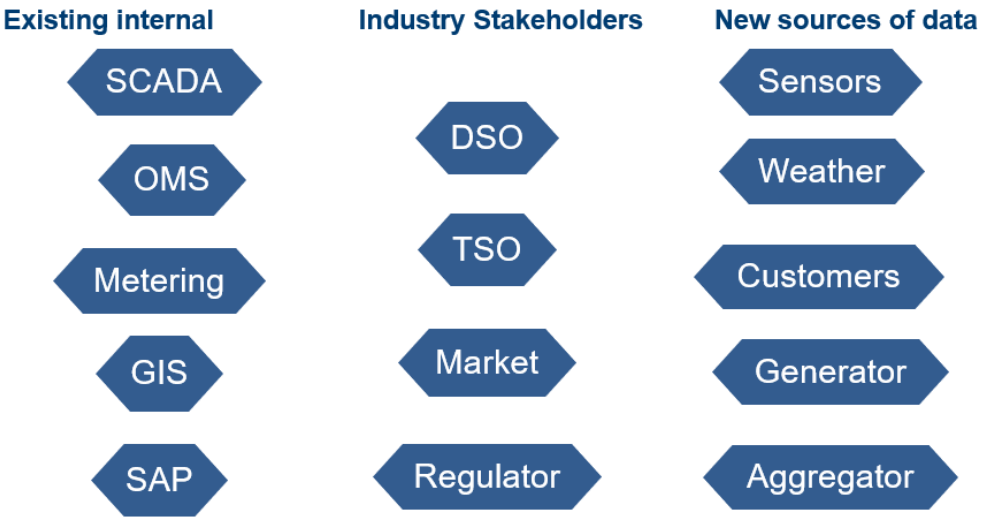


Figure A-1. Data source, corresponding networks, and categories of data

The CIM Solution

As part of this effort ESB Networks created Project SERVO – System-wide Energy Resource and Voltage Optimization Platform. This platform implements the International Electrotechnical Commission (IEC) common information model (CIM) which was used to capture all distribution assets and data into a single model. This platform enabled several use cases:

- Asset management – load visualization and asset reporting
- Operations – increased visibility into the low voltage network
- Capacity optimization – enables DSU participation and investment offset
- System Services – voltage and frequency control

ESB Networks worked with EPRI CIM experts to make sure that their understanding of the underlying CIM model could be accurately represented as the data model was being employed within the SERVO platform.



Figure A-1. SERVO web-based application with automated data feeds

The Impact

The new SERVO platform with its underlying CIM compliant data model, provided a web-based system that had automated data feeds (an example below is shown in Figure 3).

Some of the lessons learned along the way by ESB Networks included as part of overcoming barriers to implementation include:

- Benefits to the business need to be demonstrated
- Briefings, presentations across business to inform stakeholders
- Remind stakeholders that they were not replacing existing systems but enhancing them using new technology and software
- They needed to give examples of what can be achieved

Success Stories – The Green Button Standard

Green Button leverages CIM standards in the creation of a common way to share and view energy consumption data

The Problem

Green Button is a common-sense idea that customers should be able to download their own energy usage information in a consumer- and computer-friendly standardized electronic format from their utility's secure website.



Green Button is a White House initiative by the U.S. Federal Government's Office of Science and Technology Policy (OSTP), Department of Energy (DOE), National Institute of Standards and Technology (NIST), and Council on Environmental Quality (CEQ).

Green Button promises a common experience for consumers from their energy providers, and encouraging the development of an ecosystem of applications, "apps", and service providers around the availability of standardized energy usage information (EUI).

The figure above illustrates the primary dimensions of Green Button Data through the applications and services that make use of it:

- Usage – how much energy is consumed during a billing cycle or any other period?
- History – how much energy is used as a function of time intervals – 15 minutes, hours, days, and months?
- Cost – what is the cost associated with energy usage?

Green Button has significant momentum in the United States with its rate of marketplace penetration and the numerous and varied applications the availability of energy usage information has inspired. Canada has followed suit and adoptions are underway in other countries.

The CIM Solution

Green Button got its start in the UCAIug Open Automated Data Exchange (OpenADE) Task Force which developed an initial set of requirements for automated data exchange between utilities and customer authorized third parties.

In late 2009, the U.S. National Institute of Standards and Technology (NIST) established the Smart Grid Interoperability Panel (SGIP) to facilitate and accelerate the development of

standards for the Smart Grid under its mandate from the United States Congress in the Energy Independence and Security Act of 2007. Among its early efforts was Priority Action Plan 10 (PAP10) whose goal was to standardize energy usage information.

The PAP10 made the observation that this data was very similar to meter data. A modern object-oriented view of this information was sought. The SGIP itself was not a Standards Development Organization (SDO) and arranged to convene representatives of several relevant SDOs including TC57WG14 to collaborate on the problem. The North American Energy Standards Board (NAESB) volunteered to take the lead role in establishing a standard model for Energy Usage Information based on the OpenADE and related sources of requirements. Additionally, ZigBee Smart Energy Profile 2.0 was under development with similar goals but greater scope.

All parties involved recognized that the Common Information Model (CIM) had the potential to be a canonical model of information for many aspects of the Smart Grid. However, the scope of CIM itself was large and its focus was the utility enterprise.

The PAP10 team capitalized on the thinking and modeling expertise of the CIM committee and derived a profile of IEC 61968-9 that would meet the requirements for EUI. NAESB produced REQ.18/WEQ.19 that is this profile. Then the OpenADE task force brought implementation requirements to NAESB which produced REQ.21 Energy Services Provider Interface (ESPI) which produced a syntax and protocol for RESTful exchange of the semantics of REQ.18. The ESPI standard adds the dimension of customer authorization of third party access to their resources at a utility. It combines the CIM meter data model with the Atom publishing protocol and the OAuth third party authorization protocol to provide these standardized services.

At around the time that this technology was maturing and being ratified in NAESB, Aneesh Chopra, then Chief Technology Officer of the United States, unveiled the Green Button Initiative. Aneesh and the team from OSTP/DOE/NIST collaborated with the three California independently owned utilities (IOUs) to establish Green Button based on the NAESB standards. This was the logical next step in the administrations drive to free up data via web technologies with adequate security and privacy protections, building on the first such effort, Blue Button, which produced an open specification and capability for standardizing personal medical records in the Veterans Administration.

Thus, the core information for exchange by Green Button is based closely on the CIM metering model.

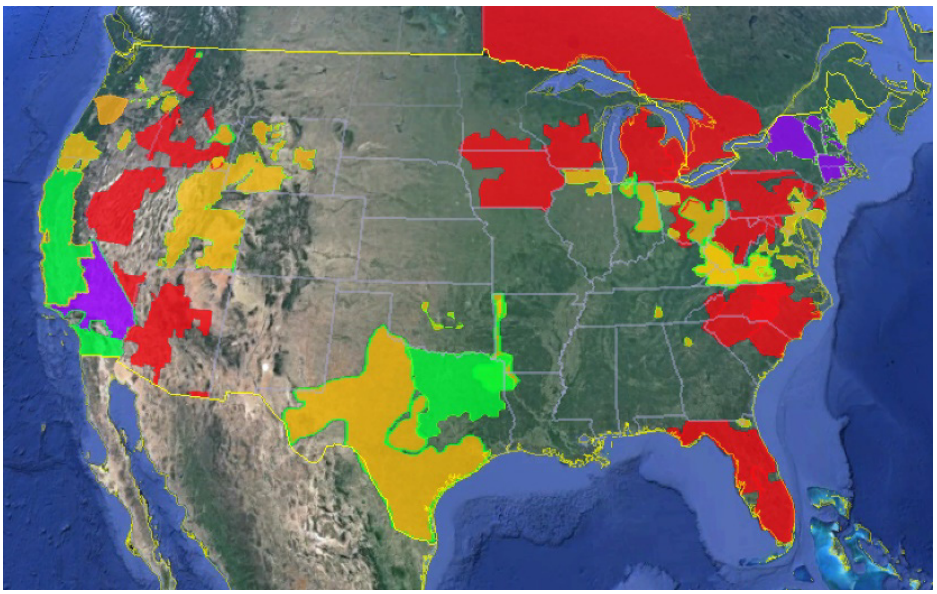
The Impact

The early results have been phenomenal. There are now tens of millions of American consumers with access to this CIM-based data. There are dozens of utilities and third party services providers exchanging data based on this standard. All this was accomplished in less than two years from the start of the Green Button Initiative. As you can see from the following graphic there is quite a large penetration achieved in a very short amount of time (30+ million customers).

In addition, in 2012, the Department of Energy sponsored a Green Button “Apps for Energy” contest, which challenged the industry to develop innovative new apps that help consumers use their Green Button data to understand their energy usage better and enable them to make energy saving decisions that save them money and reduce carbon emissions. The program was a phenomenal success with over 12,000 people actively following the challenge and dozens of new apps developed that help turn the standards effort into practical applications that help the American people.

In late 2012, Canada followed the U.S lead as the province of Ontario adopted Green Button. At a media event on November 21, 2012, the Honorable Chris Bentley, Ontario Minister of Energy, announced a partnership with MaRS Discovery District to launch a Green Button initiative in the province. Like in the U.S., Green Button adoption in Canada has been rapid, 9 months after the Minister’s announcement; over 2.6 million Ontarians now have access to their Green Button data. Ontario too has seen the value of consumer apps for saving energy, on Sept 8+9, 2013the MARS Discovery District held a Green Button “hackathon” to develop innovative new Green Button apps.

The significant impact of the Green button effort is a testament to the common sense of the solution, the elegance of the technology based on core pieces of knowledge that took years to develop by a cadre of experts (that is, CIM), and the assembly of a voluntary industry collaboration to solve a very real problem in a short amount of time.



Teeter 2013

Color Legend

Green:

Implemented 1Q2012

Purple:

Implemented 1Q2013

Tan/Yellow:

Committed for 4Q2013

Red:

Targeted for 2Q2014

Success Stories – Consumers Energy

Consumers Energy leverages IEC Common Information Model (CIM) for Enterprise Integration and a general enterprise semantic model with a Network Connectivity Model

The Problem

Consumers Energy has been adopting IEC CIM standards since 2008 when they started its service gap analysis as part of the AMI solution effort. The intention was to deploy a solution that is based upon existing vendor systems using open industry standards such as IEC CIM where available. The goal was to perform the gap analysis between these vendor applications and standards-based services, and work with vendors to ensure that there is sufficient coverage and support for industry standards from vendor products perspective to deliver an open and integrated AMI solution for Consumers Energy. As a result, interoperability could be achieved among various systems and applications, and common semantics can be established to represent power system objects in the company.

With the current technology landscape changes due to new applications to support ADMS functions, addressing the future demand of DER and Enterprise Analytics with respect to the Network Connectivity Model was going to be required. The challenge was how to ensure a consistent Network Model across all aspects of the enterprise.

The CIM Solution

The standards that involved are mainly IEC 61968-1, IEC 61968-9, and IEC 61968-100. The IEC 61968-1 Interface Reference Model (IRM) is used to group functional requirements. Its business functions and abstract components are used as actors to describe application/systems interaction using UML sequence diagrams. IEC 61968-9 is the base to define the Metering and Control context models in XML schemas (XSDs) for the integration.

After the initial success of the service gap analysis, Consumers Energy started to build an Enterprise Semantic Model (ESM) based on the IEC CIM. Displayed in Sparx Systems' Enterprise Architect (EA), the UML classes and attributes in the standard became building blocks for message payload definitions for integrations. This model-driven methodology included in Xtensible Solutions' MD3i Framework was adopted. The framework includes a set of plug-ins that enables users to work within EA's user interface through each step of the methodology. As a result, a semantically consistent common language is provided for exchanging data information across platforms and systems within the enterprise. Major benefits the company recognized are:

- Eliminating duplicate work on integration
- Maximizing the reusability of a common data model
- Lowering the cost on overall integration and support
- Facilitating the composition and consumption of information across multivendor landscapes
- Leveraging vendor's CIM-based solutions and SOA approaches for integration

Utilizing this approach and the IEC CIM, the decision was to establish a Network Connectivity Model that will be utilized by different applications and sources. To decouple and minimize the amount of dependency on specific technology or solutions.

The Impact

As a major contributor to the Open Smart Grid (OpenSG) user group, Consumers Energy also worked closely with other utilities in North America to promote the CIM standards and provide user requirements for CIM model improvement and extensions. Recognizing the need for common Web service definitions (WSDLs), Consumers Energy provided resources working on the IEC 61968-100 and leading the effort on defining common Web service templates for the Implementation Profiles for the standard.

Overall IEC CIM standards fit well in the approach Consumers Energy put together for top-down business process gap analysis, integration services design, and message payload and web service definitions for integration. Consumers Energy has continued to use this methodology and associated tools for integration work even in the generation of a Network Connectivity Model in 2017 and will continue in the foreseeable future.

Success Stories – Public Service Enterprise Group – Long Island (PSEG-LI)

Public Service Enterprise Group - Long Island leverages IEC Common Information Model (CIM) for Enterprise Information Management and Semantic Integration initiatives.

The Problem

PSEG-LI embarked on a SmartGrid program that comprises the implementation of new applications and the integration of these applications with legacy applications.

Key business drivers are to reduce the cost and complexity of the development and maintenance of integration solutions and data repositories, enable the ability to integrate PSEG-LI data among multiple service providers and implement “Best of Breed” applications.

Some of PSEG-LI’s architectural goals are (a) an event-driven enterprise and (b) provide a robust yet agile and loosely-coupled architecture along service-orientated architecture (SOA) principals. The loosely-coupled architecture provides the following benefits:

- Changes to one system have minimal impact on other existing systems
- Allows flexibility and agility to implement improved or new functionality at much reduced cost/effort/risk
- Allows the assimilation of data required for holistic decision making, analysis, planning, risk management, and so on.
- Also allows for the development of new reports and functionality not previously available in any of the off-the-shelf applications

PSEG-LI avoids proprietary integration solutions, rather targeting and embracing semantic interoperability that leverages standards-based architecture to drive down effort and cost for new integration and integration maintenance. This semantic interoperability is key to enabling this de-coupling of the architecture.

The Solution

The PSEG-LI approach comprises an end-to-end model-driven methodology for designing and implementing a "common model" based on the IEC Common Information Model (CIM) to provide the targeted loosely-coupled system integration. This is consistent with the goals of the International Electrotechnical Committee (IEC) Technical Committee (TC) 57 Working Group (WG) 14.

LIPA has solved many of the challenges that utilities will face when adopting the "common model" approach and implement a scalable solution for the governance, processes and methodology of managing this solution in a real-world environment with multiple changing pieces.

PSEG-LI selected Xtensible Solutions' MD3i 2.0 methodology and framework for implementing its integration and data warehouse solutions. Key characteristics of this approach are:

- End-to-End Model-Driven approach
- Paradigm Shift compared with the conventional approach
- Bridges the chasm between design, development and run-time
- Increased Agility, Responsiveness, Speed
- Decreased Time, Cost, Risk
- Enabler for implementation of new functionality, processes and analytics solutions

Key components of the methodology are:

- Processes and Governance
- Centrally Managed Semantic (Data) Model (EXM) based on IEC CIM
  - Heterogeneous interfaces mediated through common model
  - Capability to manage private extensions and other modifications needed to the standard CIM model
  - The ESM also supports all design of database projects, to ensure consistent semantics between data-in-flight and persisted data.
- Centrally Managed Exchange Model (EXM)
  - Service Definition, Semantic Mapping and Business Rules
  - Orchestration and Exception handling
  - Integrate and Reuse Business Rules, transformations, mappings
  - Automate gap analysis, documentation

  - No coding is performed, it generates “ready-to-go” services for deployment
- Centrally Managed Development and Run-Time Deployment
  - Generate ready-to-go SOA services
  - Continuous testing
  - Deploy into any Java runtime environment
  - Automate impact analysis on change, across all deployed services and the common model (ESM)

In addition, PSEG-LI has implemented a Solution Development Life Cycle (SDLC) used on projects, enabling PSEG-LI to control projects and ensure that the LIPA architectural goals are not subverted.

The Impact

The methodology has helped PSEG-LI establish a layered and loosely-coupled architecture with the business benefits of being more agile and responsive to business and regulatory changes, leveraging services on new projects (enabling re-use with minimal refactoring).

Projects comprised integration solutions and persistent data stores (ODS’s and Data Warehouses)

The PSEG-LI Model-Driven Semantic Integration approach has consistently performed under budget and on time under complex and challenging conditions. Changes to implemented solutions are done in a fraction of the time traditional SOA approaches would take.

All new projects are adopting the architecture and methodology. The trend of reduced cost and improved delivery speed is based on:

- Model-driven approach + governance + processes + tools for “automated/integrated” development, testing, implementation, and maintenance of the model
- Reuse of data and interfaces across company systems and SOA

Success Stories – Idaho Power Company (IPC)

Idaho Power uses the CIM as part of the integration design and governance processes.

The Problem

When Idaho Power Company (IPC) began to transform its business with the implementation of Smart Grid capabilities, it became apparent that they would first need to put in place a robust system integration framework to handle all the new system interactions resulting from the multiple planned system replacement projects. Service Oriented Architecture (SOA) coupled with mature interoperability standards seemed to be the best approach to achieve a scalable and extensible solution.

IPC quickly realized that key to a successful outcome was to have in place a set of governance policies and practices prior to starting the first integration project, which was to replace their Outage Management System. A critical component of governance was the choice of international standards for all system interactions; standards that are independent of any particular vendor's proprietary platform. The CIM standards, as recommended by NIST, were selected for further evaluation to see if they could be used to not only handle the information exchanges for each system interaction point, but that the tools to develop and maintain the multiple system interaction definitions needed by IPC were available with good vendor support.

The CIM Solution

The CIM standards comprise an information model and multiple profiles (or message definitions) that are derived from the model, ensuring a single source definition of each data element for all information exchanges. Based upon the CIM UML model, IPC developed an Enterprise Semantic Model (ESM) which provided the capability to include private extensions and other modifications needed to the standard CIM model. The CIM-based ESM then provided a common set of semantics and data definitions from which all needed system interactions could be defined as XML schemas. The integration framework, then, comprised an Enterprise Service Bus (that is, TIBCO) with data adapters at each system interface to transform the data from the internal system representation to the ESM representation.

To develop, manage and maintain the ESM and message definitions based on the ESM, IPC selected Xtensible's MD3i Framework operating on the Sparx Enterprise Architect (EA) platform, which is also used by the IEC standards body to manage and maintain the CIM UML model. After first conducting a proof of concept project using MD3i, Xtensible was selected to:

1. Develop a version of the MD3i process to meet the needs of IPC's integration design process
2. Develop a multi-user, Oracle server-based Enterprise Architect (EA) modeling tool environment for use by IPC system analysts to develop and maintain the ESM and system interactions
3. Provide training on the CIM and UML modeling within EA using the MD3i Add-Ins to execute the integration design process for ESM modeling and design
4. Provide guidance and advice as well as development of selected system integrations following the IPC integration design process

The XML schemas for each system interaction were then used to create the System-to-ESM mappings for the ESB system adapters.

The Impact

The development of an integration design process and governance policies built around a CIM-based ESM and system interface definitions was key to managing multiple system integration projects. Since each new system integration is able to build off the existing ESM from earlier projects through reuse, the amount of work to define system interactions decreases with each new project. The consequence is that the CIM standards, with the proper tools to manage their

use, provide not only a scalable, maintainable integration framework, but one that becomes more cost effective with each new system integration.

Success Stories – Oracle Utility Data Model (OUDM)

Oracle takes a standards-based approach using CIM for Enterprise Analytics

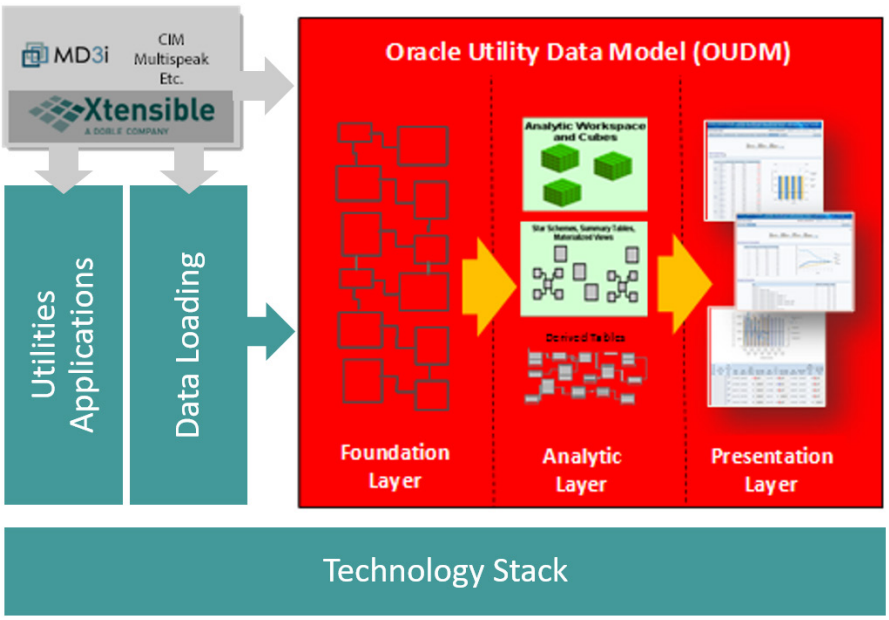
The Problem

A demand for Enterprise Analytics within the utility space has continued to grow, from customer engagement to asset management, to an increased demand in grid operations with the growth of DER and the market changes.

The challenge exists that each technology solution(s) provides analytics that sites on top of each given solution, not necessarily across different domains within an organization. Thus, limiting the perspectives in which the data and be view and insights generated.

The CIM Solution

The CIM standards comprise an information model and multiple profiles that is independent of the technology solutions(s); rather thus focusing on the business functions that a utility. Although very extensive in the Marketing Operations, Transmission and Distribution areas, so parts of the IEC CIM are less developed for example Customer. The solution was to utilize the IEC CIM areas that are full defined and harmonize it with other models and establish a physical implementation of the model that can be utilized for enterprise analytics, the OUDM. Partnering with Xtensible's utilizing the MD3i toolset and standards, a model was established and created a foundational layer of 3NF, thus giving a path to developing the analytics layer which then can presented in a technology agnostic presentation layer.



The end solution includes the following comments based off the IEC CIM:

- Logical Data Model, Physical Data Model (3NF and STAR schemas)
- Pre-build OLAP
- Pre-built Mining Model
- Intra ETL among schemas
- Intra ETL for OLAP and Mining Workflow
- Base with Reference and Aggregate
- Metadata
- Dashboard and Reports

The Impact

The development of the OUDM has resulted in an CIM based model that can be physically implemented on a box that "connects the dots" between data sources across different business areas within the organization. This allows for developing Enterprise Analytics through a standards-based IEC CIM approach. Warehousing and data lake efforts are supported, providing a jump-start to Enterprise Analytics initiatives.

Success Stories – Sempra Energy

Sempra Energy uses CIM to support their OpEx 20/20 and Smart Metering programs, reducing the cost of systems integration, maintenance, and support.

The Problem

Sempra Energy Utilities (SEU) goal is to manage its vast business data, information, and knowledge as corporate assets. To that end Sempra established an Enterprise Information Management (EIM) strategy and business case in 2007 to realize the value of these corporate assets in support of optimized business performance.

Sempra IT established their Service Delivery Program (SDP) in support of two large business programs (OpEx 20/20 and Smart Metering) to ensure that IT services across all current and future programs are consistent and sustainable. A key part of the SDP service portfolio is process integration and information management capabilities using Service-Oriented Architecture (SOA) supported and directed by the EIM strategy.

Sempra realized that key to the success of the service orientated architecture (SOA) integration in delivering sustainable process integration and business intelligence for OpEx 20/20 and Smart Metering was adopting a "common model" approach for defining common interface definitions, providing loose-coupling at the data level.

The CIM Solution

The CIM standards comprise an information model and multiple message definitions that are derived from the model, ensuring a single source definition of each data element for all information exchanges. Based upon the CIM UML model, Sempra developed its own Enterprise Semantic Model (ESM), referred to as the SIM, which contained all the required extensions and other modifications needed to the standard CIM model. The CIM-based ESM then provided a common set of semantics and data definitions from which all needed system interactions could be defined as XML schemas.

The Sempra integration framework comprised an Enterprise Service Bus (Oracle). The integration pattern used most of the time comprised data adapters at each system interface to transform the data from the internal system representation to the ESM representation.

Sempra engaged Xtensible Solutions as a trusted advisor for developing the EIM strategy and associated business case. Based on the EIM strategy, Xtensible Solutions was engaged for the Smart Metering and OpEx 20/20 programs to provide support to the SDP for the SOA-based integration delivery.

To this end, Xtensible implemented Xtensible's MD3i framework to manage, develop and maintain the Sempra SIM (Sempra Information Model) which is based on the IEC Common Information Model (CIM). The framework and methodology comprised:

- Governance and processes to manage the SIM extensions, as needed
- A toolset (plug-ins running on Sparx Enterprise Architect (EA) that manages the multiple reference models, common model, data lineage and mapping

Xtensible also helped train Sempra resources in order to establish a data modeling function within the data architecture group.

The Impact

The methodology supported the implementation of new applications such as Asset Management, Work Management, Mobile Workforce management, Outage Management. The development of an integration design process and governance policies built around a CIM-based ESM and system interface definitions was key to managing multiple system integration projects. Since each interface definition is based off the SIM common model, and each adapter transformed the proprietary interface to the common model representation, Sempra achieved the de-coupling of applications they desired. Sempra have also achieved re-use of services through this approach which has saved time and cost. Since each new system integration is able to build off the existing ESM from earlier projects through reuse, the amount of work to define system interactions decreases with each new project.

Success Stories – Southern California Edison Uses CIM for Green Button Support

Southern California Edison (SCE) supported Green Button development based on CIM standards.

Problem

As part of the NIST SGIP PAP10 drive to define a common format and protocol for exchanging energy usage information, groups within UCAIug OpenSG and NAESB needed an information model that stakeholders from a wide variety of companies, industries, and backgrounds could agree on.

Solution

SCE engaged Xtensible Solutions to support OpenADE and NAESB groups working on the Energy Services Providers Interface (ESPI)/Green Button specification. The goal was to choose objects from the meter reading package of the IEC's CIM as the basis for the objects to be exchanged. Some of the advantages included:

- Technology Neutral – Because the model is defined in UML, it can be translated into a variety of implementation languages. XML was chosen for the payload representation.
- Extensibility – The ability to adapt the CIM objects allowed it to be used with the ATOM publishing protocol. This defined the exchange mechanics using a commonly supported RESTful open standard, using XML over HTTP/S.
- Compatibility – Another advantage of the CIM was that it was familiar to utilities and meter manufacturers. In addition, the Smart Energy Profile 2.0 for Home Area Networking (HAN) communications was also using the CIM as the basis for messaging, making it easier to translate between the two protocols.

- International Support – There are many ways to build an information model. Because the CIM is built using a consensus development process, the groups did not have to spend time inventing something new.
- Community/Tools – Because the CIM has a mature community, efficient ways to use the model had been developed through several iterations of tools and processes. SCE and PG&E funded Xtensible Solutions to help lead and refine the model within these Smart Grid HAN standards.

Impact

With assistance and guidance from SGIP and other groups, UCAIug is developing a testing and certification program for Green Button standards to ensure that implementations are interoperable with each other. This program will allow energy consumers to grant access to automated periodic transfers of their data to allow for active management of smart appliances and other energy management services.

Though the industry is just at the beginning of this new frontier, millions of customers from many utilities now have access to their usage data in a standard format. In 2012, SCE, along with PG&E and SDG&E, were among the first utilities to implement the standard, making usage data available to most Californians. This data can be shared with a growing number of providers of energy services.

Success Stories – PNNL Uses CIM for GridAPPS-D™

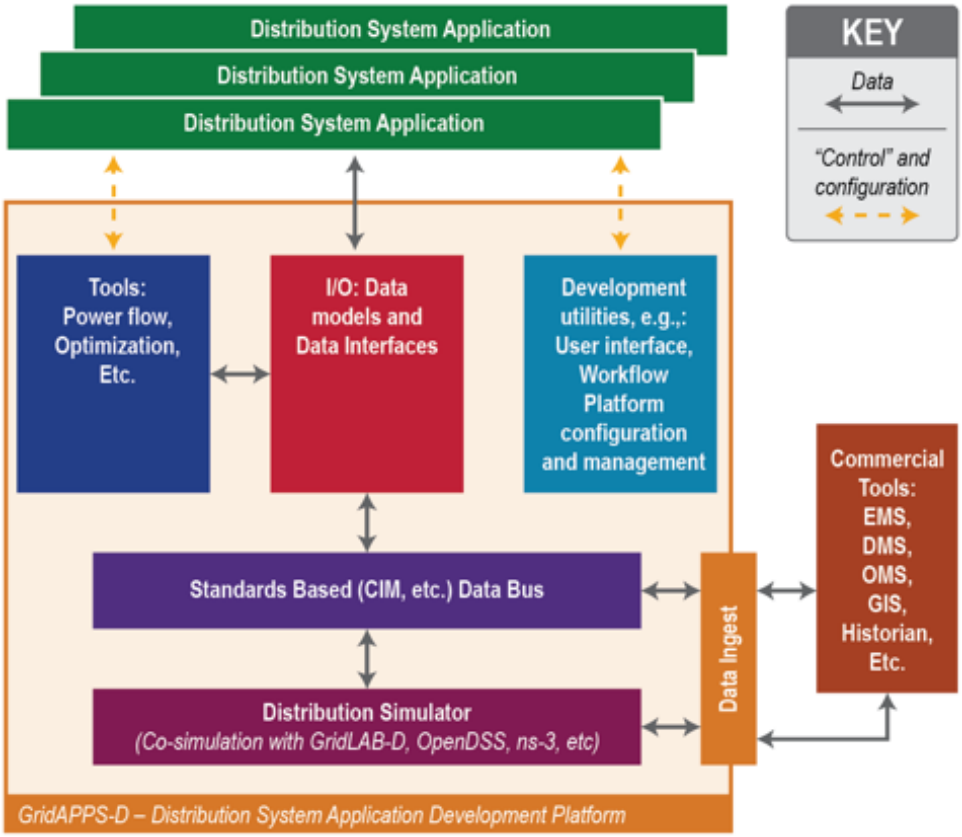
GridAPPS-D applied the CIM standards to standardize the application programming interface for an open-source distribution system planning, operations, control and management application development platform.

The Problem

Modern electric power distribution systems include growing numbers of automated devices and systems generating increasing amounts of data. In addition, distribution systems are becoming more complex and operationally challenging with increasing penetration of distributed energy resources. This situation calls for a new generation of distribution system planning, operations, control and management applications that are able to take full advantage of the growing quantity of data and that can do so while reducing the time and cost to integrate and deploy the applications.

The Solution

GridAPPS-D has been developed as an open-source, standards-based platform for development of this new generation of applications. The WG14 IEC 61968-100 based platform provides integrated access to data with a uniform application programming interface based on the Common Information Model and the use of the Java Messaging Service (JMS) publish – subscribe mechanisms. The conceptual design of the platform is shown in the figure below.



The 61968-100 standards-based data bus uses triple-store database technology to address the challenge of querying a variety of network models. Triple-store technology directly matches the IEC 61970-501 (CIM-XML) RDFS form of subject-predicate-object triples associated with EMS (61970) and DMS (61968) CIM representations, enabling the possibility of portable applications that can be applied across a multitude of distribution networks. GridAPPS-D uses triple-store technology from blazegraph™ combined with conventional relational database technology for meta-data and with InfluxDB an open-source tool supporting distributed real-time data acquisition and management that include CIM-based universally unique identifiers (mRIDs) used to federate and cross-reference data collected across the GridAPPS-D data bus.

The Impact

GridAPPS-D development began in September 2016. Version 1.0 was released in May 2018 offering core functionality as defined in the GridAPPS-D functional requirements specification. Several example applications are being developed and will be tested in late 2018 to document the benefits of taking advantage of the variety of data available in a modern electric power distribution system for problems such as Volt/VAR optimization, improved load forecasting, advanced microgrid control, state estimation and transactive energy systems.

The use of the CIM to standardize the application programming interface and messaging along with triple-store technology enables the possibility of portable applications. Applications developed using this approach should be able to be deployed at lower cost on any vendor

system or platform that complies with the GridAPPS-D programming model. By establishing a common approach, the benefits of advanced applications should be available to a wider variety of utilities.

GridAPPS-D is available at https://github.com/GRIDAPPSD/GOSS-GridAPPS-D and http://gridappsd.readthedocs.io/en/latest.

Development of GridAPPS-D has been funded by the U.S. Department of Energy's Office of Electricity Delivery and Energy Reliability's Advanced Grid Research Division.

B. ANSWERS TO SECTION QUESTIONS

Answers to Section Questions

Section 1

1. a
2. c
3. c
4. b
5. d

Section 2

1. a
2. a
3. b
4. c
5. d

Section 3

1. b
2. d
3. a
4. c
5. a

Section 4

1. d
2. a
3. a
4. b
5. c

Section 5

1. c
2. d
3. b
4. b
5. d

Section 6

1. d
2. a
3. d
4. b
5. a

Section 7

1. d
2. b
3. b
4. a
5. d

Section 8

1. b
2. a
3. a
4. b
5. d

Section 9

1. a
2. b
3. d
4. c
5. a

Section 10

1. a
2. c
3. d
4. c
5. a
6. c

Section 11

1. a
2. d
3. b
4. a
5. c

Section 12

1. d
2. b
3. c

Section 13

1. d
2. e
3. a
4. b
5. a
6. d

Section 14

1. a
2. b
3. b
4. b
5. d
6. b

Section 15

1. b
2. a
3. b
4. a
5. b

About EPRI

Founded in 1972, EPRI is the world's preeminent independent, non-profit energy research and development organization, with offices around the world. EPRI's trusted experts collaborate with more than 450 companies in 45 countries, driving innovation to ensure the public has clean, safe, reliable, affordable, and equitable access to electricity across the globe. Together…shaping the future of energy.

Program:

Information and Communication Technology, P161

3002032389

© 2025 Electric Power Research Institute (EPRI), Inc. All rights reserved. Electric Power Research Institute, EPRI, and TOGETHER…SHAPING THE FUTURE OF ENERGY are registered marks of the Electric Power Research Institute, Inc. in the U.S. and worldwide.

EPRI
3420 Hillview Avenue, Palo Alto, California 94304-1338 USA
800.313.3774 ▪ 650.855.2121 ▪ askepri@epri.com ▪ www.epri.com