EPRI | ELECTRIC POWER RESEARCH INSTITUTE

Common Information Model Primer

Third Edition

2015 TECHNICAL REPORT

Common Information Model Primer

Third Edition

EPRI Project Manager
J. Simmins

EPRI | ELECTRIC POWER RESEARCH INSTITUTE

3420 Hillview Avenue
Palo Alto, CA 94304-1338
USA

PO Box 10412
Palo Alto, CA 94303-0813
USA

800.313.3774
650.855.2121

askepri@epri.com

www.epri.com

3002006001
Final Report, June 2015

DISCLAIMER OF WARRANTIES AND LIMITATION OF LIABILITIES

THE FOLLOWING ORGANIZATION, UNDER CONTRACT TO EPRI, PREPARED THE FIRST EDITION OF THIS REPORT.

Open Grid Systems Ltd.

NOTE

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

Acknowledgments

The following organization, under contract to the Electric Power Research Institute (EPRI), prepared the first edition of this report.

Open Grid Systems Ltd.
3/2, 25, Camphill Avenue
Glasgow, Lanarkshire, G41 3AU
Scotland

This report describes research sponsored by EPRI.

EPRI would like to acknowledge research work undertaken at the University of Strathclyde's Institute for Energy and Environment and A.W. McMorran's paper, *An Introduction to IEC 61970-301 & 61968-11: The Common Information Model*, January 2007, which has influenced this report.

Editor, Third Edition
J. Simmins, EPRI

This publication is a corporate document that should be cited in the literature in the following manner:

Common Information Model Primer: Third Edition.
EPRI, Palo Alto, CA: 2015.
3002006001.

Product Description

The *Common Information Model Primer* explains the basics of the Common Information Model (CIM) standards to help operations professionals better understand how electric systems are modeled in the applications they use. CIM standards currently have three primary uses: 1) to facilitate the exchange of power system network data between organizations, 2) to allow the exchange of data between applications within an organization, and 3) to exchange market data between organizations. The Second Edition of this primer was updated with a case study that follows a utility through its journey of discovery, learning, and application of the CIM for grid modeling and integration. Questions were added to the end of each section for the reader to reinforce learning. This Third Edition includes a section on inexpensive tools for applying the model described in the narrative.

Background

The CIM was developed by EPRI in North America and is now a series of standards under the International Electrotechnical Commission (IEC) (IEC 61970, IEC 61968, and IEC 62325). The CIM was initiated as part of EPRI's Control Center Application Program Interface (CCAPI) project. Originally, the aim was to provide a common definition for power system components for use in 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. IEC standard 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 extends both of these models with IEC 62325-301 to cover the data exchanged between participants in electricity markets. These three standards—IEC 61970-301, IEC 61968-11, and IEC 62325-301—are collectively known as the CIM for power systems.

Objectives

To present the CIM in a format that will particularly benefit individuals lacking experience with standards, UML models, or system integration.

Approach

The material contained in this report was accumulated over a period of years from several short courses given on the subject, EPRI archives, careful reading and understanding of the standards, research performed in pursuit of a doctorate degree, and feedback from practitioners who have used the CIM for systems integration.

Results

Beginning with a historical perspective, this primer describes how the CIM originated and grew through the years. The functions of various working groups of Technical Committee 57 of the IEC are described. The process of how an IEC standard is created is also outlined.

The basics of the Unified Modeling Language (UML) are detailed to introduce the reader to the language of the CIM. Then, building on commonly understood objects (basic shapes), the concepts that underline the CIM are carefully built step by step. The reader is then transported into the world of power systems where concepts that were developed previously are applied to the complexities of the electric grid.

The resulting primer synthesizes material from an array of sources to reduce an extremely complex power system model in UML into straightforward concepts that build one upon another.

Applications, Value, and Use

Anyone who is not an expert in the CIM or UML will find this primer extremely helpful in understanding the leading integration technology for back office applications in the utility industry. IT professionals will find this work to be a valuable foundation to build upon in using the CIM for application integration and data management.

Keywords

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

Abstract

The *Common Information Model Primer* takes on the task of explaining the basics of the Common Information Model (CIM) (IEC 61970, IEC 61968, and IEC 62325). Beginning with a historical perspective, the primer explains how the CIM originated and grew through the years. The primer describes the functions of various working groups of Technical Committee 57 of the International Electrotechnical Commission (IEC) and outlines the process of how an IEC standard is created.

The basics of the Unified Modeling Language (UML) are detailed to introduce the reader to the language of the CIM. Then, building on commonly understood objects (basic shapes), the concepts that underline the CIM are carefully built step by step. The reader is then transported into the world of power systems where the previously developed concepts are applied to the complexities of the electric grid.

Through each chapter the reader will follow a case study that describes how a utility has used the CIM for integration. Questions and answers are provided to reinforce reader understanding of key CIM concepts. This Third Edition of the primer also includes a section on inexpensive tools for applying the model described in the narrative.

Executive Summary

The impetus for this report was the result of a combined EPRI–Gartner Group survey of the use of the Common Information Model (CIM) in utilities throughout the world. EPRI report 1020103, *Development of the Common Information Model for Distribution and a Survey of Adoption: CIM Development and Testing Activities in 2010,* explains that one of the barriers preventing the CIM from being ubiquitous has been the lack of educational materials, primarily introductory materials. This report is one result of several EPRI projects in 2011, 2012, and beyond to facilitate adoption of the CIM, which is expected to be of great benefit to the industry and public alike.

This primer builds on common concepts, such as shapes, and takes the reader into the Unified Modeling Language (UML) model. The primer examines how the standard works, specific examples of creating the model from power system assets, and how messaging and extensions of the CIM are created and used.

Readers who spend the time to delve into this work will come away with a deep understanding of concepts surrounding the CIM. This foundational understanding of the CIM should help the operations professional understand how the electric systems are modeled in the applications they use. IT professionals will find this work to be a valuable foundation to build upon in using the CIM for application integration and data management.

Table of Contents

List of Figures

List of Tables

Section 1: Introduction

Learning Objectives

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

Since deregulation in North America, Europe and elsewhere in the world, there has been an increasing need for power companies to exchange data on a regular basis. This information exchange promotes reliable operation of the interconnected power networks owned and operated by various utilities. Power companies use a variety of formats to store their data, whether it is asset and work scheduling information in a proprietary internal schema within a database, topological power system network data within a control system, or static files used by simulation software.

While much of this data is only required within a company, there is often a need to exchange the data both internally between different applications and externally with other companies. The large number of proprietary formats used by these applications requires a myriad of translators to import and export the data between multiple systems. This exponential growth in complexity when integrating increasing numbers of applications and exchanging between multiple companies has driven the requirement for a common format that covers all the areas of data exchange in the electrical power domain.

The IEC standard 61970-301[1] is a semantic model that describes the components of a power system at an electrical level and the relationships between each component. The IEC 61968-11[2] extends this model to cover the other aspects of power system software data exchange such as asset tracking, work scheduling, and customer billing. The CIM for Electricity Markets then extends both these models with IEC 62325-301[3] 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 Common Information Model (CIM) for power systems and currently have three primary uses: to facilitate the

[1] "IEC 61970 Energy management system application program interface (EMS-API) - Part 301: Common Information Model (CIM) Base", IEC, Edition 3.0, August 2011

[2] "IEC 61968 Application integration at electric utilities - System interfaces for distribution management - Part 11: Common Information Model (CIM)", IEC, Edition 1.0, July 2010

[3] "IEC 62325 Framework for energy market communications -Part 301: Common Information Model (CIM) Extensions for Markets", IEC, Draft

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.

The development of the CIM began primarily in North America, where the North American Electric Reliability Council (NERC) adopted the CIM as the format for exchanging network data between transmission companies. The majority of the application integration activities have similarly taken place within North American utilities however in recent years the adoption of the CIM by the European Network of Transmission Systems Operators for Electricity (ENTSO-E) and by a number of European, Asian, and South American utilities has seen its use grow beyond its roots in North America.

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, as he transitions from learning 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 systems
    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 systems

   b. Distribution systems

   c. Market information exchange

   d. Gas Systems

4. IEC 61968 cover the definitions for:

   a. Power systems

   b. Distribution systems

   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

Section 2: General Background

Learning Objectives

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

Since the advent of the modern digital computer, power system engineers have utilized its capabilities in a variety of areas, whether it be performing complex analysis calculations on a power system or to control its operations in real-time. All of these applications require the operator to digitally store and exchange data about the system.

Power System Data Formats

Large enterprise systems such as an Energy Management System (EMS), Asset-Management System, Work Management or Advanced Metering Infrastructure (AMI) use database schemas for defining the structure of the data storage data, often custom-written to reflect the operator's specific requirement. Offline applications for performing load-flow, steady-state estimation and fault-level analysis simulations similarly use application-specific file formats that represent the data required by each application.

These offline analysis file formats are often a legacy of the original coding of products written in an era when disk space and memory was severely limited. As such the formats will often be simple, column-oriented, fixed width or tab-delimited structures that favor compactness over verbosity and directly mirror the application's original internal data structures (e.g. COBOL record layouts).

These file formats have tended to grow over the years as more data is added and work perfectly when used only for exchanging the data required for that particular application. However, fixed formats do not lend themselves very well to extensibility as the addition of extra items can potentially break any software trying to parse the data when it finds unknown entries. The lack of verbosity within the format is beneficial for keeping the files small and compact, but the tradeoff is that the formats are not self-describing. Further, if the documentation is proprietary and not publicly available it may be difficult to determine the attributes and characteristics necessary to build an interface that leverages the file format.

Systems Integration

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. Such a scenario is not limited to power system applications but also a common practice within the software industry. However, this is usually a minor irritation since each new version of a vendor's software contains import facilities to convert previous versions of the file format into the new format.

Problems occur when companies need to exchange data between software applications from different vendors, and/or have multiple versions of the same software running within their company. Such a scenario requires a company to either:

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-detail 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, requires three things:

- A highly detailed model to describe the power system

- A file format capable of storing extended data without affecting the core data
- Power system software vendors and utilities to either adopt and embrace this data model and format either for economic or regulatory reasons

The Common Information Model (CIM) for Power Systems has the potential to meet the first requirement of the above list while the eXtensible Markup Language (XML), combined with the Resource Description Framework (RDF) offers a means of fulfilling the second requirement. The remaining requirement can be considered 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 the CIM Interoperability tests and the popularity of the format is spreading.

Modeling Data

> "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 data that can then realized in a number of different technologies. File formats, database schemas, and internal application data structures can all be derived from a model, a core part of a Model Driven Architectures (MDA)[4] in software engineering.

The Three Models

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.

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

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.

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, the Unified Modeling Language (UML) has the concept of *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*[5]. 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.

Defining Models

How information models are defined will be covered later, but 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 harmonies internal and external models into the conceptual model.

---

[5] 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

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.

Separating 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 being 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.

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.

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, and reducing errors as using a “model is king” approach, when the model can be used to generate database schemas as well as interfaces, re-imagines the applications from the data out, rather from the interface in.

Section 2 Questions

1. When using 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 how:
    a. the application or system stores the data
    b. the data that is exposed to the user
    c. the abstract definition of the data is described
    d. the data is stored in a table

3. The external model describes how data is:
    a. the application or system stores the data
    b. the data that is exposed to the user
    c. the abstract definition of the data is described
    d. the data is stored in a table

4. The conceptual model describes how data is:
    a. the application or system stores the data
    b. the data that is exposed to the user
    c. the abstract definition of the data is described
    d. 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

Section 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

History

Exchanging power systems data between utility companies is always problematic when proprietary formats are used. In the past a company would traditionally use a single software system, whether it is a custom in-house solution, or 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 and the emergence of smarter grids, there is now a greater need to be able to share such power system data between 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 power system 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) is an open standard for representing power system components originally developed by the Electric Power Research Institute (EPRI) in North America and now a series of standards under the auspices of the International Electrotechnical Commission (IEC). The standard was started as part of the Control Centre 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

IEC Technical Committee 57 develops standards for "Power systems management and associated information exchange". The committee was established in 1965 and contains a number of working groups that develop and maintain standards including CIM, IEC 61850[6] and IEC 60870-6/TASE.2[7].

The CIM is 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 "Interoperability within TC 57 in the long term" and harmonizes work across the TC57 working groups. This is to ensure that there is no duplication of effort and to promote interoperability between the standards.

WG13: Energy Management System Application Programming Interface

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.

WG14: Systems Interfaces Distribution Management Systems

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 exchanged between systems within the distribution utility.

---

[6] "IEC 61850 Communication networks and systems in substations - Part 1: Introduction and overview", IEC, Edition 1.0, April 2003

[7] "IEC 60870 Telecontrol equipment and systems - Part 6-503: Telecontrol protocols compatible with ISO standards and ITU-T recommendations – TASE.2 Services and protocol", IEC, Edition 2.0, April 2002

The original electrical model was expanded to include the modeling of unbalanced, low and medium voltage distribution networks along with extensions for Asset Management, Work Management, Customer Management, AMI, Geographical Information Systems (GIS), Maintenance and Construction, Network Planning and a number of other systems. This has significantly increased the scope of the CIM, and although the extensions originated as being for the distribution utilities, many are also applicable to the transmission operators.

This part of the CIM, published as IEC61968-11, is then used to derive interfaces primarily for systems integration within the utility. How these messages are derived will be discussed later.

WG16: Deregulated Energy Market Communications

The CIM for Markets Extensions (CME) was created under Working Group 16 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.

Standards Process

As an IEC standard, the CIM and all related standards must go through the IEC standards process to be published. This process contains a number of steps:

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.

Working Draft & 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.

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

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.

Time Scales

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.

As such it is not uncommon for users to work on draft documents; however, this must be undertaken with the knowledge that a draft 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 possible 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, e.g. 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 how often the desperate messages are heard; "it takes too long", "it changes too frequently". The fact is that the CIM integration supports the change management capabilities of any vendors interface, using service level and XSD versioning, and 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

Section 4: Core Technologies

Learning Objectives

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

Unified Modeling Language

The Unified Modeling Language (UML)[8] 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[9] 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.

---

[8] For the most recent UML specifications see http://www.omg.org/spec/UML/Current

[9] http://www.uml.org/

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[10]. The radius of the circle would also be 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.

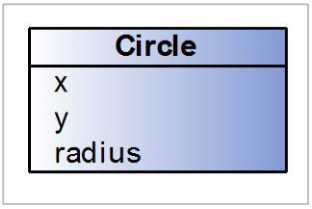


Figure 4-1
The Circle Class

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 etc. 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 etc. 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* etc.

---

[10] 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 centre of the circle. This would be documented within the UML to make it clear to users what the coordinate represents.

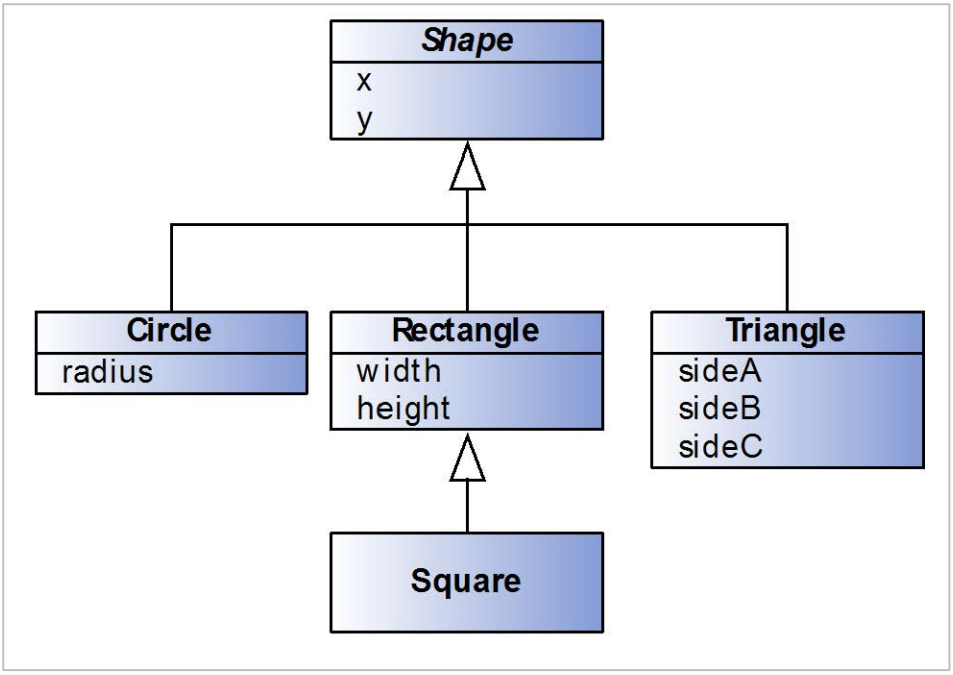


Figure 4-2
Class hierarchy for diagram shapes

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.[11]

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 *side A*, *side B,* and *side C*.

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,

[11] 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.

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 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.

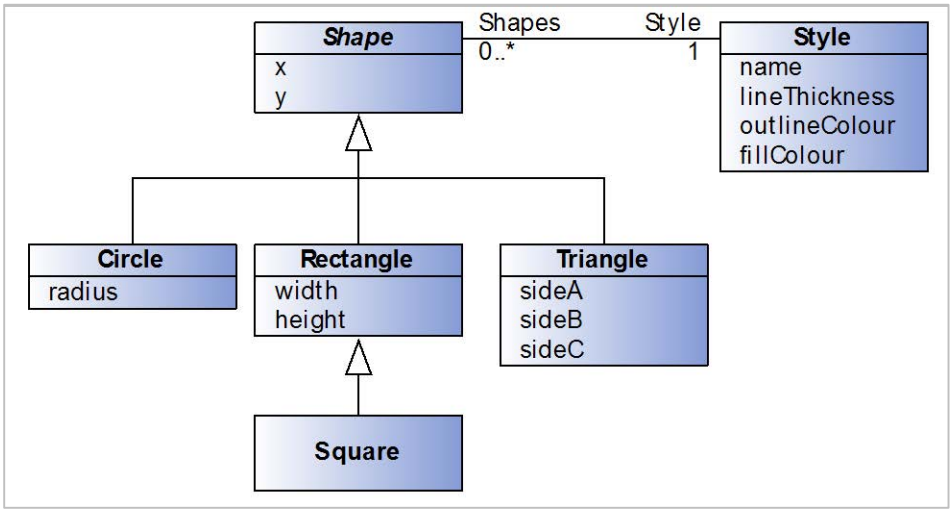


Figure 4-3
Class hierarch of diagram shapes with a Style

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*.

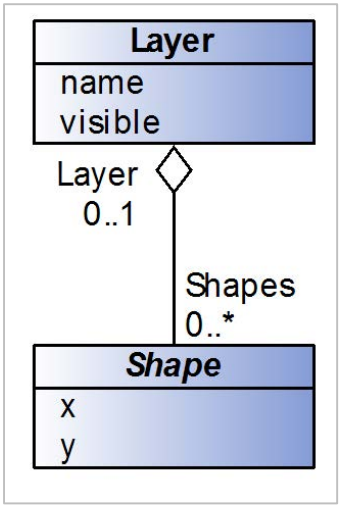


Figure 4-4
Aggregation Example with Layer and Shape

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*.

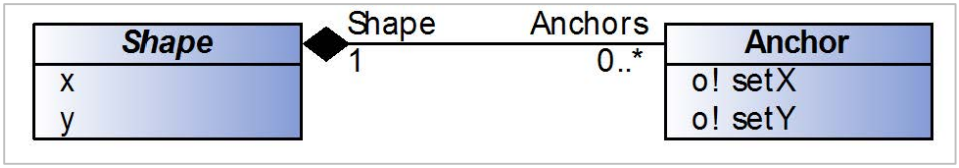


Figure 4-5
Composition Example of Shape and Anchor

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

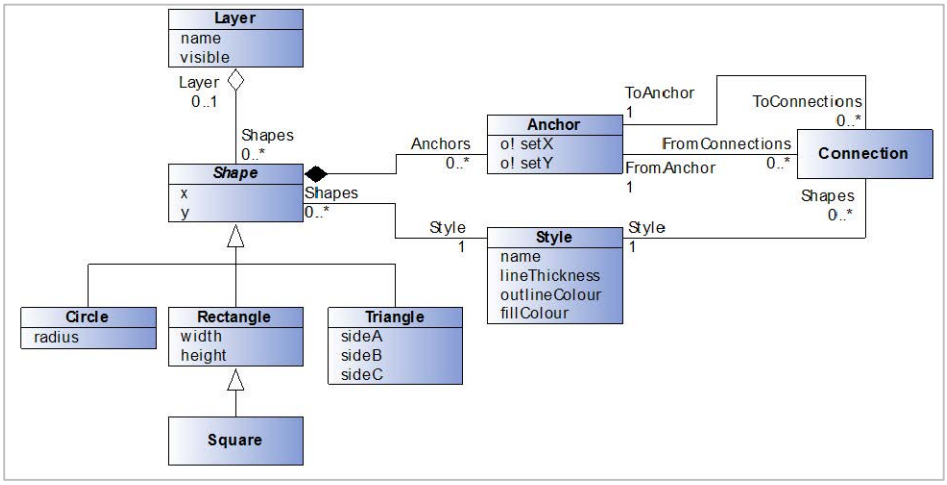


Figure 4-6
Class Diagram for Shapes

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.[12] 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.

---

[12] 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

XML Syntax

XML, the eXtensible Markup Language, is a "universal format for structured documents and data"[13] 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[14] 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[15] designed for both on and offline storage and transfere 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[16] 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.

---

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

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

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

[16] 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 can 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)[17] format or the newer XML Schema Definition (XSD)[18]. 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 (i.e. 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.

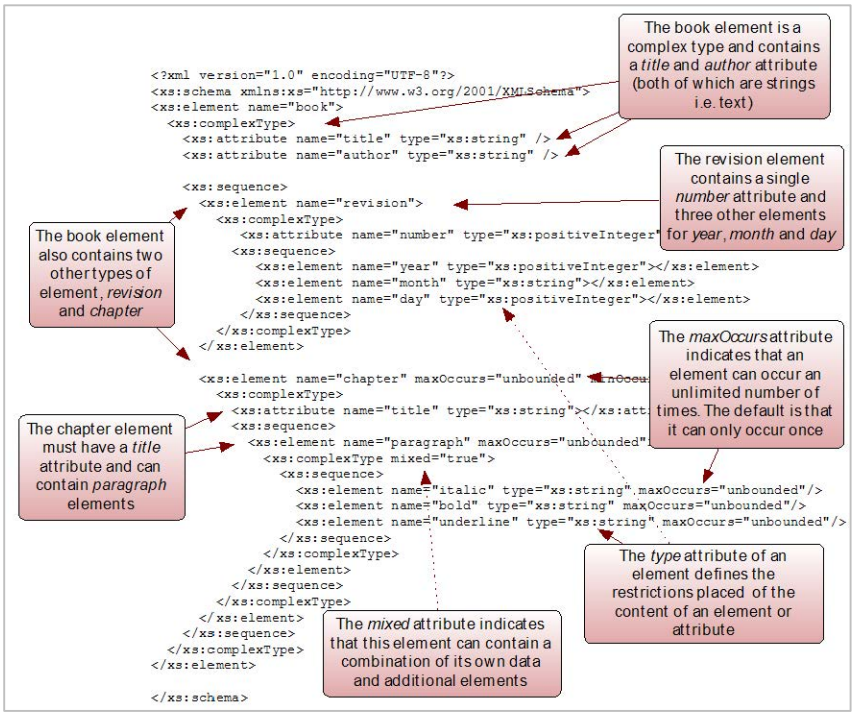


Figure 4-7
Annotated simple XML Schema Example describing the data within a book

---

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

[18] W3C Recommendation, "XML Schema Part 0: Primer Second Edition", October 2004, available at http://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[19].

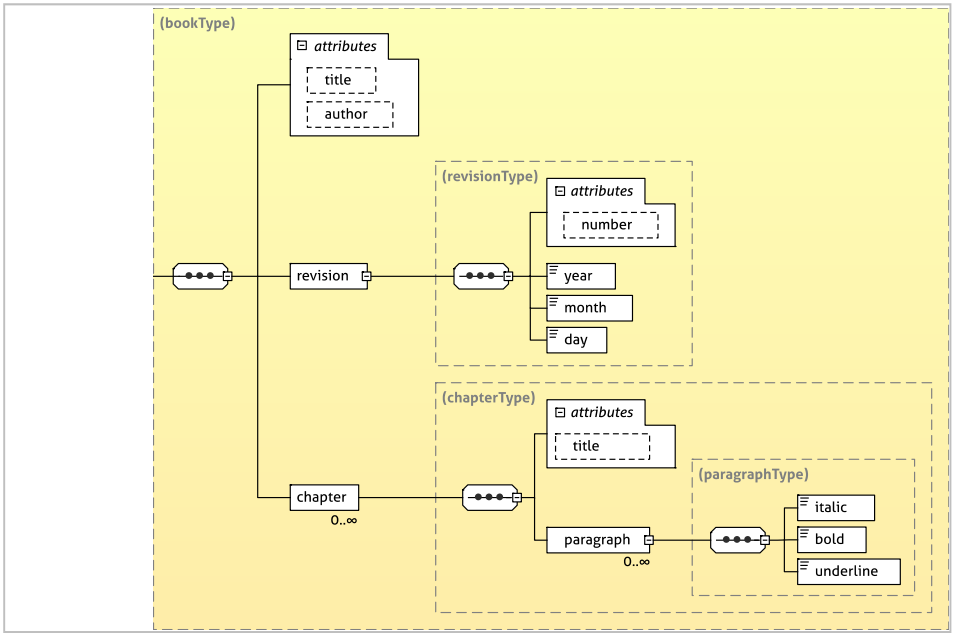


Figure 4-8
XMLSpy Style XML Schema View

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*[20]. 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

[19] XMLSpy is a commercial XML Editor from Altova, http://www.altova.com/xmlspy.html

[20] 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.

namespace, identified by a *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 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

The Resource Description Framework (RDF)[21] 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

[21] W3C Recommendation, “Resource Description Framework”, February 1999, available at http://www.w3.org/TR/REC-rdf-syntax/

former book before the latter, there is no standard way to do this using the basic XML constructs[22].

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* 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>
```

[22] 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.

```
            <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)[23] 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, RDF Schema allows resources to be

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

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 Innovation Electric Utility, has already been exposed to the "acronym soup" of syntactic representation. The integration effort IEU 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 IEU 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 an 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

Section 5: CIM UML

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 incorporate 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 an engineer the format of the Common Information Model (CIM) may at first appear confusing compared with a flat file format. 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 (i.e. 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.

Since the CIM v15 draft was finalized in 2011 a construct has also been 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*.

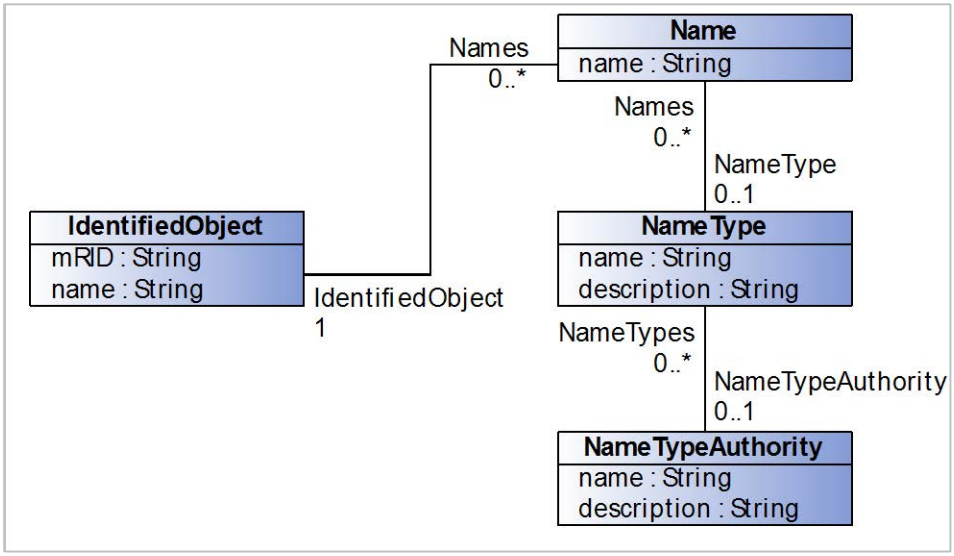


Figure 5-1
IdentifiedObject and the Name Classes

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 (e.g. 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 (e.g. *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 EdF's (Électricité de France) 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. This new approach 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.

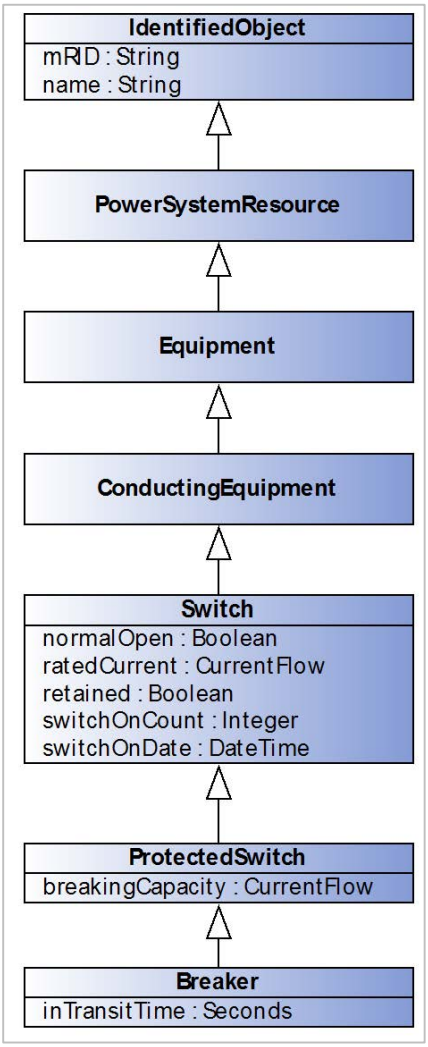


Figure 5-2
Breaker Class Hierarchy in CIM

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[24]; and attributes to record the number of types s 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*.

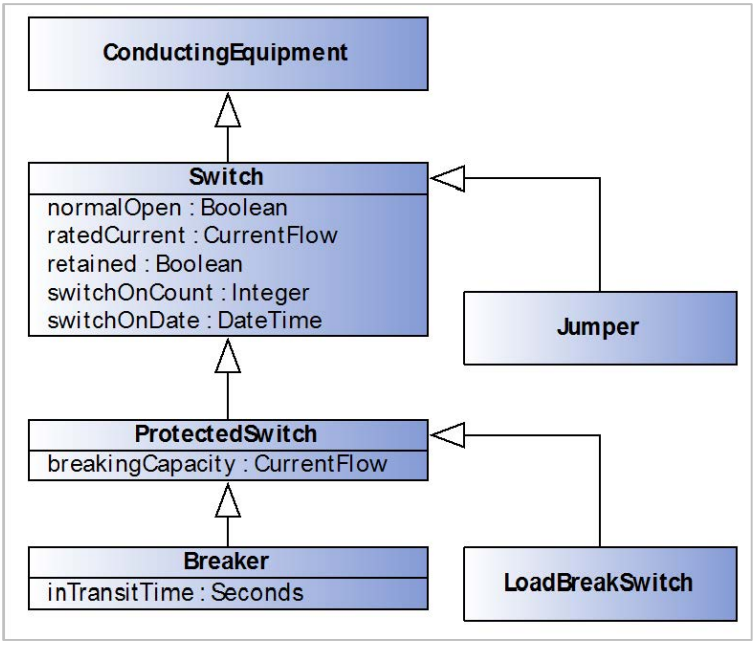


Figure 5-3
Switch Class with expanded subclass hierarchy

[24] 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, pp229-235

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.

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.

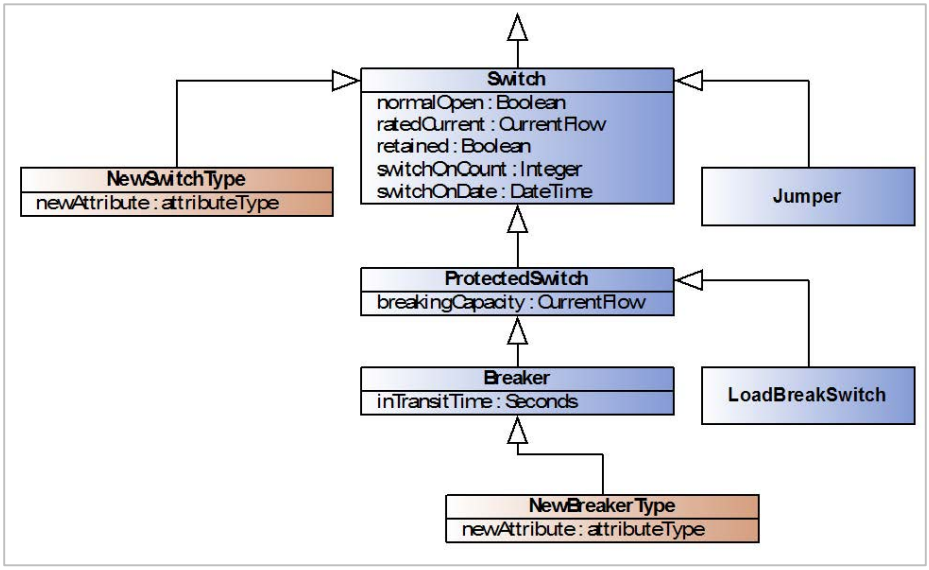


Figure 5-4
Switch class diagram with new types of Switch and Breaker

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 *PowerSystemResourc*e 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:

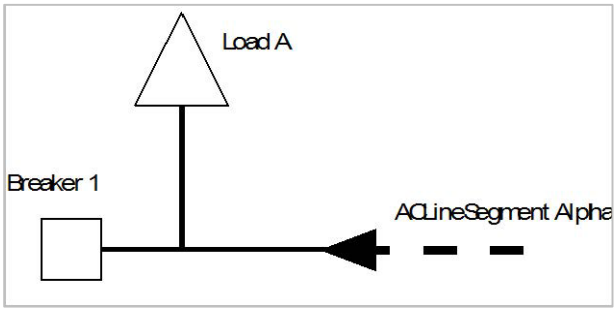


Figure 5-5
Connectivity Example Circuit

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.

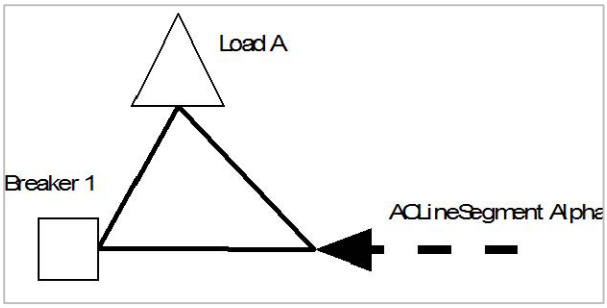


Figure 5-6
Connectivity Example Circuit with Direct Associations

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.

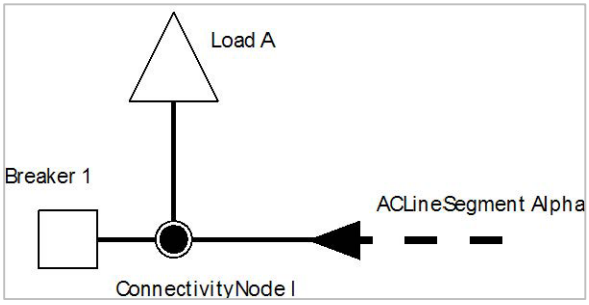


Figure 5-7
Connectivity Example Circuit with Connectivity Node

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.

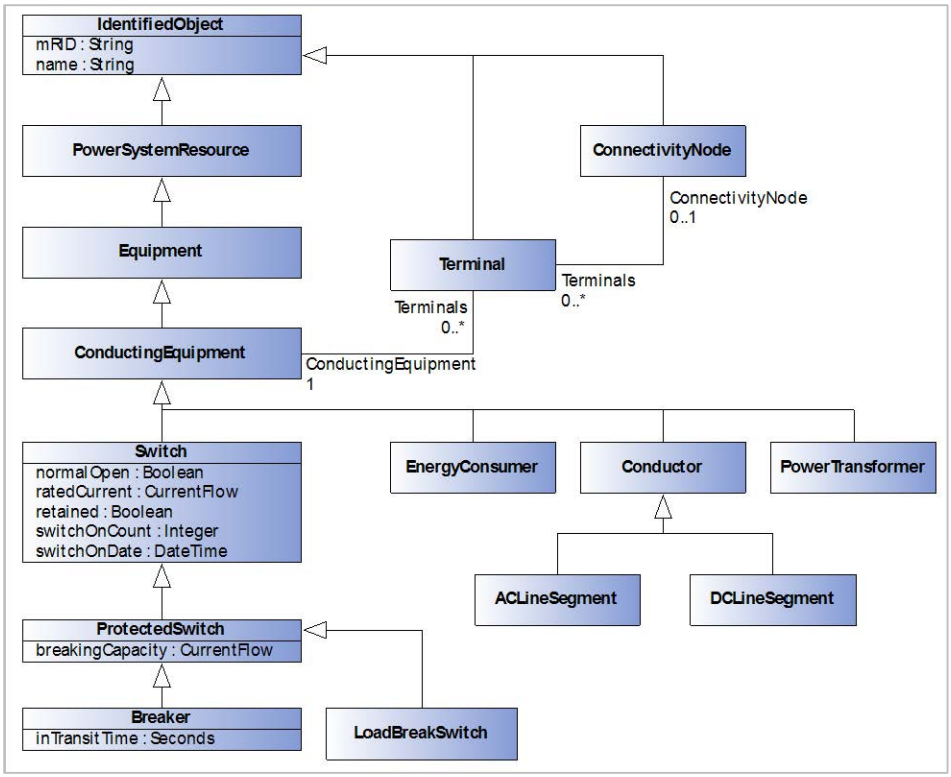


Figure 5-8
Conducting Equipment and Connectivity class diagram

The relationship between the *Terminal*, *ConnecitivtyNode* 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.

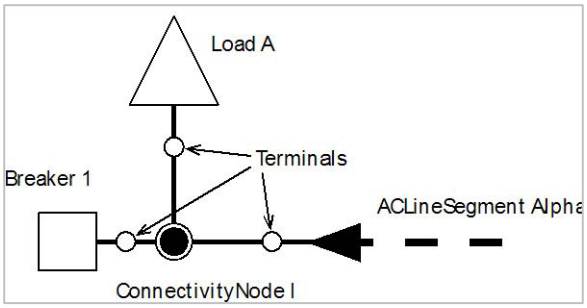


Figure 5-9
Connectivity Example Circuit with Connectivity Node and Terminals

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.

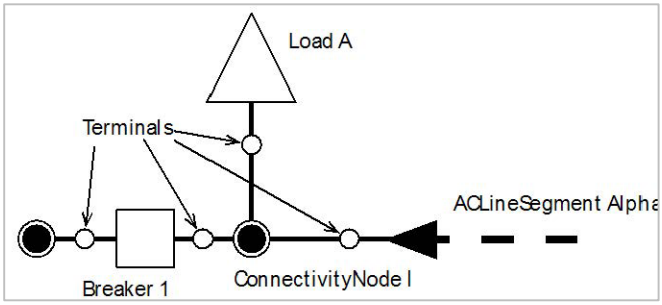


Figure 5-10
Connectivity Example Circuit with Additional Breaker Terminal

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.

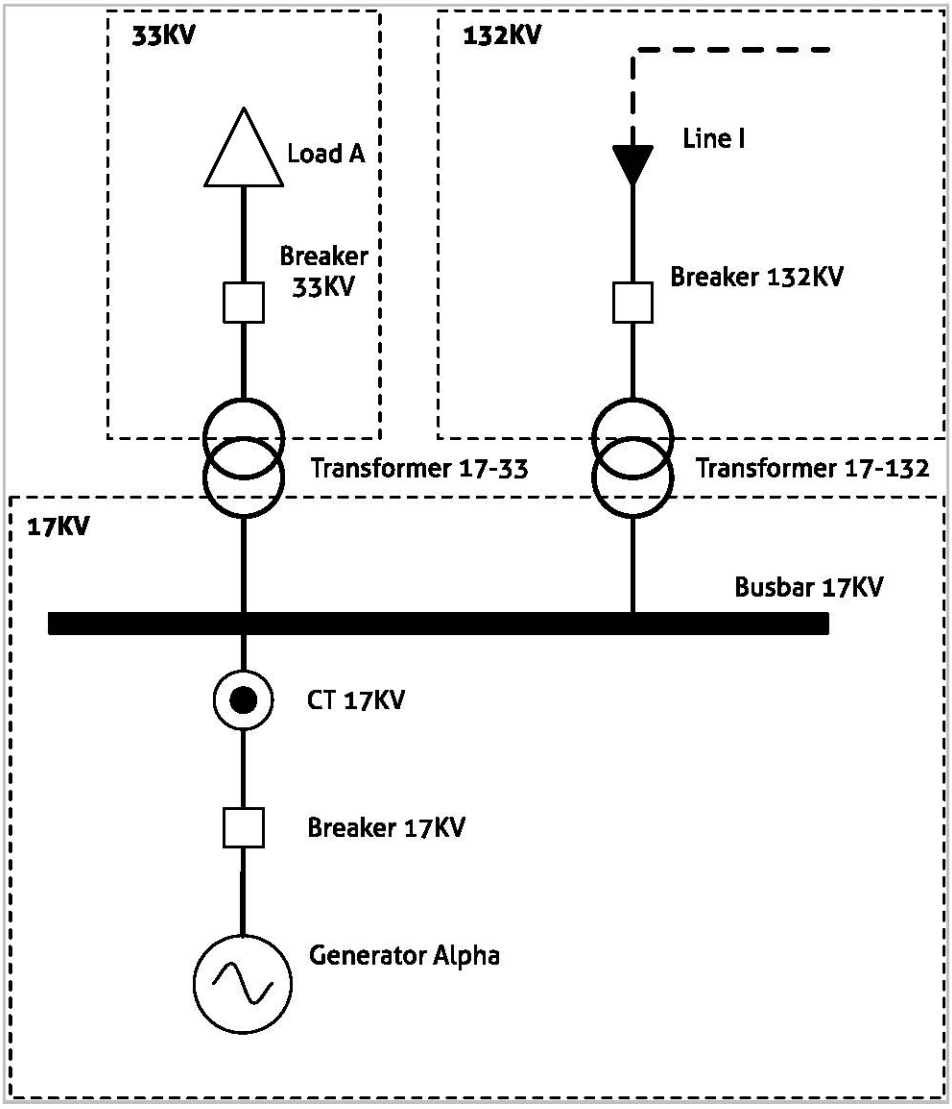


Figure 5-11
Example Circuit as a Single Line Diagram

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.

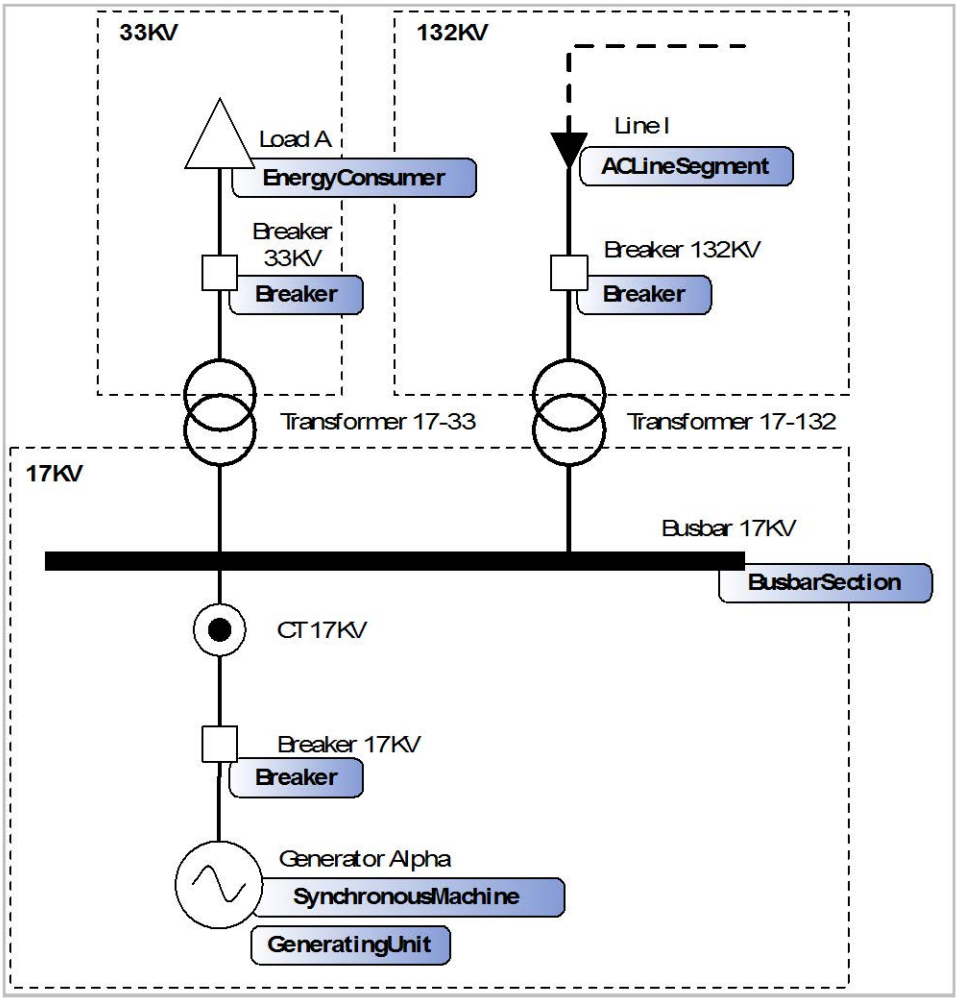


Figure 5-12
Example Circuit with Partial CIM Class Mappings

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.

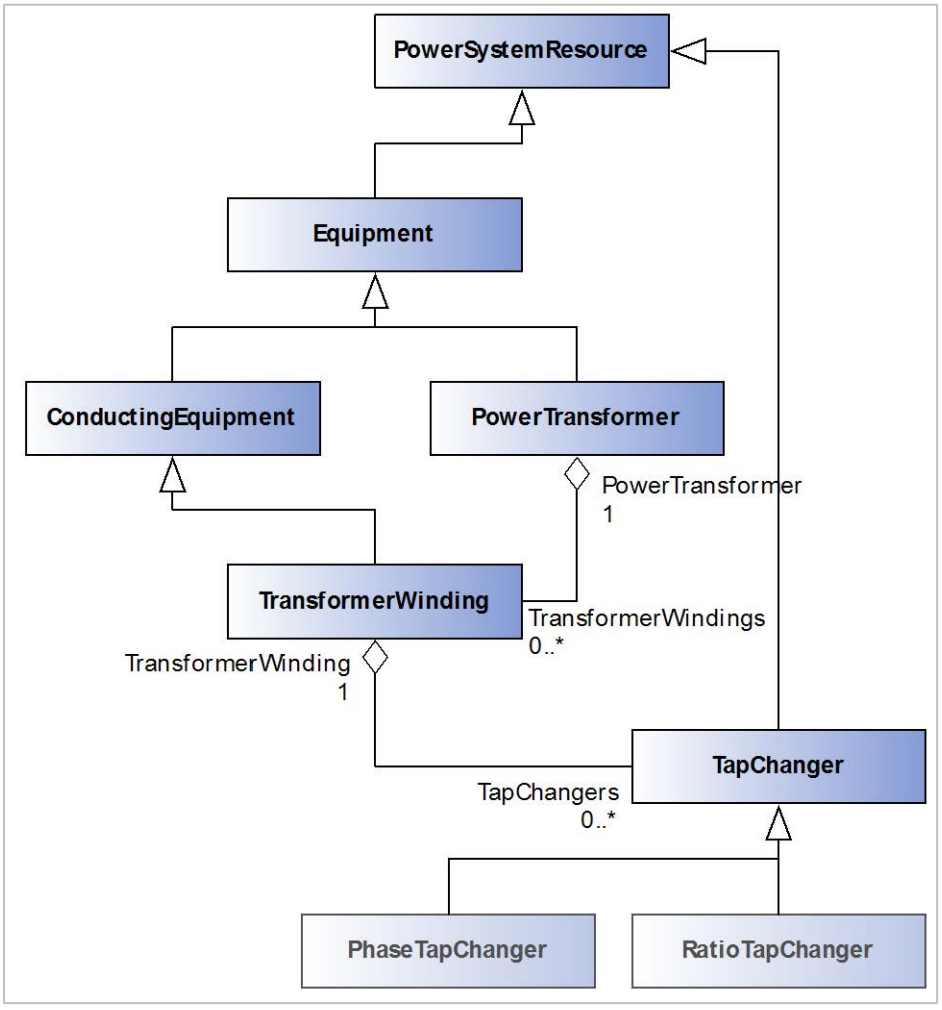


Figure 5-13
Transformer Model (pre-CIM v15)

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[25], meaning that a *PowerTransformer* is made up on 1 or more *TransformerWindings* which in turn can be made up of zero or more *TapChangers*.

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*.

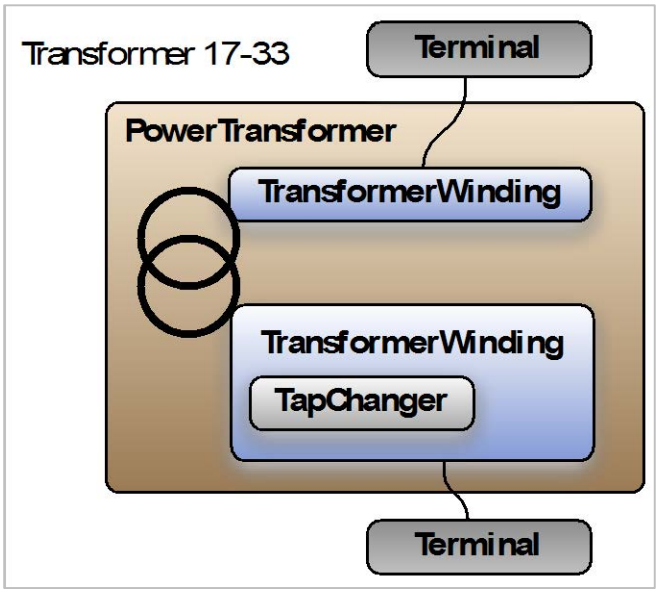


Figure 5-14
CIM Transformer Instance Example

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.

---

[25] 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 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 (e.g. 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[26].

- 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

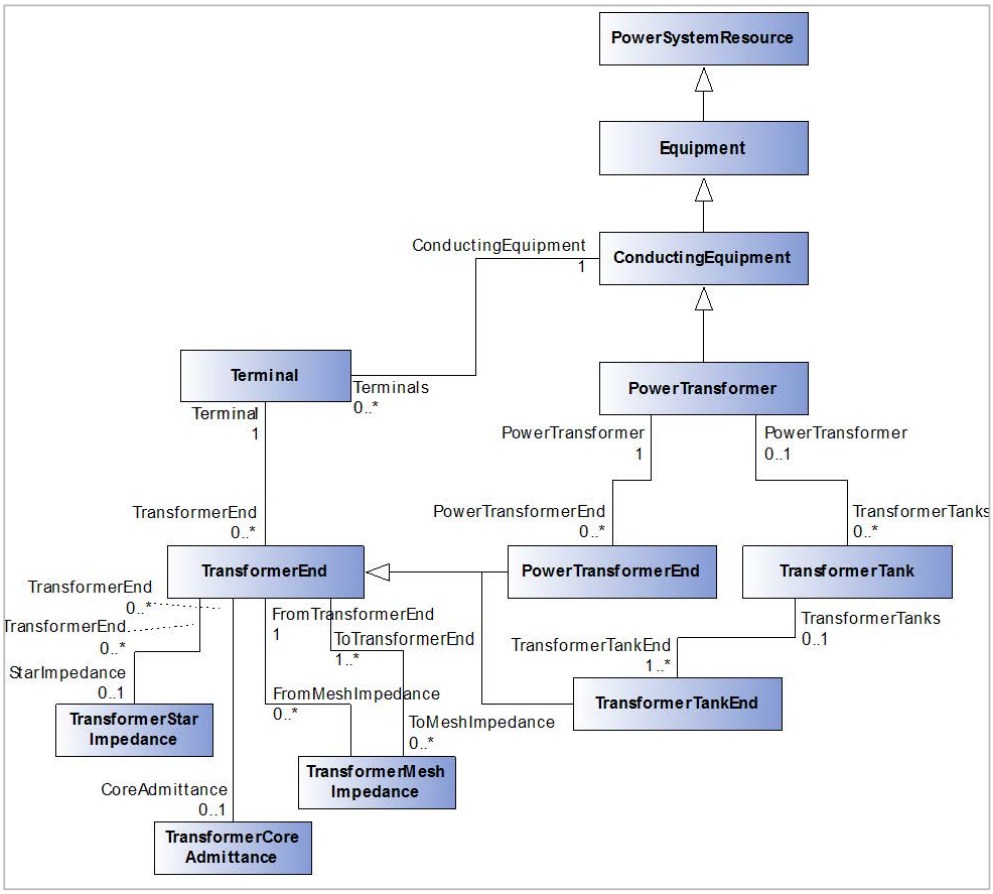


Figure 5-15
Transformer Class Diagram CIM15+

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

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

multi-phase transformers (e.g. pole-mounted distribution transformers) and when modeling a balanced transmission level power transformer the changes are minor.

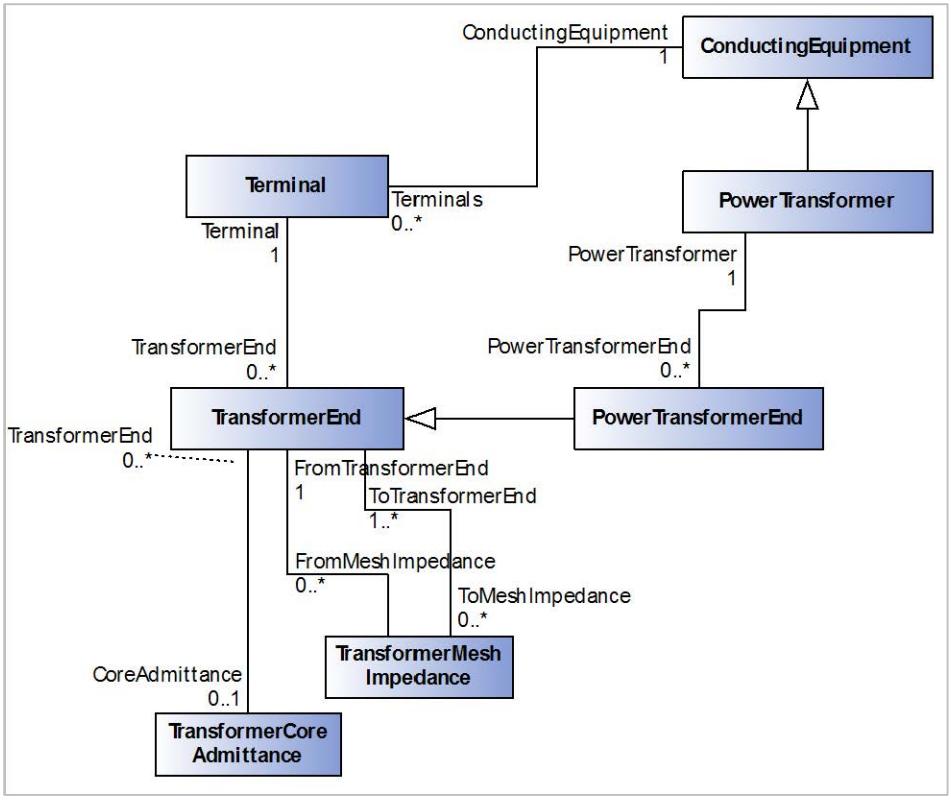


Figure 5-16
Balanced Power Transformer Classes

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*.

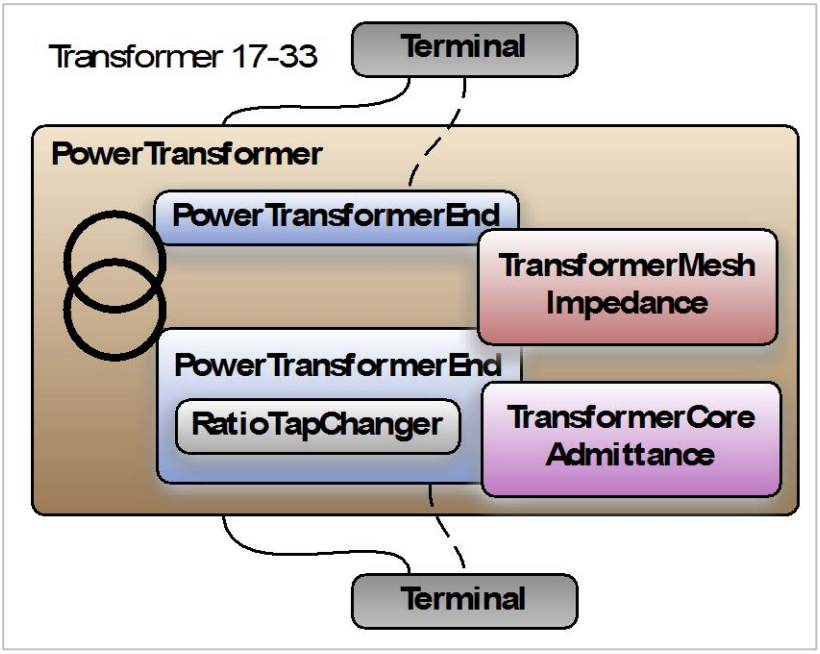


Figure 5-17
CIM v15 Transformer Instance Example

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

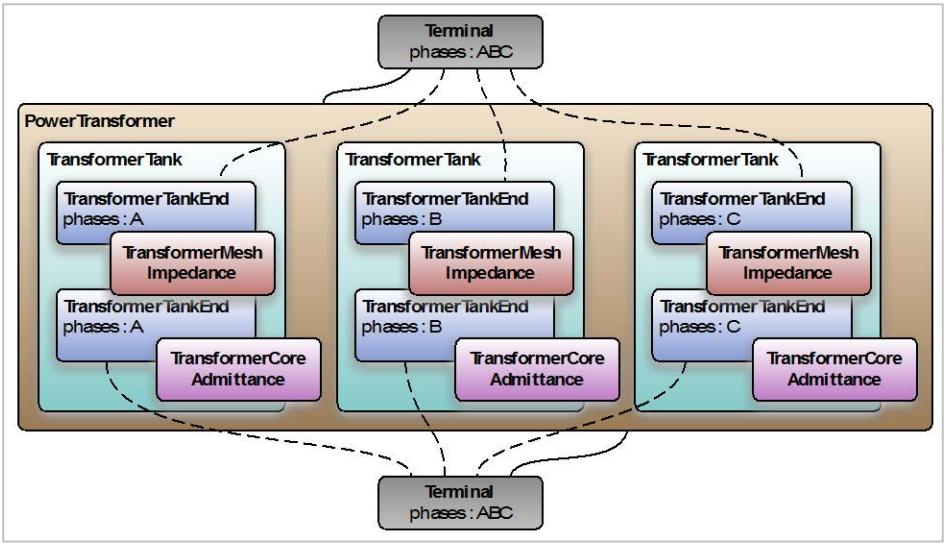


Figure 5-18
CIM v15 Transformer with multiple tanks instance example

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[27] 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.

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

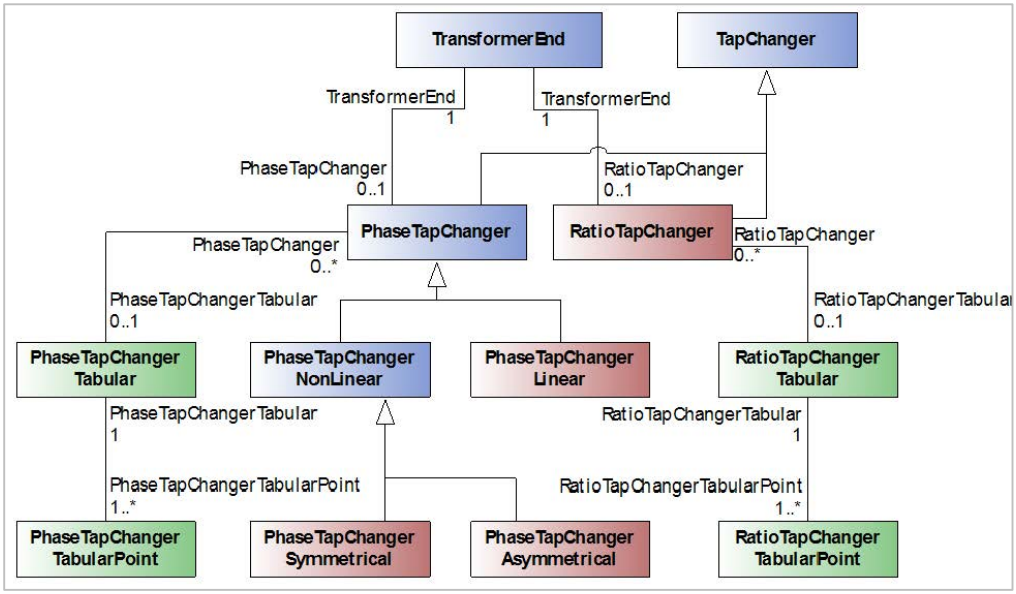


Figure 5-19
Tap Changer Class Diagram CIM v15

The *Tap Changer* model shown in Figure 5-19 at first appears to be very complex but this is primarily due to the classes 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 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 (e.g. 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 & 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[28] etc.

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 etc.) or the operating mode of a *Synchronous Machine* (generator or condenser).

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.

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

CIM Representation

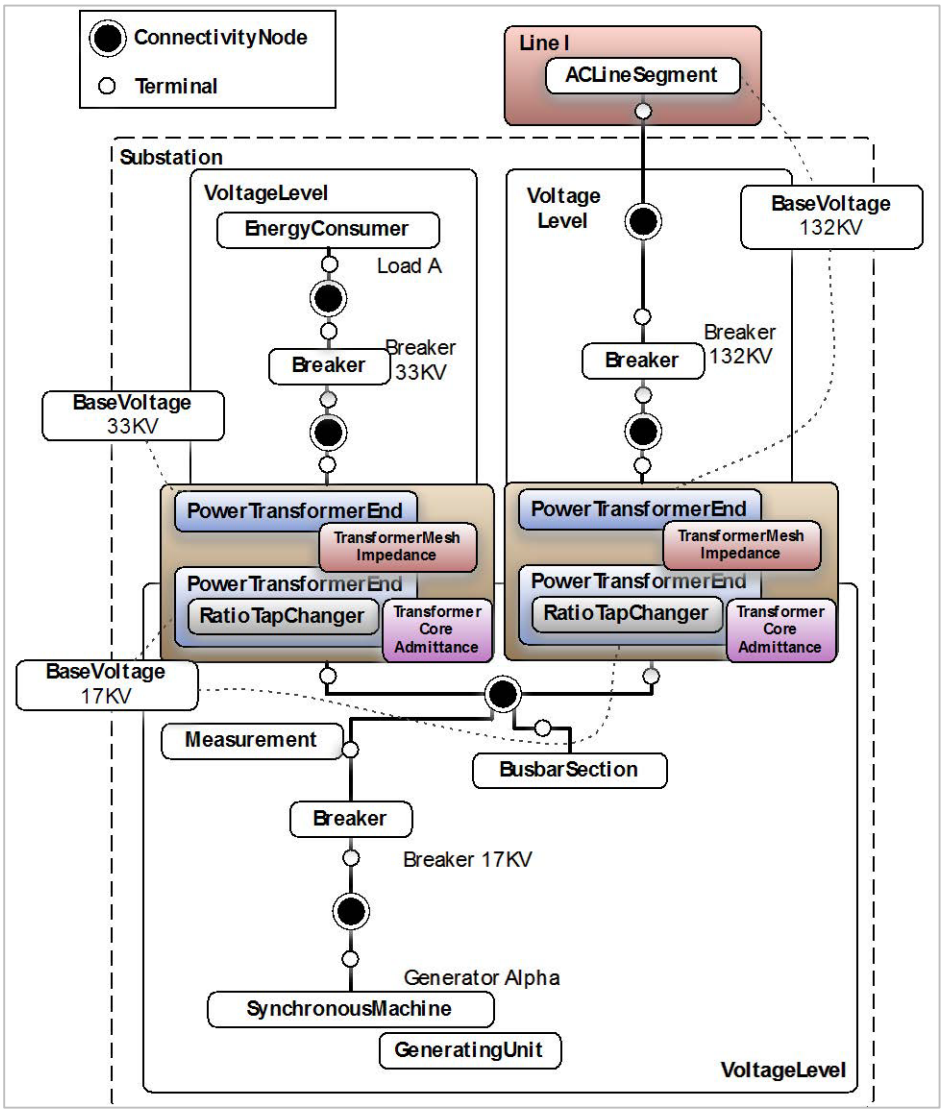


Figure 5-20
Complete CIM Representation

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 etc. 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.

Asset vs. Functional

The Asset View

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, etc. 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.

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 etc. from the perspective of procurement, maintenance, accounting etc. This view of the network includes the manufacturer, model, purchase date, installation date, maintenance history etc. 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 (e.g. trucks, buildings, tools etc.) 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.

Some assets may also perform multiple functions from the functional network perspective; a single piece of physical equipment may be functioning as a *Breaker* and a *Measurement* from the CIM perspective. 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.

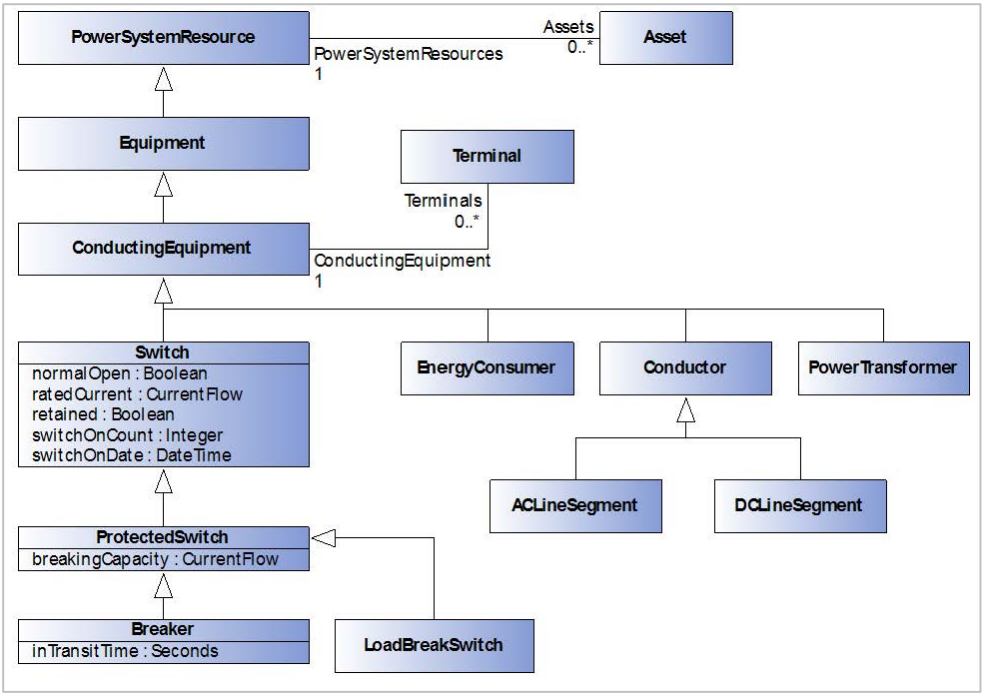


Figure 5-21
Excerpt of Functional Model with Asset Class

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.

Changing an Asset

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 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.

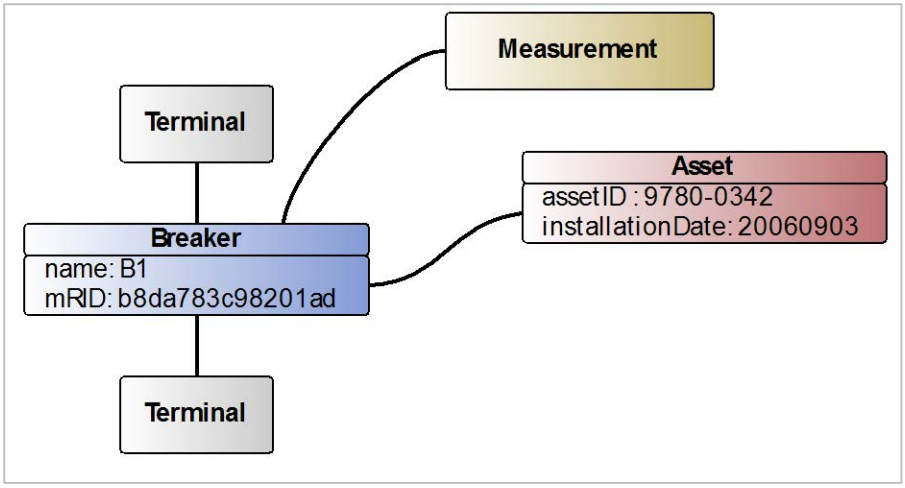


Figure 5-22
Breaker-Asset Instance Example

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,* it's *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.

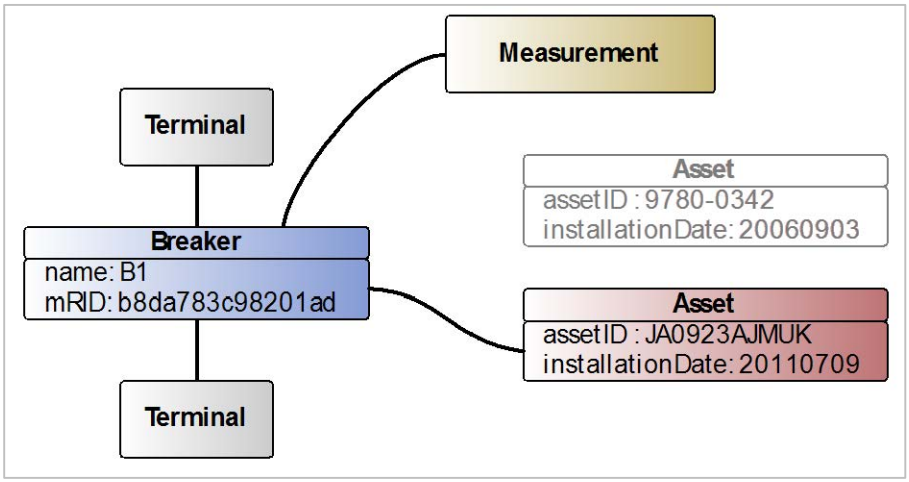


Figure 5-23
Breaker-Asset Change Example

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.

Diagram Layouts in CIM

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 CIM Diagram Layout (CDL) format extends the CIM to provide a mechanism for describing the layout of electrical schematics independent of the source or target systems' vendor. This standard will allow graphics and network data to be exchanged simultaneously, thus ensuring that schematic views are synchronized with the electrical model across multiple systems. The standard is intended to allow 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 when developing the CDL 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 CDL 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.

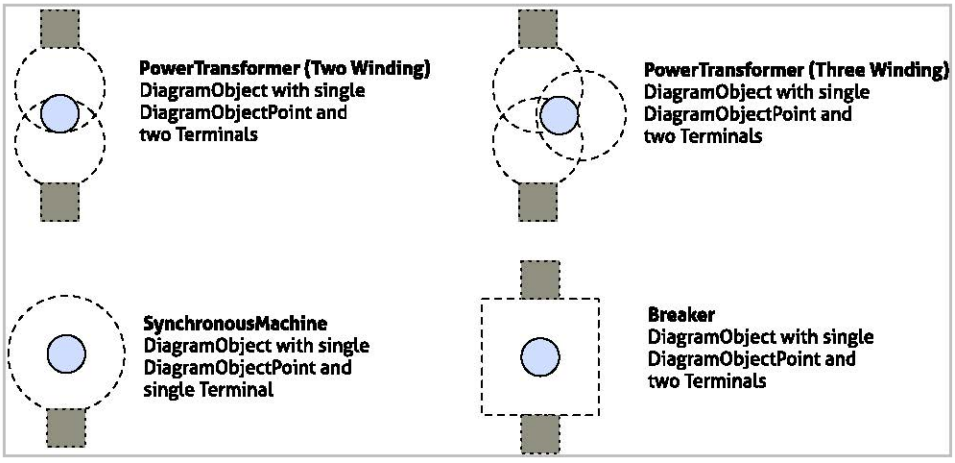


Figure 5-24
CIM Diagram Layout Single Point Examples

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.

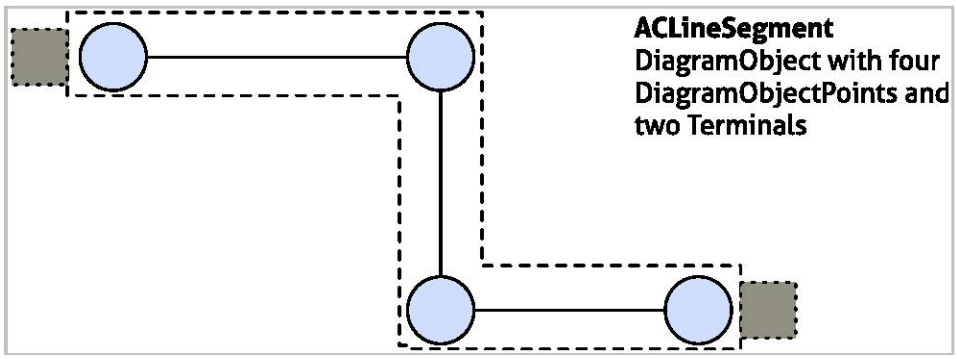


Figure 5-25
CIM Diagram Layout Multiple Point Example

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 are 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 UML

The CDL 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.

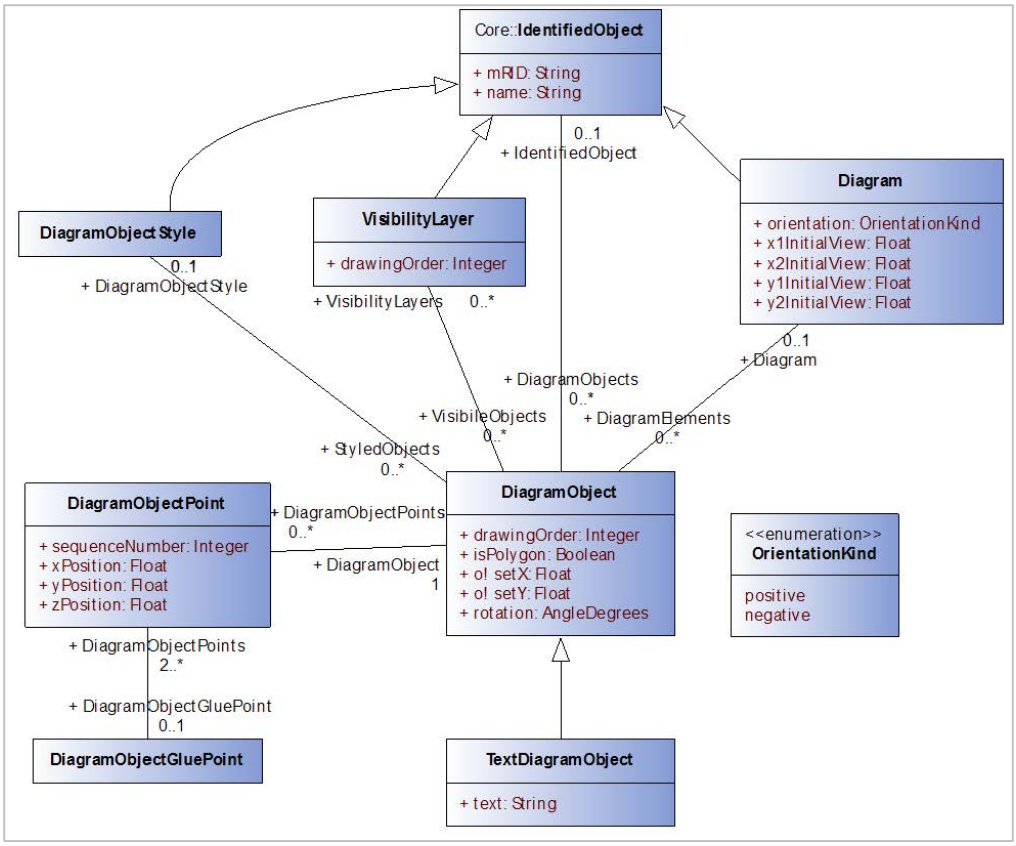


Figure 5-26
CIM Diagram Layout Class Diagram

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 (e.g. 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, but 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 (e.g. 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 CDL 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 similar construct to the CDL 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 (e.g. Latitude Longitude with Decimal Degrees, Lambert I-IV, GB Ordnance Survey Grid Codes etc.).

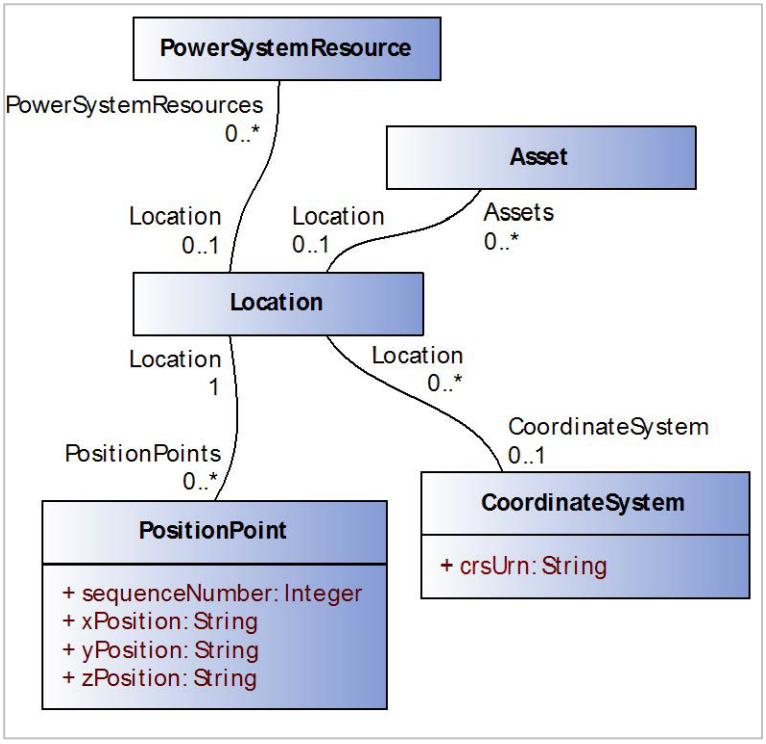


Figure 5-27
CIM Location Class Diagram

As with the CDL 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 etc. 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.

Unbalanced Multi-Phase Network Modeling

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 (i.e. current, voltage, reactive power, tap position, switch status etc.) to each phase.

When there is a single CIM device 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

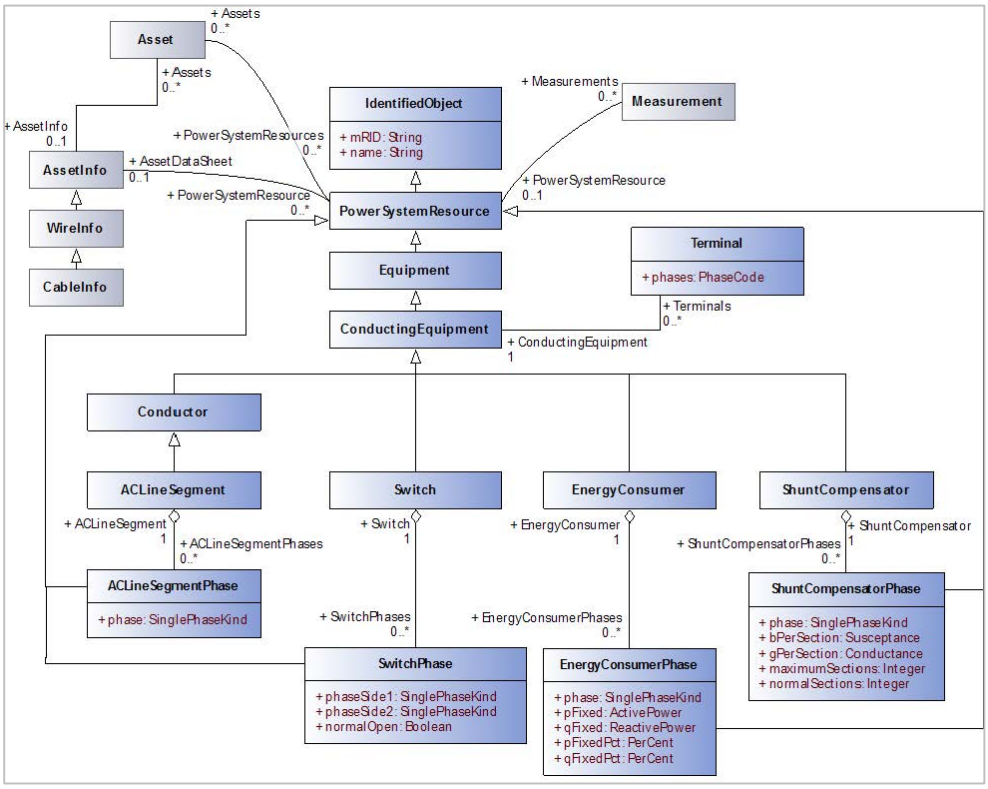


Figure 5-28
Unbalanced three-phase network components

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[29] 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 User 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.

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*.

[29] 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

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 etc.

Case Study

Jeff Kimble arrives at his first CIM Users Group meeting. On 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 firehouse of information". In one day attendees get an introduction to the CIM UML, the tool that is used to maintain it (Sparxsystems 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 profiles is a "subset of the model, used to perform meaningful work, such as being 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 the use of network model exchange because the RDF can be used to describe how various assets are related to each other, which is important in network model exchange. 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

Section 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 covered by the Interface Reference Model (IRM) as shown in Figure 6-1. This has meant that the model itself has grown as its scope has increased, from a core set of less than 100 classes to describe a balanced electrical model from the perspective of an EMS, to a model of 600+ classes with thousands of attributes and associations.

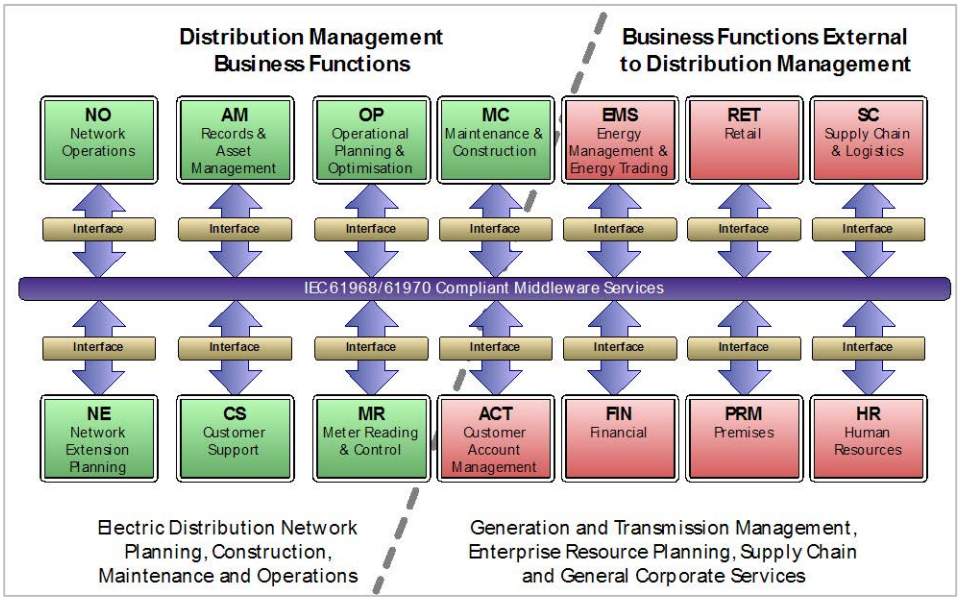


Figure 6-1
IEC61968-1 Interface Reference Model

It is this all-encompassing model that new users often encounter, either as the IEC standard documents for 61970-301 and 61968-11, or the UML model itself from Enterprise Architect or in XML Meta Interchange (XMI) format. Since the model includes every class and element it can appear overly complex and impractical for those seeking simple implementations or viewing the CIM for the first time.

Information to Implementation

The CIM is 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 incompatible. As such, the model needs to be restricted for each context into a profile, a subset of the full model that is derived to define the data exchanges required for each interface. 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.

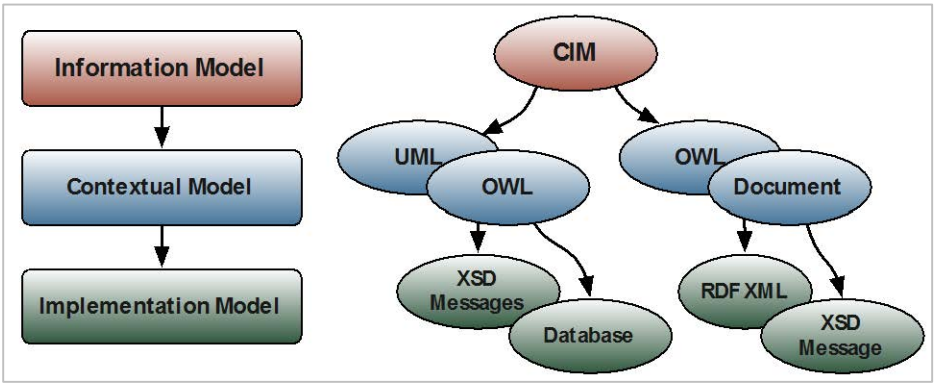


Figure 6-2
Information, Contextual & Implementation Models

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

When viewed as its profile, a CIM interface is no longer a complex model of 600+ classes but is instead a small number of classes with a subset of their attributes and associations. When multiple interfaces are to be implemented by a single system, the use of a single overall model from which the profiles are derived means that the data definitions are re-used, not duplicated. 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 or market systems.

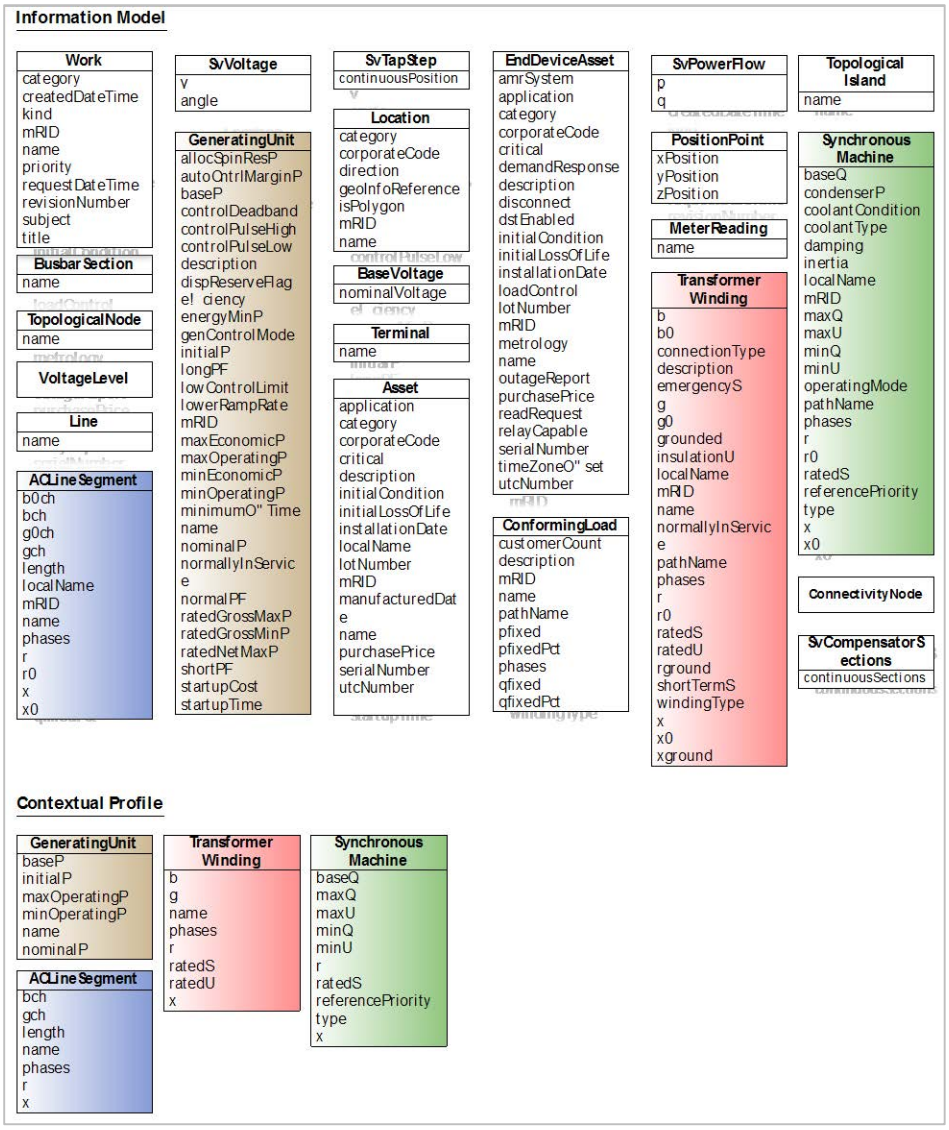


Figure 6-3
Example Information to Contextual Models

As shown in Figure 6-3, the addition of this *context* (i.e. *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.

Profile 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 systems or low-voltage pole-mounted 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.

CIM Standard Profiles

Normally the CIM would not be implemented in its entirety. IEC standard documents such as 61970-452[30] (Common Power System Model), 61968-13[31] (Common Distribution Power System Model) and ENTSO-E[32] are examples of profiles derived from the overall CIM model used to exchange 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.

Common Power System Model

The original CIM profile (now profile group) is IEC 61970-452, the Common Power System Model which is used for the exchange of node-breaker electrical power system models. Originally aimed at exchanges between EMS it has since grown to encompass the exchanges between multiple systems within the transmission environment.

The key profiles are Equipment, Topology, State Variables and proposed profiles for Analog and Discrete Measurements (from SCADA) and the recent addition of the Diagram Layout profile. Together they all constitute the data exchanges between systems as part of the EMS network analysis process as shown in Figure 6-4.

---

[30] "Energy management system application program interface (EMS-API) - Part 452: CIM Transmission Network Model Exchange Profile", IEC, Draft

[31] "Application integration at electric utilities - System interfaces for distribution management - Part 13: CIM RDF Model exchange format for distribution", IEC, Edition 1.0, June 2008

[32] "Interoperability Test 'CIM for System Development and Operations' 2011", ENTSO-E Final Report, 15th August 2011

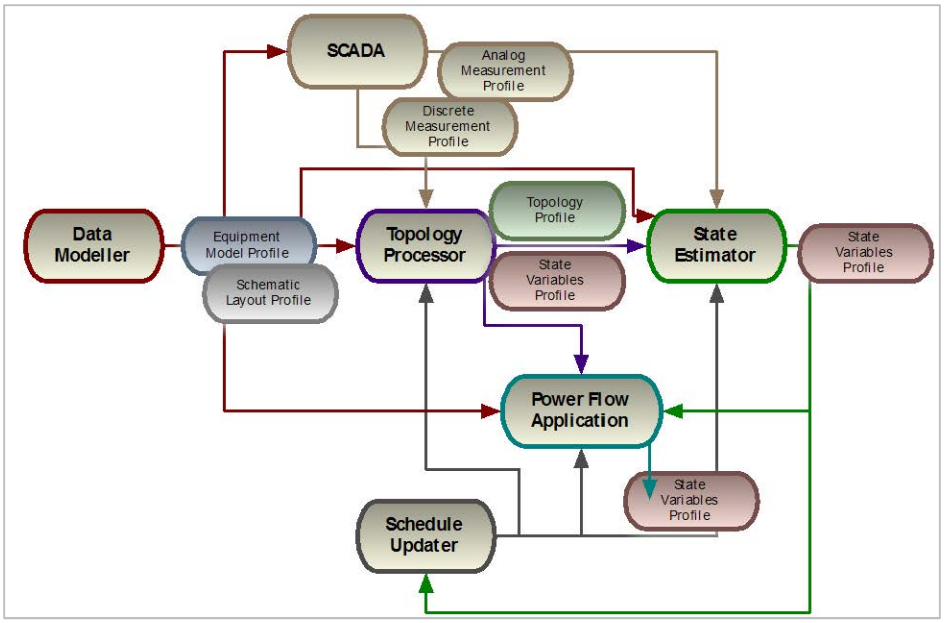


Figure 6-4
EMS Network Analysis Systems and Data Exchanges

Each instance of a profile (i.e. 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 *Terminals* and/or *TopologicalNodes* 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 etc. 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[33].

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.

ENTSO-E

The European Network of Transmission Systems Operators for Electricity (ENTSO-E) started an initiative in November 2007[34] 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

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

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

based exchange. ENTSO-E exchanges involve 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 2011 the second edition of the second version of the ENTSOE-E CIM XML was interoperability tested. 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 standard is 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 v15) and by modularizing the profiles and data in this way applications can import only the data required for their own internal processes.

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 so that provides one area to begin his CIM journey. Alternatively, 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. In the context of the CIM, a profile is a:
    a. Subset of the model that may be more restrictive in its definition
    b. 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

Section 7: CIM RDF 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[35] 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>
```

[35] 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="ConductingEquipment">
        <rdfs:label xml:lang="en">ConductingEquipment</rdfs:label>
        <rdfs:subClassOf rdf:resource="#Equipment"/>
</rdfs:Class>

<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:

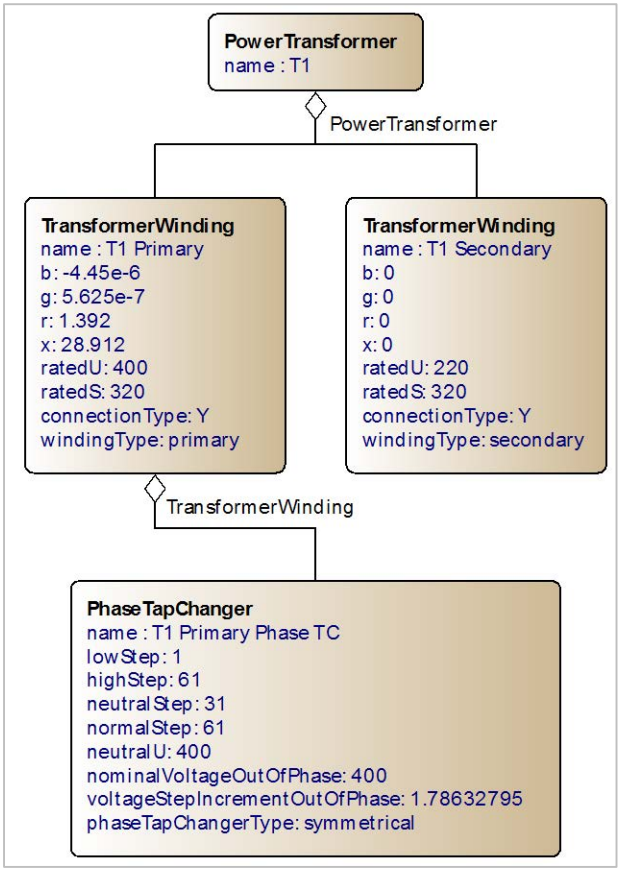


Figure 7-1
Transformer Shown as four CIM Objects with attributes

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 (e.g. 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[36]. In RDF every node has to have its own unique ID in the form of the *rdf:ID* (i.e. 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. i.e. 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[37] (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 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.

---

[36] 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 (i.e. a perfect example of *composition* as described in the background to UML in Section 4) so it does not needs it own, persistent ID as it is the curve itself that is the persistent entity.

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

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 & 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 & Efficiency

The nature of XML, its verbose format where the XML tag elements often take up more space then 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[38] lossless data compression algorithm, itself based on LZ77[39] and Huffman coding[40].

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.

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.

---

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

[39] Jacob Ziv and Abraham Lempel, "A Universal Algorithm for Sequential Data Compression," IEEE Transactions on Information Theory, May 1977, Volume 23, Number 3, pp. 337–343

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

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.[41]

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.

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>
```

---

[41] 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 (e.g. 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 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.

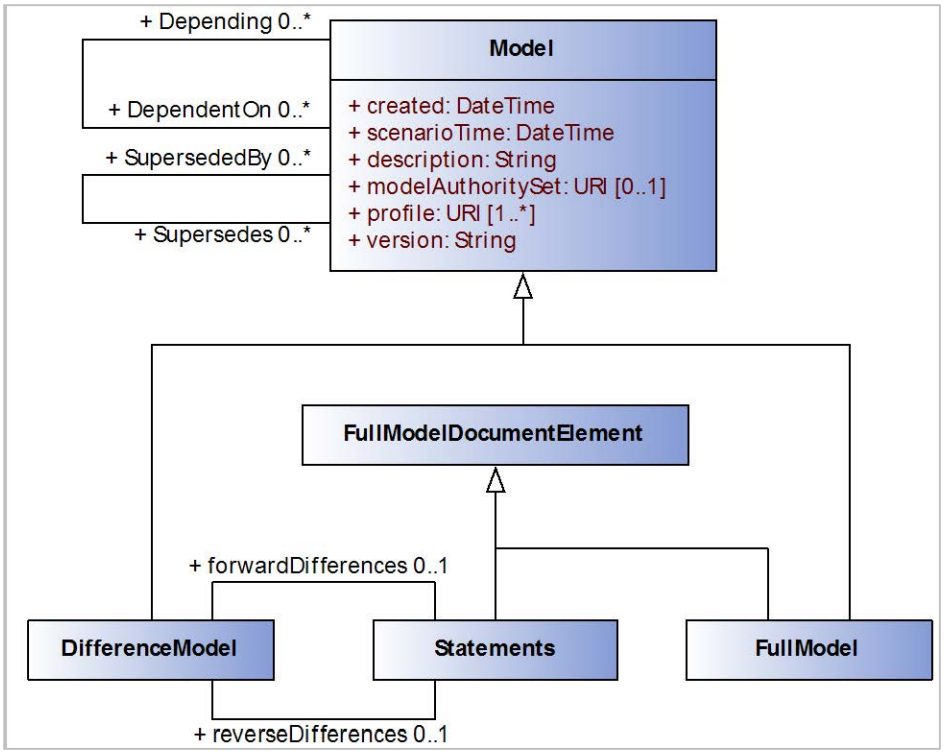


Figure 7-2
CIM Header UML Model

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[42].
- 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 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

[42] 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.

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.

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 the size the CIM specifications also include guidance on how to send *difference model* that contain only the information that describes the changes between a previous version 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 the 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

Section 8: CIM XSD Messages

Learning Objectives

- Understand how to map application interfaces using the CIM
- Understand how to create a 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 will be a number of computer applications that must communicate with each other. 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 number 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.

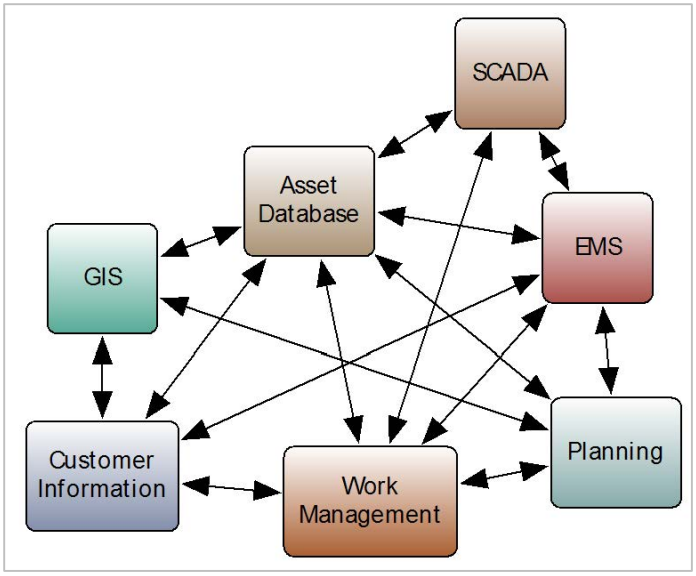


Figure 8-1
Communications between enterprise applications

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

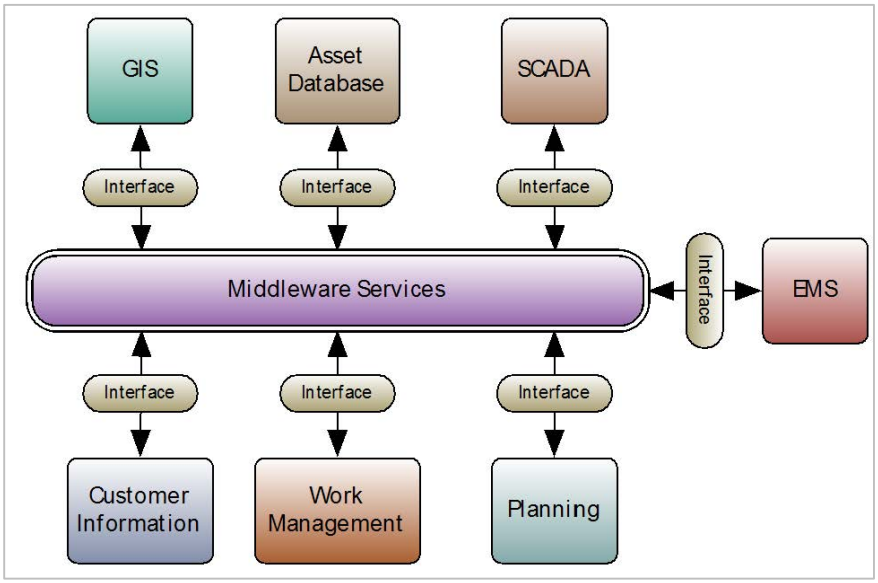


Figure 8-2
Enterprise Service Bus model for inter-application communication

Enterprise Application Integration (EAI)[43] replaces these dedicated links with a single communication link called a "message bus". 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[44]. 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.

Naming & Design Rules

The IEC has defined *Naming and Design Rules*[45] (NDR) to define how the CIM UML is mapped into XML Schema messages. These rules follow the United Nations/Center for Trade Facilitation and E-Business (UN/CEFACT) rules for deriving profiles from an information model and the NDR ensure there is a standard approach for building XML Schemas from the CIM.

In essence the NDR 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 etc. This removes ambiguity for deriving schema and ensures multiple users and vendors will produce compatible schema. The NDR is currently a draft IEC standard and the reader is urged to refer to this document for more explicit details on how the CIM UML maps to XML Schema.

A number of free, open-source and commercial tools implement these naming and design rules including:

- CIMTool by Langdale Consultants
- CIMConteXtor by Zamiren
- CIM EA by Xtensible Solutions

---

[43] D. Linthicum, Enterprise Application Integration, Addison Wesley Longman, Reading, Massachusetts, 2000.

[44] G. Robinson, "Model Driven Integration (MDI) for Electric Utilities", Proceedings of Distributech, Miami Beach, Florida, USA, March 2002

[45] IEC 62361-100: Naming and Design Rules for CIM Profiles and XML Schema Mapping

For the purposes of illustration an example of how to derive an XSD will be shown using CIMTool.

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
- 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.

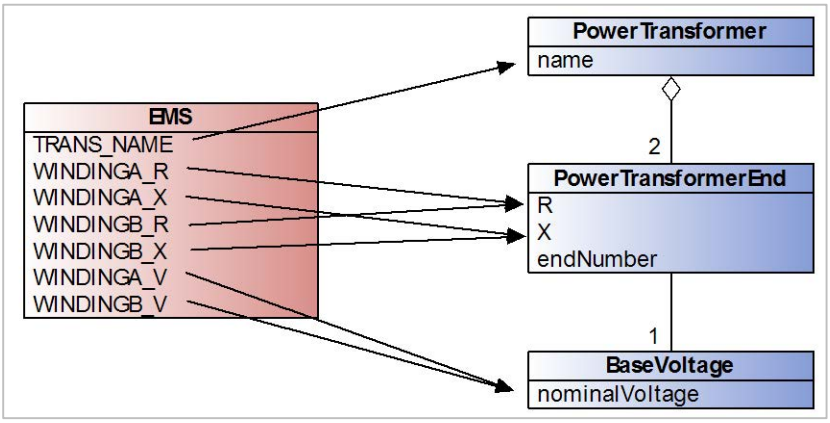


Figure 8-3
CIM Interface Mapping

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[46] 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.

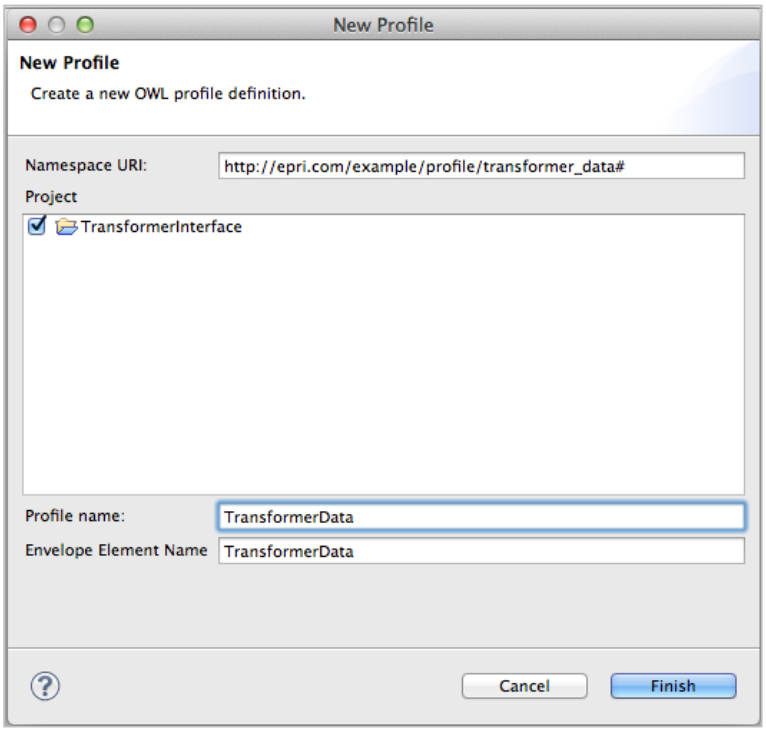


Figure 8-4
Creating a new CIMTool profile

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

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.

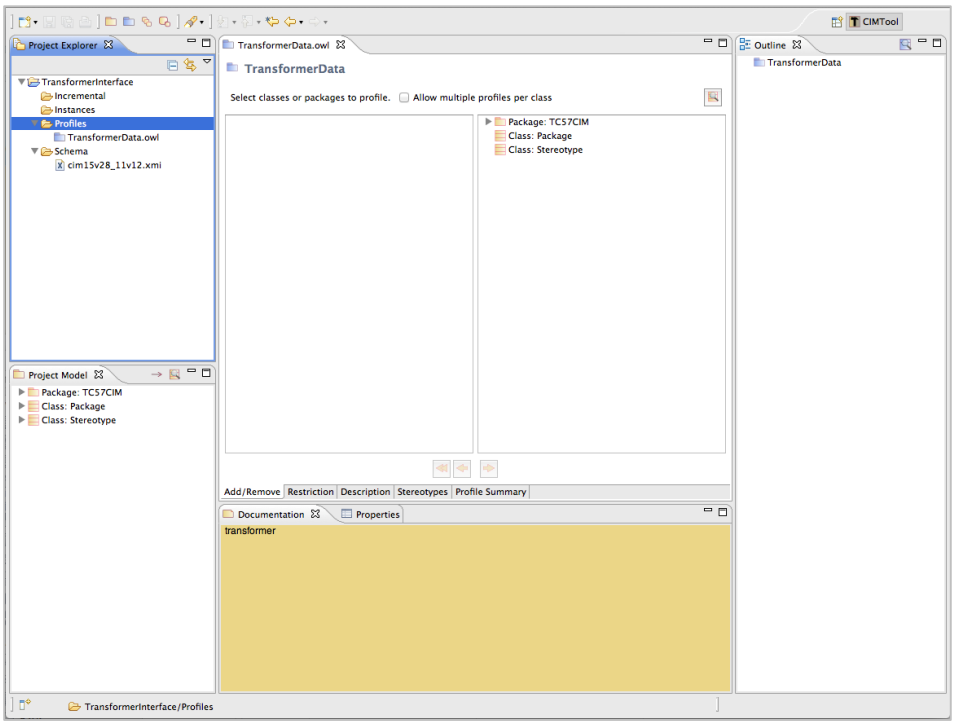


Figure 8-5
Empty Profile

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.

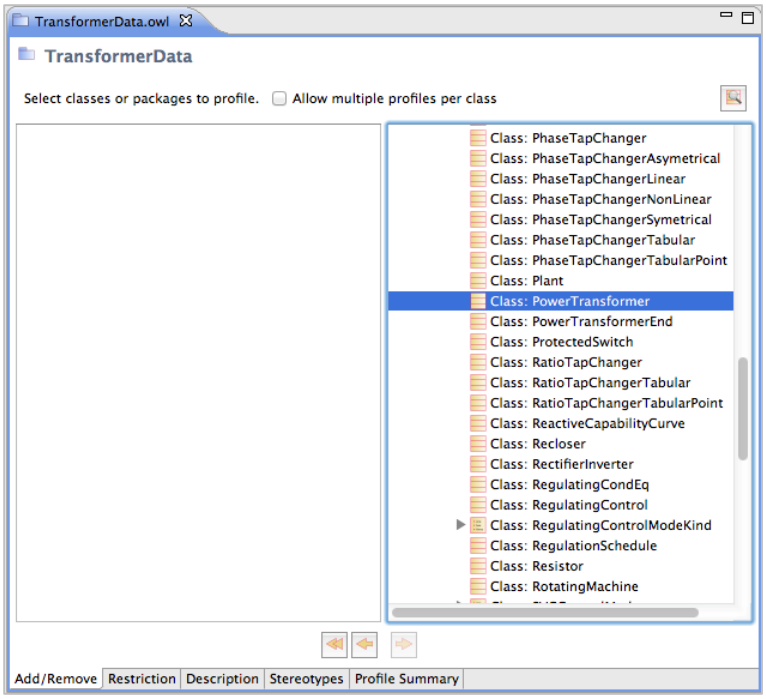


Figure 8-6
Selecting a class to be added to the profile

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.

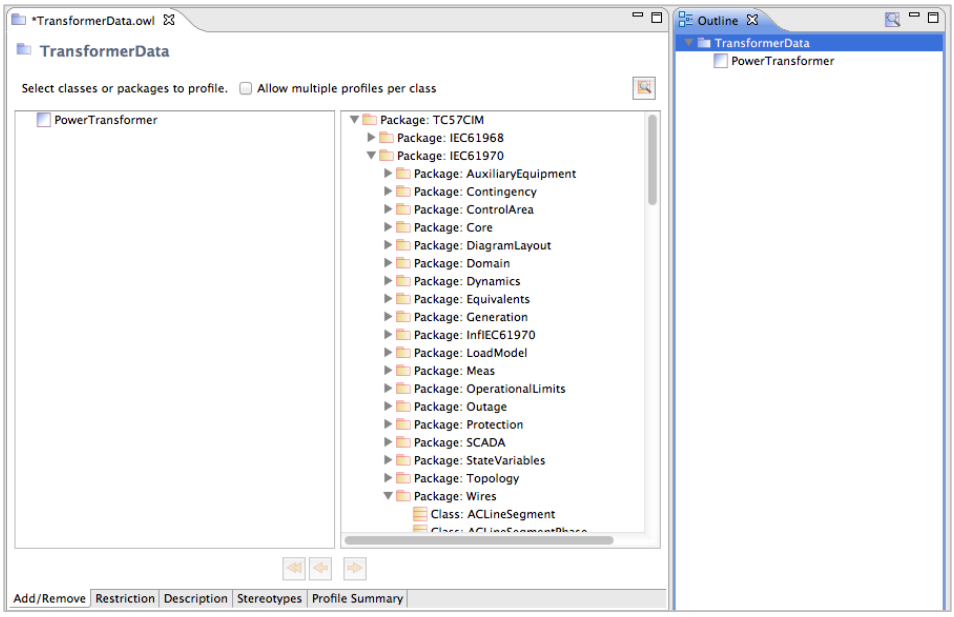


Figure 8-7
Profile with PowerTransformer class

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.

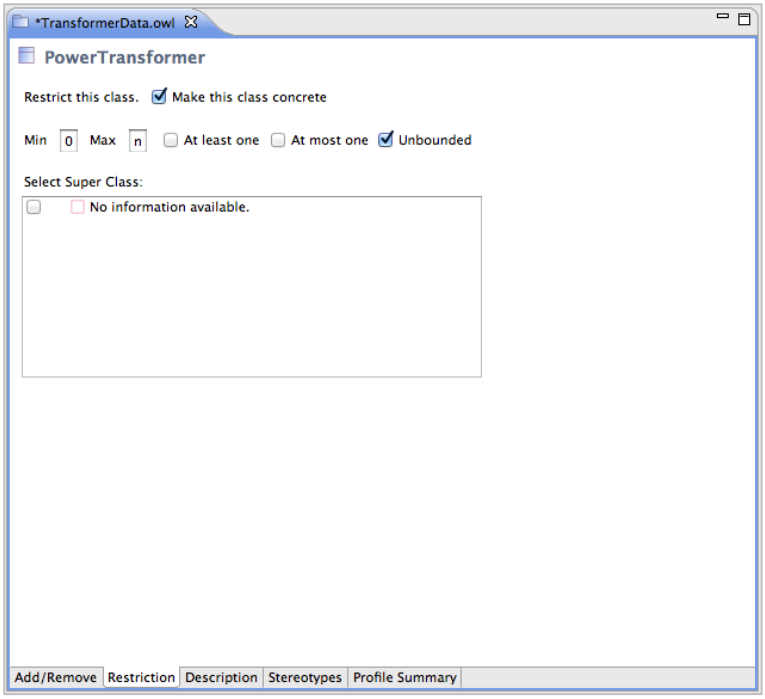


Figure 8-8
Marking the PowerTransformer class as concrete

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.

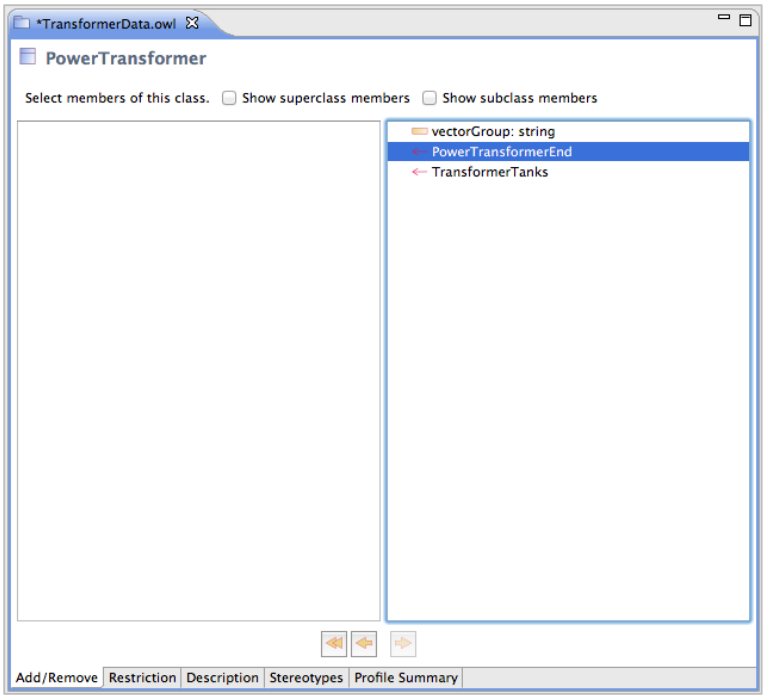


Figure 8-9
Add PowerTransformerEnd association for PowerTransformer class

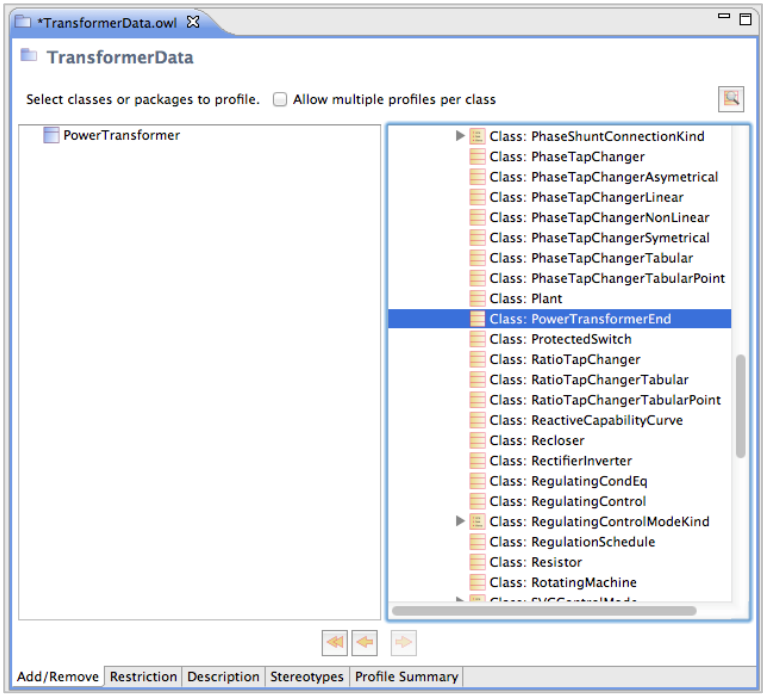


Figure 8-10
Add PowerTransformerEnd class to message

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.

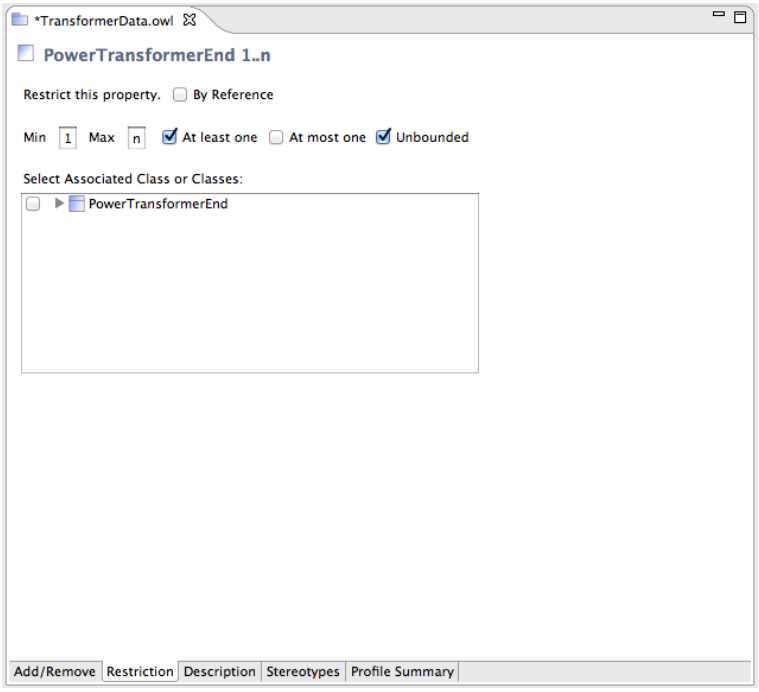


Figure 8-11
Select associated class for PowerTransformerEnd association

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.

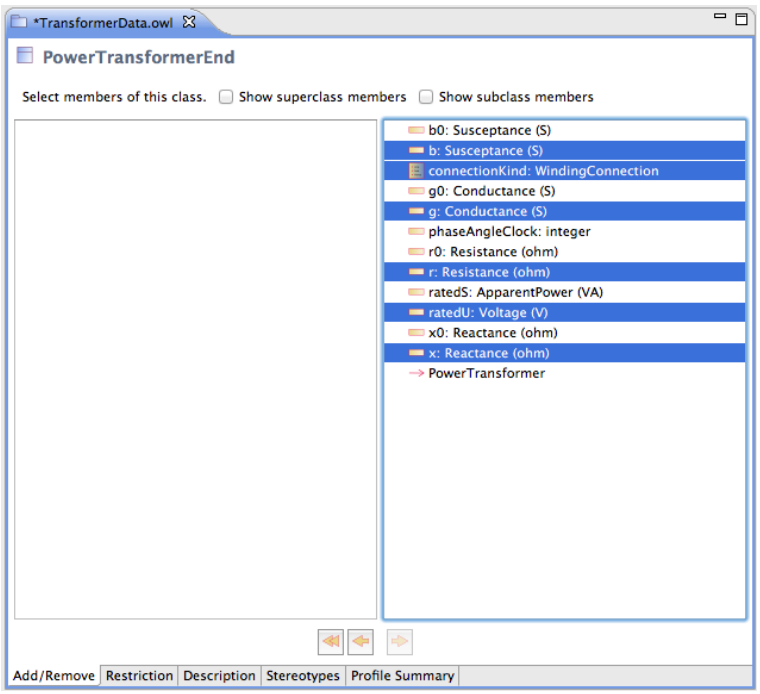


Figure 8-12
PowerTransformerEnd Native Attributes

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.

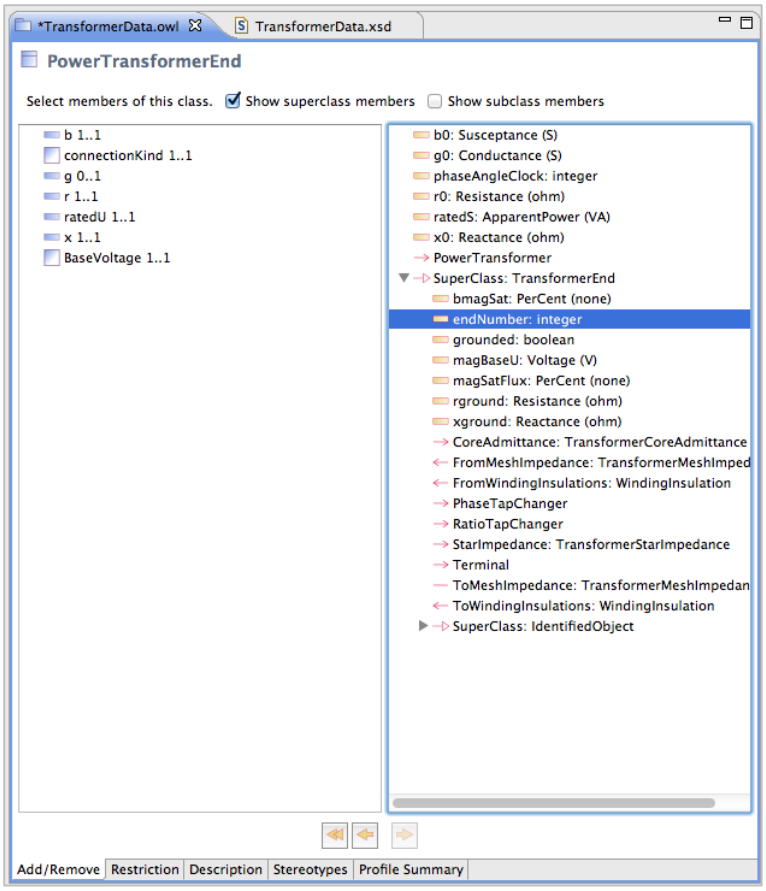


Figure 8-13
Add inherited attribute from TransformerEnd

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.

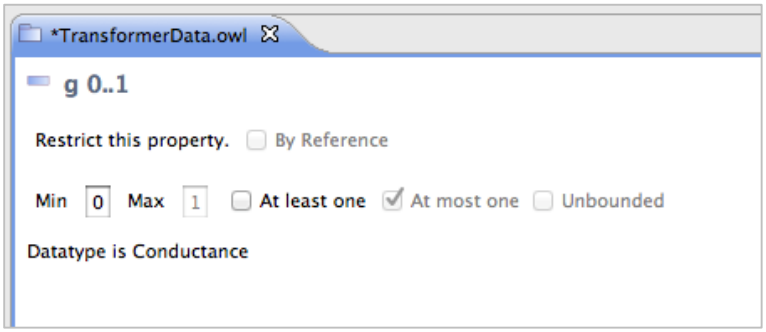


Figure 8-14
Optional PowerTransformerEnd attributes

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 etc.) to be specified. The resulting message outline is shown in Figure 8-15 below.

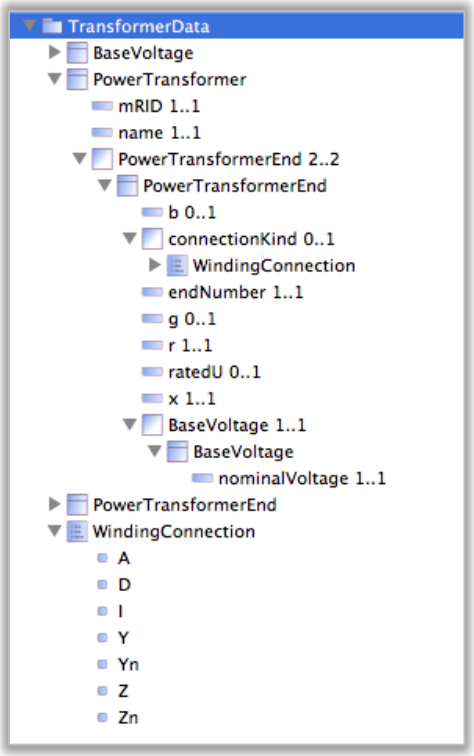


Figure 8-15
TransformerData Message Outline

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.

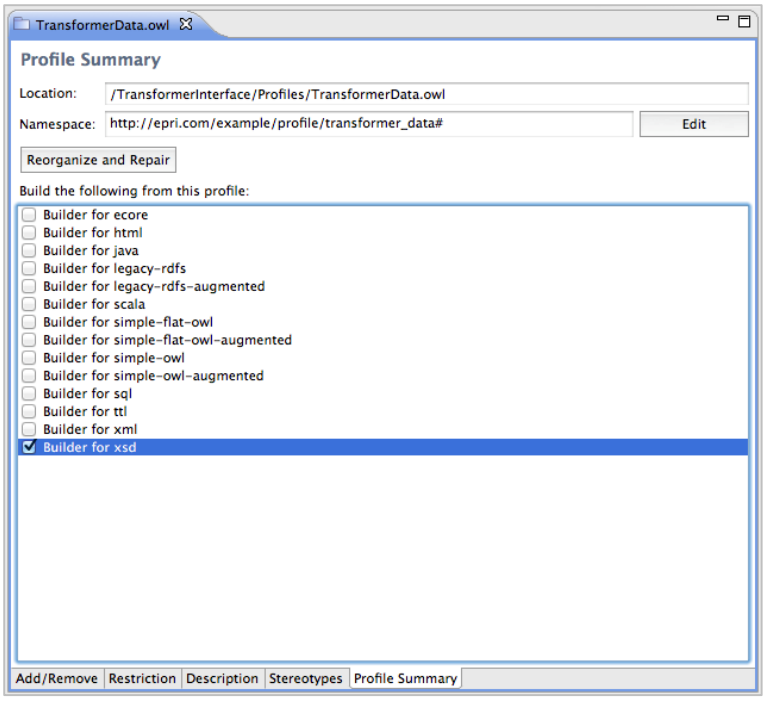


Figure 8-16
Enable XSD Builder in CIMTool

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:

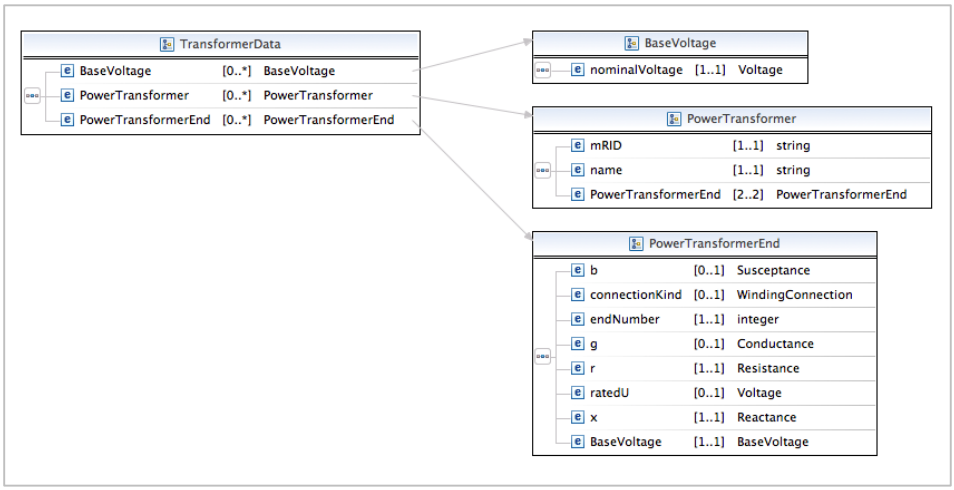


Figure 8-17
TransformerData XSD Schema Element

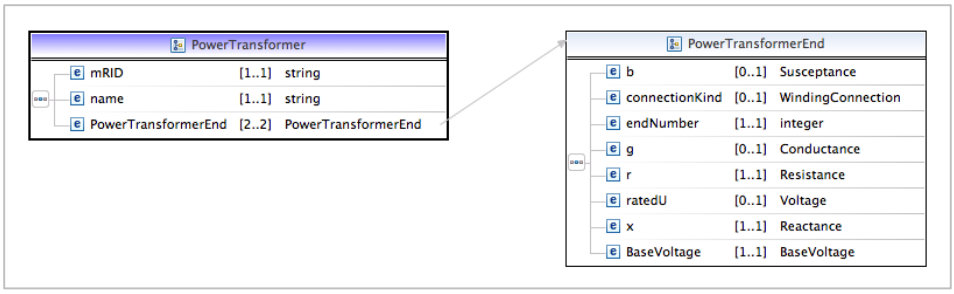


Figure 8-18
PowerTransformer XML Schema Element

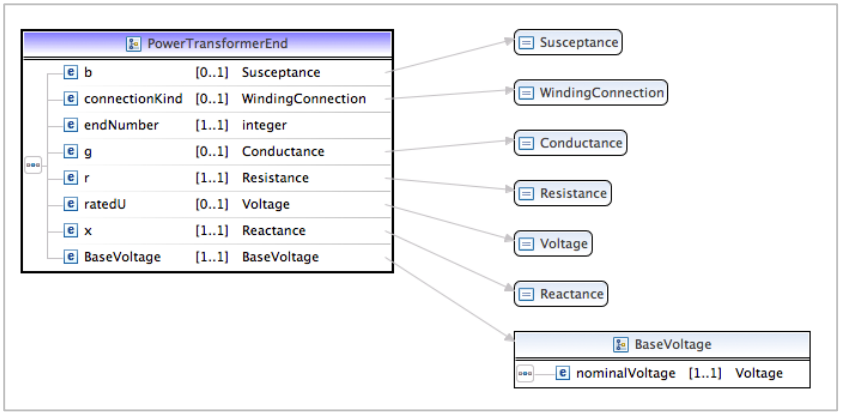


Figure 8-19
PowerTransformerEnd XML Schema Element

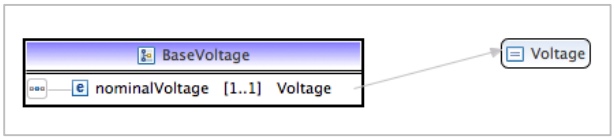


Figure 8-20
BaseVoltage XML Schema Element

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.

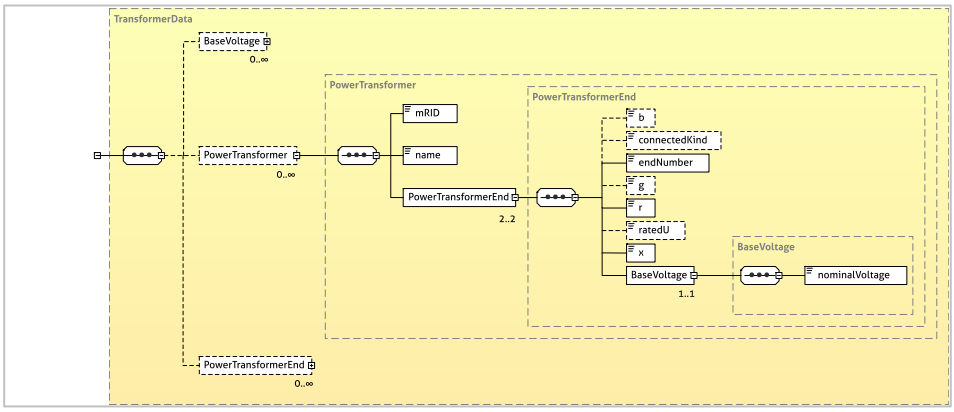


Figure 8-22
XMLSpy style XML Schema view

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 (e.g. 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 for 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 as a reference model is 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. An 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. the Naming and Design Rules

   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

Section 9: Transforming / Mapping 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

In Section 8 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 CIM.

In some cases the requirement is only to import or export in which case the transformation is one-way, but for many applications there is a need to import and export CIM and ensure that there is consistency between the importing and exporting (i.e. ensure that a native element 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 only using CIM. 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 etc. as shown in Figure 9-1.

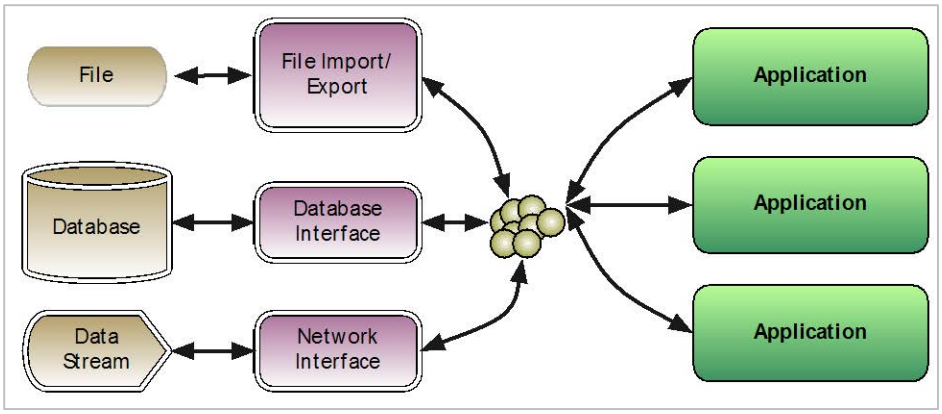


Figure 9-1
Multiple Serialization Formats/Locations

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)[47] is shown below:

[47] 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 http://www.omg.org/spec/QVT/1.0/

```
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;
}
```

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[48] format[49] 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:

- The PSS/E data structure must be derived to create a PSS/E meta-model
- A parser for PSS/E RAW files must be written to create the format-agnostic data objects
- A transform between the CIM classes and the PSS/E meta-model elements
- 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).

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 (e.g. 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 (e.g. 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 (e.g. 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

---

[48] 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.

[49] 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, pp229-235

read into the format-agnostic objects the index references are resolved and become direct, in-memory associations (i.e. 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 (e.g. 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 9-7oolean 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 9-9oolean 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 structures with a meta-model it makes is easier for users to 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 has 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 because business logic will not needed to be created to capture the state data to preserve information as it passes from system to system.

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 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 one directional
    c. the data has a defined structures 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 in one directional

Section 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*[50] 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

[50] 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.

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).

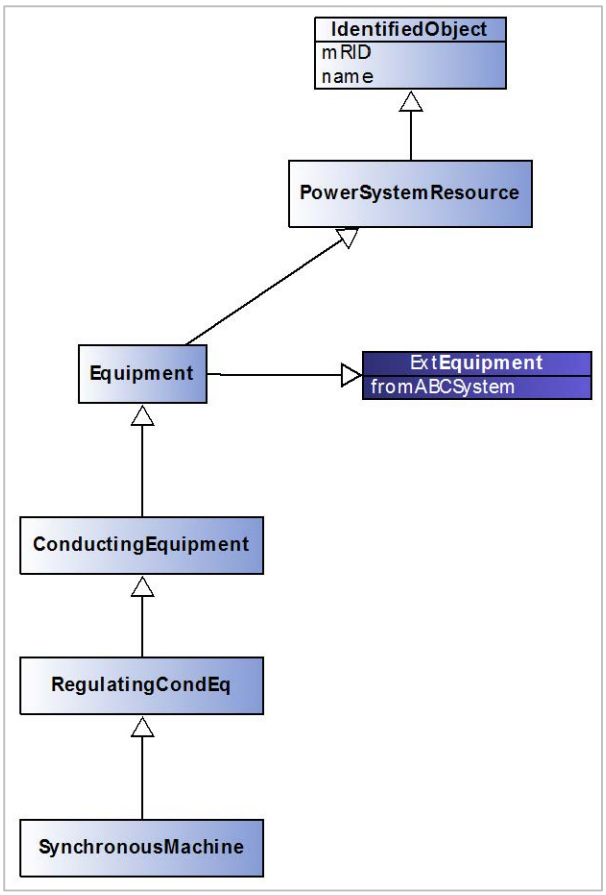


Figure 10-1
Adding an extension to an existing class

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.

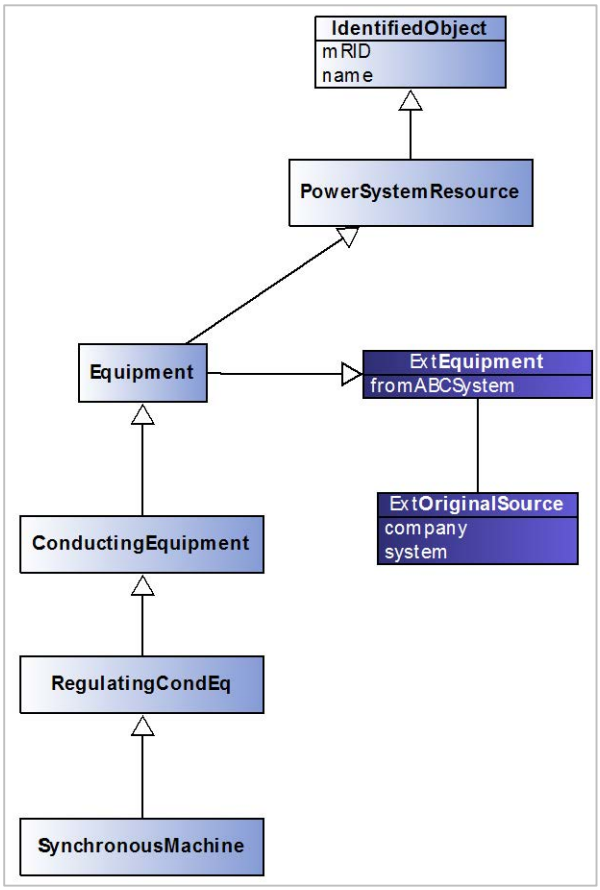


Figure 10-2
Adding a new class to the model

To accomplish this a new class, *ExtOriginalSource*[51] 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.

[51] 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.

For this example 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 will be used. 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 (e.g. at a gas-fired power station).

Adding New Classes

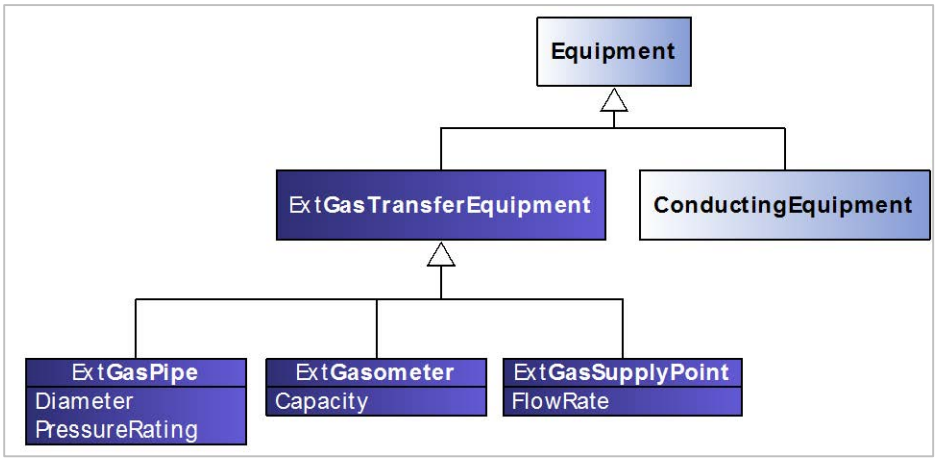


Figure 10-3
Gas Transfer Equipment classes

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 it capacity
- *ExtGasSupplyPoint* to define a point on the network where gas is supplied

Refining Existing Classes

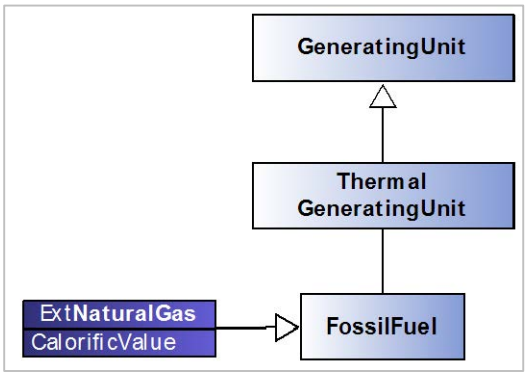


Figure 10-4
Refinement of CIM Fossil Fuel class for Natural Gas

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.

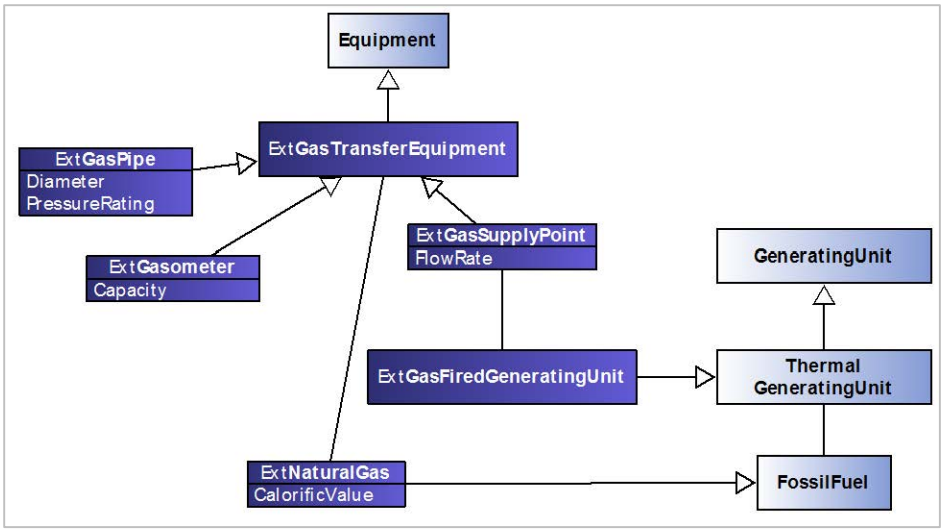


Figure 10-5
Connecting the Gas and Electrical models

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.

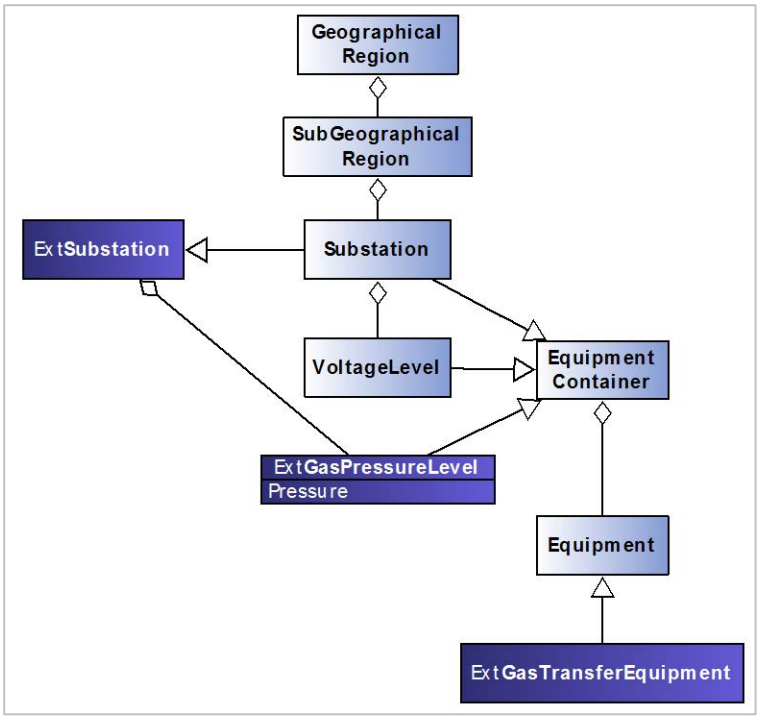


Figure 10-6
Gas Pressure Level Containment Classes

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.

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 – e.g. by exporting a *ExtGasFiredGeneratingUnit* as a *ThermalGeneratingUnit* and ignoring any attributes/associations introduced in the *ExtGasFiredGeneratingUnit* class).

Internal vs. Global Extensions

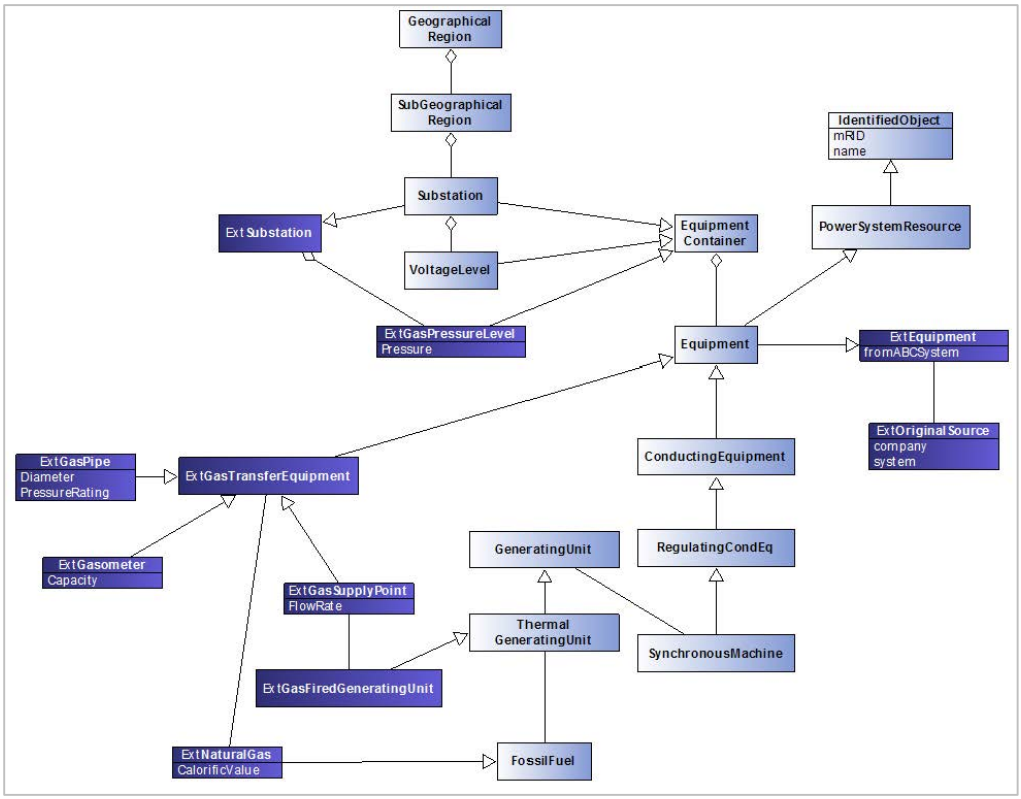


Figure 10-7
Extended CIM UML

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.

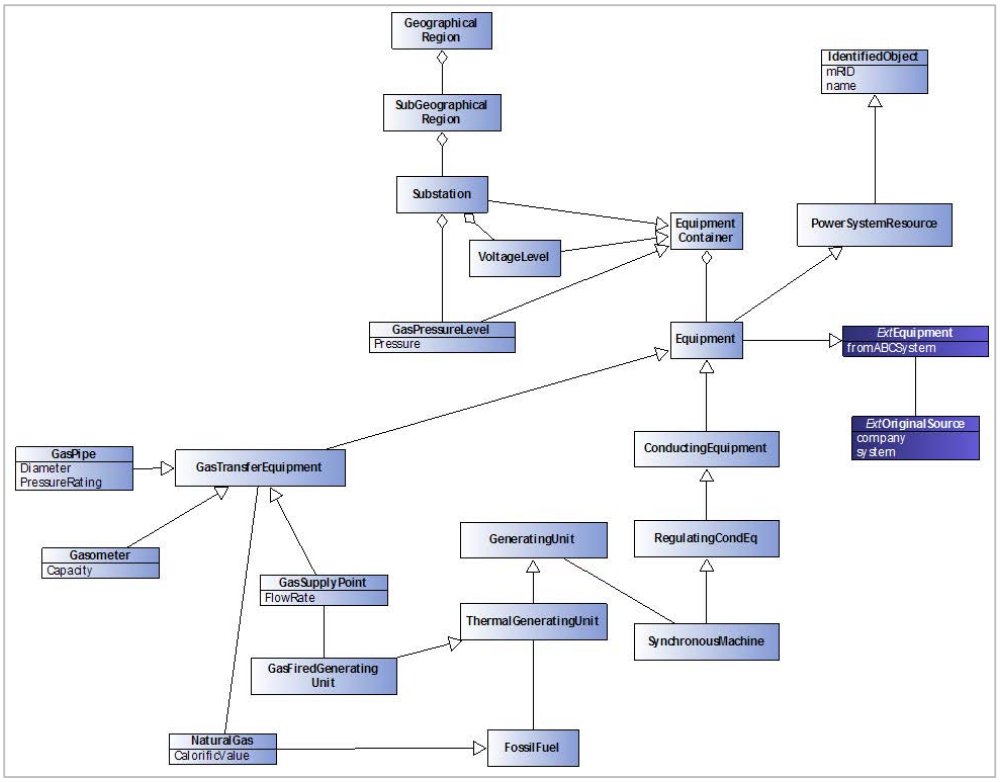


Figure 10-8
Future CIM with integrated extensions

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 in to the standard and so reducing the complexity, time and cost to the utility (and their vendors) in maintaining a custom data model.

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 are when CIM-like methods are applied to other domains, for example, water, or gas utilities (because the CIM is an electrical system model).

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, e.g. 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

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 others

Section 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 it was reference (albeit by design) but it didn't inform the user on how the CIM could be leveraged for systems integration. Even when profiles were created for some of the standards, e.g. IEC 61968-9 Meter Reading and Control it was difficult for practioners to go from understanding what was contained in a profile, and how to use it to integrate two systems. Much as the analogy that has been used with the CIM is that it is a reference, like a dictionary, it is not a writing *guide*. That is, the reader could look up the meaning of the words, but there was no guidance on how to write a sentence. For this reason the learning curve that was already associated with such a large, comprehensive model was aggravated by providing limited guidance on use for systems integration.

For these reasons the Working Group that supports CIM for distribution (IEC 61968) where application 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, feedback their findings to the SDO, in effect stating, "yes, and…"

To this end IEC 61968-100: Application integration at electric utilities – System interfaces for distribution management – Part 100: Implementation Profiles was created. This standard provides guidance on how to use IEC 61968 for utility application integration, specifically for Part 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, e.g. 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.

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.

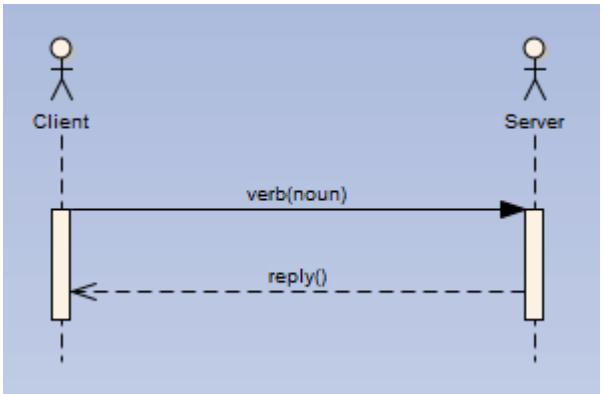


Figure 11-1
Basic request / response pattern

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.

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[52].

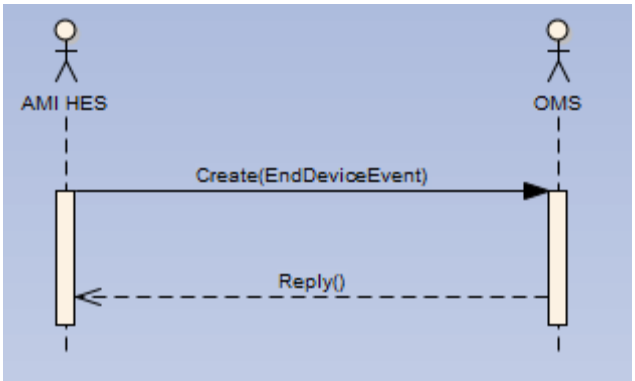


Figure 11-2
Basic request / response pattern using the Create verb and the EndDeviceEvent object.

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 vase 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.

[52] 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.

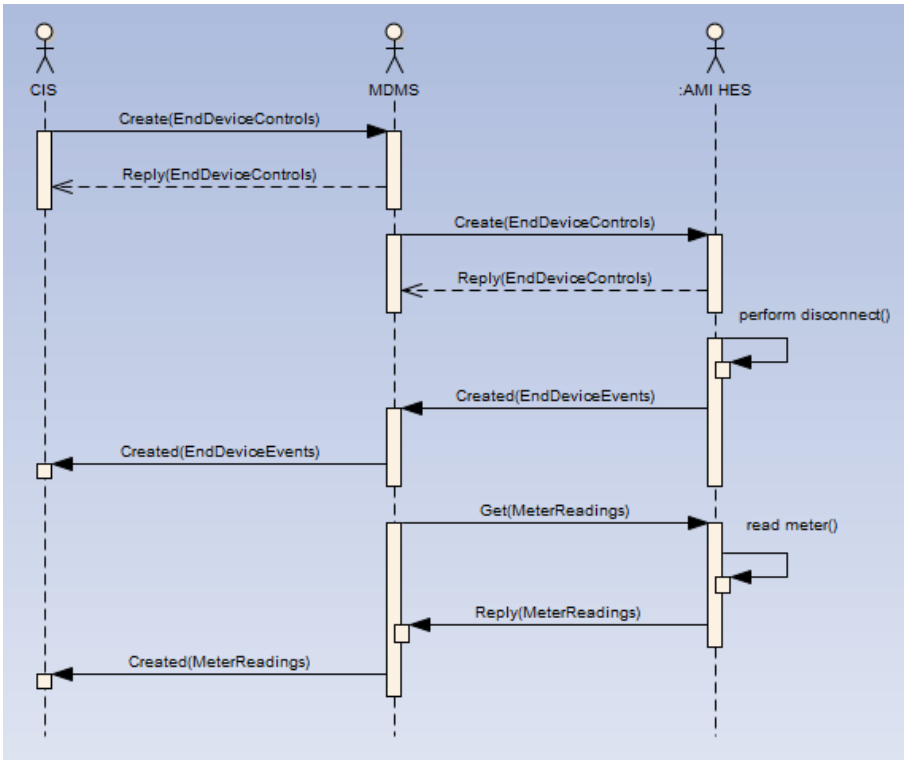


Figure 11-3
Example of message integration required for a remote discconect of a smart meter

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.

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 is 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. An 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

Section 12: Free and Open Source Tools

MODSARUS®: MODelling SmartGrid ARchitecture Unified Solution

MODSARUS has been developed as an add-in of Enterprise Architect (EA) software from Sparx Systems, already chosen by IEC TC57 for developing the Common Information Model (CIM)-related standards. MODSARUS is leveraging the standardized modeling approach as well as a Model Driven Architecture approach in an integrated UML platform in order to facilitate the development of Smart Grid data exchanges from any reference data models. The core idea of the approach is to produce an automatic implementation code from UML models designed by architects employing graphic user interfaces to guide humans and shadow modeling rules complexity.

CIMTool

CIMTool is a commonly encountered tool for creating profiles (a collection of classes that fully describe the information to be modeled or exchanged) from the larger enterprise semantic model, in this case the IEC CIM. CIMTool is open source, meaning a copy can be obtained at no charge from the CIMTool Download Page. It is implemented as a plug-in for the Eclipse platform. CIMTool is most commonly used for creating new profiles, editing those created in EA, and checking for inconsistencies. The tool can also validate messages against profiles.

CIM EA

CIM EA is a free, but not open source add-in to EA. CIM EA extends EA to provide an environment in which developers can manage the IEC CIM, CIM Profiles, and CIM-based artifacts.

http://www.cimea.org/

EPRI Use Case Repository Software

The EPRI Use Case Repository Software is an open source application to import use case information from the IEC use case template into and from Sparx Systems EA or other computer-aided software engineering (CASE) tools. This software serves as the heart of the “use case process.” Use cases define

applications so as to determine the specific requirements for communications infrastructure, new technologies, and information integration.

https://sourceforge.net/projects/ucrepoclientapp/

Appendix A: 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

Section 11

1. a
2. d
3. b
4. a
5. c

Appendix B: CIM Related Success Stories

Success Stories – The Green Button Standard

Case Study: 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

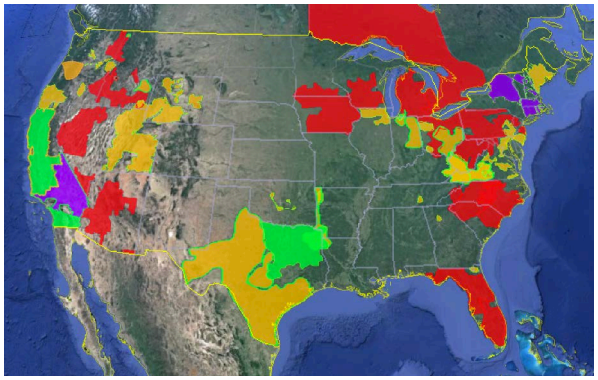
The early results have been phenomenal. There are now tens of millions of American consumers with access to this CIM-based data. There are dozens of utilities and third party services providers exchanging data based on this standard. All this was accomplished in less than two years from the start of 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 (i.e. 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

Case Study: Consumers Energy leverages IEC Common Information Model (CIM) for Enterprise Integration and a enterprise semantic 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.

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

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 is continuing to use this methodology and associated tools for integration work in 2013 and the foreseeable future.

Success Stories – Long Island Power Authority

Case Study: Long Island Power Authority (LIPA) leverages IEC Common Information Model (CIM) for Enterprise Information Management and Semantic Integration initiatives.

The Problem

LIPA has 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 LIPA data among multiple service providers and implement "Best of Breed" applications.

Some of LIPA'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, etc.
- Also allows for the development of new reports and functionality not previously available in any of the off-the-shelf applications

LIPA 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 LIPA 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.

LIPA 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 & 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, LIPA has implemented a Solution Development Life Cycle (SDLC) used on projects, enabling LIPA to control projects and ensure that the LIPA architectural goals are not subverted.

The Impact

The methodology has helped LIPA 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 LIPA 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)

Case Study: 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 (e.g., 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 – Sempra Energy

Case Study: 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

Case Study: 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.

Export Control Restrictions

Access to and use of EPRI Intellectual Property is granted with the specific understanding and requirement that responsibility for ensuring full compliance with all applicable U.S. and foreign export laws and regulations is being undertaken by you and your company. This includes an obligation to ensure that any individual receiving access hereunder who is not a U.S. citizen or permanent U.S. resident is permitted access under applicable U.S. and foreign export laws and regulations. In the event you are uncertain whether you or your company may lawfully obtain access to this EPRI Intellectual Property, you acknowledge that it is your obligation to consult with your company's legal counsel to determine whether this access is lawful. Although EPRI may make available on a case-by-case basis an informal assessment of the applicable U.S. export classification for specific EPRI Intellectual Property, you and your company acknowledge that this assessment is solely for informational purposes and not for reliance purposes. You and your company acknowledge that it is still the obligation of you and your company to make your own assessment of the applicable U.S. export classification and ensure compliance accordingly. You and your company understand and acknowledge your obligations to make a prompt report to EPRI and the appropriate authorities regarding any access to or use of EPRI Intellectual Property hereunder that may be in violation of applicable U.S. or foreign export laws or regulations.

The Electric Power Research Institute, Inc. (EPRI, www.epri.com) conducts research and development relating to the generation, delivery and use of electricity for the benefit of the public. An independent, nonprofit organization, EPRI brings together its scientists and engineers as well as experts from academia and industry to help address challenges in electricity, including reliability, efficiency, affordability, health, safety and the environment. EPRI also provides technology, policy and economic analyses to drive long-range research and development planning, and supports research in emerging technologies. EPRI's members represent approximately 90 percent of the electricity generated and delivered in the United States, and international participation extends to more than 30 countries. EPRI's principal offices and laboratories are located in Palo Alto, Calif.; Charlotte, N.C.; Knoxville, Tenn.; and Lenox, Mass.

Together...Shaping the Future of Electricity

Program:

Information and Communication Technology

3002006001

Electric Power Research Institute
3420 Hillview Avenue, Palo Alto, California 94304-1338 • PO Box 10412, Palo Alto, California 94303-0813 USA
800.313.3774 • 650.855.2121 • askepri@epri.com • www.epri.com