Using the Common Information Model for Power Systems as a Framework for Applications to Support Network Data Interchange for Operations and Planning

Alan W. McMorran

Submitted for the Degree

Of

Doctor of Philosophy

Institute for Energy and Environment

Department of Electronic and Electrical Engineering

University of Strathclyde

Glasgow, G1 1XW

Scotland, UK

June 2006

Acknowledgements

I would like to thank Professor Jim McDonald for taking a chance and giving me the opportunity to study within his research group and for his tenacious support in every aspect of my work. A special thank you to Dr Graham Ault, without whose guidance, encouragement, support, friendship and unwavering faith in me, I would have undoubtedly missed out on this life-changing opportunity.

I would also like to thank my colleagues in the Advanced Electrical Systems group, in particular Dr Ian Elders, whose advice and patience have contributed immeasurably to the success of my research; to Robert Currie for being a true friend and drinking companion over the last three years; and to Dr Andrew Willshire for taking the time to help me ensure that this thesis is of a suitable literary quality and for spotting all the spelling mitsakes. The research group has been more than just a workplace, and I cannot thank everyone enough for the invaluable experiences and eclectic, vibrant discussions I have been involved in.

This research work has been funded in part by National Grid, and I thank Jenny Cooper and Ciaran Morgan for their contributions to this work and for the financial and technical support National Grid has provided me over the last two years.

A special thanks to Jan for her endless support and encouragement, and for being there with Hobbes and Fraizer to pick me up whenever I was feeling down. Thanks to David and Lorna for being the best brother and sister anybody could ever ask for, and for ensuring that my feet stayed firmly fixed to the ground whenever I was at risk of getting overly carried away.

To my parents John and Catherine, for their love, inspiration, encouragement and support throughout my life I owe a debt beyond measure. If I achieve nothing else in my life, I only hope that I have made them proud.

Abstract

The Common Information Model (CIM) is an object-oriented representation of a power system used primarily as a data exchange format for power system operational control systems and as a common semantic model to facilitate enterprise application integration. The CIM has the potential to be used as much more than an intermediary exchange language and this thesis explores the use of the CIM as the core of a power systems toolkit for storing, processing, extracting and exchanging data directly as CIM objects.

This thesis looks at the evolving nature of the CIM standard and proposes a number of extensions to support the use of the CIM in the UK power industry while maintaining, where possible, backwards compatibility with the IEC standard. The challenges in storing and processing large power system network models as native objects without sacrificing reliability and robustness are discussed and solutions proposed.

A number of applications of this CIM software framework are described in this thesis aimed at facilitating the use of the CIM for exchanging data for network planning and operations. The development of novel algorithms is described that use the underlying CIM class structure to convert power system network data in a CIM format to the native, proprietary format of an external analysis application. The problem of validating CIM data against pre-defined profiles and the deficiencies of existing validation techniques is discussed. A novel validation system based on the CIM software framework is proposed that provides a means of performing a level of validation beyond any existing tools. Algorithms to allow the integration of independent power system network models in a CIM format are proposed that allow the automatic identification and removal of overlapping areas and integration of neighbouring networks.

The development of an application to dynamically generate network diagrams of power system network models in CIM format via the novel application of existing, generic data visualisation tools is described. The use of web application technologies to create a remotely-accessible tool for creating power system network models in CIM format is described.

Each of these applications supports a stage of the planning process allowing both planning and operational engineers to create, exchange and use data in the CIM format by providing tools with a native CIM architecture that can adapt to the evolving CIM standard.

Table of Contents

Table of Figures

Abbreviations

AJAX	Asynchronous JavaScript and XMLHttpRequest
API	Application Programming Interface
BSD	Berkeley Software Distribution
CAD	Computer Aided Design
CCAPI	Control Centre Application Programming Interface
CIM	Common Information Model
CPSM	Common Power System Modelling
DMS	Distribution Management System
DNO	Distribution Network Operator
DTD	Document Type Definition
EAI	Enterprise Application Integration
EMS	Energy Management System
EMTP	ElectroMagnetic Transients Program
EPRI	Electric Power Research Institute
ERP	Enterprise Resource Planning
GIS	Geographic Information System
GMR	Geometric Mean Radius
HTML	Hyper Test Markup Language
HTTP	Hyper Text Transfer Protocol
IEC	International Electrotechnical Commission
IT	Information Technology
JDBC	Java Database Connectivity
JDO	Java Data Objects

kV	Kilovolt
MFLOPS	Million Floating Point Operations
MVAr	Megavolt Ampere Reactive
MW	Megawatt
NERC	North American Reliability Council
OAG	Open Applications Group
ODMG	Object Database Management Group
OWL	Web Ontology Language
PNG	Portable Network Graphics
PSS/E	Power System Simulator for Engineering
RDF	Resource Description Framework
RDFS	RDF Schema
RMI	Remote Method Invocation
SCADA	Supervisory Control and Data Acquisition
SGML	Standard Generalized Markup Language
SQL	Structured Query Language
SVG	Scalable Vector Graphics
TNO	Transmission Network Operator
UML	Unified Modelling Language
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
XML	eXtensible Markup Language
XSLT	eXtensible Stylesheet Language Transform

1 Introduction

1.1 Motivation and Context for Research

Since deregulation, both in the UK and internationally, there has been an increasing need for power companies to exchange data on a regular basis. This is to ensure the reliable operation of the interconnected power networks owned and operated by a number of different utilities. Power companies use a variety of different formats to store their data, whether it be 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 power electrical 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 [2] extends this model to cover the other aspects of power system software data exchange such as asset tracking, work scheduling and customer billing. These two standards, 61970-301 and 61968 are collectively known as the Common Information Model (CIM) for power systems and currently have two primary uses: to facilitate the exchange of power system network data between companies; and to allow the exchange of data between applications within a company.

The development of the CIM has primarily taken place in North America, where the North American Electric Reliability Council (NERC) has 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.

This has resulted in aspects of the CIM's design being focussed on the needs of American utilities, which do not always correspond directly with those of the UK transmission and distribution companies. While network operators in the UK are also required to exchange information, unlike their North American counterparts the regulator in the UK has not specified a standard format the utilities must use. The Grid Code[3] defines the data that must be exchanged between system operators but does not state the format or medium that must be used.

The first part of this thesis discusses possible changes and extensions to the CIM to allow the representation of data important to UK network operators. This is to address some of the perceived deficiencies in the CIM that prevents it from accurately modelling the UK network at a level that would allow it to be adopted as the common format for exchanging network data between UK network operators.

Both the applications described previously use the CIM as an intermediary data exchange format, whether it is transmitting small segments of data between applications or entire power system network models between companies. The focus of the research outlined in the second part of thesis is the use of the CIM beyond this current data exchange application and utilising the CIM architecture for creating, editing, exchanging, validating and visualising power system network data for both planning and operational applications.

This entails the creation of extensions to the CIM standard to support the use of power system networks in a CIM format for planning, and the development of a software framework to allow this data to be natively created, stored, edited and processed.

The framework requires:

- Novel methods of storing the data that allows instant access, supports fast conversion to other formats, and concurrent access from multiple sources.
- A schematic drawing application that allows the user to create new power system networks natively in the CIM format using a familiar interface. The underlying format must be full, non-abstracted CIM but provide the user with an interface that is equivalent to that of a standard power system network design package, while concealing the complexity of the underlying data if it is not required.
- The development of algorithms to utilise the structure of the CIM to create methods for converting CIM data into other formats for export to external

simulation and analysis software that are linearly scalable (i.e. the execution time is proportional to the size of the network)

- The design and implementation of an extensible engine for validating CIM data against any number of pre-defined profiles.
- A way for a user to dynamically create a schematic of the power system network from CIM data that contains no graphical information

Such tools will provide the utilities with the ability to utilise the CIM immediately for operational purposes, thus reducing the impact of such a major transition, and allow planning engineers to create, exchange, validate, integrate and visualise full power system network models. Existing commercial tools utilise the CIM as an exchange format, while the open source tools available are concerned with checking the validity at a very basic level, or generating schema for use in application integration applications.

There are no tools currently on the market specifically designed to allow the user to deal with power system networks in CIM format natively for planning purposes. This presents a number of challenges for developing novel methods for storing, processing, validating, editing and converting the data that scales linearly and is capable of easily coping with any future changes to the CIM. It is these challenges that are addressed by the research recorded in this thesis.

1.2 Principal Research Contributions

There are a number of key contributions that have been made by the work described in this thesis.

Several extensions to the CIM have been proposed both to support the representation of equipment within the UK electrical network at a level of detail beyond that currently available in the CIM, and to facilitate the use of power system network models in CIM format for planning applications. The approach taken in making these extensions has been to minimise the changes to the existing CIM standard. This facilitates backwards compatibility, which simplifies the integration of any extensions with existing software.

The use of an underlying CIM architecture for the software framework has provided a powerful foundation for providing both storage and processing of data in a native CIM format. This use of the CIM as the basis of a software framework is itself a novel application of a standard that until now has been used exclusively for

exchanging static data. Applications built on this framework have shown that data in CIM format can be used for more than just data exchange and power system models in CIM format can be natively created, edited, processed and exported.

It has been demonstrated that the CIM format, in conjunction with an extension to define points in the network that can be used for an external connection, allows power system models in CIM format to be used for planning applications. Using the developed software framework and its remote multi-user access architecture, users can remotely upload new or modified network sections and automatically identify possible connection points within a larger base network model; integrate these models into a single, coherent model; then export this data into a format usable by an existing analysis application.

These previous novel developments have required the development of new algorithms that utilise the CIM representation of a power system network to allow the conversion and processing of the native CIM data. These algorithms are essential to all the CIM based applications described in this thesis, most notably the tool to export power system data in a format compatible with PSS/E, a commercially available power system simulator, and the automatic integration of power system models in CIM format.

These algorithms, combined with the CIM based software framework has allowed the rapid creation of tools for a number of applications:

- A CIM power system network design web application. This design tool is a novel application of the popular AJAX technique for creating interactive web applications. The web-based nature of the tool provides a means of natively creating and sharing power system network models in a CIM format with embedded schematic information.

- A modified version of an open source generic data visualisation tool that, when used with network models in CIM format, allow the user to dynamically generate network topology diagrams at differing levels of abstraction.

- A novel method of validating CIM data against pre-defined profiles by defining logical rules and validating each object individually. This is a completely different approach to other members of the CIM community who have chosen to implement an Extensible Markup Language (XML) schema

based validation system that has proven itself less flexible and unable to express all the required constraints.

1.3 Industrial Applications of Research Contributions

The use of the CIM for network data exchange by all utilities that fall under the jurisdiction of NERC has required all the major power system software vendors to create import and export modules that are capable of dealing with CIM data. Regular Interoperability Tests are organised to ensure compatibility between the products from each vendor. This involves each party creating a network model file from their own applications and then exchanging it with every other user who in turn imports the file into their software.

Until recently the only validation that could be performed on these files was at a very basic level based on early XML schemas, and as such there was a requirement for a validation tool that could express every requirement stated. To aid the participants the validation tool described in this thesis was made available online and was used by the participants during and in the weeks preceding the last Interoperability Test. The flexibility of the CIM software framework allowed the validation engine profile and error reporting mechanism to be designed, written and deployed in under two weeks. Since being made available as an online web application in January 2006, the tool has been used by engineers from a number of companies and institutions including: ABB, Areva, EDF, ELIA, ESB National Grid Ireland, General Electric, KEMA, LS Industrial Systems, National Grid, Siemens, SISCO, SNC Lavalin, Scottish Power, Subnet Solutions, Western Area Power Administration and Xtensible Solutions.

A similar validation tool produced by one of the largest power system software vendors was used alongside the application described herein at the last CIM Interoperability Test. This tool was unable to perform the same level of validation as the application described in this thesis. This resulted in engineers from one department of the company having to use the validation tool described in this thesis to validate their test model since the tool developed by another department was unable to perform the validation to the same level of accuracy.

Allowing public access to this one outcome of this research has brought international recognition to the project and University, including invitations to present at CIM User Group meetings and to join the IEC Working Group responsible for the creation of the IEC 61968 standard, as well as a number of

enquiries from commercial companies interested in the technology and its novel application of the CIM standard.

1.4 Thesis Outline

The remainder of this thesis has been divided into eight principal chapters:

Chapter Two provides some background on the existing techniques and technologies used within the research work. This includes a basic description of the modelling language the CIM is expressed in, along with a description of the CIM itself and the different methods used to encapsulate the data

Chapter Three covers extensions to the CIM, including those proposed by IEC working groups and academics, and the extensions proposed to allow the CIM to successfully cover the major requirements of UK utilities.

Chapter Four describes the framework used to construct native CIM applications while Chapter Five provides details on the application to convert CIM data to a proprietary format for power system simulation. Chapter Six describes how the validation engine was designed and implemented while Chapter Seven details how the CIM structure can be utilised to allow the automatic integration of power system network models in CIM format. Chapter Eight discusses how diagrams of network models in CIM format can be automatically generated and presents a method of graphically creating new power system network models in CIM format.

Finally, Chapter Nine summarises the principal conclusions of the research work, highlighting the main achievements and proposing further research and development work to build on the existing outcomes.

1.5 Associated Publications

The following publications have arisen from the work described in this thesis:

1.5.1 Journal Publications

A.W. McMorran, G.W. Ault, C. Morgan, I.M. Elders, J.R. McDonald, "A Common Information Model (CIM) Toolkit Framework Implemented in Java", IEEE Transactions on Power System, February 2006, Volume 21, Number 1, pp.194-201

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 System, February 2004, Volume 19, Number 1, pp.229-235

1.5.2 Conference Publications

A.W. McMorran, G.W. Ault, C. Morgan, I.M. Elders, J.R. McDonald, "A Common Information Model (CIM) Toolkit Framework Implemented in Java", IEEE Power Engineering Society General Meeting, 18-22 June 2006, Accepted for presentation

A.W. McMorran, "The Common Information Model as a Software Framework", CIM User Group Meeting, Carmel Indiana, 1-4 November 2005

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 Power Engineering Society General Meeting, Denver Colorado, 6-10 June 2004 p.116 Vol.1

A. Dysko, A. W. McMorran, G. M. Burt, G. Ault, J. R. McDonald, "Web Services Based Distributed Dynamic Protection System Simulation And Testing", Developments in Power System Protection Conference, Amsterdam, The Netherlands, April 2004

A.W. McMorran, G.W. Ault, G.M. Burt, J.R. McDonald, "Web Services Platform For Power System Development Planning", UPEC 2003: Proceedings of the 38th International University Power Engineering Conference, Thessaloniki, Greece, September 2003

2 Background

2.1 Chapter Introduction

This chapter describes the pitfalls of the traditional methods of storing power system data, then introduces the concepts behind class modelling and how this approach is used to define a power system in the IEC 61970 Common Information Model (CIM) standard. The use of the Extensible Markup Language (XML) to encapsulate this data for the exchange of both full power system models and inter-application messages is then described.

2.2 Power System Data Formats

Since the advent of the modern digital computer, power system engineers have utilised the capabilities of this tool 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.

Large-scale Energy Management Systems (EMS) and asset-management systems 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 and fault-level analysis simulations use application-specific file formats that represent the data required by each application.

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 own 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.

The deregulation of the power industry, however, has resulted in multiple utilities, running software from a number of different vendors, 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 traditionally 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. Of course, such a scenario is not limited to power system applications. Changing the file format for each new software version is common practice within the software industry but usually only causes 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 commerical 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, all of the major power system application vendors are active participants in the CIM Interoperability tests and the popularity of the format is spreading.

This chapter will provide some background on the CIM and the CIM RDF XML format. To understand the structure of the CIM, however, it is important to have an understanding of class hierarchies within the object-oriented software paradigm and the benefits of using such an approach to model the components of a power system. The following sections will provide some general background on class hierarchies, followed by more detailed background information on the CIM. Finally, it will be shown how this data can be represented in the RDF XML format.

2.3 Class Hierarchies and UML Class Diagrams

When building any system to represent data, whether it be a software architecture or a database schema, the design of the system will define how extensible and scalable the system is, and ultimately, whether it succeeds or fails at its given task. This chapter provides an introduction to the concept of Class Hierarchies and how they are used in system design, along with the Unified Modelling Language (UML)[17].

Within a system, a class represent a specific type of object being modelled. A class hierarchy is an abstract model of a system defining every type of component within a system as a separate class. A class hierarchy should reflect the real-world structure of the system.

While a full description of UML is outwith the scope of this thesis, UML class diagrams provide a useful means of visually representing object hierarchies. This section will provide a simple case study to show how a class hierarchy representing a small segment of a University system can be constructed independently of the final platform on which the design will be utilised.

3.1 Classes

h class can have its own internal attributes and relationships with other classes. h class can be instantiated into any number of separate instances, known as ects, each containing the same number and type of attributes and relationships, with their own internal values.

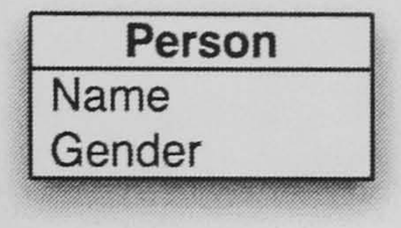


Figure 2.1 The Person Class

simple example of a class is that of a Person as shown in Figure 2.1. The Person ss contains two basic attributes: *Name* and *Gender*. If the system being created re to represent every person in the University, it would require only this single ss since every person within the University can be represented at the most basic el by the attributes defined in Person.

a University containing 10,000 students and staff, the system would create 000 separate instances of the Person class, each containing a value for Name and nder independent of the other 9,999 instances (although not necessarily unique).

he system is required to store more information than just a person's Name and nder, and differentiate between staff, students and the different types of each, n the class diagram becomes more complex.

3.2 Inheritance (Generalisation)

heritance (also known as Generalisation) defines a class as being a sub-class of other class. As a sub-class, it inherits all the attributes of its parent, but can also ntain its own attributes.

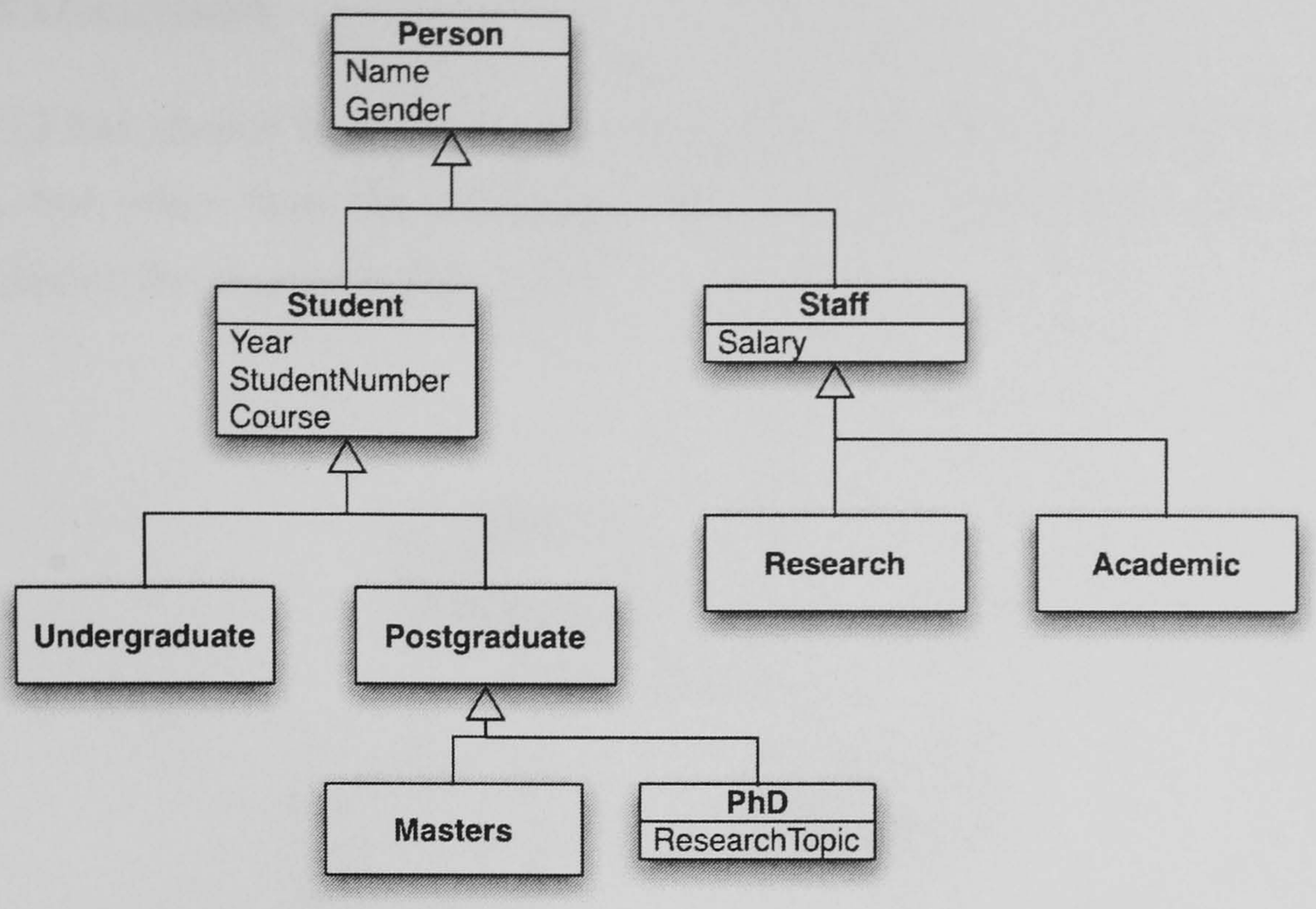


Figure 2.2 Class Hierarchy of people at a University

Figure 2.2 provides a class hierarchy to represent some of the different types of people that exist within a University system. This diagram, as with all subsequent class diagrams uses standard UML symbology. Student and Staff are both sub-classes of Person. A Student is still a person and still has a Name and Gender, but has additional attributes to denote the year they are in, their student number and the course they are studying. Similarly, if someone is Staff they are still a Person, but have gained a new attribute to indicate their salary.

The Student class itself has two sub-classes, Undergraduate and Postgraduate, both inheriting all the attributes of Student (and in turn, of Person), but independent of each other. The Postgraduate class also has two subclasses, Masters and PhD. A PhD is a Postgraduate and a Student and a Person, with all of their attributes, but with the addition of its own ResearchTopic attribute.

The Staff class also has two sub-classes, Research and Academic, both of which retain all the attributes of Staff and Person.

So while a PhD is a Postgraduate, a Student and a Person, not all Students are PhDs. Similarly, an Academic is Staff and a Person, but not every member of Staff is an Academic.

2.3.3 Association

Section 2.3.2 has shown how a class hierarchy can be formed to describe sub-classes of Person, but other than the inheritance there are no other relationships defined between classes the classes in Figure 2.2.

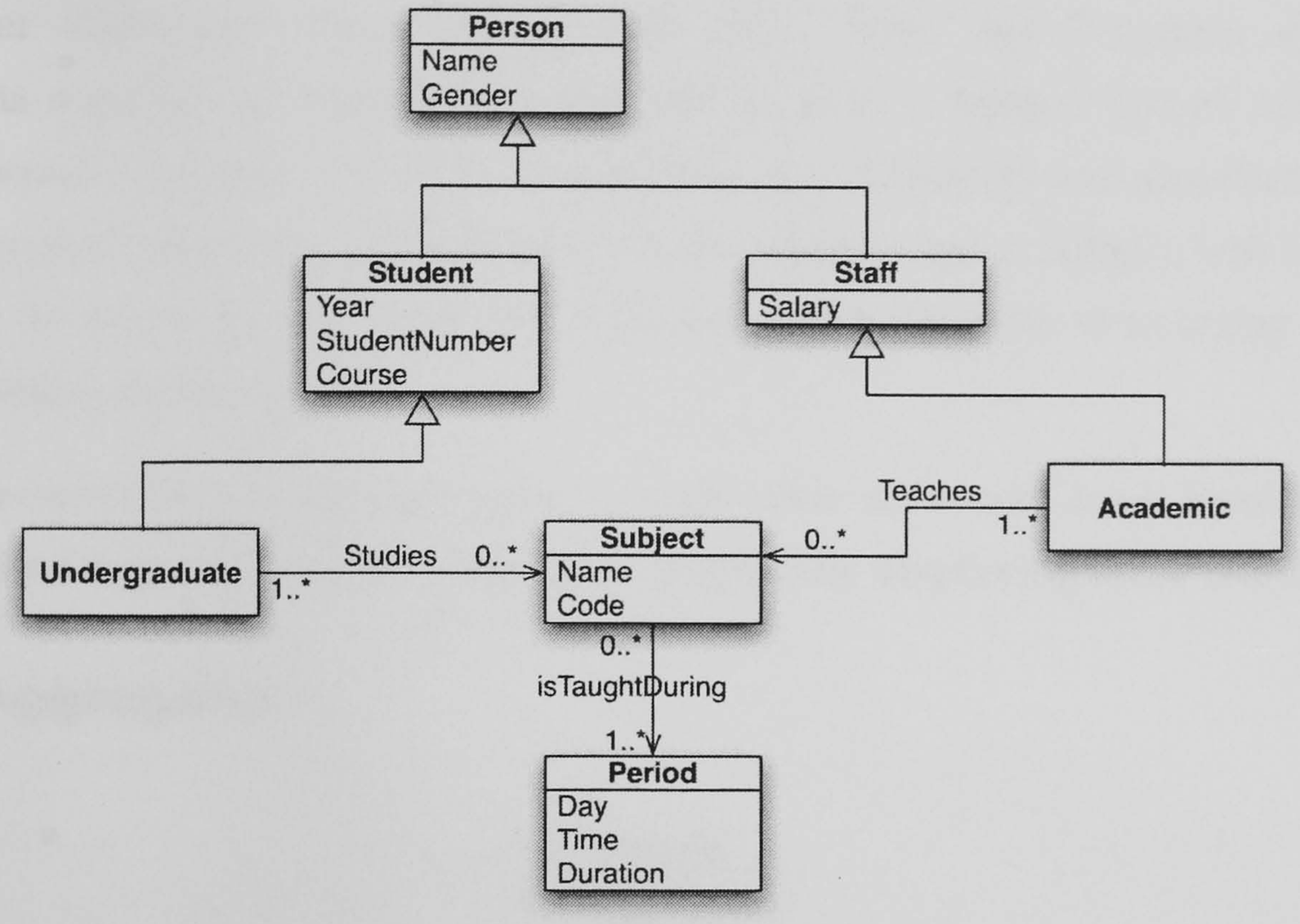


Figure 2.3 Class hierarchy of students, staff and subjects

In Figure 2.3 we have introduced two additional classes: Subject and Period. Neither of these classes are a type of Person, and as such do not inherit from the Person class. The Subject class does however, have a relationship with the Undergraduate and Academic classes.

An Undergraduate can study a number of subjects and an Academic can teach a number of subjects. These relationships are shown on the diagram as associations between the classes.

For the Undergraduate-Subject association, the role is given as "Studies" while the location of the arrowhead indicates that it is the Undergraduate who *Studies* the Subject (if the arrow were reversed, it would mean that the Subject studied the Undergraduate).

At each end of the association link is the multiplicity. For the Undergraduate-Subject association, these indicate that a Subject must have from 1 to many (1..*) Undergraduates, but an Undergraduate can have from 0 to many (0..*) subjects

(since it is possible that some Undergraduates may not be studying any subjects due to industrial placements, sabbaticals etc. It is assumed that a subject will not exist in the system if no students have chosen to study it).

Similarly, a Subject will have from 1 to many Academics who teach that Subject, but an Academic may teach from 0 to many Subjects (since not all Academics have to teach).

The other additional class, Period, with Day, Time and Duration attributes, represents a particular timetable period and as such a Subject has an association with a Period. As with the other associations, the Subject-Period association has a role, *isTaughtDuring* and multiplicities which indicate that a Subject will be taught during 1 to many periods and that a Period will have from 0 to many Subjects taught during its time-period.

This demonstrates how classes relate to each other on a very basic level and how UML Class Diagrams provide a means of graphically displaying these relationships.

2.3.4 Aggregation

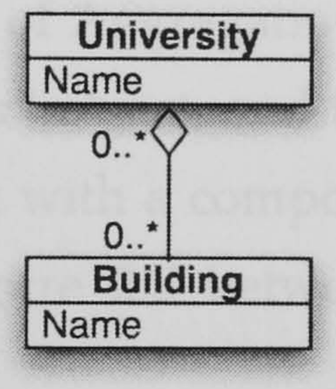


Figure 2.4 Class Hierarchy of a University and Building

The Aggregation relationship defines a special kind of association between classes, indicating that one is a container class for the other. In the example shown in Figure 2.4, two new classes University and Building have been introduced, each very simple classes containing only a single attribute to denote their name. The multiplicity on the diagram operates in the same manner as to that of the Association, indicating that a Building can be part of 0 or more Universities (we are assuming that some Universities operate joint schools and not every building within the system will necessarily be part of a University). The second multiplicity indicates that a University can contain 0 or more buildings (0 if it operates solely by remote learning for example).

Unlike a simple Association relationship the line denoting a relationship on the diagram contains a diamond instead of an arrowhead. This indicates that the two

classes have an Aggregation relationship. This can be thought of as "The University is made up of 0 or more Buildings", indicating that the relationship is stronger than a simple association. The clear diamond, however, indicates that the two are not completely inter-dependent, and that if the University were destroyed the buildings would still exist (assuming the destruction was not a literal demolition but instead indicated that the University had ceased to exist).

2.3.5 Composition

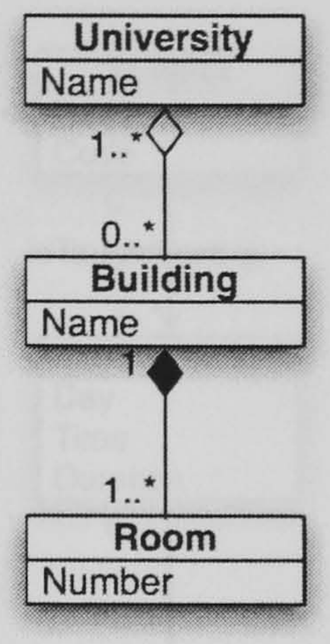


Figure 2.5 Class Hierarchy of a University, Building and Room

Composition is a specialised form of Aggregation where the "contained" object is a fundamental part of the "container" object, and that if the "container" is destroyed, all the objects that are related to it with a composition are similarly destroyed. An example of this is shown in Figure 2.5 between the new Room class and the Building class.

The line here has a solid diamond, indicating that the relationship is a composition. The multiplicity states that a building will have 1 or more rooms (since even an empty building can be thought of as one giant room) and that a room will be contained within 1 building only. This reflects the real world makeup of rooms and buildings. If a building is destroyed then the rooms within it are also destroyed.

Any system that implements this design will know that if a Building object is destroyed, any Room objects that are contained within that particular instance will also be destroyed.

2.3.6 Summary

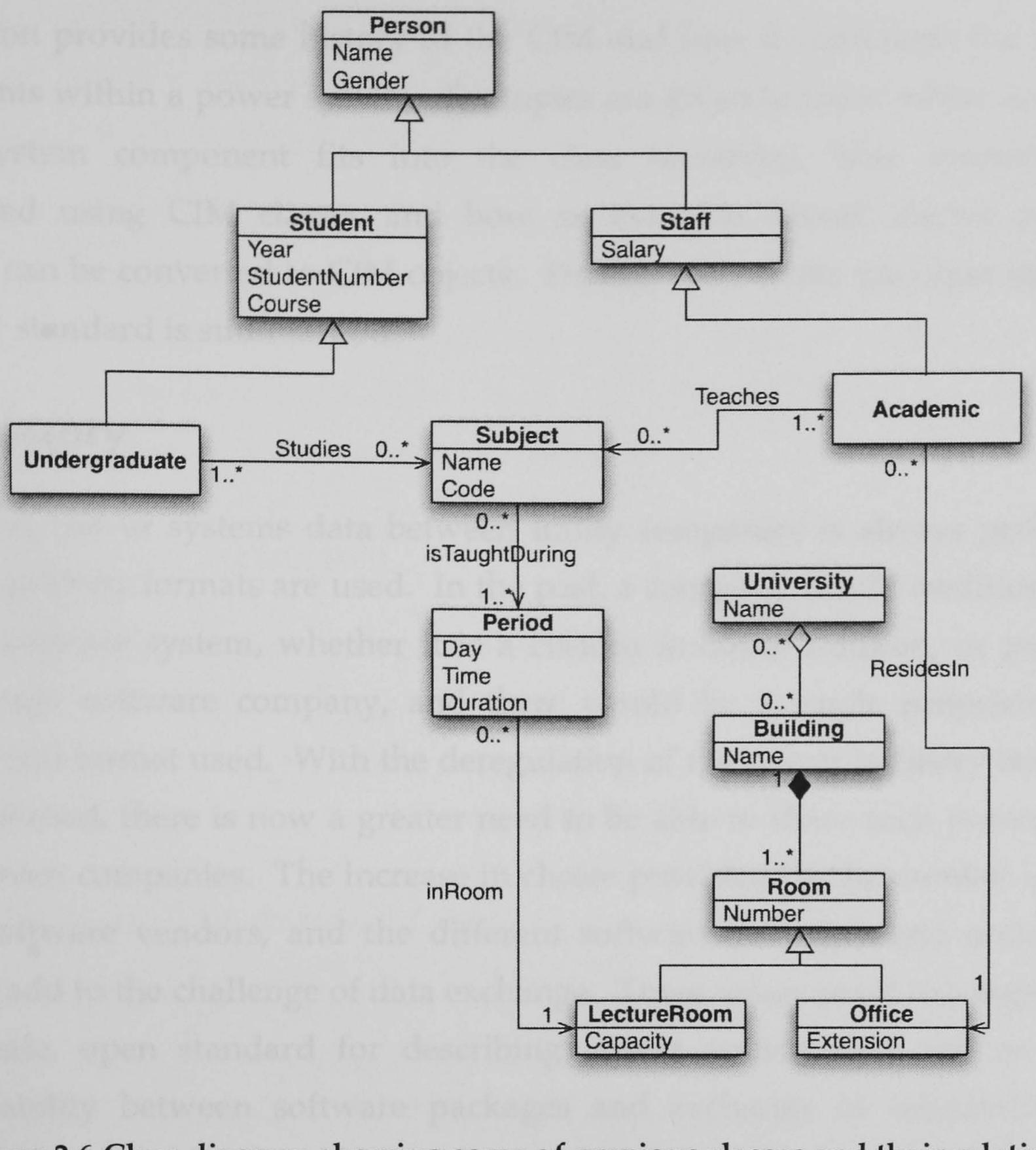


Figure 2.6 Class diagram showing some of previous classes and their relationships

The previous sections should have provided you with a basic understanding of what a class hierarchy is and how this can be represented on a class diagram. Figure 2.6 shows some of the classes from the previous sections (along with two extra sub-classes of Room), and how the separate diagrams in Figure 2.3, Figure 2.4 and Figure 2.5 all relate to each other. It should be clear that the system could be extended further to incorporate more details about the University system such as timetables for students and staff or the computing facilities available in each room. Both of these examples would make use of the existing classes by association along with the introduction of new classes.

These fundamentals of the class system are essential in the understanding of the CIM as described in the following sections.

2.4 The Common Information Model for Power Systems

This section provides some history of the CIM and how it represents the common components within a power system. Examples are given to show where a common power system component fits into the class hierarchy, how connectivity is represented using CIM classes and how an example circuit, shown as a line diagram, can be converted to CIM objects. Finally, each of the packages in the IEC 61970-301 standard is summarised.

2.4.1 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 both in the UK and abroad, there is now a greater need to be able to share such power system data between companies. 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)[1][2] is an open standard for representing power system components developed by the Electric Power Research Institute (EPRI) in North America. The standard was developed as part of the IEC TC57 WG13 on developing a Control Centre Application Programming Interface (CCAPI) to provide a common model for describing the components in power systems for use in a common Energy Management System (EMS) Application Programming Interface (API). 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.

The data model itself is language-independent, 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.

For an engineer the format of the Common Information Model (CIM) may at first appear confusing compared with a flat file format. This chapter will explain how the CIM was created using a class structure to describe components of a power system network; the advantages of this approach; and how a power system network model can be translated into a number of CIM objects.

2.4.2 CIM Class Structure

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 Naming class[1] so for this section it can be considered the base class for the hierarchy.

2.4.2.1 Example: The Breaker Class

A simple example will 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 as a "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"[1]. To understand how this fits into the CIM class hierarchy the Breaker can be thought of at different levels of abstraction.

---

[1] There is ongoing discussion within the CIM User Group on the use of the Naming class itself and whether a single super-class should replace it. The Naming class breaks a strict class hierarchy since no object can be thought of as being a "Naming" at its highest level. Suggested solutions include the removal of the Naming class itself and its replacement with a CIMObject class, or retaining the Naming class but have it as an associated class to a CIMObject class. This approach breaks the current hierarchy since no other class will inherit from Naming, but instead, any CIM class can have one or more Naming objects associated with each instance. These discussions are continuing and at the current time no changes have been finalised for the next version of the CIM.

At the most detailed level it is a Breaker, but since a breaker's most basic functionality is the ability to be open or closed it can be described as a specialised 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 type 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 and a type of Switch. This corresponds to a class inheritance structure shown in Figure 2.7 below.

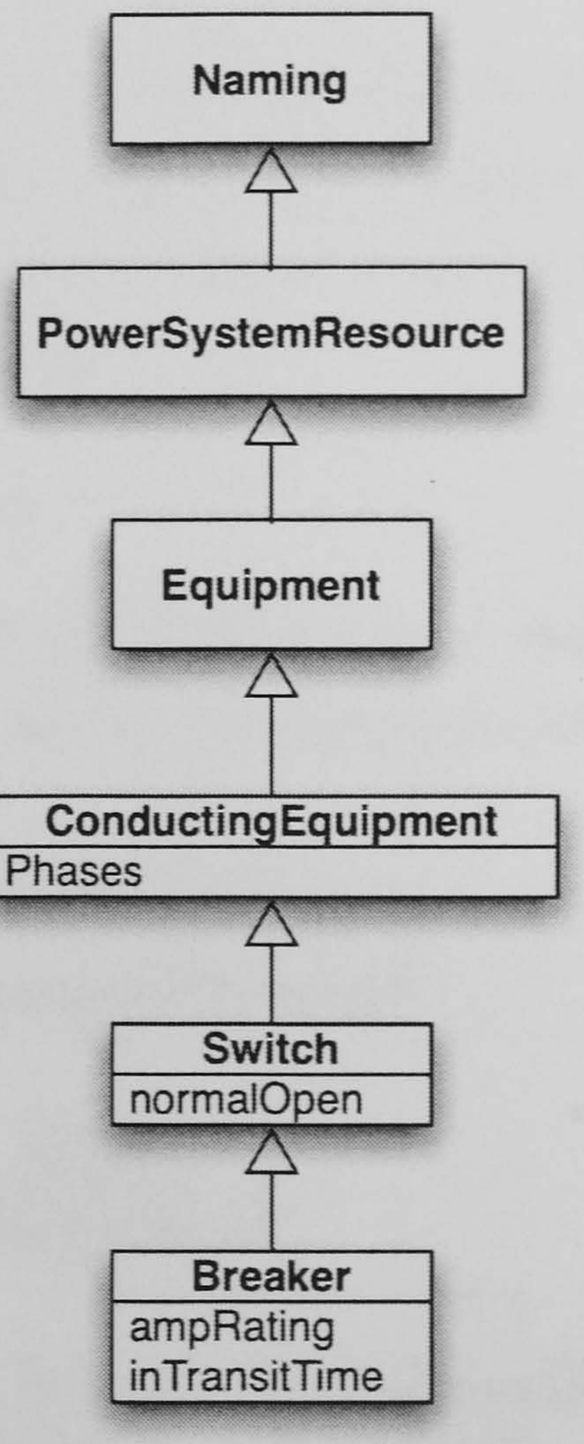


Figure 2.7 Breaker Class Inheritance Hierarchy

The Naming 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 organisational entity such as a SubControlArea.

- 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 and contains an attribute to denote the phases (A,B,C,N or any combination of each).
- 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.
  - Breaker, a specific sub type of Switch, with additional attributes to define the current rating and transit time.

As with the University system example in Section 2.3, all subclasses inherit the attributes from their parent class, and as such a Breaker will contain a normalOpen, from the Switch class, and phases attribute, from the ConductingEquipment class, as well as its own native attributes.

2.4.2.2 Subclasses of Switch

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

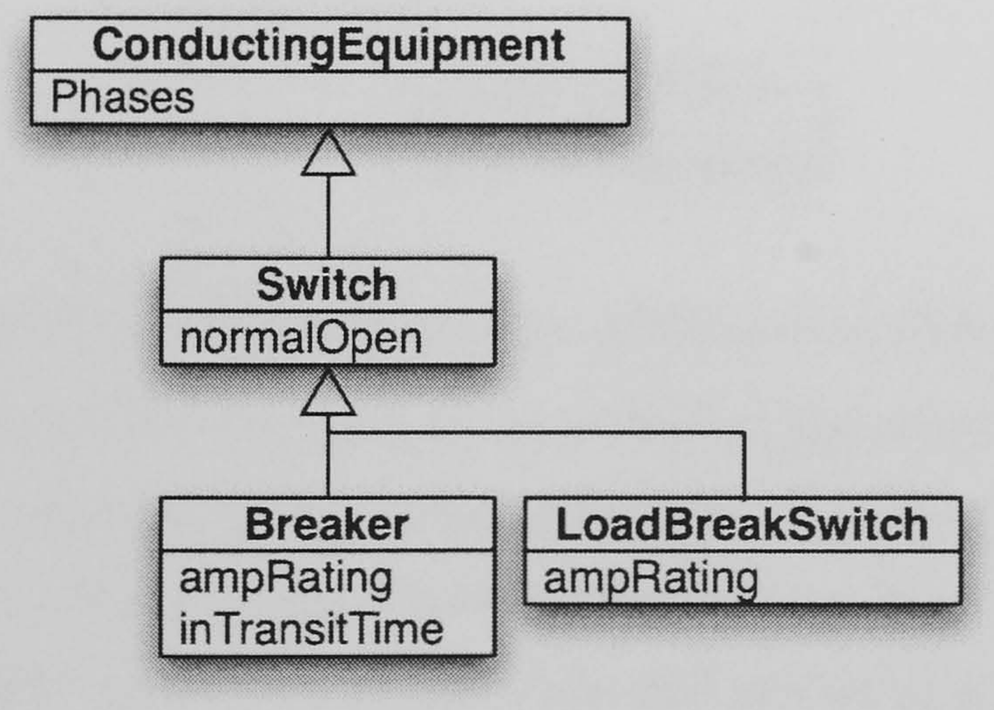


Figure 2.8 Switch class with Breaker and LoadBreakSwitch subclasses

Figure 2.8 shows an example of how the LoadBreakSwitch class, a subclass of Switch fits into the class hierarchy. Both Breaker and LoadBreakSwitch inherit from Switch, so they both contain a normalOpen 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 Switch, a piece of Conducting Equipment, a piece of Equipment, a Power System Resource or just a Naming entry.

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 open or closed to determine the status of the network. 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 proceeds accordingly.

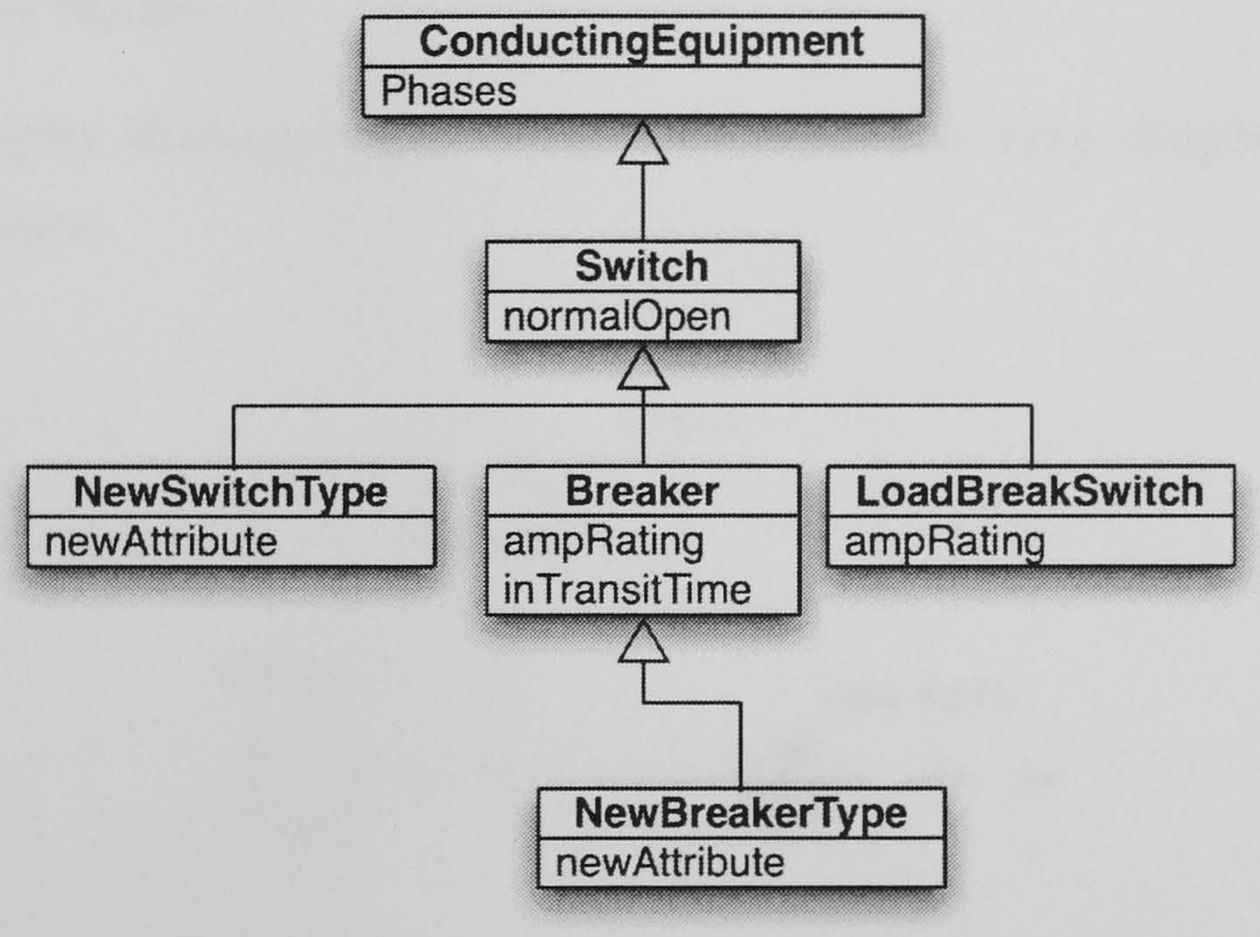


Figure 2.9 Switch Class diagram with new subclasses of Switch and Breaker

If a new type of Switch, NewSwitchType is added to the standard at a later date as shown in Figure 2.9, assuming the original Switch class is not modified, then the software will still be able to treat NewSwitchType as if it were a Switch 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 2.9), it is still a type of Switch (since its parent class, Breaker is a subclass of Switch) and can be treated as Switch or a Breaker by the software.

As has been shown, this use of an inheritance hierarchy to define components allows classes within the system to be defined as specialised subclasses of a general parent class until the desired level of detail has been reached, from the generic PowerSystemResource right down to the Breaker or LoadBreakSwitch class.

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

2.4.2.3 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 2.10 below.

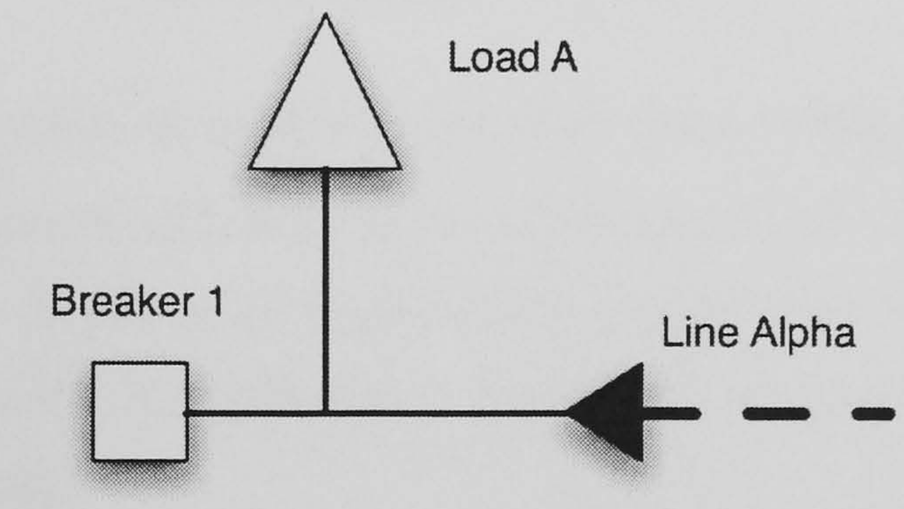


Figure 2.10 Connectivity Example circuit

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 or DC 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 contain associations to Load A and Line Alpha; Load A contain associations to Line Alpha and Breaker 1; and Line Alpha contain associations to Breaker 1 and Load A would result in the interconnections being defined as shown in Figure 2.11.

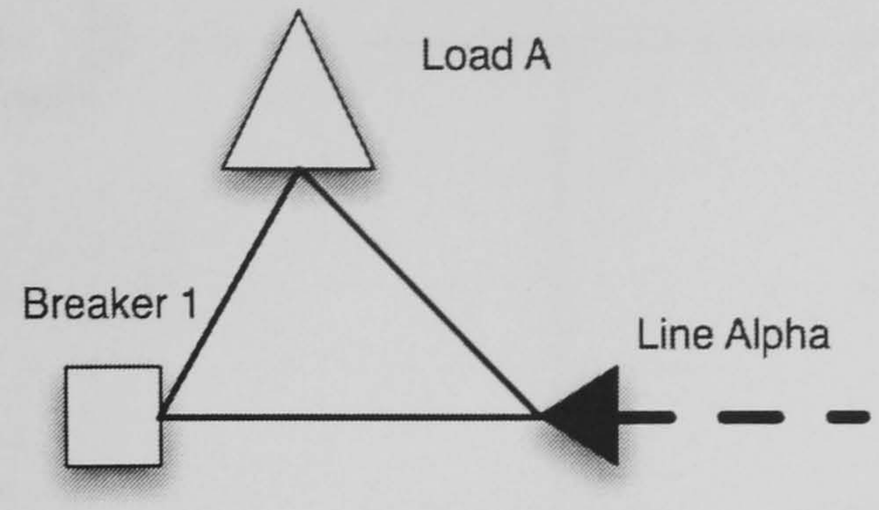


Figure 2.11 Connectivity Example circuit with direct associations

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 2.12.

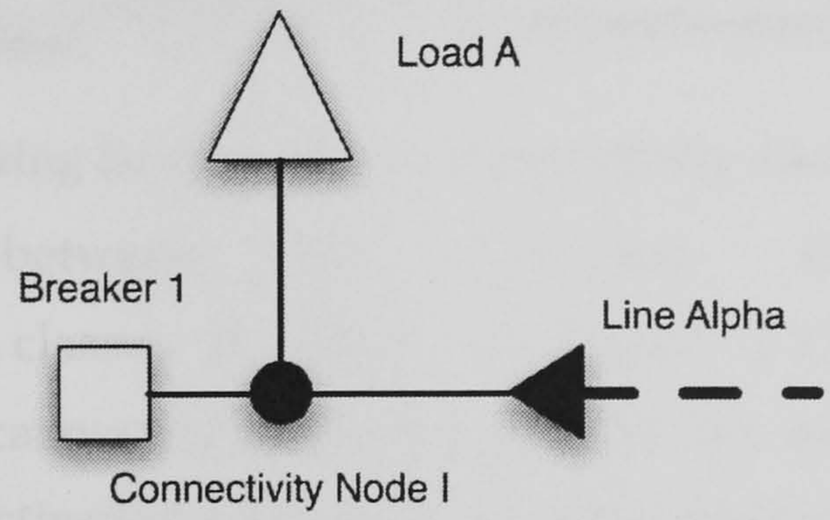


Figure 2.12 Connectivity Example circuit with Connectivity Node

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.

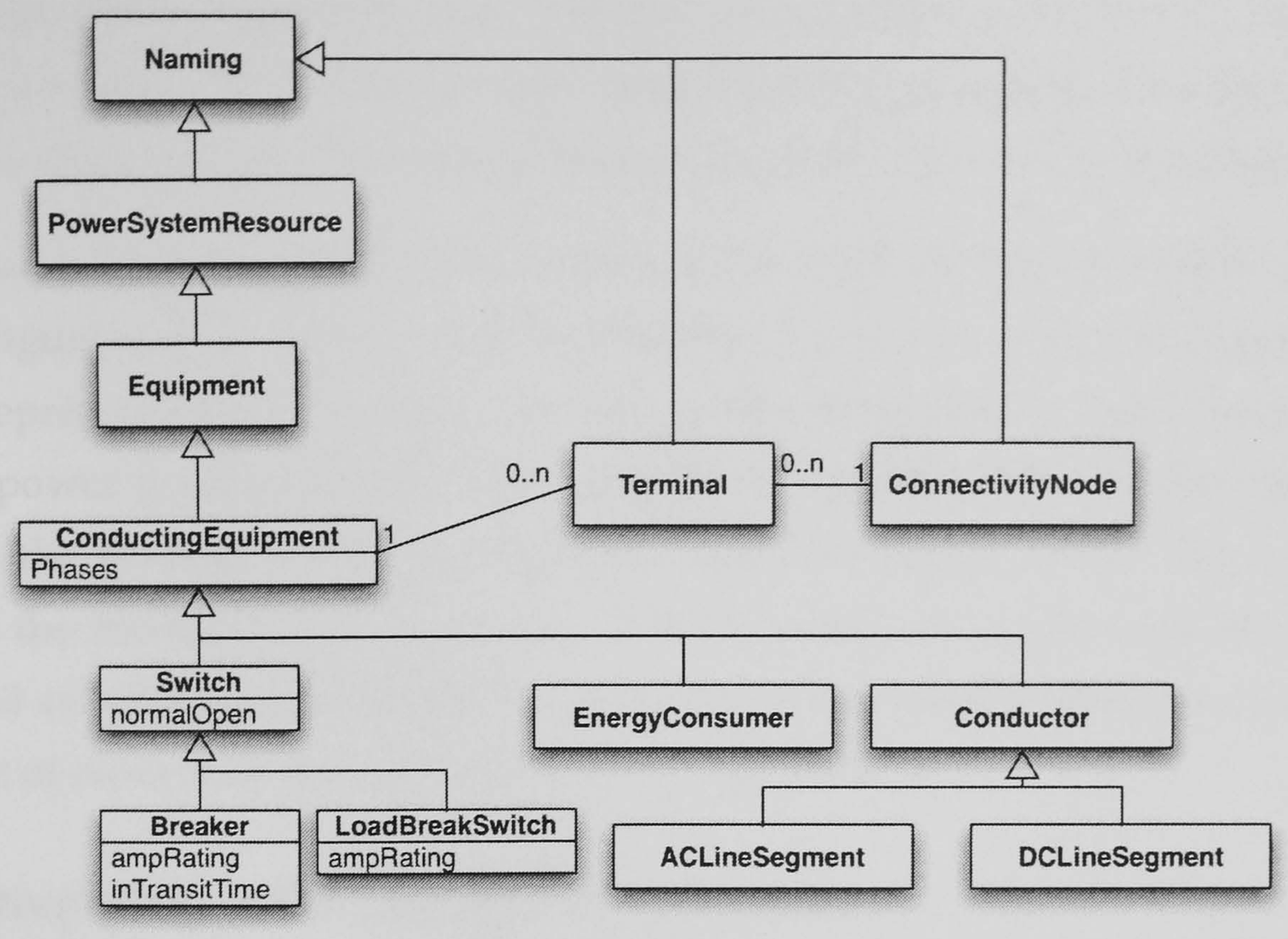


Figure 2.13 Conducting Equipment and Connectivity class diagram

The relationship between the Terminal, ConnecitivtyNode and ConductingEquipment classes is shown in Figure 2.13. Since only pieces of conducting equipment carry current on the network, the association to the Terminal class is from the ConductingEquipment class with a multiplicity of 0..n since a piece of conducting equipment can have zero or more connections to the network. The corresponding Terminal to Conducting Equipment relationship has a multiplicity of 1 since a Terminal can only ever be associated with one Connectivity Node. 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 connectivity relationship between the terminals, conducting equipment and connectivity nodes is illustrated in Figure 2.14a) below.

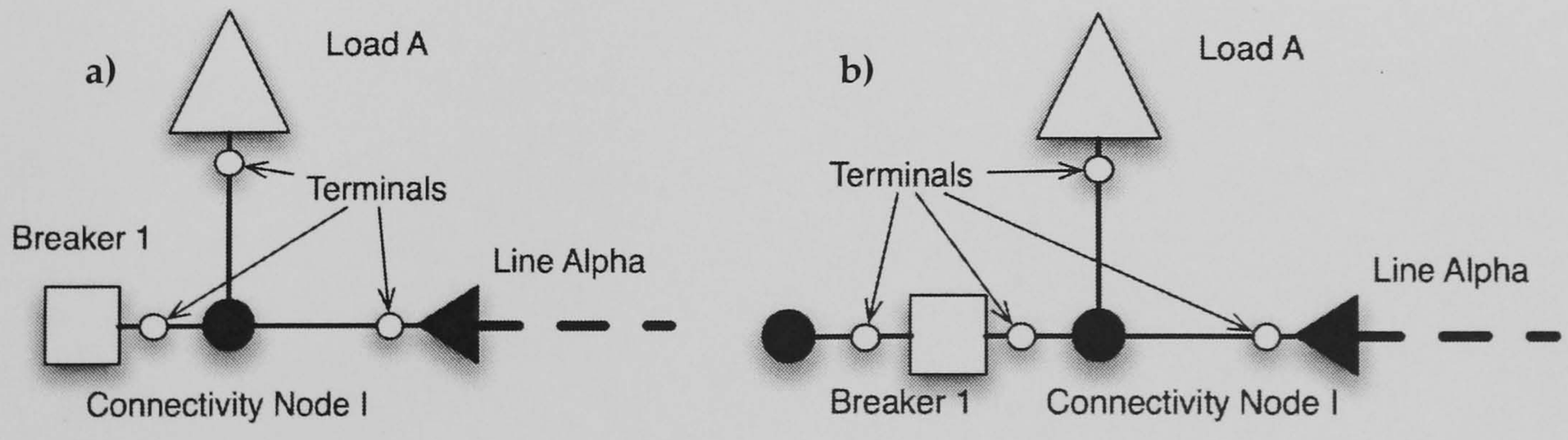


Figure 2.14 Connectivity Example circuit with Connectivity Node and Terminals

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 power flows, currents and voltages.

The importance of allowing the measurement point to be defined so exactly can be shown in Figure 2.14b). 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.

2.4.3 Converting a Circuit to CIM Objects

The previous chapters 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 modelled by converting a standard line diagram into CIM objects.

2.4.3.1 Identifying the CIM Classes

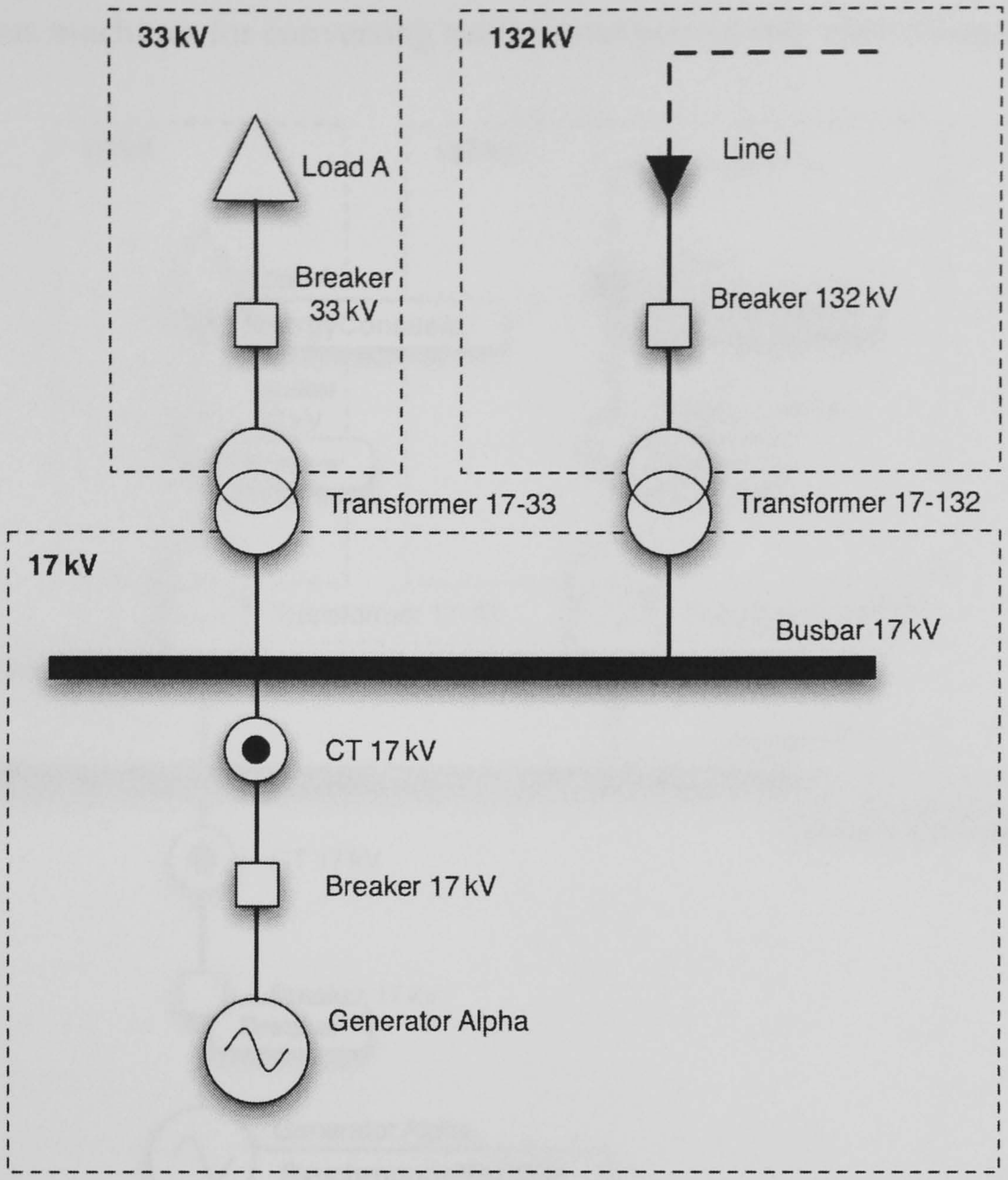


Figure 2.15 Example Circuit as a line diagram

The circuit shown in Figure 2.15 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, as stated in Section 2.4.2.3 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"[1]. 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"[1]

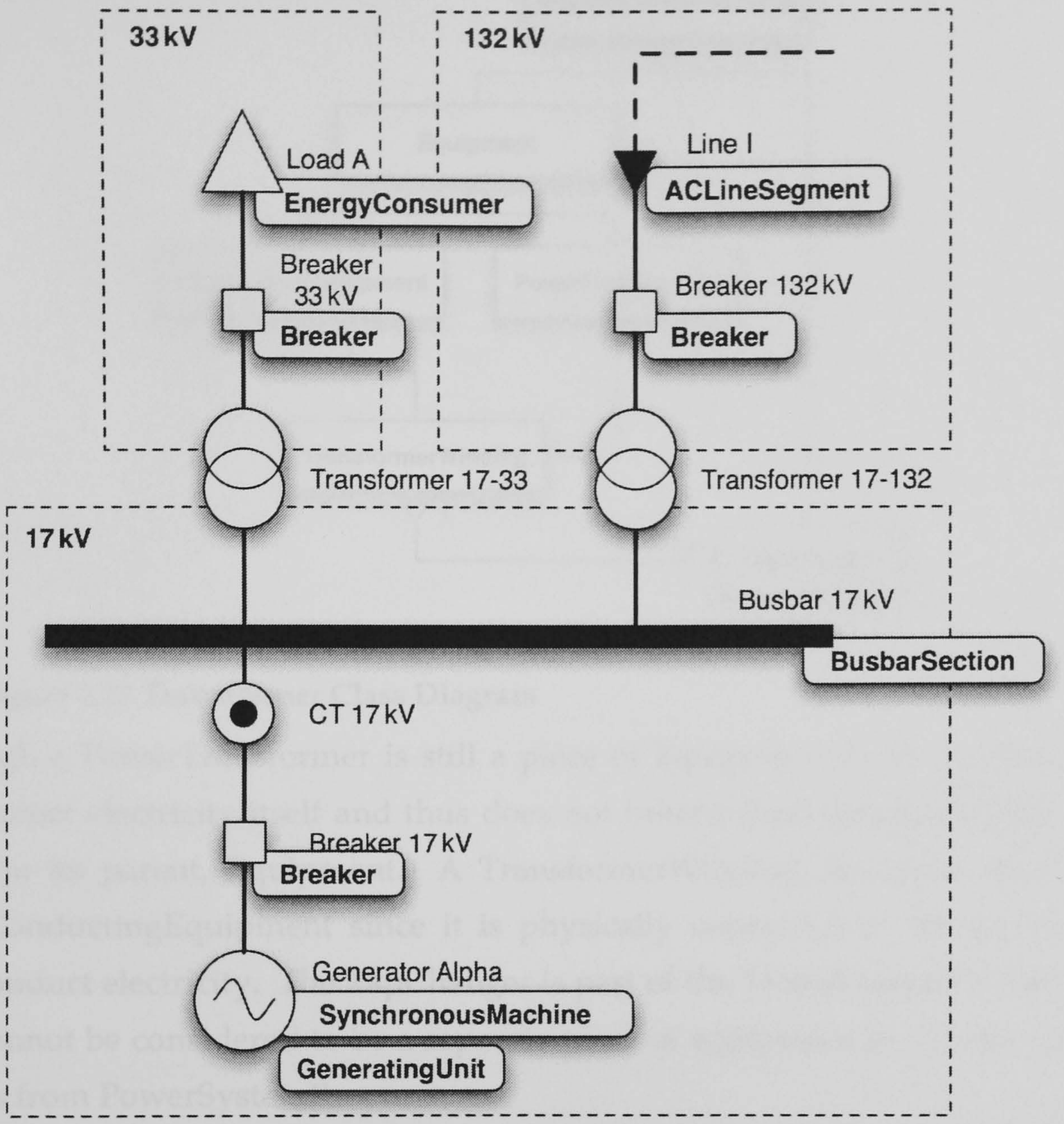


Figure 2.16 Example Circuit with partial CIM Class mappings

These mappings are shown in Figure 2.16, leaving only the two power transformers and current transformer to be mapped to CIM classes.

2.4.3.2 Representing Power Transformers as CIM Objects

A power transformer is not mapped to a single CIM class, instead it is split down into a number of components with a single PowerTransformer *container* class. Thus a two-winding power transformer becomes two TransformerWinding objects within a PowerTransformer container. If a tap changer is present to control one of the windings then an instance of the TapChanger class is associated with that particular winding while still being contained within the PowerTransformer instance. The

UML class diagram for the classes that form a transformer is shown in Figure 2.17 below.

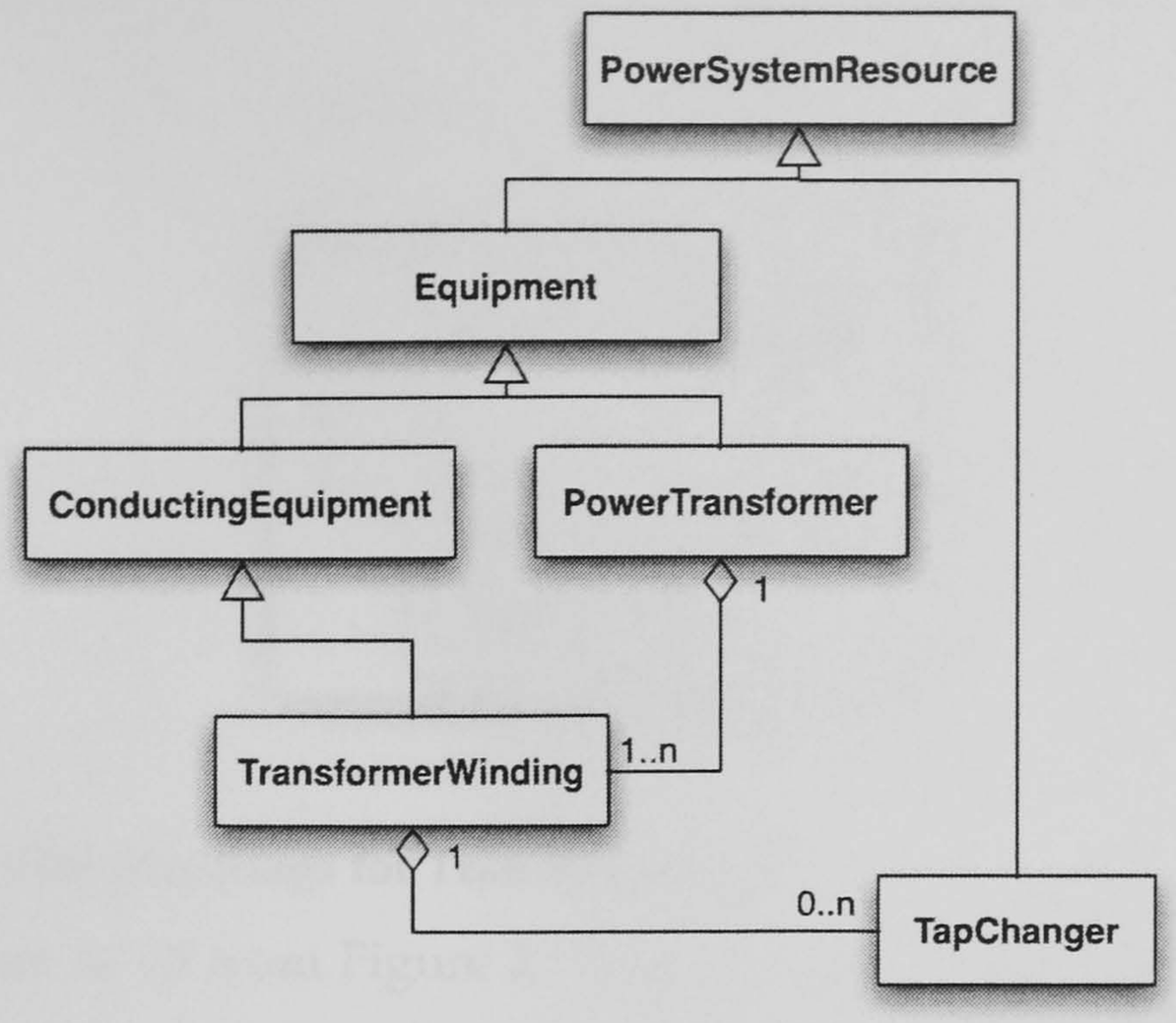


Figure 2.17 Transformer Class Diagram

Although a PowerTransformer is still a piece of Equipment in the system, it does not conduct electricity itself and thus does not inherit from ConductingEquipment but from its parent, Equipment. A TransformerWinding, however, does inherit from ConductingEquipment since it is physically connected to the network and does conduct electricity. 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.

The PowerTransformer and TransformerWinding classes have an aggregation relationship[2], 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.

---

[2] 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.

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.

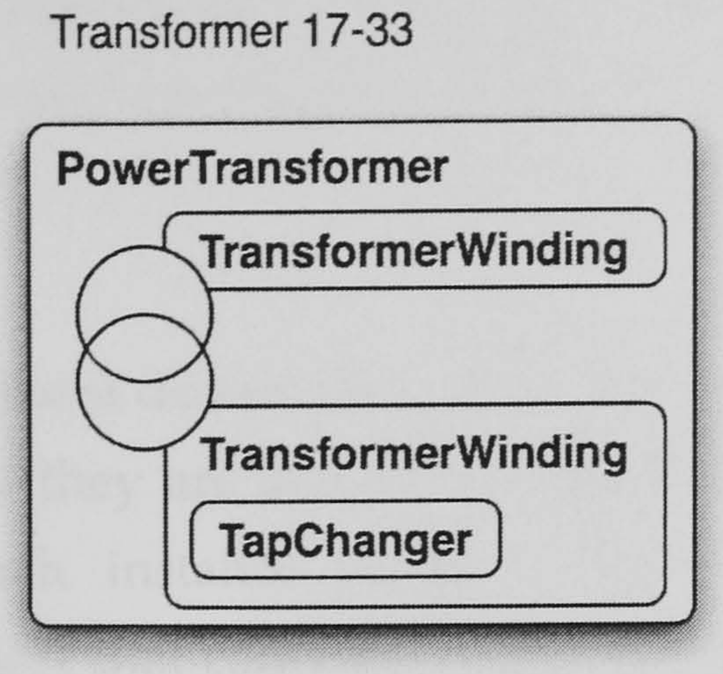


Figure 2.18 CIM Mappings for Transformer 17-33

Thus, Transformer 17-33 from Figure 2.15 can be represented as 4 CIM objects: two TransformerWindings, one TapChanger and one PowerTransformer as shown in Figure 2.18.

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

2.4.3.3 Representing a Current Transformer as a CIM Object

The current transformer CT 17kV does not map directly to a piece of conducting equipment in the CIM hierarchy as would be expected. The current transformer's purpose is to measure the current at its location in the network, and as such when modelling the network it is the measurement from that location that is modelled rather than the piece of equipment doing the measuring.

This involves creating an instance of the Measurement class to measure the current at a particular terminal. As described in Section 2.4.2.3, each piece of conducting equipment has one or more terminals to represent the points at which it connects to the network. By associating a Measurement object with a particular terminal and defining the measurement taken by that instance to be current then the Measurement object will reflect the role played by the current transformer.

2.4.3.4 Defining 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 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 itself has an associated BaseVoltage object that contains a single attribute to define the nominal voltage of that particular group of components. A BaseVoltage instance 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"[1].

In the example network shown in Figure 2.15, 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, is itself a container, not a piece of conducting equipment. 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, however, 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"[1]. 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"[1] it would be inaccurate to define it as being inside a specific voltage level within a substation. As such, the AC and DCLineSegment classes contain a direct association to the BaseVoltage class to define their nominal voltage level.

2.4.3.5 Equivalent CIM Representation

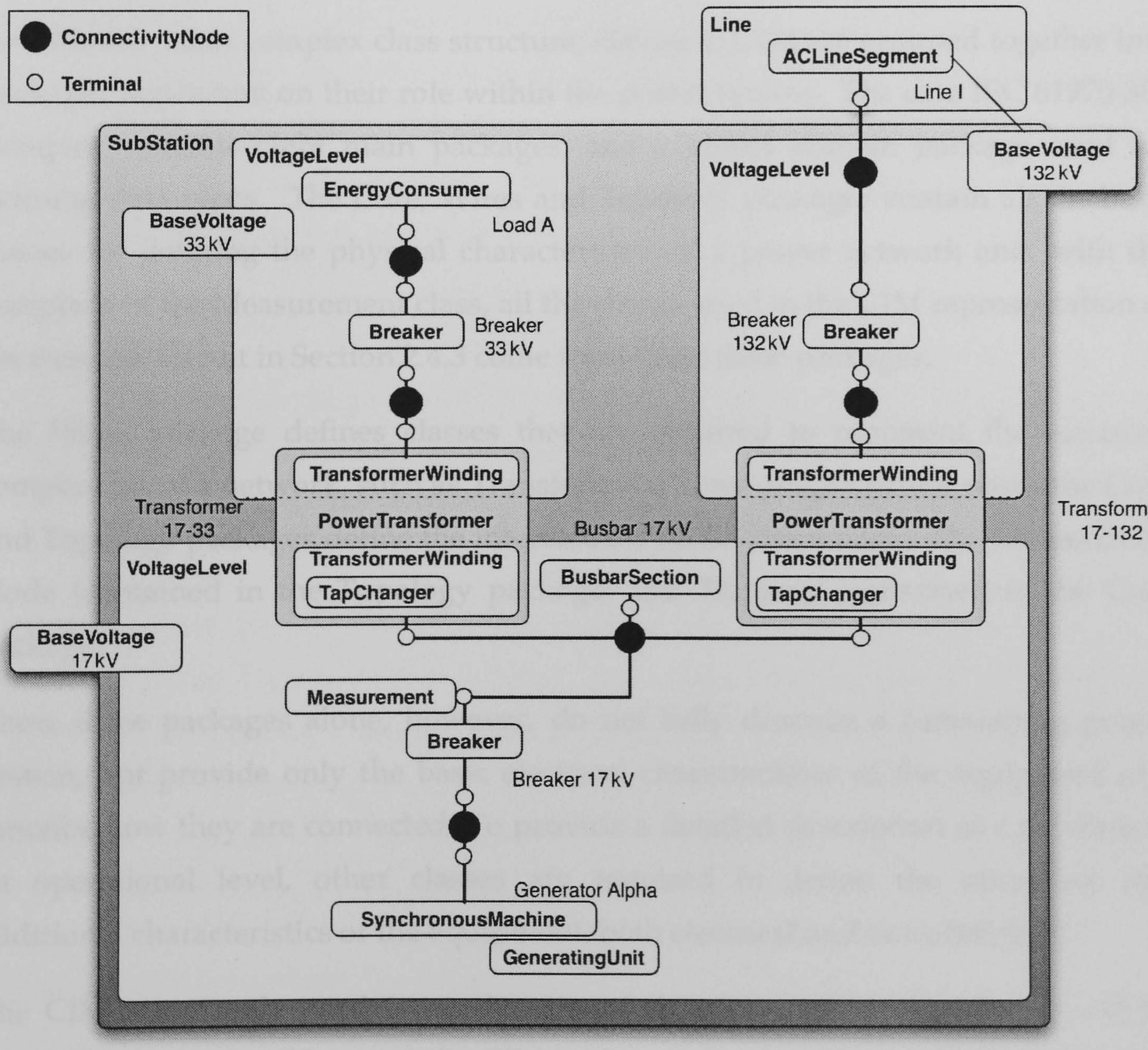


Figure 2.19 Example Circuit with full CIM Mappings

When fully converted to CIM objects, the original example circuit shown in Figure 2.15 is translated into the 45 CIM Objects shown in Figure 2.19. 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 busbar in the system. This reflects the positioning of equipment in the physical system, since a voltage transformer will often measure voltages at the busbars within a substation.

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 and load curves, but for now it is enough to understand how an existing network representation can be mapped to CIM objects.

2.4.4 IEC 61970-301 CIM Packages

As with any other complex class structure, classes in CIM are grouped together into packages dependent on their role within the power system. The core IEC 61970-301 standard contains eight main packages, and 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 and, with the exception of the Measurement class, all the classes used in the CIM representation of the example circuit in Section 2.4.3 come from these three packages.

The Wires package defines classes that are required to represent the electrical components of a network, such as Transformers, Lines and Switches, while the Core and Topology packages define the interconnection of components: The Connectivity Node (contained in the Topology package) and Terminal (contained in the Core package).

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 operation and additional characteristics of the equipment, both electrical and non-electrical.

The CIM is not only used for exchanging full power system models, as will be covered in more detail later on, the CIM is also used as a common model for defined business process messages. As such, a number of the packages contain classes that

are used for business processes and not for defining the properties of a full power system model in CIM format.

2.4.4.1 Core

The Core package contains the parent PowerSystemResource class, from which all other classes concerned with the physical properties of the network inherit (including all classes relating to physical pieces of equipment, as well as *Equipment Containers* which are used for organising pieces of equipment into groups, such as specific VoltageLevels, or equipment contained within a specific Substation).

2.4.4.2 Wires

The Wires package defines all pieces of equipment electrically connected to the network, as well as supporting classes for defining additional properties and arrangement of objects. This includes classes for the components that are physically connected to the network at the points of power generation and consumption (Energy Consumer and Synchronous Machine, as previously mentioned), as well as several classes that detail the arrangement and settings for Power Transformers, properties for Lines (comprised of one or more Line Segments), and other pieces of conducting equipment including Switches, Busbar Sections and Regulating Conducting Equipment (Compensators).

2.4.4.3 Generation

The Generation package is split into two sub-packages, Production and GenerationDynamics.

Production

The Production package is used for defining various kinds of generators, and includes a class hierarchy for defining the components of Thermal and Hydro generators. The package also includes definitions of production costing information such as Cost Curves and Net to Gross curves. To define power generation unit in the CIM requires an association of a production class object with a SynchronousMachine, a class contained within the Wires package.

GenerationDynamics

The GenerationDynamics package contains the description of Prime Movers, including turbine types, and classes that define various types of steam supplies,

such as Pressurised or Boiling Water Reactors for nuclear power stations, and different types of Fossil Fuel Boilers for coal oil and gas fired boilers.

2.4.4.4 LoadModel

The LoadModel package deals with modelling energy consumers through curves and associated data. The EnergyConsumer class (and its subclasses) within the Wires package define the physical connection point between the network and customer. Instances of the EnergyConsumer class also contain associations to Load Demand Models and Schedules for non-conforming load (e.g. large industrial loads, or power station services).

2.4.4.5 Topology

The Topology package, together with the Terminal class, provides definitions of how equipment connects together in the form of Connectivity Nodes. The Topological Node class is comprised of Connectivity Nodes connected by closed switches (and for many applications can be considered analogous to a bus in a bus-branch representation). The Topological Island class contains all electrically connected Topological Nodes, and as such a fully interconnected power network should contain only one Topological Island.

2.4.4.6 Measurement

The Meas (Measurement) package is used to define the Measurements being taken from a particular Power System Resource. There are two ways of connecting Measurements:

The first option is to associate a measurement instance with a Power System Resource, which covers measurements not related to electrical connectivity including temperature or weight.

The second option, as described in Section 2.4.3.3 is to associate the Measurement with a Terminal. This is used for measurements dependent on connectivity, such as current or voltage where the Terminal defines the point of the network that the measurement is to be taken from. The Measurement class acts as a Current or Voltage Transformer for measuring the current or voltage at a point in the network, however it does not represent a piece of physical equipment.

2.4.4.7 Outage

The Outage package define schedules for the planned network configuration, and includes classes that associate with a Switch to define its state at a particular time. This is used primarily for defining business process messages, however it could be used to change network configurations at specific times during a simulation.

2.4.4.8 Protection

The Protection package defines the settings and parameters of pieces of protection equipment that operate Switches. The classes defined in this package are used to describe the behaviour of the Switch: its current limit, delay from detection of abnormal conditions to operation, maximum and minimum limits.

2.5 The eXtensible Markup Language (XML)

2.5.1 XML

XML, the eXtensible Markup Language, is a "universal format for structured documents and data"[3], which is quickly becoming the standard for storing machine-readable data in a structured, extensible format that is accessible over the internet. XML is actually a meta-language[3] that allows the user to design their own markup language to describe the structure of the data.

XML is a subset of SGML, the Standard Generalized Markup Language[7] designed for both on and offline storage and transferral 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.

The XML syntax 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 other elements contained within these two tags are classed as "children" of the parent element.

2.5.1.1 Simple XML Example

As an example, a simple XML tag-syntax to store a book can 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>
```

---

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

```
    <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>
```

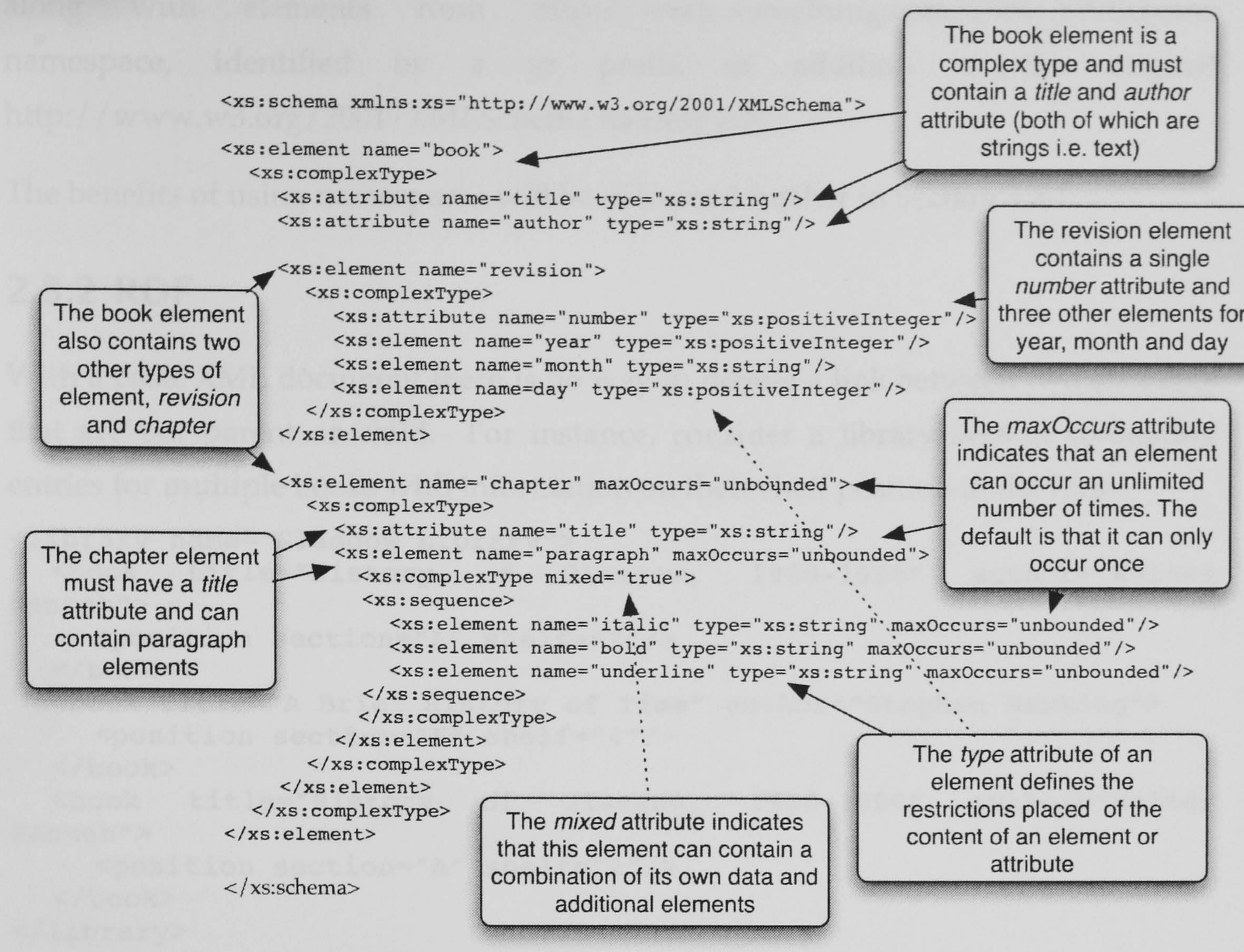
Here the *book* element contains its own attribute to describe the *title* and *author*, with a child element to describe the *revision* of the book, plus several *chapter* elements. The *chapters* in turn contain elements for each *paragraph*, which themselves contain mixed data of other elements and 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.

2.5.1.2 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)[4] format or the newer XML Schema[5]. The XML Schema 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; and if they have default values.

Using the previous book example, a simple XML Schema 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 2.20




```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="book">
  <xs:complexType>
    <xs:attribute name="title" type="xs:string"/>
    <xs:attribute name="author" type="xs:string"/>

    <xs:element name="revision">
      <xs:complexType>
        <xs:attribute name="number" type="xs:positiveInteger"/>
        <xs:element name="year" type="xs:positiveInteger"/>
        <xs:element name="month" type="xs:string"/>
        <xs:element name=day" type="xs:positiveInteger"/>
      </xs:complexType>
    </xs:element>

    <xs:element name="chapter" maxOccurs="unbounded">
      <xs:complexType>
        <xs:attribute name="title" type="xs:string"/>
        <xs:element name="paragraph" maxOccurs="unbounded">
          <xs:complexType mixed="true">
           <xs:sequence>
             <xs:element name="italic" type="xs:string" maxOccurs="unbounded"/>
             <xs:element name="bold" type="xs:string" maxOccurs="unbounded"/>
             <xs:element name="underline" type="xs:string" maxOccurs="unbounded"/>
           </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:complexType>
    </xs:element>
  </xs:complexType>
</xs:element>

</xs:schema>
```

Figure 2.20 Annotated simple XML Schema Example describing the data within a book

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*[4]. An XML document can contain elements from multiple namespaces simultaneously, each of which denote a seperate XML Schema 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.other.com/2005/ABSchema"
xmlns:yz="http://www.something.com/2004/YZ-Schema">
```

Indicating that the XML document may contain elements from the http://www.other.com/2005/ABSchema namespace, identified by an *ab* prefix,

---

[4] 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.

along with elements from http://www.something.com/2004/YZ-Schema namespace, identified by a *yz* prefix in addition to the original http://www.w3.org/2001/XMLSchema namespace.

The benefits of using namespaces will be discussed further in Section 3.2.1.

2.5.2 RDF

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

```
<library name="Glasgow Library">
  <book title="History of Glasgow, 1900-1950" author="Walter
Hannah">
    <position section="A" shelf="2"/>
  </book>
  <book title="A Brief History of Time" author="Stephen Hawking">
    <position section="E" shelf="4"/>
  </book>
  <book title="History of Glasgow, 1950-2000" author="Walter
Hannah">
    <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 *History of Glasgow, 1900-1950* and *History of Glasgow, 1950-2000* books to indicate that reader may wish to read the former book before the latter, there is no standard way to do this using the basic XML constructs.

The Resource Document Framework (RDF)[25] is an XML schema used to provide a framework for data in an XML format by allowing relationships to be defined between XML nodes. Each element can be assigned a unique *ID* attribute under the RDF namespace http://www.w3.org/1999/02/22-rdf-syntax-ns# (which 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*.

2.5.2.1 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*.

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.strath.ac.uk/libraries/2006/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.strath.ac.uk/libraries/2006/library-schema#">
<lib:library lib:name="Glasgow Library">
  <lib:book lib:title="History of Glasgow, 1900-1950"
lib:author="Walter Hannah" rdf:ID="_entry0001">
    <lib:position lib:section="A" lib:shelf="2"/>
    <lib:sequel rdf:resource="#_entry0003"/>
  </lib:book>
  <lib:book lib:title="A Brief History of Time" lib:author="Stephen
Hawking" rdf:ID="_entry0002">
    <lib:position lib:section="E" lib:shelf="4"/>
  </book>
  <lib:book lib:title="History of Glasgow, 1950-2000"
lib:author="Walter Hannah" rdf:ID="_entry0003">
    <lib:position lib:section="A" lib:shelf="2"/>
    <lib:sequelTo rdf:resource="#_entry0001"/>
  </lib:book>
</lib:library>
</rdf:RDF>
```

As shown, the RDF provides a means of showing relationships between elements outwith the standard parent-child relationship. The schema contains additional elements that go beyond the simple *ID* and *resource* attribute as will be shown in next section, but it is these features of the framework that are utilised when expressing the CIM in XML format.

2.5.2.2 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[26] provides the user with a means of describing specific kinds of resources or classes. The RDF Schema does not provide a vocabulary for a specific application's classes like *lib:sequel* or *lib:sequelTo*, or properties like *lib:title* and *lib:author*. Instead, the RDF Schema 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:title* will be used in describing a *lib:book*, or that *lib:sequel* is an element of *lib:book* and should indicate a reference to another *lib:book* entry.

In essence, the RDF Schema 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 defined as instances of one or more classes and for these classes to be organised in a hierarchy.

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

```
<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="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 class of *book* is defined, then the two properties *sequel* and *sequelTo* are defined. Each of these properties has their domain (the class the property is within) referencing the *book* class, as does their range (the class of element the property refers to). 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 modelled 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.

2.5.3 CIM RDF XML

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. Using the previous example of the Transformer class hierarchy shown in Figure 2.17, the resulting RDF Schema takes the form:

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

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

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

<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.MemberOf_PowerTransformer">
  <rdfs:label xml:lang="en">MemberOf_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.

2.5.3.1 CIM RDF XML Example

As an example, the simple Transformer example from Figure 2.18 will be extended to include attributes for each object. This produces four objects with their own internal data as shown in Figure 2.21 below:

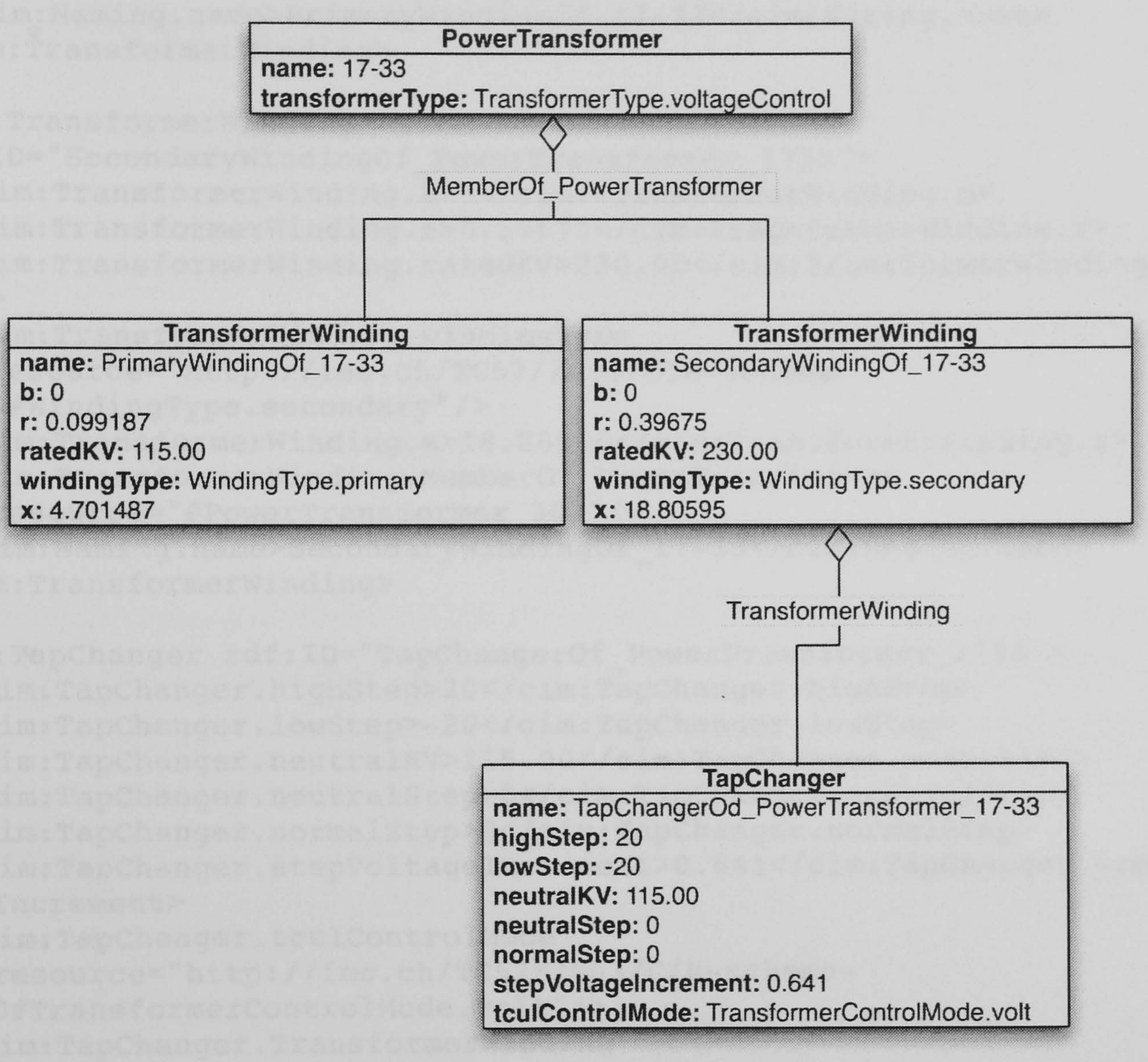


Figure 2.21 Transformer shown as four CIM Objects with attributes

Each of these objects can then be expressed as an XML node using the CIM RDF Schema given the namespace http://iec.ch/TC57/2003/CIM-schema-cim10# and prefix *cim*:

```
<rdf:RDF xmlns:cim="http://iec.ch/TC57/2003/CIM-schema-cim10#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

<cim:PowerTransformer rdf:ID="PowerTransformer_1733">
  <cim:PowerTransformer.transformerType
rdf:resource="http://iec.ch/TC57/2003/CIM-schema-
cim10#TransformerType.voltageControl"/>
  <cim:Naming.name>17-33</cim:Naming.name>
</cim:PowerTransformer>
```

```
<cim:TransformerWinding
rdf:ID="PrimaryWindingOf_PowerTransformer_1733">
   <cim:TransformerWinding.b>0</cim:TransformerWinding.b>
   <cim:TransformerWinding.r>0.099187</cim:TransformerWinding.r>
   <cim:TransformerWinding.ratedKV>115.00</cim:TransformerWinding.rat
edKV>
   <cim:TransformerWinding.windingType
rdf:resource="http://iec.ch/TC57/2003/CIM-schema-
cim10#WindingType.primary"/>
   <cim:TransformerWinding.x>4.701487</cim:TransformerWinding.x>
   <cim:TransformerWinding.MemberOf_PowerTransformer
rdf:resource="#PowerTransformer_302"/>
   <cim:Naming.name>PrimaryWindingOf_17-33</cim:Naming.name>
</cim:TransformerWinding>

<cim:TransformerWinding
rdf:ID="SecondaryWindingOf_PowerTransformer_1733">
   <cim:TransformerWinding.b>0</cim:TransformerWinding.b>
   <cim:TransformerWinding.r>0.39675</cim:TransformerWinding.r>
   <cim:TransformerWinding.ratedKV>230.00</cim:TransformerWinding.rat
edKV>
   <cim:TransformerWinding.windingType
rdf:resource="http://iec.ch/TC57/2003/CIM-schema-
cim10#WindingType.secondary"/>
   <cim:TransformerWinding.x>18.80595</cim:TransformerWinding.x>
   <cim:TransformerWinding.MemberOf_PowerTransformer
rdf:resource="#PowerTransformer_302"/>
   <cim:Naming.name>SecondaryWindingOf_17-33</cim:Naming.name>
</cim:TransformerWinding>

<cim:TapChanger rdf:ID="TapChangerOf_PowerTransformer_1733">
   <cim:TapChanger.highStep>20</cim:TapChanger.highStep>
   <cim:TapChanger.lowStep>-20</cim:TapChanger.lowStep>
   <cim:TapChanger.neutralKV>115.00</cim:TapChanger.neutralKV>
   <cim:TapChanger.neutralStep>0</cim:TapChanger.neutralStep>
   <cim:TapChanger.normalStep>0</cim:TapChanger.normalStep>
   <cim:TapChanger.stepVoltageIncrement>0.641</cim:TapChanger.stepVol
tageIncrement>
   <cim:TapChanger.tculControlMode
rdf:resource="http://iec.ch/TC57/2003/CIM-schema-
cim10#TransformerControlMode.volt"/>
   <cim:TapChanger.TransformerWinding
rdf:resource="#PrimaryWindingOf_PowerTransformer_302"/>
   <cim:Naming.name>TapChangerOf_PowerTransformer_17-
33</cim:Naming.name>
</cim:TapChanger>

</rdf:RDF>
```

The PowerTransformer.transformerType and TapChanger.tculControlMode 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. The ability to include additional data within the standard CIM RDF XML (commonly shortened to CIM XML) by using multiple schemas and namespaces simultaneously will be covered further in Section 3.2.1.

2.6 XML Messaging

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.

2.6.1 Existing Inter-Application Communication Infrastructure

Within large companies 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 links 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 communication links to be defined and implemented, further increasing the complexity of the overall system with a corresponding financial penalty.

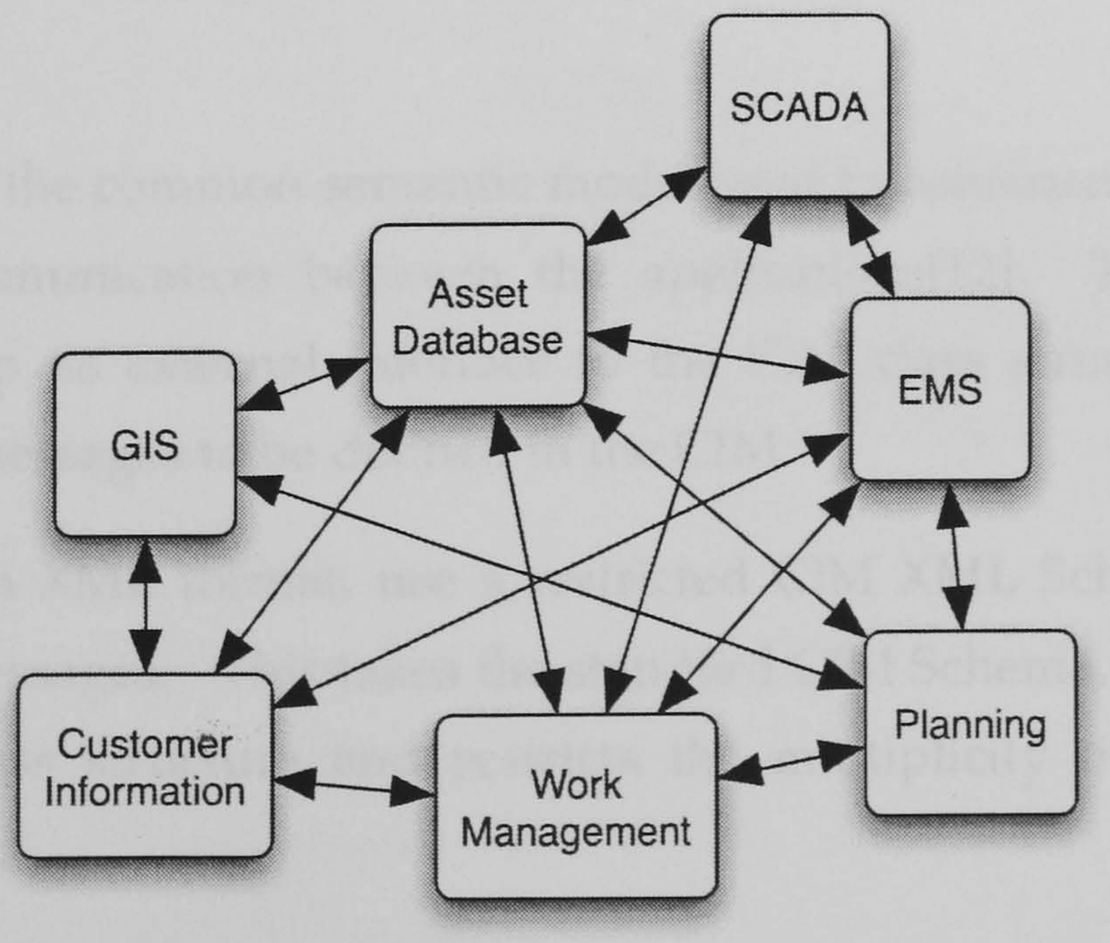


Figure 2.22 Communication links between enterprise applications

As illustrated in Figure 2.22, even for a small section of the overall IT system, this can result in a large number of inter-application communication links. As companies expand their IT infrastructure or replace existing applications with products from other vendors they must define new interfaces for each communication link, a process that is both time consuming and expensive.

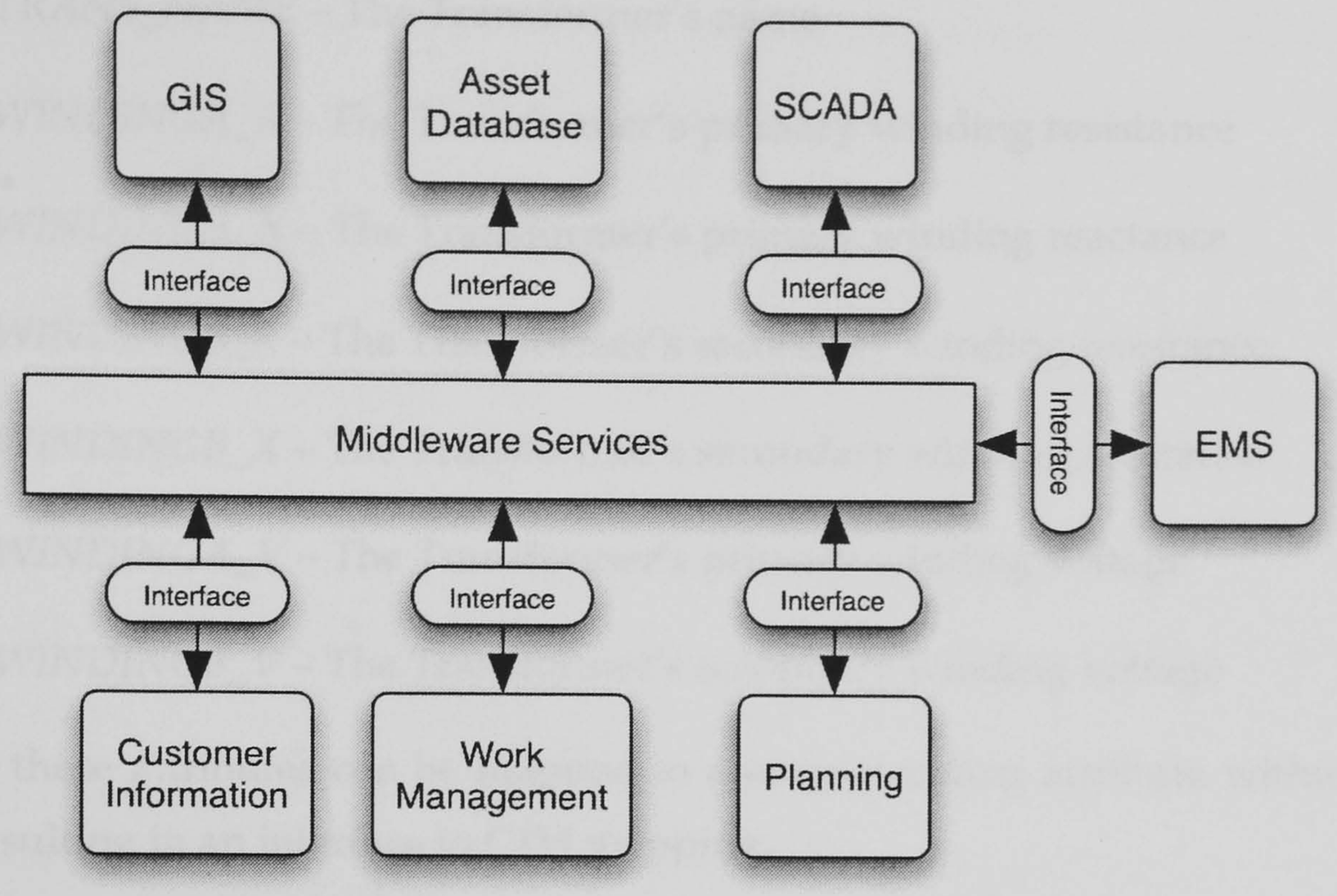


Figure 2.23 Enterprise Application Bus model for inter-application communication

2.6.2 The Message Bus Concept

Enterprise Application Integration (EAI)[11] 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.

The CIM provides the common semantic model used to construct the messages that are used for communication between the applications[12]. 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.

2.6.3 Mapping Application Interfaces to the CIM

This can be illustrated using a simple example. An EMS application's external interface requires the user to access data on the transformers within the system. The EMS application's interface attributes 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.

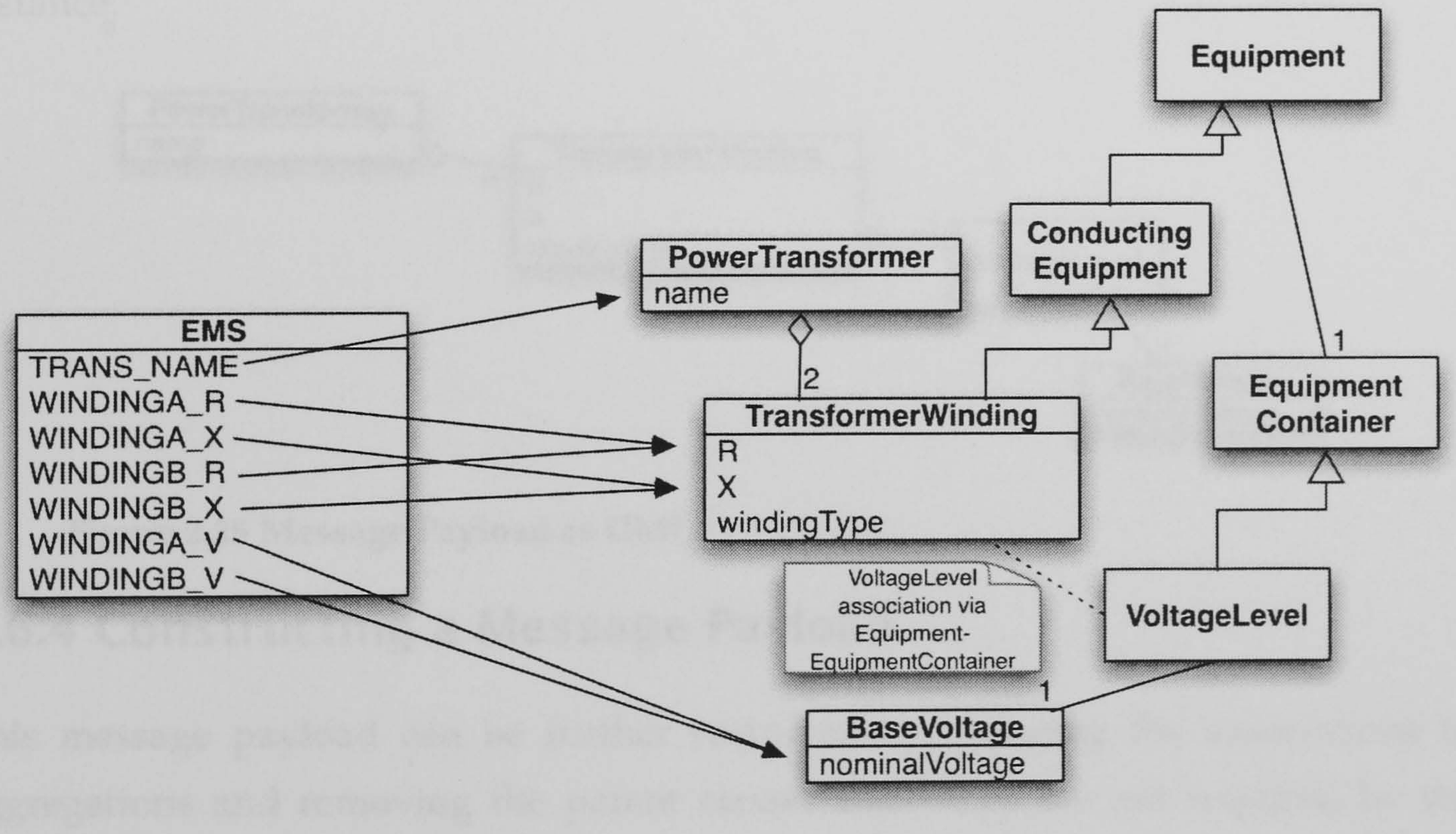


Figure 2.24 CIM Interface Mapping

This mapping, shown in Figure 2.24, 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 TransformerWinding 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 TransformerWinding class present in the message, with the windingType 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 TransformerWinding via the VoltageLevel class whose relationship with TransformerWinding is defined in the Equipment - EquipmentContainer association further up the class hierarchy. This is because the TransformerWinding class itself does not contain a direct VoltageLevel association within the CIM, but instead inherits a MemberOf_Equipment container association from the Equipment class (via ConductingEquipment), and since VoltageLevel is a subclass of EquipmentContainer this can be used to provide the required association to VoltageLevel.

Both the EquipmentContainer and BaseVoltage associations have their multiplicity changed from 0..1 to 1 requiring each VoltageLevel to have one BaseVoltage instance associated with it and each piece of Equipment (in this case TransformerWinding) to have an association to a single EquipmentContainer instance.

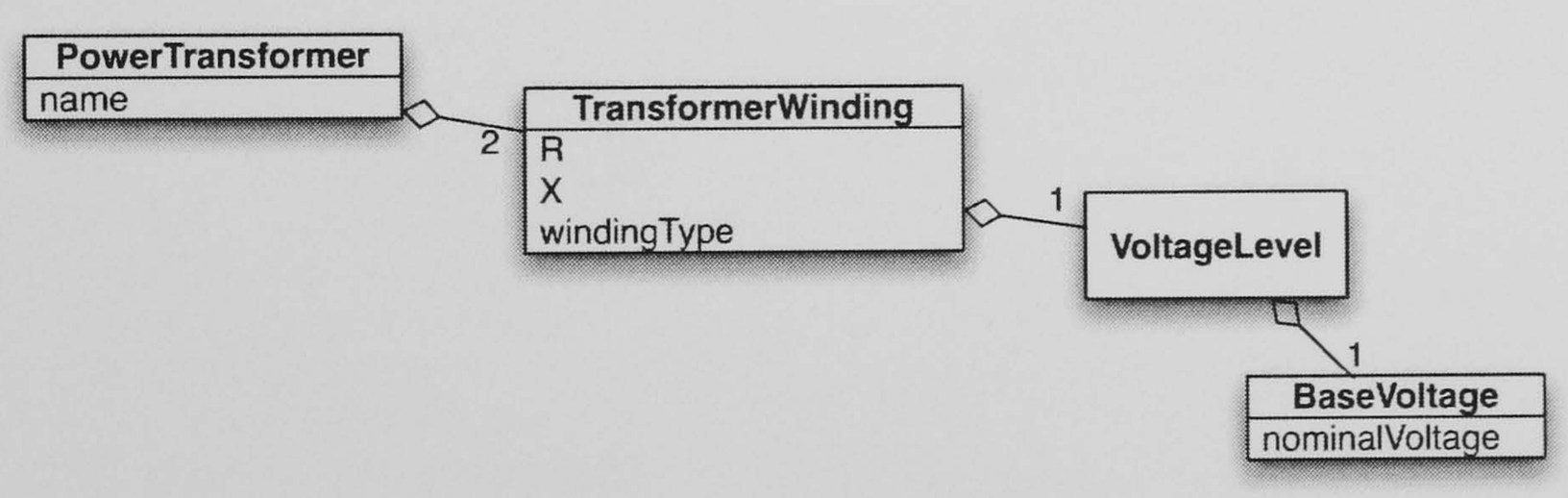


Figure 2.25 Message Payload as UML

2.6.4 Constructing a Message Payload

This message payload can be further restricted by changing the associations to aggregations and removing the parent classes since they are not required by the actual message content. Thus the message payload can be represented as the modified class structure shown in Figure 2.25.

Thus a single two winding transformer containing the desired attributes can be represented in XML as:

```
<cim:PowerTransformer>
  <cim:Naming.name>Transformer SGT1</cim:Naming.name>
```

```
<cim:PowerTransformer.Contains_TransformerWindings>
   <cim:TransformerWinding.r>0.23</cim:TransformerWinding.r>
   <cim:TransformerWinding.x>0.78</cim:TransformerWinding.x>
   <cim:TransformerWinding.windingType>WindingType.primary
        </cim:TransformerWinding.windingType>
   <cim:Equipment.MemberOf_EquipmentContainer>
      <cim:VoltageLevel.BaseVoltage>
         <cim:BaseVoltage.nominaVoltage>400
          </cim:BaseVoltage.nominalVoltage>
      </cim:VoltageLevel.BaseVoltage>
   </cim:Equipment.MemberOf_EquipmenContainer>
</cim:PowerTransformer.Contains_TransformerWindings>
<cim:PowerTransformer.Contains_TransformerWindings>
   <cim:TransformerWinding.r>0.46</cim:TransformerWinding.r>
   <cim:TransformerWinding.x>0.87</cim:TransformerWinding.x>
      <cim:TransformerWinding.windingType>WindingType.secondary
           </cim:TransformerWinding.windingType>
      <cim:Equipment.MemberOf_EquipmentContainer>
         <cim:VoltageLevel.BaseVoltage>
            <cim:BaseVoltage.nominaVoltage>275
                 </cim:BaseVoltage.nominalVoltage>
         </cim:VoltageLevel.BaseVoltage>
      </cim:Equipment.MemberOf_EquipmenContainer>
   </cim:PowerTransformer.Contains_TransformerWindings>
</cim:PowerTransformer>
```

This XML message in turn has an XML Schema to describe the payload contents:

```
<xs:schema xmlns:cim="cimBase" xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element minOccurs="1" maxOccurs="1" name="PowerTransformer">
 <xs:complexType>
  <xs:complexContent>
   <xs:extension base="cim:PowerTransformer">
    <xs:sequence>
     <xs:element minOccurs="1" maxOccurs="1"
         name="Naming.name" type="xs:string"/>
     <xs:element minOccurs="2" maxOccurs="2"
         name="PowerTransformer.Contains_TransformerWindings">
      <xs:complexType>
       <xs:complexContent>
        <xs:extension base="cim:TransformerWinding">
         <xs:sequence>
          <xs:element minOccurs="1" maxOccurs="1"
              name="TransformerWinding.r" type="xs:float"/>
          <xs:element minOccurs="1" maxOccurs="1"
              name="TransformerWinding.x" type="xs:float"/>
          <xs:element minOccurs="1" maxOccurs="1"
              name="TransformerWinding.windingType" type="cim:WindingType"/>
          <xs:element minOccurs="1" maxOccurs="1"
              name="TransformerWinding.MemberOf_EquipmentContainer">
           <xs:complexType>
            <xs:complexContent>
             <xs:extension base="cim:VoltageLevel">
              <xs:sequence>
               <xs:element minOccurs="1" maxOccurs="1"
                   name="VoltageLevel.BaseVoltage">
                <xs:complexType>
                 <xs:complexContent>
                  <xs:extension base="cim:BaseVoltage">
                   <xs:sequence>
                    <xs:element minOccurs="1" maxOccurs="1"
                        name="BaseVoltage.nominalVoltage" type="xs:float"/>
                   </xs:sequence>
                  <xs:extension>
                 </xs:complexContent>
                </xs:complexType>
               </xs:element>
              </xs:sequence>
             </xs:extension>
            </xs:complexContent>
           </xs:complexType>
          </xs:element>
         </xs:sequence>
        </xs:extension>
       </xs:complexContent>
      </xs:complexType>
     </xs:element>
    </xs:sequence>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>
</xs:element>

</xs:schema>
```

This schema refers to another *cimBase* schema that contains the definitions for the CIM classes, their attributes and associations and for the enumerations and data types such as *WindingType*.

The major difference between the CIM XML Message and the CIM RDF XML is that instead of having each CIM object as an independent XML element that is then linked using the RDF *ID* and *resource* attribute, the elements are contained within each other. This way a PowerTransformer element contains child elements to describe the TransformerWindings, that in turn contain child elements to denote the voltage.

For the scope of the message, this means that the PowerTransformer element contains all the required data making it simpler to then transform the element into another format if that is required. As will be described later, importing and converting CIM RDF XML data is more challenging, but given the highly interconnected nature of that representation it is not possible to represent it as nested XML elements for anything other than the most simple network models.

2.6.5 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. 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 not only a common data format but crucially provides a common semantic model, which provides consensus on the interpretation of each class and attribute.

This CIM XML messaging approach has been applied by a number of large utilities both in the UK and the USA and has proven to be a flexible and scalable system. As well as exchanging data on logical power system components, extensions to the core 61970-301 CIM classes contains packages and classes to allow the definition of business processes such as work scheduling, customer invoicing and financial trading arrangement as a CIM XML messages.

While the work detailed in the rest of document is concerned with using 61970-301 CIM data in the form of full power system models initially encoded as CIM RDF XML, an understanding of the XML messaging application of the CIM standard is beneficial. Much of the work being undertaken to extend the CIM is concerned with using it to define message payloads for exchanging between applications in an EAI environment. This is why only a small subset of the overall CIM class structure is used when representing a logical power systems model.

2.7 Chapter Summary

This chapter has introduced the concept of classes and class hierarchies along with their basic relationships that define how classes relate to each other: inheritance, association, aggregation and composition. The benefits of using this approach to define the components of a power system were then demonstrated along with an example of how a simple power system, represented as a line diagram, can be mapped to CIM Objects. The extensible markup language, its resource document framework subset and schemas were then introduced to demonstrate how the CIM class structure is mapped and the data encapsulated in an XML format. Finally the primary uses of the CIM were discussed: for encapsulating entire power system models as CIM RDF XML; and exchanging data between applications as CIM XML Messages. All of this forms an essential foundation on which the novel developments in this thesis are built.

3 Extensions to the Common Information model

3.1 Chapter Introduction

The object oriented nature of the CIM, combined with the adoption of XML as the primary method of encapsulating the data for exchange purposes, allows the CIM to be extended and enhanced to include additional data not provided for by the original IEC standard classes. Such extensions have been proposed by a number of parties including official IEC working groups, academic institutions and software vendors. This chapter describes the methods used to allow these multiple standards to co-exist within a single CIM network model and then summarises a number of these extensions. Finally extensions are proposed by the author to enhance the CIM to allow the accurate modelling of the UK electrical network, and to support the use of power system network models in CIM format for planning applications.

3.2 Methods for Coping with Multiple CIM Standards

Since, there is often a need to extend the current IEC 61970-301 CIM standard, this creates the problem of having an extended version of CIM co-existing with the IEC standard CIM. To accomplish this, a means of identifying which standard each item of data comes from is required. This identification should exist within the data format being imported into a CIM based application and the application itself must be able to identify which standard each item of data comes from and how to deal with it. In this project the industry standard method of exchanging CIM data, CIM XML format, is used. This approach utilises the Resource Document Framework (RDF) syntax to represent the associations between nodes.

3.2.1 XML Namespaces

Multiple standards can be used within one file by utilising the XML (eXtensible Markup Language) namespace system, whereby XML nodes are prefixed with a short string that corresponds to a namespace Uniform Resource Identifier (URI) in the head of the document denoting a separate XML schema. For the IEC 61970-301 standard CIM data, the *cim* prefix is used with a URI that refers to a unique XML Schema (in this case also an RDF Schema). Thus the root element of the XML document contains:

```
xmlns:cim="http://iec.ch/TC57/2003/CIM-schema-cim10#"
```

This indicates that the XML Namespace prefix *cim* corresponds to the schema identified by that particular URI. The URI is used to provide a unique identifier for the schema, but does not resolve to a specific file or addess. Their purpose is purely to provide a unique identifier for the namespace and schema, and this format was adopted because the formatting of URIs is well documented. Any unique string could be used, however, but with the example given above, it allows a quick interpretation of the standard to be made:

- *iec.ch* – the standard is from the IEC (the *.ch* refers to Switzerland where the IEC is based)
- *TC57* – the standard was developed by Technical Committee 57 (Power Systems Management and Associated Information Exchange)
- *2003* – the standard is the 2003 version
- *CIM-schema-cim10#* - a brief description of the standard, and version number

Prefixing nodes and attributes with *cim* indicates that they are part of this IEC standard, and as such, any application that is importing the data can deal with any namespace that it recognizes.

Extending and changing the standard therefore requires that any new or modified classes are identified as being of a different standard, since simply adding a *cim* prefix to them would indicate that they are part of the original IEC standard schema, when this is not the case.

For the extended standard proposed in this project, the prefix *ngt* is used, with a namespace of "http://eee.strath.ac.uk/2006/Strath-CIM-Schema11#" denoting that the schema is from the University of Strathclyde's EEE Department; is the 2006 revision and denoted as version 1.1 of CIM.

The head of the XML document therefore now contains three namespaces to denote the nodes and attributes that are either:

- From the original IEC CIM schema
- From the extended Strathclyde CIM schema
- From the RDF schema.

This creates a head node, the start of the RDF document, denoted by *rdf:RDF*. Within this head node, the URIs of the other namespaces used within the document are included. Thus the opening RDF node contains:

```
<rdf:RDF xmlns:cim="http://iec.ch/TC57/2003/CIM-schema-cim10#"
xmlns:stcim="http://eee.strath.ac.uk/2006/Strath-CIM-Schema11#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
```

Contained within this *rdf:RDF* node is all the network information formatted as the following example shows:

```
<cim:Disconnector rdf:ID="DisSwitch_2">
  <cim:Switch.normalOpen>false</cim:Switch.normalOpen>
  <cim:ConductingEquipment.Terminals rdf:resource="#Terminal_23A"/>
  <cim:ConductingEquipment.Terminals rdf:resource="#Terminal_22B"/>
  <cim:Equipment.MemberOf_EquipmentContainer
rdf:resource="#VoltageLevel_2"/>
  <cim:Naming.name>DisSwitch_2</cim:Naming.name>
</cim:Disconnector>

<stcim:NetworkConnectionPoint rdf:ID="NCP_1">
   <cim:Naming.name>NCP_1</cim:Naming.name>
   <cim:EquipmentContainer.Contains_Equipments
rdf:resource="#DisSwitch_2"/>
   <stcim:NetworkConnectionPoint.External_Terminal
rdf:resource="#Terminal_NCP2X1"/>
</stcim:NetworkConnectionPoint>
```

This example shows two nodes, a Disconnector and a NetworkConnectionPoint, the former being a standard IEC CIM class, the latter from the Strathclyde CIM standard. The mixture of *stcim* and *cim* prefixes in the second node is because the Network Connection Point class is a child class of the IEC CIM EquipmentContainer class, and as such these attributes are inherited from it (or its parent class, Naming).

This system allows data from the two standards to co-exist within the same document. Existing applications that are not able to cope with the extended Strathclyde data will simply ignore the nodes that are of the "http://eee.strath.ac.uk/2006/Strath-CIM-Schema11#" namespace and only import those identified as being from the IEC CIM standard.

3.3 IEC Proposed Extensions to CIM

The IEC TC57 Working Groups 13 & 14 are proposing additional extensions to the CIM standard to include additional data. This includes the proposed IEC 61968 standard under development by WG14: System Interfaces for Distribution Management which is designed to facilitate inter-application integration among application software sharing information as part of a company's Distribution Management System.

3.3.1 IEC 61970 Extensions

As well as the core packages that form the IEC 61970-301 standard, there are additional packages under the 61970-302 and 303 standard that are focussed on exchanging data between companies and for dealing with Supervisory Control and Data Acquisition (SCADA) applications.

3.3.1.1 IEC 61970-302

The 61970-302 packages, Reservation, Financial and Energy Scheduling are used primarily for exchanging data between companies.

The **Energy Scheduling** package provides a model to represent the data exchanges made between companies when scheduling the transfer of electricity and the resulting transactions. These transactions include recording power that is generated, consumed and lost as well as the sale and purchase records.

The **Financial** package contains classes to represent the legal entities (e.g. generators, consumers, operators, transmission providers) involved with the exchange of electricity along with the settlement and billing agreements that are required.

The **Reservation** package represents the transmission services and paths used to exchange electricity. This includes classes to represent the Ancillary Services. These relate to various aspects of insuring that the production of energy matches consumption of energy at any given time. Such services are critical to the security and reliability of the interconnected network.

These three packages are primarily aimed at the application of CIM to XML Messaging described in Section 2.6 and are not usually included with a full power system model in CIM format.

3.3.1.2 IEC 61970-303

The 61970-303 standard is focused on providing a model to represent the information required by SCADA applications. This SCADA package links in with the Measurement package and provides definitions of the SCADA units that link to the Control class for providing Measurement Values within the Measurement package.

3.3.2 IEC 61968 Extensions

The IEC 61968-11 draft standard[2] contains extensions to the CIM aimed at covering the data requirements of distribution management systems, and contains six packages. The Core2 package contains packages intended to be included in the initial version of CIM, but was not included in the standard submitted to the IEC since they were not deemed to be of sufficient maturity at the time.

Within the draft of the 61968 packages are a number that have an impact on the work undertaken to develop application using a CIM software architecture. A large number, however, are aimed at the business process messaging application of the CIM and are not of direct relevance when exchanging power system model data.

3.3.2.1 Activity Records

The **Activity Records** package contains the classes to "record the activity for an Asset, Location, Power System Resource, Customer, Erp Contact (e.g., operator, market participant, etc.), or other object at a point in time"[2]. Such activites can be events that have already occurred, or details of pre-planned activities. An Activity Record is associated with a Power System Resource, and "the relationship records events regarding the logical function being provided by the resource in the electrical network"[2]. The relationship between the Asset object and the Power System Resource objects allows the recording of the asset's history, independently of where it is currently being used in the electrical network. The Location object records events associated with the geographical location of the asset. The future state activities are used, for example, when generating units must be scheduled for maintenance or when a transformer is scheduled for refurbishment.

3.3.2.2 Assets

The **Assets** package and its child packages of **Asset Basics, Point Asset Hierarchy** and **Linear Asset Hierarchy** provide classes for describing both the asset-

management level of data for existing Power System Resources, and stand-alone assets that are independent of the function of the power network.

3.3.2.3 Location

The **Location** package defines the position of a Power System Resource, allowing for single coordinate points or multiple coordinates for defining a specific area (such as a substation site).

3.3.2.4 Outage

The **Outage Package** within the proposed 61968 standard is more concerned with Asset management and record keeping than the Outage package in the 61970 standard detailed in 2.4.4.7. Whereas in the 61970 Outage package the classes are used to define planned outages and the current status of the equipment, in the 61968 the classes are used to create an Outage Record for an item of Power System Resource. This Record is made up of 0..n Outage Steps (i.e. each Outage incident creates a new Outage Step). This package contains the necessary classes and parameters required to create a complete record of equipment outages, including the ability to specify which work crew was dispatched to deal with the outage.

3.3.2.5 Additional Packages

Of the remaining packages, there are few classes that are of direct interest to representing working power system models in a CIM format. The remaining packages are there to model business processes and the exchange of information related to customers (e.g. pricing, billing and accounts), documents (e.g. work orders, trouble tickets, safety documentation) and to facilitate the integration of the Enterprise Resource Planning (ERP) standards proposed by the Open Applications Group (OAG) with the CIM. While these extensions are required for companies to implement CIM based messaging and using the CIM as their company's global information model, they are not of direct relevance to the work described in this document.

3.4 Other Proposed Extensions to the CIM

3.4.1 CIM Extensions for Electrical Distribution

As well as the continuing work by the IEC Working Groups to extend the CIM, there have been recommended extensions to CIM for representing Electrical Distribution and IEEE Radial Test Feeders published by Wang, Schulz and Neumann in the IEEE Transactions on Power Systems[8].

Wang, Schulz and Neumman focus on adding extensions to the CIM to cover the data requirements for electrical distribution power flow as well as modifying the existing classes for distribution lines, loads and introducing specific distribution devices. The majority of the work involves extending the existing CIM classes to allow it to accurately describe unbalanced multiphase connections. This is required since the CIM standard is based on exchanging data for balanced three phase transmission networks, which is sufficient for the exchange of data at the transmission level. For distribution management systems, where analysis is also conducted on unbalanced three phase, two phase or single phase systems, there is a requirement to accurately represent data for these networks.

3.4.1.1 Line Model

The paper proposes some significant changes to the Line model, including the modification of existing classes, the separation of data into separate associated classes, and the addition of new classes.

Conductor

All the attributes from the Conductor class, bar length are removed and put into a separate BalancedThreePhase subclass of the new ConductorImpedence class that has a 1..1 relationship with Conductor. The purpose of this is to allow the model to accommodate one, two and asymmetrical three phase systems, whereas the existing CIM classes are used to define a balanced three-phased system.

Conductor Impedance

As mentioned above, this class, and its child classes, SinglePhase, TwoPhase, ThreePhase and BalancedThreePhase are used to define the impedance of a

conductor. Separating the data into a separate associated class "provides the flexibility for a conductor to have an appropriate impedance representation"[7].

ConductorType

This class is omitted entirely from the modified package, with the Conductor->ConductorType->WireAttangement relationship replaced with Conductor->WireArrangement. This introduced the problem of breaking the existing CIM hierarchy by modifying the relationship of existing classes.

WireArrangment

The WireArrangement class in the standard CIM contains the mounting point information, defining the coordinates to specify the positioning of each wire on a tower. As part of the modifications to the CIM proposed by Wang, Schulz and Neumman, the mounting point information is removed from the class and put into its own separate class. The WireArrangement also has a phase attribute added, since each conductor potentially has multiple WireArrangement instances, one for each phase. This attribute is used to identify which phase each WireArrangement refers to, allowing for a single ACLineSegment object to be classed as the Line for a single circuit, but still include the information about the layout of each individual phase's cable.

WireType

Each WireArrangement instance has an associated WireType, as with the standard verion of the CIM, but the paper removes the attributes from the parent WireType, and creates child classes of WireType for OverheadConductor and Cable, with all the original WireType attributes included in the OverheadConductor class, and the Cable class (and its subsequent child classes, ConcentricNeutral and TapeShielded), include only the attributes specific to them. The only concern with this approach is that it makes the attributes too specific, ignoring that resistance, ampRating and radius are attributes common to overhead conductors and cables, and as such have a place in the parent WireType class.

3.4.1.2 VoltageRegulator

The addition of a VoltageRegulator and LineDropCompensator to the model is used to represent the voltage regulation equipment in a power system, with the proposed

classes capable of representing single, two and three phase regulators as well as the connectivity between the regulators, transformer windings and tap changers.

The paper considers a step-type voltage regulator, which "is fundamentally an auto-transformer with many taps in the series winding"[7]. The addition of the line-drop compensator allows for the automatic voltage changing with resistance and reactance attributes to describe the equivalent impendence between regulator and the load centre. The association with a tap changer object provides the information about which tap changer the compensator controls.

3.4.1.3 Load Model

As with the VoltageRegulator and Line models, the changes to the Load Model are intended to allow the representation of the individual loads on each phase, since the current CIM representation assumes the load is balanced across all phases. This is realised by introducing a DistributionLoad class which is comprised of 1..n PhaseLoads, which are defined as a being a percentage of the overall DistributionLoad.

3.4.1.4 Summary

These changes are designed to show how the current CIM standard can be extended so as to allow the modelling of lines at the distribution where it cannot be assumed that every line is balanced three phase. A significant number of the changes proposed to the Line model by Wang, Schulz and Neumman provide part of the proposed solution to the deficiencies of the CIM in relation to detailed zero sequence impedance calculations that will be discussed in section 3.5.2

3.4.2 CIM For Market Operations

An additional package for the CIM standard is under development by the IEC Working Group 16 as part of the IEC 62325, Framework for Deregulated Electricity Market Communications standard. This working group is looking at extending CIM to facilitate market operations. This involves adding classes for allowing bids and setting clearing parameters aimed at the business process messaging application of the CIM. This development is still in draft form and, since it does not affect the properties of the physical power system directly, is not of direct relevance to the work described in this document.

3.4.3 Common Graphics Exchange

There has been a request for proposals from the Control Centre Application Programming Interface (CCAPI) task force to establish a common methodology for exchanging graphical data between power system applications[9]. The format required by the working group however is aimed at the exchange of data between EMS systems, and as such is aimed at a Proprietary Format -> CIM Graphics -> Proprietary Format system, with CIM as a framework exchanging information on the symbols used along with detailed layout information. No official extensions have been proposed to solve this problem; however this, request for proposals relates to work that will be described in Section 8 of this document.

3.5 Extensions Proposed to Support the Research Work Discussed in this Thesis

It has always been acknowledged by the task force that defined the original CIM that the base classes, while providing a high level of detail, will not always meet the requirements for all power system applications and as such, the CIM would evolve over time. This is reflected in the extensions already under development as previously detailed in section 3.3.

While the CIM Base, part 301 of the IEC 61970 standard for Energy management application program interface provides a highly detailed, object-based data representation of a power network, there are several areas that were investigated as part of the program of research work detailed in this document.

Four of these possible extension areas have been identified in collaboration with National Grid (the GB System Operator), for investigation as part of this project:

- The requirement for highly detailed line information to allow the calculation of zero-sequence impedance from multiple segment values
- Modelling an Auto-transformer
- Representing fault constraints
- Defining the points of interconnection between Network Operators for the purposes of model integration.

For each of these problem areas, solutions are proposed and discussed. Minimising the changes to the existing CIM data structure, so as to preserve compliance with the IEC 61970-301 CIM 1.0 standard is a priority.

3.5.1 Requirement for Enhanced Line and Transformer Models

3.5.1.1 Enhanced Line Model

The CIM's roots in the North American EMS industry has resulted in the model being focussed on exchanging data to allow accurate load flow calculations of fully balanced three phase transmission systems. This level of detail is often insufficient for post-fault and stability analysis, which is an important issue for power engineers in the UK. The UK transmission network is far more interconnected in comparison to its North American counterpart which, combined with the reduced geographical spread of the network, results in lower impedance connections and more generating devices in relatively close proximity. This means that short-circuits in the network can result in a higher level of short-circuit current flowing to the earth point than in a less interconnected, geographical dispersed network. This means that short-circuits have a more serious impact on overall network operations in the UK than in North America and the network must be modelled to a level of detail that allows post-fault and stability analysis to be performed to a high degree of accuracy.

The UK transmission network is more interconnected than its North American counterpart in part due to the practise of installing turn-ins to other substations. This involves splitting a transmission line and redirecting it to another substation so that, for example, a double circuit on a route will consist of one continuous circuit from substation A to substation B and another circuit that is split into two by being redirected to substation C.

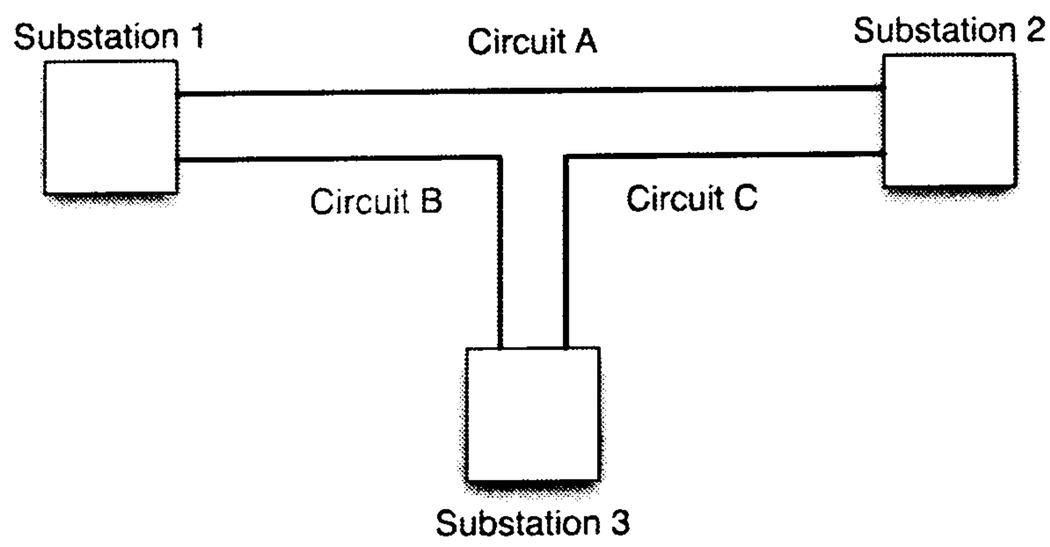


Figure 3.1 Turn-in Circuit Example

An example of a double circuit connection where one circuit is split by another substation is shown in Figure 3.1. Here Circuit A and B will be on the same tower until Circuit B is branched off to Substation 3. Similarly, Circuit C will rejoin Circuit A's transmission route and become a neighbouring circuit on the remaining transmission towers.

For load flow purposes it is sufficient to give Circuits A, B and C positive and zero sequence impedance values that are based on the average across their length. For post-fault and stability analysis, however, such values are insufficiently accurate to allow accurate results to be produced.

Using Carson's formula [13], the zero-sequence impedance of an individual span can be calculated from a wire's length, phase and shield GMR, phase and shield impedance, its height above the ground and the proximity of other conductors, along with their phase.

In a situation where Circuits B and C are energised and Circuit A is de-energised the zero sequence impedance of A will determine the zero-sequence current induced within this circuit from the magnetic fields of Circuits B and C when they are in close proximity on a transmission tower. Given that the current flow in Circuits B and C is likely to differ, by having accurate information on the zero sequence impedance of each individual span in Circuit A it is possible to accurately calculate the zero-sequence current flow in the entire circuit.

If there is a fault in Circuit B, unless Circuit A is fully grounded along its entire length the induced current in A from the section where it shares its tower with Circuit C, can affect the resulting post-fault current in B. So for accurate post-fault analysis of a fault knowing the induced current in a neighbouring circuits will allow the engineer to more accurately analyse the resulting post-fault condition of the network.

As such, there is a requirement for the CIM line model to allow the modelling of transmission lines to the level of detail that will allow zero-sequence impedance values to be calculated on a per-span basis and thus for zero-sequence currents to be calculated based on these values.

3.5.1.2 Auto-Transformer Modelling

A transformer in the CIM is modelled as having two or more windings, electrically isolated with a magnetic coupling. An Autotransformer by comparison has a single winding with fixed connection on one side and a variable connection on the other. For load flow purposes an autotransformer can be represented as a two winding transformer, but for post-fault and stability analysis, this masks the differences in electrical properties between the two systems that can impact on the stability of the network.

Autotransformers have a low initial cost and size in comparison to a double-wound transformer when the ratio of transformation is less than 2[14]. This makes them an attractive option for utilities when there is a relatively small difference in voltage level. Autotransformers, however, do have some disadvantages related to their construction.

The electrical continuity of the two windings and the fact that part of the winding is common to both sides results in the "leakage field between the primary and secondary windings [being] small and the reactance correspondingly so" [14]. Earthing the low voltage neutral point of an autotransformer also earths the high voltage neutral point since this point is common to each side and thus provides an additional low-resistance path to ground under fault conditions. This means that the autotransformer, without the installation of external protection in the form of reactors[14], is more likely to fail under external short circuit conditions with a high short-circuit current than a double-wound transformer[15].

The electrical continuity between the two voltage levels means that in three-phase transmission network, autotransformers fail to suppress harmonic currents between voltage levels unlike a double-wound transformer[14]

These issues must be considered in post-fault and stability analysis, but are not required for load-flow analysis. As such it is beneficial to have a transformer model that reflects the physical connectivity within the transformer, something currently lacking in the CIM.

3.5.2 A Line Model to allow the Calculation of Zero-sequence Impedance Values

3.5.2.1 Problem

The Common Information Model was developed in North America, where the transmission network's configuration can differ significantly from that of the UK. The differences in geography means that the UK transmission network contains significantly more complex transmission line configurations. Lines are generally shorter in the UK, and there is more interconnectivity and branching than in the North American network. As such the standard CIM representation for a Line is insufficiently detailed for the UK network operators to accurately calculate the zero-sequence impedance of a transmission line based upon its component parts. Due to this deficiency, it is proposed that the Line package be extended to include the additional detail that is currently lacking.

Currently, CIM contains a Line class, which itself is made up from multiple AC or DC Line Segments. Typically, (based on the example CIM network models used at the latest Interoperability Test), a Line is generally represented as containing no more than one or two Line Segments, often spanning several kilometres. The level of detail required by British utilies for the calculation of zero sequence impedance values, will involve each Line Segment representing the span between two towers, substantially increasing the complexity and size of the data-set required.

The issue of increasing the complexity of the Line model has been looked at previously for the distribution network as detailed in section 3.4. While the extensions proposed for distribution systems provide some of the additional complexity required, they still lack classes to describe the additional components whose parameters will be required to calculate an accurate zero-sequence impedance value for a circuit.

The extensions proposed to the Line model include some of the changes proposed for the line model in a distribution network, described in section 3.4.1.1. It was, however, felt that some of the changes proposed for distribution systems diverged from the IEC standard without providing enough added benefit, and hence the original standard is used for the base rather than the distribution system version.

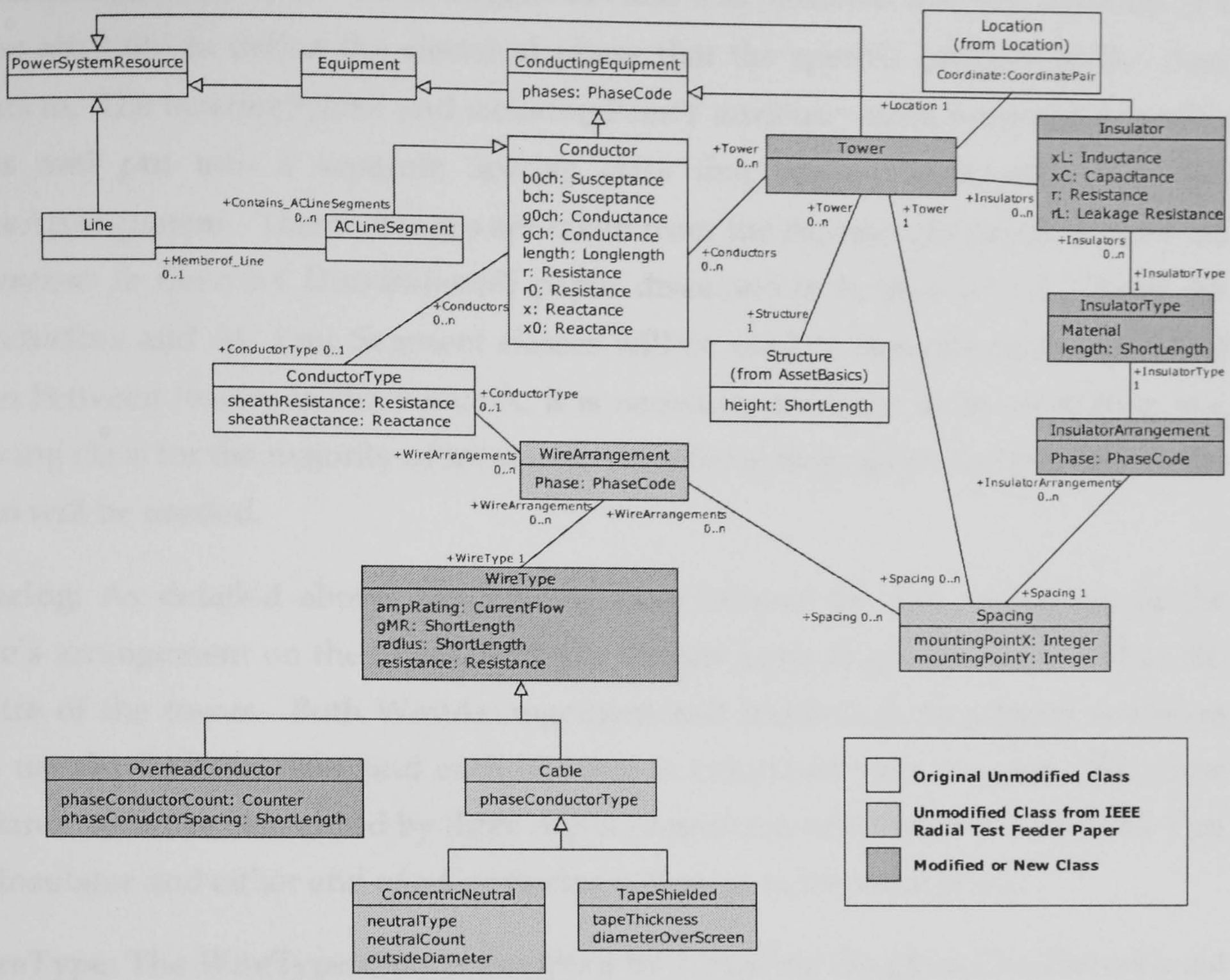


Figure 3.2 Proposed class hierarchy for an extended line model for allowing the calculation of zero sequence impedence

3.5.2.2 Proposal

The proposal is to extend and modify the Line Model of the CIM to allow the accurate representation of a line with details about the arrangement of individual phases at each tower on a line. This includes the addition of a set of classes to describe the insulators on a tower; some modifications to the existing conductor classes; and the addition of a tower class to represent a single tower on the line. A class diagram showing the proposed modifications is show in Figure 3.2. The class diagram is marked to show which original CIM classes remain unchanged, and which are new or modified. A detailed list of the changes and additions proposed is given below:

Conductor: The Conductor class remains unchanged, since it was felt that there was nothing to be gained by modifying a class that contains all the required attributes.

ConductorType: As with the Conductor class, the ConductorType class remains unchanged.

WireArrangement: The WireArrangement class was modified with the addition of a *Phase* attribute to define the electrical phase that the specific instance of the class refers to. The *moutingPointX* and *mountingPointY* attributes were removed from this class and put into a separate *Spacing* class that has an association with the WireArrangement. These changes are taken from the changes proposed in the *CIM Extensions to Electrical Distribution[8]* paper discussed in Section 3.4.1.1. Since the Conductors and AC Line Segment classes will be used to describe each individual span between towers in the network, it is necessary for there to be more than one spacing class for the majority of instances, since the arrangement at either end of the span will be needed.

Spacing: As detailed above, the Spacing class defines the X-Y coordinate of the wire's arrangement on the tower, with the Origin given at ground level and in the centre of the tower. Both WireArrangement and InsulatorArrangement instances can use the Spacing class, and each instance is associated with a tower. The same instance could be referenced by three objects simultaneously, since it is possible that an Insulator and either end of a Conductor will meet at the same point.

WireType: The WireType class is modified by removing the phaseConductorCount and phaseConductorSpacing attributes. Subclasses of WireType, *OverheadConductor* and *Cable* are created with the removed attributes reinstated in the *OverheadConductor* class, and a phaseConductorType attribute added to the *Cable* class. It was felt that the *ampRating*, *gMR* (Geometric Mean Radius. "If the conductor is replaced by a thin walled tube of radius GMR, then its reactance is identical to the reactance of the actual conductor"[1].), *radius* and *resistance* attributes would be common to all children of WireType, and as such these attributes were retained in the WireType class.

Insulator: For the proposed Line model to contain all of the data required to successfully calculate zero-sequence impedance values for a line it should contain objects to describe the insulators that hang from a *Tower* and provide electrical insulation against leakage from the lines into the *Tower* (and hence into ground). The Insulator model will follow that of the *Conductor* model, with an associated InsulatorType class and an association with a Tower, which it physically connects to.

InsulatorType: Like the corresponding ConductorType class in the Conductor model, the InsulatorType has a 0..n inheritance relationship, and as such can be associated with multiple instances of the Insulator type. This is because the class

may contain data that is constant across multiple instances, such as length and material type, as opposed to instance specific attributes such as Reluctance, Susceptance and Conductance which will likely change between insulators.

InsulatorArrangement: As with the *WireArrangement* class, this is used to define the phase associated with the Insulator instance. This class will be associated with the corresponding WireArrangement class so that the conductor that is physically connected to the Insulator can be ascertained. As with the WireArrangement class, the InsulatorArrangement will have an associated Spacing class to define the physical position of the insulator with respect to the tower.

Tower: Since the Tower class is not part of the conducting equipment on the network, it inherits from the super-class Power System Resource. It is possible that this will be deemed to be part of the Asset Package rather than Power System Resource, and as such would inherit from the Structure class rather than be associated with an instance of the class. Both systems have merit, however the draft nature of the 61968 standard, and the fact that in this model the tower itself is directly associated with pieces of network Equipment, has led to the inheritance stemming from the Power System Resource class, and associated with an instance of Asset (in this case, the Structure child class). The Tower is associated with multiple instances of the conductor and insulator classes, but due to it not actually being a part of the *Conducting Equipment*, is not part of the Topology itself and has no associated Terminals or Connectivity Nodes. The Tower will also have multiple Spacing classes associated with it that are used to define the physical spacing of wires and insulators on the tower itself.

Figure 3.3 illustrates how a small portion of a Line would now be represented as objects. The solid black lines show physical connectivity, the dashed lines showing associations. A tower has no physical connection to the electrical portion of the network, since although the Tower is physically connected; its electrical impact on the network is accounted for by the Insulator itself and as such is ignored. An AC Line Segment representing a span between tower associates with two tower instances. Below the left AC Line Segment in Figure 3.3 is expanded to demonstrate that it is composed of a single Conductor Type, which itself is made of three Wire Arrangements, one for each phase of the circuit. The corresponding decomposition of an Insulator is also shown, and one of the Spacing objects for each tower is shown, with the Wire Arrangement and Insulator Arrangement instances that share an association with it. For this example each tower would have three spacing

instances per circuit, denoting points at which Wire Arrangement and Insulator Arrangement (i.e. phases) physically connect.

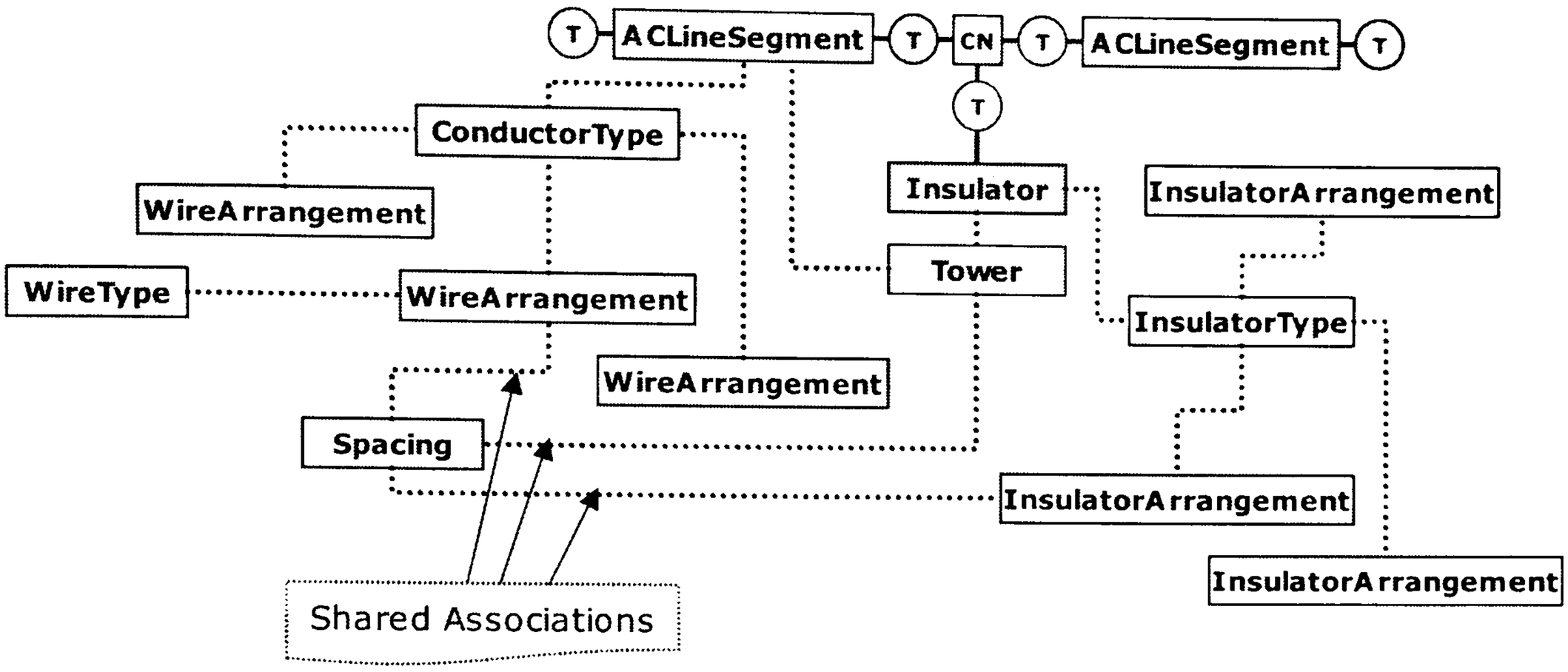


Figure 3.3 Proposed CIM Object Representation for a Section of a Line

3.5.2.3 Conclusion

This solution allows for a line to be represented down to the individual spans between towers. Using the associations between instances of the spacing, tower and wire arrangement classes, an accurate representation of the positioning of the phases of all the circuits that each tower carries can be constructed. This, combined with the data within each instance to describe the properties of the line provides sufficient detail to calculate the zero sequence impedance of an entire line.

3.5.3 Modelling an Auto-Transformer as CIM Objects

3.5.3.1 Problem

An Auto Transformer is a Power Transformer that has a physical connection between the windings, unlike a normal transformer where the windings are electrically isolated. In the CIM, the standard arrangement for a transformer is for each winding to have a single Terminal (physical connection to the network) and the windings to be contained within a Power Transformer container object.

There are several options for modelling an Auto-Transformer in the CIM using either the existing standard, or the modified standard version.

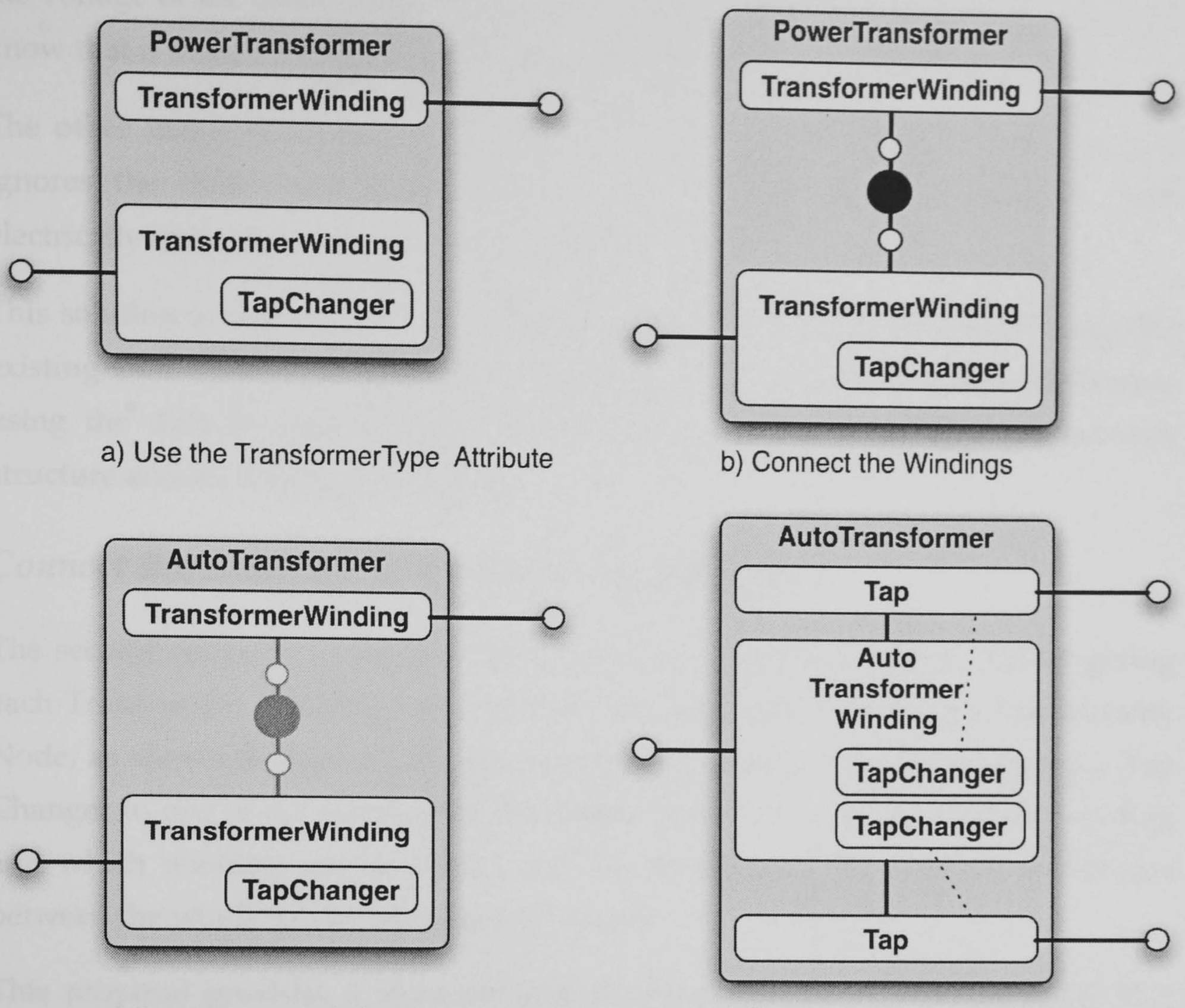


Figure 3.4a)-d) Proposals for modelling an Auto-Transformer as CIM Objects

3.5.3.2 Proposals

Use the TransformerType Attribute

The first option is to use the existing Power Transformer model as shown in Figure 3.4a) and to set the Power Transformer's transformerType attribute to Auto. This method of defining an auto transformer would require software to be written to note this change in type, and have knowledge of any special characteristics that an Auto-Transformer has over a Transformer that has no physical connection between the windings.

By assigning a Tap Changer to one of the windings it is possible to denote which is the static winding and which winding can be altered. However it should be noted that given that the two windings represent a single connected winding in reality, any changes to the voltage in one winding will be reflected by an inverse change in

the voltage of the other winding. Any software that alters the tapped voltage must know that it must subsequently alter the other winding correspondingly.

The other major problem is that by representing the transformer in this way, it ignores the differences in physical construction between a Transformer with electrically isolated windings, and an auto-transformer.

This solution is undoubtedly the simplest to implement, since it does not alter the existing CIM classes, but has a major weakness in that it relies on any application using the data to cope with the inconsistencies between the physical network structure and the CIM representation.

Connect the Windings in the Power Transformer

The second option is to connect the Transformer Windings within CIM by giving each Transformer Winding two terminals and connecting them via a Connectivity Node, as shown in Figure 3.4b). As with the proposal in 4.2.1, by assigning a Tap Changer to one of the windings it is possible to denote which is the static winding and which winding can be altered, but the problems of the changes in voltages between the windings described in 4.2.1 remain.

This proposal provides a more accurate model of the auto transformer than that described in 4.2.1 since the two inter-connected windings can be thought of as a single winding that is split into two parts. This representation, however, has the additional problem of being contrary to the normal Transformer representation in CIM where a Transformer Winding has a single Terminal associated with it. As such, this could lead to problems with third party software that uses the CIM standard approach for dealing with transformers.

Create a new Subclass of Power Transformer

The third option is to create a new subclass of Power Transformer that inherits all the attributes of the Power Transformer class, but can introduce its own parameters that define the characteristics of the Auto-Transformer. As shown in Figure 3.4c), this would allow the interconnection as given in either Figure 3.4a) or Figure 3.4b), as well as the introduction of the additional parameters required to accurately model an Auto-Transformer. It does, however, introduce the problem of deviating from the standard and introducing a custom class where the requirement for it is questionable.

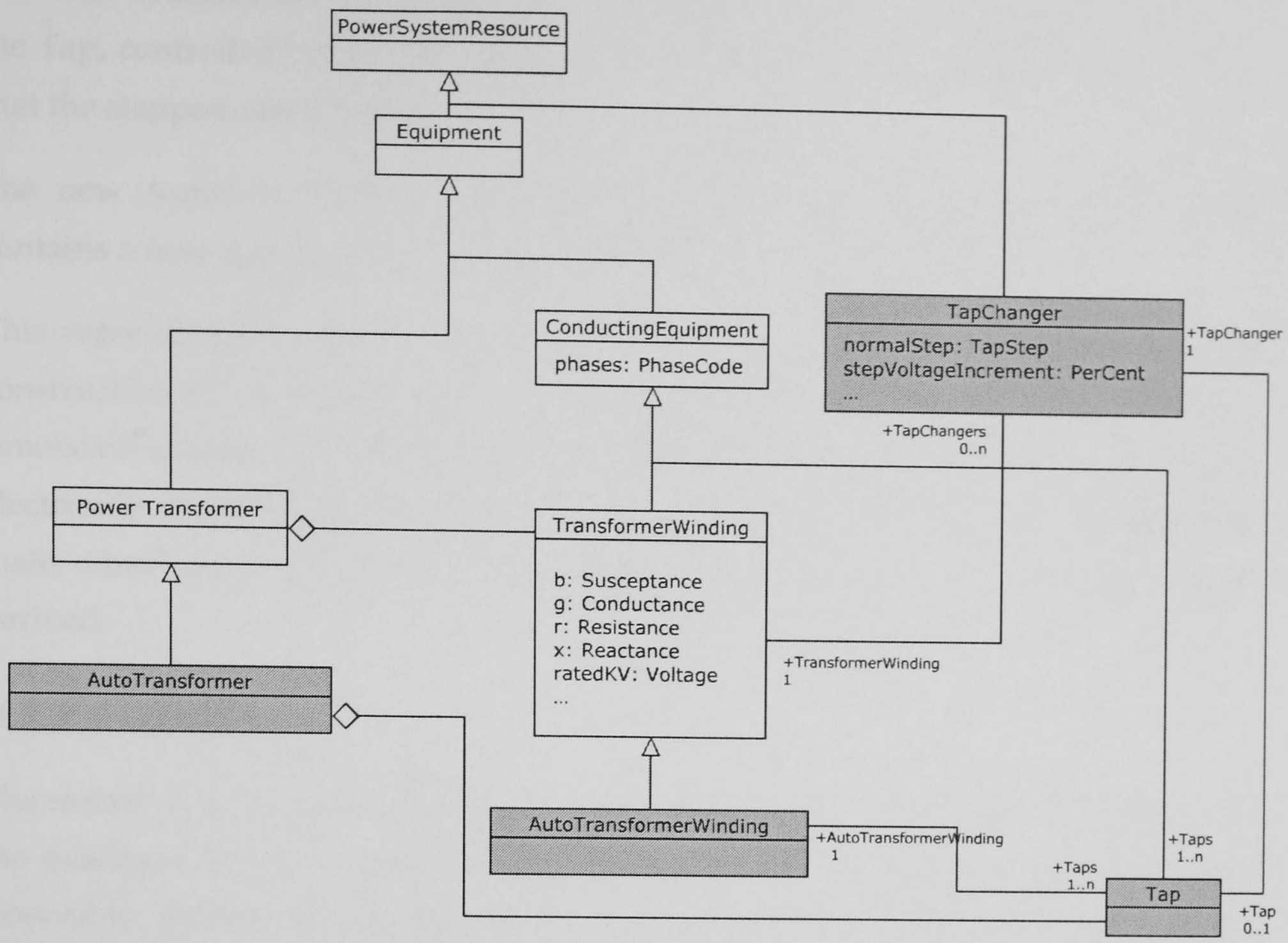


Figure 3.5 AutoTranformer, AutoTransformerWinding and Tap Class hierarchy

Create Multiple New Classes

The fourth option introduces two new classes to help represent the Transformer and offers the most extensible solution to the problem. Rather than represent the single auto-transformer winding as two connected windings, as is the case in the other proposed solutions, this proposal involves having a single winding with a single terminal and then 1..n instances of a new Tap class associated with it as shown in Figure 3.4d).

The AutoTransformerWinding class is a subclass of the existing TransformerWinding class, and is used to represent the special case of a transformer with a single winding. The main change to the TransformerWinding class is the introduction of a relationship between it and the Tap class, used to define the secondary connection to the winding to provide the step-down voltage. Since each Tap is also associated with a TapChanger the TapChanger class is altered to include a 0..1 association with a Tap.

The Tap class inherits from Conducting Equipment and is associated with a single Auto Transformer Winding and a single TapChanger. The Terminal connected to

the Auto Transformer Winding will always have the full Transformer voltage, but the Tap, controlled by the Tap Changer, is used to define the point on the winding that the stepped down voltage is taken from.

The new AutoTransformer class inherits from the PowerTransformer class and contains a new aggregation relationship with the newly formed Tap class.

This representation allows for the model to more accurately reflect the physical construction of the network and, by having a single fixed winding rather than two connected windings, it also allows for the modelling of an auto transformer with an electrically isolated tertiary winding, since the changes to a tap will not affect the main winding's voltage, from which the tertiary winding's own potential will be derived.

3.5.3.3 Conclusion

The model proposed in Figure 3.4d), while involving the most significant changes to the existing CIM transformer representation of the four options, offers the most extensible method for modelling an Auto-transformer. It closely models the physical connectivity of the transformer itself and the associations with the Taps. The use of a subclass of the existing Power Transformer class allows for the accurate representation of more complex Auto-transformers that include tertiary windings or multiple taps, allowing the model to cope with several auto-transformer configurations.

This option is the most complex to implement (ultimately requiring additional functionality to allow conversion of these new classes into their equivalent IEC 61970 CIM classes for backwards compatibility) but the benefits of accurately modelling the physical construction of the transformer are more important.

3.5.4 Representing Fault Ratings & Constraints

3.5.4.1 Problem

Resources within the power network will contain ratings which define the operating constraints of the equipment. These will include maximum and minimum ratings for a number of attributes, including current and voltage. The present version of the CIM is oriented at defining the thermal constraints of equipment, but for the CIM to meet the requirements of the UK utility companies, the ability to accurately define electrical fault constraints will be required.

3.5.4.2 Proposal

An option for including fault rating data is to include the ratings as attributes within the Equipment and ConductingEquipment classes. Extra attributes can then be added to any existing child classes where required. The problem with this approach is that it limits the number of constraints defined for each rating on a piece of equipment (e.g. Cyclic, Post Fault), and it is possible that the number of constraints specified could vary between ratings, and even between instances of the same class. It is therefore preferable to have a more adaptable solution.

It is proposed that a separate Rating class will be created, as shown in Figure 3.6, which will contain attributes to define the constraint on a piece of equipment as a separate associated object. It is possible that a fault constraint for a piece of equipment will be defined as multiple instances of the Rating class, containing both static values and curves that denote the constraint according to other parameters (e.g. current vs. time).

This RatingCurve class will inherit from the standard Curve Schedule class, but is likely to require its own parameters, such as the DC Component of a waveform, to allow accurate results to be calculated using the formula. As such a new child class of Curve Schedule is proposed to meet this requirement.

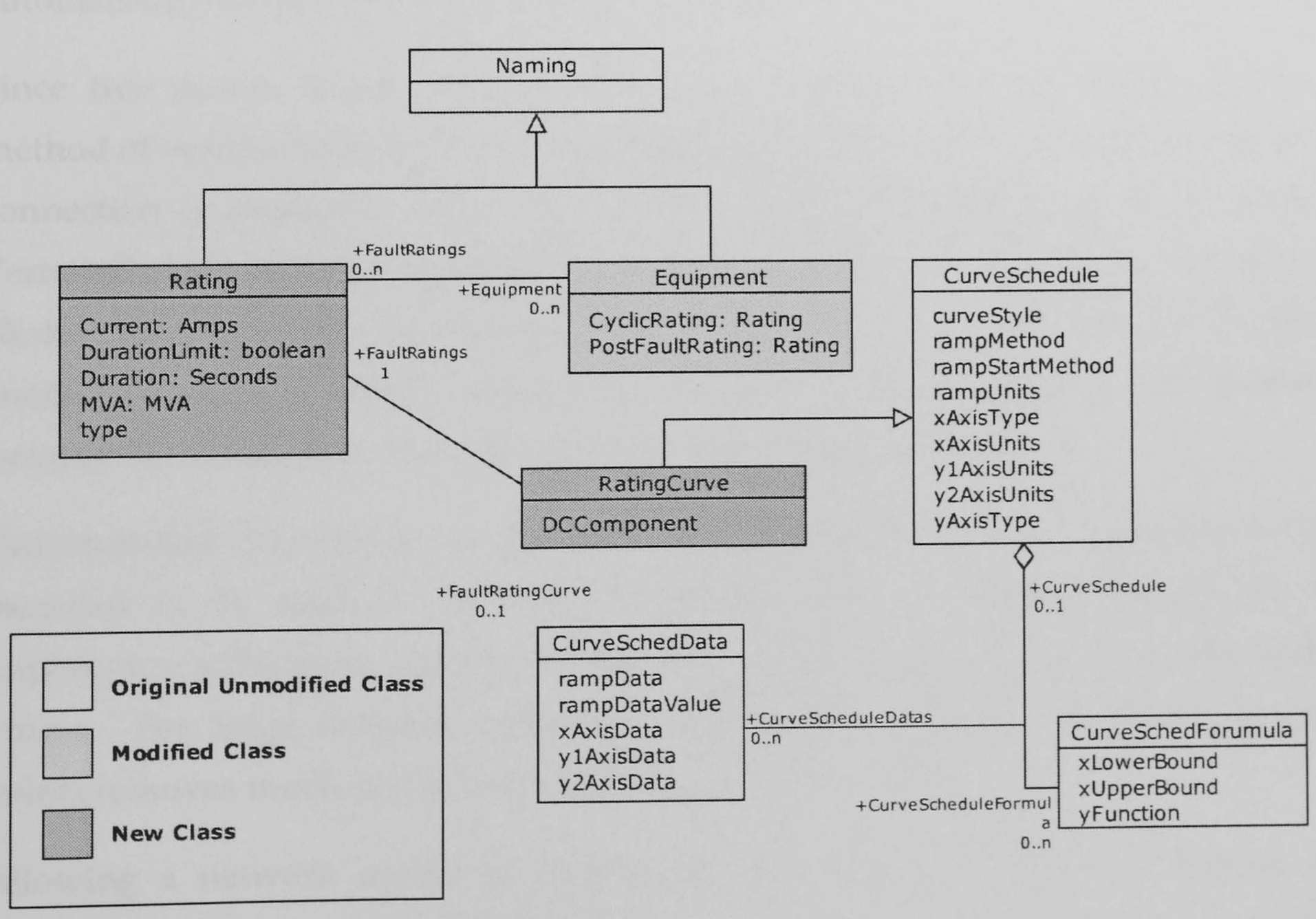


Figure 3.6 Proposed Rating class diagram

3.5.4.3 Conclusion

A 0..n relationship between the class's particular rating attribute and the Rating class allows for the Fault Constraints to be defined as a combination of specific values and formulae, thus allowing flexibility in defining the constraints of a piece of equipment, whether they be Post Fault, Cyclic or any other required Ratings.

3.5.5 Defining Network Interconnection Points

3.5.5.1 Problem

The ability to take two or more existing power system models represented in the CIM and join them together to form a single, interconnected power system model is of fundamental importance to this project. A system that would allow the automatic reception, interrogation and amalgamation of power system models from third parties with the existing transmission network model would obviously be of major benefit to network operators and to planning engineers.

When connecting two power system models together it is necessary to identify the points on each network that are to be electrically connected and from these connection points Voltage Level, Substation and Topological nodes can then be automatically combined should that be required.

Since two power system models may share multiple inter-connection points, a method of initially identifying which points of the network are intended for external connection is required. The most obvious method for doing so is to create a Terminal for a piece of equipment that does not itself connect to a Connectivity Node. This isolated Terminal can then be used to connect the network model to another network model by identifying the other network model's corresponding isolated Terminal, then creating a Connectivity Node to join the two.

Unfortunately there exists the potential for isolated Terminals to exist that are not intended to be used as network connection points. Relying purely on the importation software to correctly identify the points of interconnection could lead to errors. For large network integrations, manually defining the interconnection points removes much of the benefit of an automated system.

Allowing a network model to incorporate the ability to identify which of the isolated terminals are intended for connection to an external network instantly

removes one of the possible areas for error, and makes the job of matching connection points simpler. This will be discussed in detail in Section 7.

3.5.5.2 Proposal

Rather than define a new piece of Conducting Equipment or Topological object that would alter the core CIM classes, a new container class, Network Connection Point, is created. This extends the existing EquipmentContainer class with additional associations to a single Terminal object and Voltage Level. The Network Connection Point can also be associated with multiple pieces of Conducting Equipment, which serves two purposes:

1. It allows a network operator to store information on which pieces of equipment exist in a specific Grid Supply Point.

2. It allows for overlap of models between networks, which aids the process of automatically matching Network Connection Points where each model contains multiple points of inter-connection.

A class diagram showing the relationships that exist between the new NetworkConnectionPoint class and the existing CIM classes is shown in Figure 3.7.

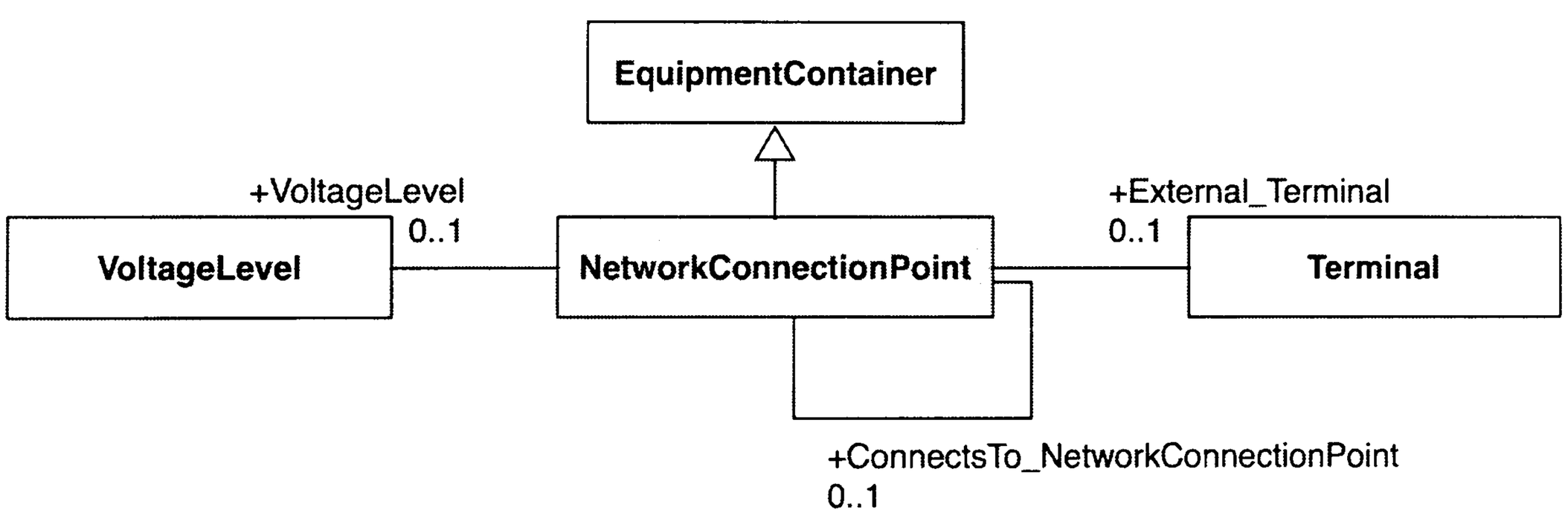


Figure 3.7 Structure of core toolkit showing interaction with external components via API

The solution adopted is to extend the existing EquipmentContainer class to form the new NetworkConnectionPoint class. The UML class diagram of this new class is shown in Figure 3.7, with two additional associations: ConnectsTo_NetworkConnectionPoint and External_Terminal. This allows the Network Connection Point to be defined in two ways depending on the network it represents:

1. As the provider of a connection, where the connection point is defined by an association with a single Terminal using the new External_Terminal association

2. As the receiver of a connection, where the connection point is defined by an association with a Connectivity Node using EquipmentContainer's existing ConnectivityNodes association.

When combining two networks, each pair of Network Connection Points, one from each network, should contain at least one External_Terminal and one ConnectivityNode. If a Network Connection Point contains multiple External_Terminal and/or ConnectivityNode associations or multiple Network Connection Points then the process must identify the best match or matches. The External Terminal and Connectivity Node, being virtual network components (i.e. they do not represent real pieces of physical network equipment), are used to represent the point at which the two networks are connected in the model. When both networks are fully connected, each External Terminal will associate with a Connectivity Node from another Network Connection Point and vice-versa. The ConnectsTo_NetworkConnectionPoint association provides a link between a Network Connection Point instance from each model.

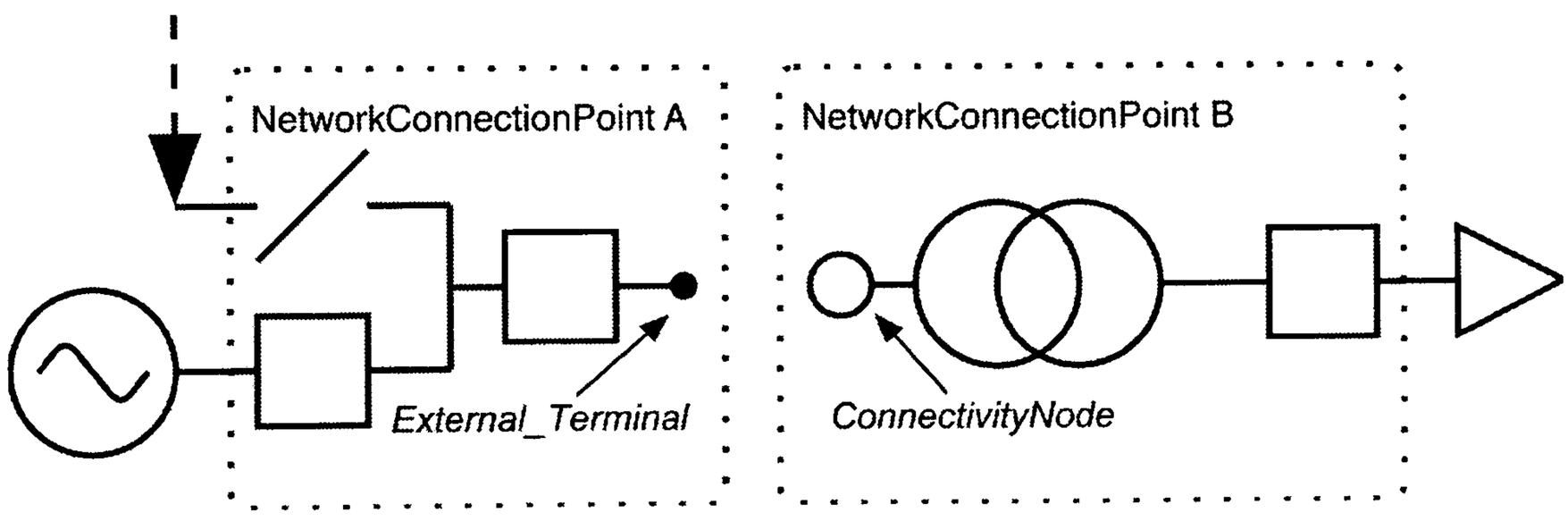


Figure 3.8 Illustration of a network connection using Network Interconnection Points

A simple example of a Network Connection Point is shown in Figure 3.8. Network Connection Point (NCP) A is the provider of the connection and contains a single External Terminal. NCP B is the receiver and contains a single Connectivity Node. Upon integration the Connectivity Node and External Terminal would become connected, thus creating a direct link between the Transformer and Breaker and joining the two network models.

3.5.5.3 Conclusion

The benefit of this additional Network Connection Point class is that, as a container, it can be excluded from the network model when exporting as IEC 61790-301 compliant CIM without affecting the network's structure or topology. Software to import and connect models can be written to function without the classes, but by including the data the process is made significantly more straightforward with the potential for fewer errors.

3.6 Backward Compatibility Issues

As discussed in Section 3.2, the Strathclyde and IEC CIM standards are able to co-exist, with XML files and software applications capable of storing data from multiple CIM standards concurrently. Since applications that import XML data can ignore nodes from unrecognised namespaces, the *stcim* prefixed nodes will simply be ignored by third party applications, which is acceptable when the classes do not directly impact upon the network layout or electrical characteristics of the model. For example, the Network Connection Point, does not represent a piece of physical equipment, and as such its inclusion is not required to create a valid connected network.

A problem occurs, however, in the case of the modified Line Model and Auto Transformer classes. If omitted from a model, the network connectivity would be incomplete since essential pieces of electrical equipment would be missing because they are from the Strathclyde standard and not the CIM standard.

For any classes such as these, it must be possible to convert them into existing CIM classes in such a way that although the higher level of detail will be lost, the essential connectivity and basic parameters are maintained.

3.6.1 Areas of Concern

The following sub-sections describe how backwards compatibility can be achieved with the proposed CIM extensions.

3.6.1.1 Line Model

Translation from the modified Strathclyde standard classes back into the original IEC standard CIM classes is required. Equipment attributes that have been moved into Strathclyde classes (such as Spacing) can be copied into the original class with

any new subclasses cast back into the original parent class from the IEC standard. The additional tower class and those that are used to model the arrangement of insulators are not essential parts of the network model, and their omission removes a level of detail but does not affect the model's connectivity.

3.6.1.2 Auto Transformers

For an Auto Transformer, the Tap, AutoTransformerWinding and AutoTransformer classes must be converted into the original TransformerWinding and PowerTransformer classes. For the AutoTransformerWinding this should be a simple case of converting it back to the TransformerWinding class, and in the process removing the associations to any Taps that are connected to it. For the Tap class the creation of a new TransformerWinding instance is required. This involves the calculation of appropriate values for each attribute in the Two-Winding Power Transformer representation and calculating electrical parameter values as close to the Single Winding-Tap Auto Transformer representation as possible. The AutoTransformer must similarly cast itself to its parent PowerTransformer class and then contain the newly created TransformerWinding classes.

3.6.1.3 Other Extensions

The other additional modifications to CIM detailed all involve the addition of classes that provide extra detail to the model, and any modifications to existing classes were to allow the addition of these extra associations. As such, the modified classes can be cast back to their equivalent classes in the original IEC specification CIM, removing the associations to the new classes in the process. Any instance of the new classes can now be ignored without breaking object associations.

3.6.2 Implementation of Backwards Compatibility

Exportation methods (to move from a CIM+Strathclyde model to a CIM only model) will be built into the classes themselves to allow CIM based applications to be extensible. Backward compatibility can be maintained by setting down stringent requirements on any new classes that modify any core CIM classes or any classes directly related to the representation of network topology. If all of these new CIM classes are capable of exporting themselves as a valid representation in IEC CIM (whether it be as a single or multiple objects), backwards compatibility, albeit at a reduced level of detail, can be maintained.

This approach is required to ensure that data in any new CIM applications is compatible with the existing third party software applications that are capable of importing and modifying only IEC CIM data in XML format. This approach will be embedded in the core application software being developed as part of this current project, rather than being part of the data standard itself.

3.7 Chapter Summary

This chapter has summarised a number of extensions to the CIM and their relevance to the work discussed in this thesis. The new extensions proposed to support the adoption of the CIM within the UK power community highlight some of the deficiencies of the current CIM standard, while illustrating how the standard can be extended to cover these gaps without breaking backwards compatibility. The final proposed extension defines a way of marking components in the network that can be used as points of connection. This has two main applications: to facilitate the automatic integration of operational models by explicitly defining the points in a network that connect to neighbouring network; to allow planning engineers to define points in a network that are suitable for connection by existing or prospective connection partners. The automatic integration of models, whether for operational or planning purposes, will be discussed further in Chapter 7.

4 Exchange & Storage of CIM Power System Models

4.1 Chapter Introduction

As mentioned previously in Section 2.4.2, the CIM defines each component of a power network as a separate class and how these classes relate to each other. The model itself does not describe functionality, only the associations for a class, the data it contains and the format of the data.

This approach, of creating a generic model of power network components allows the model to be translated into classes for exploration and manipulation within an object oriented programming language application. Languages such as Java, C++ and C# are object oriented, and the CIM structure can be used to create corresponding classes in any of these, or other object oriented languages. Each CIM class becomes a corresponding class in the target language, with the corresponding attributes and inheritances.

This chapter will describe how the CIM can be used to build a software framework in Java, thus providing a powerful mechanism for interrogating and processing power system network models in a CIM format and highlights the originality of using the CIM as the foundation of a software tool.

4.2 Power System Analysis Software Design Methodologies

Much of the power system simulation software currently in use within large utility and consultancy organisations is based on procedural programming languages, many of which date back to the late 1970s and early 1980s, when processing power and memory capacities were a tiny fraction of what is available today. Since then, programming languages have evolved, moving on from procedural, functional programming, to an object oriented design. This technique involves creating programs as instances of modules or classes, so that a program is split into a number of small, self-contained systems that are adaptable and reusable.

As mentioned, CIM is ideal as the basic framework of such object oriented software, since it defines all the basic components of the power network as objects that can be

quickly converted to classes in an appropriate object oriented programming language application. All the different components of a network can be instantiated in the computer's main memory simultaneously, creating a dynamic computer representation of the network.

With a CIM implementation of a power system, objects can communicate directly with each other, reducing the requirement for a single, centralised program or procedure to perform all the processing of data. Multithreaded software, where more than one process can run concurrently, based on an object oriented CIM network would allow a distributed simulation system that integrates the processing and storage of data.

The use of the CIM for the basis of the underlying architecture instead of a custom-designed solution, as has been used in other object-oriented power system applications [16], is to allow the framework to cope with extensions and modifications to the CIM standard without requiring a complete redesign of the software. The CIM defines the relationships between classes, and by defining rules for creating methods within each class for setting and getting the attributes or associations. The Application Programming Interface (API) for the software follows a pattern based on the UML standard, prefixing an attribute name of 0..1 multiplicity with *get* and *set* or *add* and *remove* for 0..n multiplicities. This is a common practice for object-oriented software development, and allows for simple creation of Java classes from a UML design, and as such allows the software to integrate extensions to the CIM standard with little modification. A custom designed class hierarchy is unlikely to integrate extra CIM classes without significant redesign of the class structure.

Many existing power systems analysis software packages and Energy Management Systems can import and export data in the CIM format, encapsulated in XML. However, they convert this data into or from their own internal data structure and as such, their ability to cope with extended formats or modifications to the standard is obviously limited by how the CIM maps to their own data structure.

4.3 Power Systems Toolkit Design

The CIM objects form the core data storage system for the power systems toolkit proposed in this chapter. Access to the data is via the core toolkit module's API, which maintains the integrity of the data. Additional modules can be attached to this main module, allowing the import, export and modification of the data.

The primary motivation for the design of this system is to allow it to operate as a remote application, providing multiple users access to the same data and tools concurrently. This system creates problems concerning the synchronization of data between multiple, concurrent user sessions. A remote system of this type must prevent multiple sessions from creating multiple instances of the original data if any changes are to be integrated back into the original data during runtime. These issues will be addressed further in section 4.4.

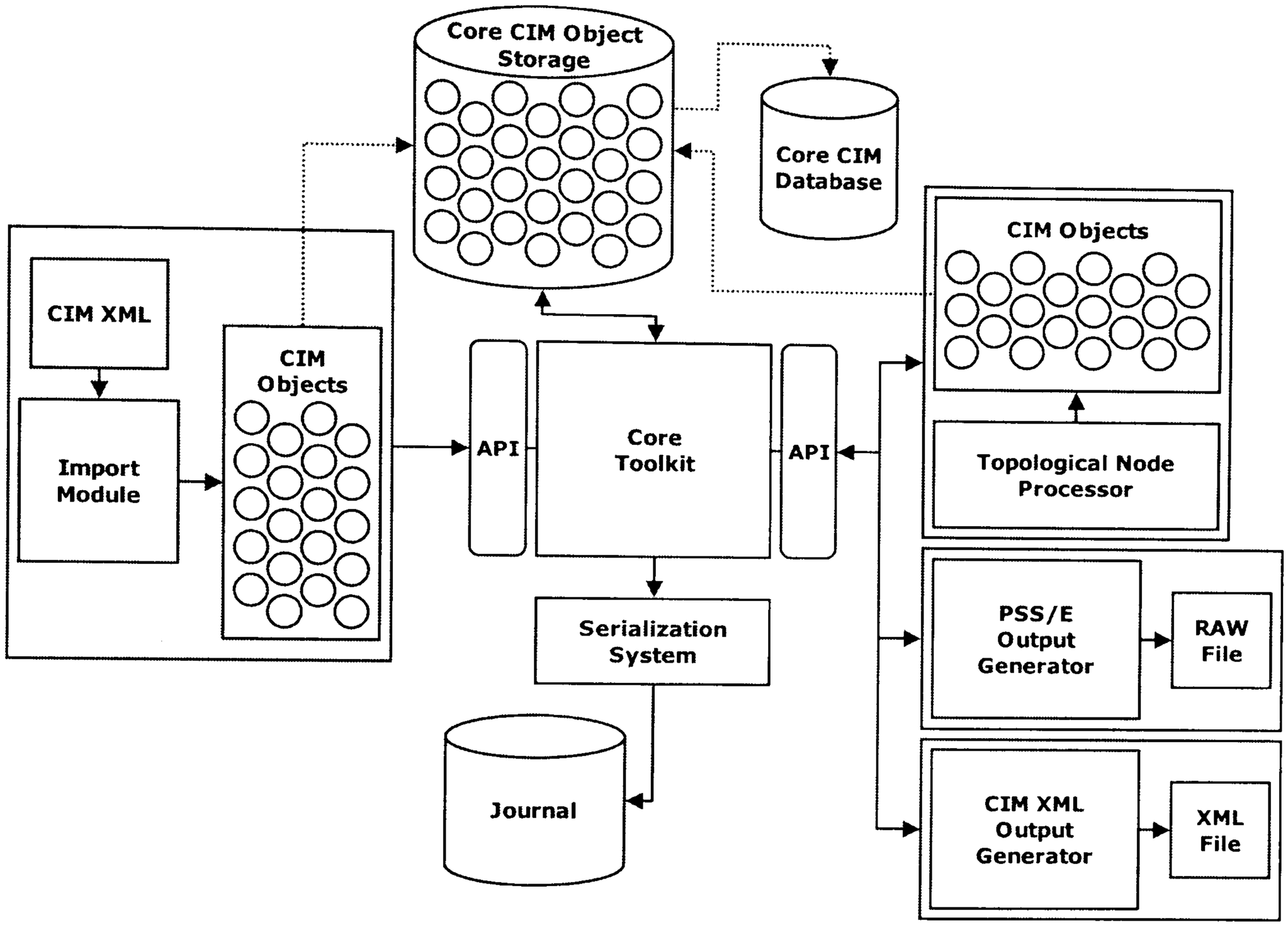


Figure 4.1 Structure of toolkit showing interaction with external component via API

Figure 4.1 shows the structure of the toolkit (the import module, core toolkit and its associated serialization module and object storage system) and three additional modules that utilise the API. The export modules (PSS/E in this case but could be any other power systems analysis application) use the API to read the core data then process it into the required output format. The Topological Processor uses the existing CIM objects to create associated Topological Node objects, which are inserted into the core data storage system, via the API.

The import module has access to the CIM class definitions, and must be able to determine which classes are required, and how many instances of each class will be required to successfully parse all the data from the source CIM XML file. The

module will then work its way through the imported file, creating instances of one or more CIM classes for each component of the network described in the source file, until the whole network has been instantiated as interconnected CIM objects.

This CIM model is then passed to the core toolkit, via the API, which integrates these objects into its core object storage system. The additional modules can then access the data through the core module's API. This way the data can be interrogated, modified or exported, while the core module maintains the integrity and tracks modifications to the data.

Accessing the data through the core model will allow the implementation of the serialization based journaling system described in section 4.4.5, since the data itself is not made available natively to the attached modules, but is accessed through the core API. This records, and serializes, all commands performed that modify the core data, as well as verifying the integrity of the attached modules, to prevent unauthorised access to the sensitive network data.

4.4 Challenges of Implementing a CIM Based Power System Toolkit

Several challenges were overcome in creating a CIM-based power systems toolkit in Java. These included: creating a Java implementation of CIM; choosing a system of data storage for the toolkit which would be both fast, flexible and allow concurrent access from multiple sources; creating a simple method of importing data from a CIM XML file into the toolkit itself; and ensuring that the software system was reliable.

4.4.1 Implementation of CIM classes in Java

An implementation of CIM in the Java programming language was undertaken to demonstrate the expected advantages of using objects to store network data. Since Java is a fully object oriented language, the UML [17] version of CIM already publicly available on the CIM User Group website[10] could be used to automatically create the base Java classes. There are tools available, both free-standing and within existing commercial design packages to translate the language-independent UML specification into a number of programming languages, and create the appropriate base functions for inserting and retrieving values from each class.

The benefit to using the CIM for the software's internal architecture rather than a custom solution is that with major power systems software vendors such as ABB, Areva and Siemens implementing CIM import and export functionality into their software, storing the data internally in the CIM format removes one extra level of translation when exchanging data between applications. The ability to cope with extensions to the CIM is also easier when the internal data storage architecture mirrors that of the standard. Any additions and modifications to the standard requires only a corresponding change to the internal data storage architecture, removing the problems of mapping a proprietary internal architecture to changes in the CIM.

When converting the CIM UML classes into Java code, functions are automatically created for adding, modifying and retrieving data in the format of *get[Attribute]()* and *set[Attribute]()* where *[Attribute]* is replaced with the name of the attribute, or *get[Associations](), add[Association]()* and *remove[Association]()* where similarly *[Association]* is replaced with the name of the association. This allows the classes to follow the standard, but can also be easily extended to create custom classes with additional custom functionality. These functions allow a single function call to perform multiple data modifications on the object, and any other associated objects. This combines the data storage and manipulation into one single entity which, while more complicated than a file or database system, provides a more comprehensive API and enables the actual software applications using the data to be simplified.

Java's performance, and thus suitability, for computationally intensive applications can be measured using the Math, Statistics, and Computational Science Division of the National Institute of Standards and Technology's SciMark2 benchmark suite[18]. This benchmark is used to measure the performance of computer systems using a series of computational kernels: a Fast Fourier Transform, Jacobi Successive Over-relaxation, Monte Carlo integration, Sparse Matrix Multiply and Dense LU Matrix Factorization. The benchmark code is available in both C and Java and returns values in MFlops (Million Floating Point Operations) per second for each kernel and a composite score.

The two matrix based benchmarks offer the most relevant comparison for this application, since the extensive use of arrays in each kernel allows the access times of the two systems for storing arrays in each language: object-references in Java and pointers in C, to be compared as well as the numerical performance. Benchmarks run recently on a Pentium 4 system using both the Java and C versions found that

while the C code was marginally faster overall, producing a composite score of 384MFlops compared with 361MFlops for the Java 1.4.2 version, this is a drop of only 6%. The full benchmark results can be found at [19].

The advantages of using Java all outweigh what is now only a small performance disadvantage over natively compiled C or C++ code: its cross-platform compatibility; the ability to easily use the same framework for graphical, command-line or server based applications; the inbuilt libraries for constructing distributed applications; its intrinsic security features; and the integrated multithreading capabilities.

4.4.2 Advantages of Storing A Power System Model as Objects

The traditional approach to storing large quantities of data for concurrent access from multiple sources is to use a database system for storing, searching and retrieval of the data. Such databases required the use of the database interface, which, while powerful for complex searching, has significant disadvantages when performing traversals of the power system network layout, like those required for converting node-breaker data in bus-branch format, at a higher level of abstraction.

Using objects written directly in the chosen object-oriented language for persistent storage offers several benefits:

- Far greater flexibility for access and manipulation of data.
- The interface is written in the native language and is fully customisable, so data manipulation and access can be fully integrated into the data storage medium.
- A standard data format can be maintained, while providing a powerful, adaptable interface for accessing the data.

The use of packages and inheritance allows for the core classes to be extended, but compatibility with previous versions to be maintained by using separate packages for each version of the CIM and the use of inheritance between packages allows backwards compatibility to be maintained where required.

One of the main functions of the toolkit is to automatically create Topological Node objects, analogous to a Bus in bus-branch format, from the more detailed node-breaker format of CIM. This process will be discussed in detail in Section 5.4, along

with the implemented algorithm. However, for now it is enough to consider the conversion as a full traversal of the network, accessing every object at least once. For comparison, the algorithm was implemented using an object-storage system for the CIM data, then using a MySQL database to store identical data.

This choice of comparison is due to the three available choices for a multi-access system: Using a database to store the data and allow it to be accessed concurrently by each session as data is required; use a persistent object-storage system for common data storage; use a database for data storage then instantiate an independent set of objects based on the database data for each user session. The latter option creates the problem of synchronizing data between multiple sessions, but for the purposes of benchmarking, operates on the same principle as the object-storage option.

With the object-storage option, the references to any connected objects are contained within the object as a direct reference. With a database, the field that refers to another entry in the database uses a foreign key to locate the entry for the other object and extract the appropriate information.

Assuming the database is fully indexed, containing data for n objects with each object having a single row and unique numeric identifier in the primary table, then a single step in a network traversal would occur in $O(log\ n)$[20]. This is because a standard binary search of an ordered index would takes place on average in $log\ n$, for a fully ordered index of n items. If the traversal takes m steps, then the database will take $O(mlog\ n)$ to complete the traversal.

For a persistent object-storage system, a single step of the traversal will take O(1), since objects contain direct references to other associated objects, and no searching is required. A full network traversal therefore takes place in m steps, or $O(m)$.

It can therefore be concluded that for a network of size n, the time taken for a complete network traversal with an object persistent storage system will increase linearly as the number of steps in the traversal m increases, but is independent of the network size n. Using a database system for data storage and retrieval however, shows a growth that is linearithmic, since the execution time is a product of the log of the network size and the average number of steps for a traversal.

4.4.3 Memory Storage Requirements for an Object-Based System

An object prevalent system stores all the data in memory. To compare the memory requirements when instantiating a power systems model as CIM objects in Java, the toolkit instantiated an increasing number of power system models in memory, and used Java's Runtime.totalMemory() to measure the memory used by the Java Virtual Machine.

The system was tested with a number of CIM models of varying complexity taken from the industry's CIM Interoperability tests. These included test models of varying sizes from ABB, Areva, Siemens, EDF and the Western Area Power Administration (WAPA). By having the toolkit instantiate multiple instances of these models in varying combinations, the amount of memory used to store networks of increasing size can be measured.

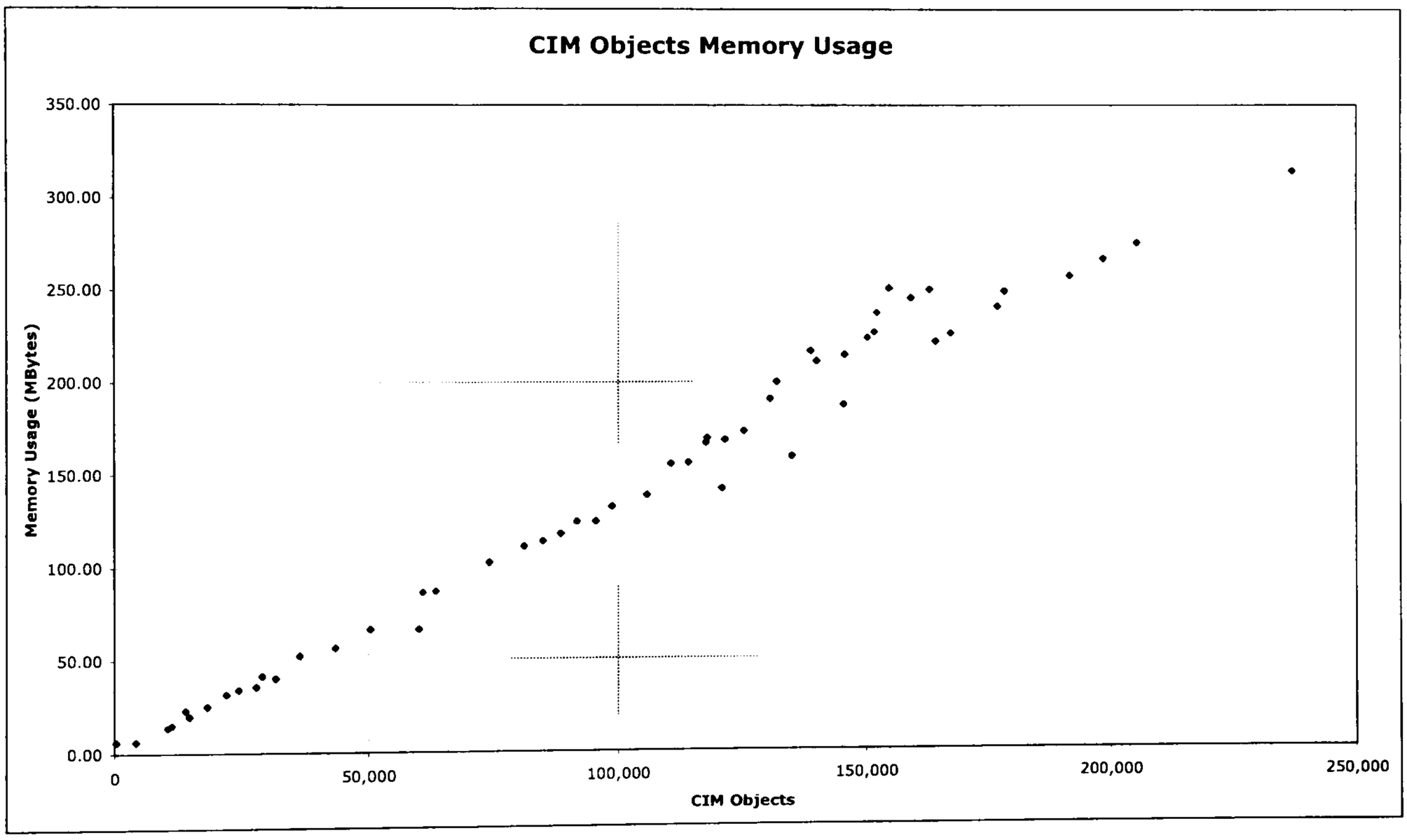


Figure 4.2 CIM Objects Memory Usage, 0 to 236,000 objects

On a computer with 4.5Gb of Physical memory, the memory usage follows a linear growth pattern. Figure 4.2 shows a graph of the total number of CIM objects plotted against the memory used to instantiate them in memory. The outliers in the graph are due to the variance in complexities between the difference models. One CIM object may contain only one or two attributes and associations, while another CIM object may contains tens or even hundreds of attributes and associations. As such,

the complexity of the CIM power system model will affect the memory used for each object.

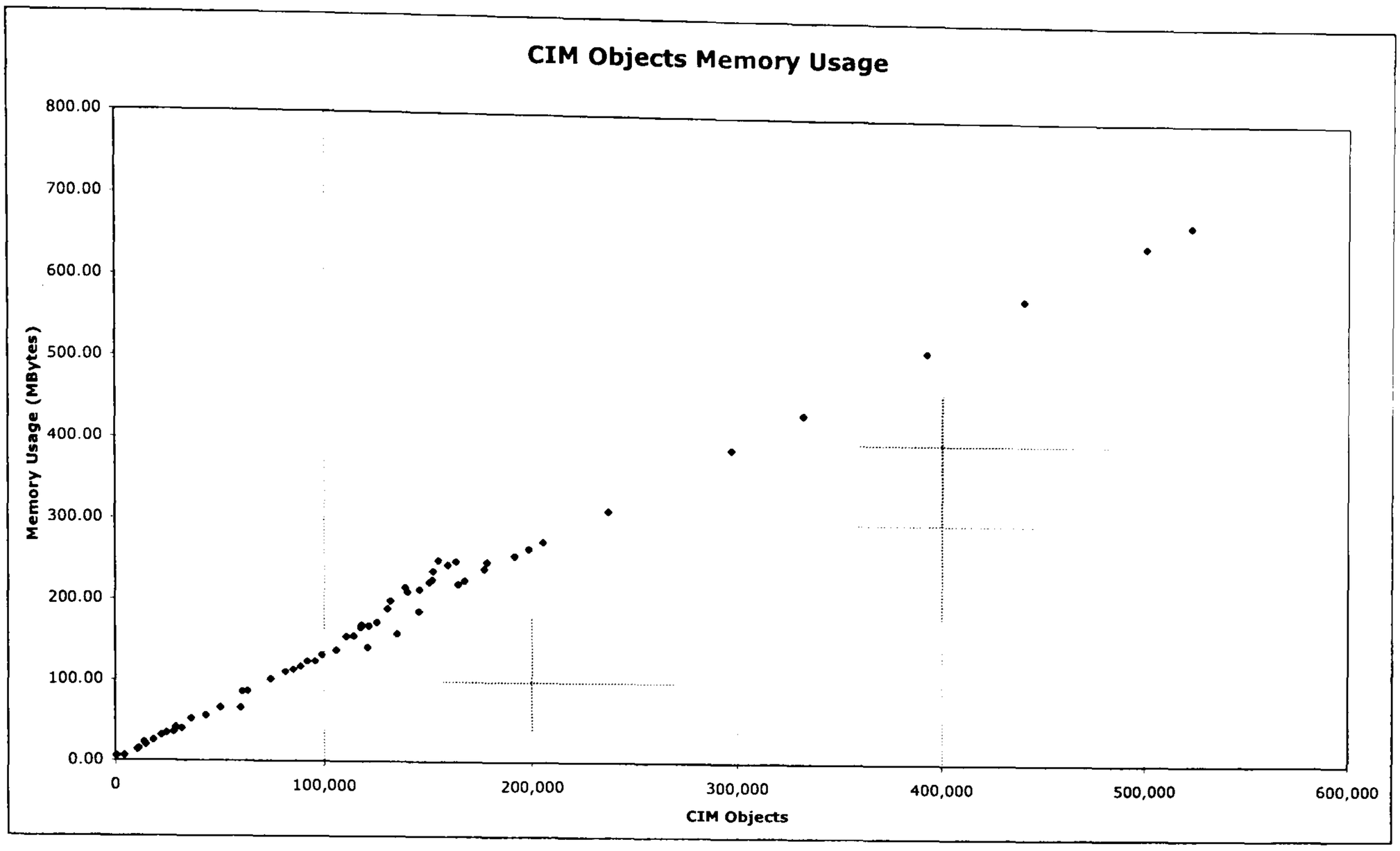


Figure 4.3 CIM Objects Memory Usage, 0 to 522,000 objects

The increase in memory usage does, however, show a linear growth in memory usage as the number of instantiated objects increases. By using multiple instances of large, real-world models, the number of instantiated objects can be increased by tens of thousands of objects each time. Figure 4.3 shows the original graph extended to over half a million objects using almost 700Mbytes of memory.

On average a CIM Object uses 1.3Kbytes of memory allowing the system to continue to store up to 1.7 million CIM objects, at which the 32bit Java Virtual Machine (the environment in which Java programs run on a computer) memory address limit of 2.2Gbytes is reached. Beyond this, Java 1.5 supports 64 bit memory addressing on compatible operating systems, and with modern 64 bit workstations and servers available at prices less than $3000, the system has the potential to scale up to tens or hundreds of millions of objects. Beyond this limit, the system could be split across multiple computers, operating as an interconnected cluster.

An alternative is to implement persistent CIM Java objects using a database with JDBC (Java Database Connectivity), which stores each object in a serialized form within a database. The problems with such a system have been discussed previously[21], where it was noted that the complexity of mapping Java objects into

tables and back increases dramatically when multiple levels of inheritance, as contained in CIM, are included.

The other system used in [21], was an ODMG (Object Database Management Group) Java binding, which has since been superseded by Java Data Objects (JDO). This is a system for storing Java objects in a database with transparent object mapping to database tables. While this approach does offer advantages over JDBC since it is simpler to implement, it still requires a database system to function, which removes much of the speed advantage of an object prevalent storage system since each object is loaded from the database.

A database storage system does offer advantages when performing complex searches across multiple object types. For example, locating all pieces of equipment of a specific voltage level would require a single database query in the database implemented previously, but to implement the same search in an object-storage system would require functionality to search through every instance of each equipment type to find any matches.

The advantages provided by the superior search capabilities of a well-designed database make the combination of an object-store with an associated read-only database the most attractive option. Making the database read-only and making any changes to data within the object-store be mirrored automatically on the database by each object removes the problem of synchronising data between the two. This system provides the advantages of complex database searches while maintaining the benefits in speed for complex data transformation provided by an object-storage system.

4.4.4 Importing CIM XML Power System Data into Java Objects

As has been mentioned previously, CIM provides a framework for creating an object based representation of a power system. Rather than interpreting CIM XML data directly, importing this data as a series of CIM objects allows greater flexibility in accessing and manipulating the data. Of course, the construction of an export module for straight translation into other, proprietary power system modelling data formats is also possible.

The model is instantiated using a generic import module that can cope with any XML components within the document that validate against the defined schemas,

and for which a corresponding class file exists. Rather than using a large compare statement to locate the corresponding class, the name of the XML components can be used to create a class file of the same name using Java's Reflection functionality. This allows the available functions of a class to be found dynamically by the program when running. As previously described in section 4.4.1, if a class is constructed so that for a variable *X* the corresponding set and retrieval methods are *setX* and *getX* then it is possible to propagate the object based on the data retrieved from the XML component.

In this way small pieces of code can be reused for importing any CIM XML node, since the import module is generic and non-class specific. As each XML component is read, the name of it is used to create an object whose class has an identical name to that of the XML component. Now the attributes in the XML component can be added into the object by using the Reflection API to locate a *set* method that matches the name of the attribute.

For example, the code below shows an excerpt from a CIM XML file containing the *g, r, x, ratedKVA* and *ratedMVA* values for a Transformer Winding, TW_1A.

```
<cim:TransformerWinding rdf:ID="TW_1A">
   <cim:TransformerWinding.g>0.04</cim:TransformerWinding.g>
   <cim:TransformerWinding.r>0.07</cim:TransformerWinding.r>
   <cim:TransformerWinding.x>0.47</cim:TransformerWinding.x>
   <cim:TransformerWinding.ratedKV>400</cim:TransformerWinding.ratedK
V>
   <cim:TransformerWinding.ratedMVA>164</cim:TransformerWinding.rated
MVA>
</cim:TransformerWinding>
```

When this data is imported into the toolkit, the import module selects the corresponding Java class, *TransformerWinding* by using the name of the XML node, *cim:TransformerWinding,* locating the Package in the CIM that contains the *TransformerWinding* class then using Java's reflection functionality to create a new instance of that class. Given that the Java class structure and XML schema are created from the same UML model, a CIM XML file that validates against the schema during the initial stage of importation, will in turn map to one of the class files, preventing the software from trying to import invalid XML data.

The Java class contains attributes that correspond to the values in the XML nodes: variables named *g, r, x, ratedKVA* and *ratedMVA,* and also contains the functions *setG, getG, setR, getR, setX, getX, setRatedKVA, getRatedKVA, setRatedMVA* and *getRatedMVA* which are used to set or retrieve the value of the corresponding variable.

The data will initially be in a String format since an XML file is plain text, but this can be converted to the appropriate data type within the class itself during the set method, keeping the import module simple and generic. For example the string "34.2" can be converted to the floating-point value *34.2* by the *set* method, since the variable in the class will be a floating-point value type rather than a string.

For object associations (where the attribute of the XML node refers to another node), it is necessary to initially store these associations as the unique String identifier for the other component. As each node is read in, an index is updated with the identifier of an object, and a reference to the object itself. Then, once the file has been fully imported and all the objects instantiated, the references in each object can be converted from a text identifier to an object association using the previously created index.

This import system results in the saved network model being instantiated as CIM Java objects, and thus any export or processing module can access and modify the data through each object's API. The associations and interconnections between the objects simplify the process of modifying multiple interconnected objects and provide a powerful API, capable of much more than simply setting or retrieving the data. A single command could result in changes to data rippling through all the associated objects automatically as functions in one object can call additional functions to modify data accordingly in associated objects.

4.4.5 Use of Serialization to Track Model/Data Changes for Security

One of the major disadvantages of storing the model in memory is that in the event of a system crash, the entire model is lost, since memory is usually volatile. This drawback, however, can be overcome, without significantly reducing the advantage provided by object prevalent data storage.

Java, and other common object oriented languages (C++, C#, CORBA), have support for serialization (saving the state of an object into a file). It is beyond the scope of this thesis to explain the intricacies of serialization, however, the basic process involves converting the current state of an object as an encoded stream of bytes, which can then either be saved to a file, or transmitted. This byte stream contains all the data required to reconstruct the original object.

Serialization can also be applied to the commands executed on the model, so a record of these commands, combined with a regular serialization and storing of the model's state at a given point in time, allows for the full restoration of the model's current state following a system crash.

To save the state of the model during a controlled shutdown of the system however, the model's state can be exported as a valid CIM XML file. This allows any upgraded version of the system containing a newer version of the CIM class hierarchy, with additional or modified classes to recreate the full network from the CIM XML file.

4.5 Extending CIM

As was discussed in section 3, the CIM, whilst comprehensive in many areas, lacks the detail required for specific areas of power system engineering. The object-based design of CIM allows for enhancements and modifications to the current standard, but it can be easily recognised that ad-hoc modifications to a standard are undesirable in the majority of cases. Open standards, such as CIM, are adopted to aid the process of exchanging data, and, so data in the standard format and structure (or a future revision), immediately becomes incompatible if the standards used are subsequently modified.

Using an object prevalent data storage system, however, can provide a compromise, allowing enhancements and extensions to CIM data, but still maintain the capability to output CIM 1.0 compliant data. Extensions to the CIM are defined in UML, and by auto-generating the appropriate class files from this UML file, these new classes can be easily integrated into the existing data storage architecture without requiring modifications to the existing importation module.

The most basic way of adding additional data is to add extra attributes to each object in addition to the standard CIM attributes. This allows for more data to be stored in each object, whilst maintaining the ability to export the standard data without introducing incompatibilities.

To extend the level of detail that can be stored in object format, it may be advantageous to have child classes for existing classes. For example, a Line object is currently made up of one or more Alternating or Direct Current Line Segments in CIM 1.0 representation. In some cases, having the resistance and reactance specified as absolute values for each line segment is not sufficient for some users of CIM, and it may be better for this value to be calculated based on the:

- length of each line segment
- type of conductor and bundles
- type of tower
- number of towers for each segment
- distance between towers
- positioning of phases on each tower
- number of circuits using each tower

Some of this data is not currently included in standard CIM. However the additional child classes and attributes proposed in the previous section can be added to the Line and Line Segment classes, as was detailed in section 3.5.2. With standard CIM, when the value of the resistance is requested from an AC Line Segment object, the absolute value stored as a variable within the object is returned to the user. Using the enhanced data objects, however, the value returned is itself calculated by a function within the object based on the values of both the object's internal values, and the child objects associated with it. This way, CIM compliance is maintained, since the resistance value is still obtainable if desired, but the greater level of detail can also be maintained, and is available by direct interrogation of the child objects. The ability to include multiple schemas in the same XML file also allows for the standard and enhanced data to be included in a single output file, without breaking compatibility.

Using the extensions to the Line Model described in section 3.5.2, towers can be represented in this extended CIM data model. The inclusion of this data allows the software to identify any points in the network where two or more circuits share a tower, which can affect the electrical characteristics of the individual phases of a circuit. A Line will span several towers, at varying distances from each other, and the span between each tower will be defined as an instance of the Conductor class. Each tower on the line is recorded with a single instance of the Tower class, itself associated with all instances of Conductor from every circuit that use the tower. A Resistance value for the Conductor would be calculated using the attributes of its associated classes (Conductor Type, Wire Arrangement, Wire Type and any instances of the Tower class) as well as the positioning of the phases from any surrounding conductors. This would allow a *getResistance* function in a Conductor object to calculate the resistance each time the function is called. This way any

changes to the attributes of any child classes or other nearby conductors would always be reflected in the value of *Resistance* returned from the Conductor object.

Using a generic import system as detailed previously allows for additional classes to be added by simply including their class descriptors in the configuration settings for the toolkit. This way, the import modules and toolkit itself do not need to be recompiled or rewritten to cope with additional classes. The network model can contain additional information in this extended format, and by ensuring that the version information for each class is correctly assigned, the model can be exported with or without the additional data included.

By structuring the extensions in such a way that, wherever possible, the extra data and classes do not break the existing standard's class associations, then these additional classes will enhance the existing data without necessarily disrupting backwards compatibility.

4.6 Java Packages

Within Java, classes are arranged into packages, a hierarchical system like the arrangement of directories or folders on a hard disk. This way, classes with the same name can exist by being part of a separate package, allowing extensions to the standard to co-exist with the standard IEC classes. This allows the software that is to interrogate, modify and integrate CIM data to be able to cope with multiple standards of CIM data where the class files will have identical names.

A separate package is created for each standard, and the importation system uses the namespaces to choose which package an object is instantiated from for each node. A simple configuration file is used to specify which namespace corresponds to which package, allowing additional packages of extensions without rewriting the importation system.

Since classes can inherit between different packages, the modified line hierarchy within a Strathclyde.Wires package can inherit from the original classes within an IEC.Wires package. This allows the Java class structure to mirror the data specification where new classes inherit from the existing IEC CIM, but also maintain all the existing classes (i.e., the Strathclyde modified WireArrangment class will exist in the Strathclyde package, while the original class can co-exist within the IEC package). This system allows the embedding of functionality to have Strathclyde CIM objects as one or more IEC CIM objects or to be simply interpreted as an instance of its parent class.

4.7 The Mercury Framework

This core framework of enhanced CIM objects has been used as the foundation for a web-based CIM power system network model toolkit, developed by the author, known as *Mercury*. The software's core is the model library, which stores the CIM Objects and provides an API, extending that of the core CIM framework for accessing, interrogating and modifying the models. The second element of the model toolkit is the server interface, which provides remote access to the library with either Web Services or by generating HTML to form the user interface through which the model library can be accessed.

4.7.1 The Model Library

The model library runs as background process on the host computer storing CIM power system models, using the Java object storage framework describe. Each model is contained within a separate instance of the *Model* class, which provides access to the underlying CIM objects as well as additional functionality for interrogating, modifying, analysing and exporting the entire model.

The library stores the multiple instances of the *Model* class, providing user-level access control; archiving facilities; the command serialization system discussed previously and an interface through which other applications can access the *Model* API.

With modules that can access the *Model* API and that of the CIM classes themselves the library can be enhanced beyond the core functionality. These "plugin" modules can export the data in additional formats and perform more complex analysis of the network.

The library also allows access to its API via the Remote Method Invocation (RMI) facilities within Java, allowing any other computer on the network that meets the security criteria to access its public functions. This allows access by command line Java applications, graphical applets on the desktop or by servlets running under Apache Tomcat or any similar J2EE compatible server engine.

4.7.2 The Server Interface

For the *Mercury* software, the model library is accessed via a number of servlets running under Apache Tomcat 5.5. Since Tomcat itself is written in Java, it maintains the platform independence that led to the initial selection of Java as the

language of choice for the framework. The servlets access the model library using the RMI API, and in turn provide a set of public functions, accessible via the HTTP POST and GET protocols. This allows the functions to be accessed by Web Services and Web Browsers. By generating HTML code to access the functions, the user's web browser becomes the user interface for accessing the models within the library.

A component's attributes can be updated using standard forms, with the server validating and converting the form's parameters, received in String format, into the appropriate format for the destination's attributes in the same manner as the initial importation process described in section 4.4.1. By using the unique IDs assigned to each object, associations can be altered by accessing functions within the servlets that receive String IDs to represent the components. The servlet then uses the Model's own internal ID index to locate the CIM object to which these textual identifications refer.

4.8 Chapter Summary

The combination of several new technologies, notably the Common Information Model and object prevalent data storage, along with established techniques for object oriented programming has created the foundations for an international-standard-based, highly extendable and scalable system for storing and manipulating power system data. The implementation of the framework in Java, which is available on a wide variety of computing platforms, allows for significant platform independence for the system.

The use of the open CIM standard for the internal software architecture is itself a novel concept, since the standard is currently used as an intermediary exchange format between applications that have their own internal storage architecture.

Combining the data storage and manipulation into a single memory resident program for long term storage, while unconventional, provides significant speed advantages over dealing with a native file or database when performing complex interrogations of the power system. It allows for the development of a powerful and significantly more extensible API than that of a traditional enterprise-level database. Using Java allows the software to be deployed in a number of ways, including a web servlet, allowing remote execution of the software. This aids the development of web services and other remote or distributed systems for power system simulation and analysis based around the toolkit.

The system provides functionality for the data to be easily exported in XML format, allowing the exchange of data between applications and companies in an open non-proprietary format, one of the major reasons for the development of the Common Information Model in the first place. The ease with which modules to modify and export the data can be constructed highlights the flexibility of the design, and the scope for extending and utilising the core framework.

5 Translation & Conversion of CIM Power System Models

5.1 Chapter Introduction

This chapter outlines the main challenges posed when translating data from CIM XML to proprietary data formats. If utilities are to adopt the CIM tools to translate CIM formatted network models into the format used by their legacy analysis applications then translation is essential. This chapter discusses these challenges and proposes solutions to some of the more complex problems such as node-breaker to bus branch conversion and the mapping of CIM attributes to proprietary file formats along with a case study to translate a small CIM XML power system model for load flow simulation.

5.2 CIM XML Translation

With the use of CIM increasing, and the acceptance of XML as the most common format for distribution, more applications of CIM XML will emerge in the near future. Widespread use will encourage industry participants to innovate within the bounds of the technology and the ability to interpret CIM XML files will become necessary for power system simulation. Several software vendors are now offering products with CIM XML compatibility built in, but it is still a relatively new format and many legacy applications have yet to implement native support.

Currently, some tools and applications support CIM XML natively, and more applications that are based on CIM, or are able to natively interpret the data, will become available. This migration to CIM XML-aware tools is, however, constrained by the availability and investment required for creating new tools or modifying existing applications. Of the four scenarios outlined in Section 2.2, this chapter is concerned with the third scenario: storing the power system data in a common format but still requiring software to translate this data into the native format for the target application.

5.3 Translation of Power System Data

One of the principal challenges in the representation and exchange of power system data is the passing of data between applications. Different applications and

software tools use different models of the power system and a variety of data formats. Moving data from one format to another requires some "translation" of the data. A method for performing such translation is presented in this section.

The most widely used power system simulation tools, such as PSS/E and EMTP, use their own proprietary data formats. Future simulation tools may be able to read CIM XML data directly, perform analysis and output directly in CIM XML. Until such applications are available, data translation will be required. Even after the introduction of CIM-based simulation tools there will still be substantial embedded investment in software using proprietary formats. Translation to and from CIM XML will be required to integrate CIM XML tools with legacy software.

In translating from CIM XML to a proprietary power system data format, there are several interrelated issues that must be resolved simultaneously:

1. Topology Format
2. Unique Component Identifiers
3. Physical Characteristics
4. Identifying a Specific Equipment Property

5.3.1 Topology Format

Power systems can be represented with models at different levels of abstraction, which results in different formats for the power system topology. "Node-Breaker" models are one of the most detailed and are used in CIM. This level of complexity is not required for the simulation of large power systems, as many of the components do not play an active role in the network's behaviour. Simulation tools most commonly use a Bus-Branch model of the power system. Figure 5.1 shows the same substation bay in these two topology formats.

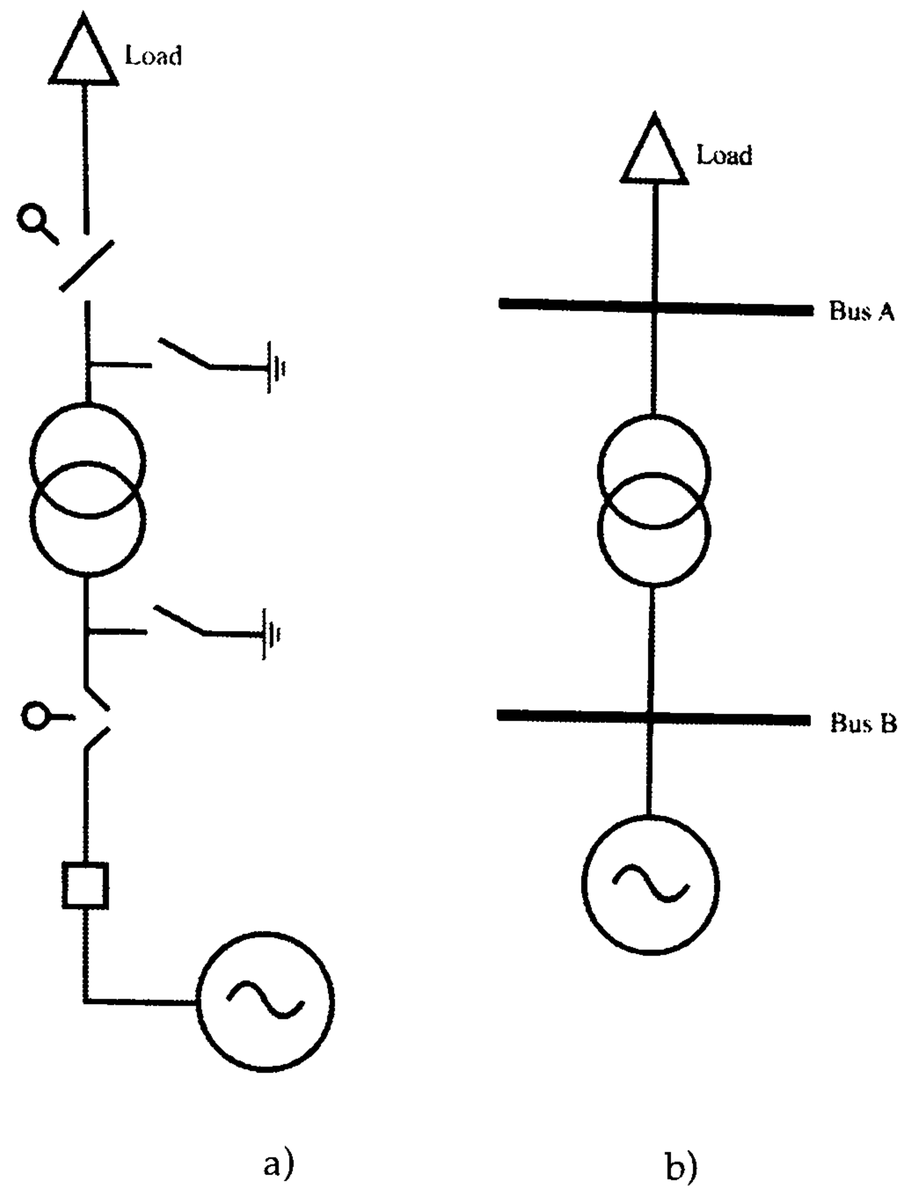


Figure 5.1 A substation feeder bay in: a) Node-Breaker format; and b) Bus-Branch format.

Devices such as ground disconnectors are used to isolate sections of a substation to allow engineers to work safely. They should be included if substation data is to be comprehensive, as required in EMS applications. However power system analysts are primarily concerned with the main system components: buses, branches, transformers, generators and loads. In the Bus-Branch format, all the switches and connections to ground are missing because under normal operational conditions, if the switches are closed, they do not directly affect the circuit's performance.

One challenge in translating from the Node-Breaker topology format of CIM XML to a proprietary Bus-Branch format is recognising the component type in CIM XML and translating to the correct component type in the proprietary format. There are three courses of action for each CIM XML component identified:

1. Translate one CIM XML component to one proprietary format component.
2. Amalgamate two or more CIM XML components into a single proprietary format component.
3. Ignore the CIM XML component and do not create a component in the proprietary data format.

The higher resolution in the CIM XML representation of a power system means that it will usually not be necessary to create two or more proprietary format components from a single CIM XML component.

The translator must convert the higher-resolution data of CIM XML to the lower-resolution format of the simulation tool whilst maintaining the integrity of the output. This process means that it is not always possible to translate back to CIM resolution from the proprietary format since the conversion and formatting is often a one-way process.

It would not be possible to convert back to the high-resolution circuit data when amalgamating several components into a single Bus, unless the legacy data about the components contained within the bus are included, or the bus type is of a standard for which the internal structure is known.

Similarly, for single numerical values calculated from two or more source values, it is, in most cases, impossible to work back to the source values if only the result is known. This is because mathematical formulae are often one way functions, where there are multiple possible combinations of factors that could have resulted in the value.

An example of this is calculating the overall branch resistance and reactance for a transformer. This requires getting values of resistance and reactance for each winding of the transformer from the CIM data, then calculating the overall resistance of the transformer using the values extracted. This formula however results in single values for resistance and reactance for the transformer, and it is not possible to work back to the individual values for each winding from this.

This shows why it is important for legacy data to be retained if data at a high level of abstraction is to be converted back to a higher detailed format.

5.3.2 Unique Component Identifiers

Unique bus identifiers in proprietary formats create problems for the translation since a unique bus may be created from several components in a CIM XML model. In Bus-Branch data formats, a bus appears as a single point with equipment connected to it and each bus has a unique identifier. In Node-Breaker data (as in CIM), a bus is not a single point but can contain switches and circuit breakers which, when closed, function as uninterrupted connections between two points. This issue may arise with other component types.

CIM has a class that is analogous to a bus, the Topological Node. This class contains a list of Connectivity Nodes, and can be used to amalgamate a group of Connectivity Nodes into one object. Connectivity Nodes are defined as "points

where terminals of conducting equipment are connected together with zero impedance" [31]. Connectivity Nodes are not physical components in a power system, but exist only within CIM data to provide a connection point for pieces of equipment. Thus, all Connectivity Nodes that are joined by closed switches, or other non-primary circuit elements can be put into a single Topological Node, which can be used to represent a bus in a CIM data structure.

In this context, primary equipment refers to any piece of coducting equipment that would be connected to a bus in a bus-branch representation. In the CIM the primary equipment is anything that is a sub class of EnergyConsumer (for loads), SynchronousMachine and EquivalentSource (for generators), TransformerWinding (the physical connection points to the network for a transformer) or AC or DCLineSegments (for lines).

When Topological Nodes are present in the CIM data, these can be directly converted into buses with unique identifiers. However, Topological Nodes are optional and not required for a valid CIM XML document. For those files that do not contain Topological Nodes, effective translation requires a means of dynamically generating the required bus data whilst maintaining the integrity of the original data.

A process has been developed to perform this task. The process uses a recursion system based around the Connectivity Node class. For each piece of primary equipment in the CIM XML file that will be connected to a bus, the recursion process identifies other pieces of primary equipment connected to it, and creates a CIM Topological Node instance that will become a bus in the Bus-Branch data.

The algorithm for this process is shown below:

```
1. For each piece of primary equipment
2.    If the equipment has not been processed
3.    For each of the equipment's unprocessed Connectivity Nodes
4.       Create a new Topological Node
5.       Add the current Connectivity Node to the current Topological Node
6.       Mark the Connectivity Node as having been processed
7.       For each of the Connectivity Node's other connected equipment
8.          If the equipment is a piece of Primary Equipment or Open Switch
9.             Stop
10.         Otherwise
11.            Find the Equipment's other Connectivity Nodes
12.            For Each of these Connectivity Nodes
13.               Go back to step 5
14.         Select the next piece of Primary Equipment and goto step 2
```

This process spiders through a network, adding in all Connectivity Nodes until it reaches another piece of Primary Equipment or open switch, which indicates an edge of the Topological Node. By repeating this process until no more paths are available, a Topological Node will be created. Repeating this process until all elements of Primary Equipment have been processed will result in a set of Topological Nodes covering the whole network.

Each Topological Node will only have external connections to the defined primary elements, and will contain one or more Connectivity Nodes. As the process executes, each Connectivity Node is marked as having already been processed, so as to keep each Topological Node unique, and prevent a Connectivity Node from existing in two or more Topological Nodes. That situation would be undesirable as it could result in pieces of non-primary equipment being linked to more than one bus.

An example of this process is illustrated in Figure 5.2a). Here, the Transformer is selected as the starting piece of primary Equipment. The Transformer contains two Windings, which, in CIM, define the physical connection points to the network.

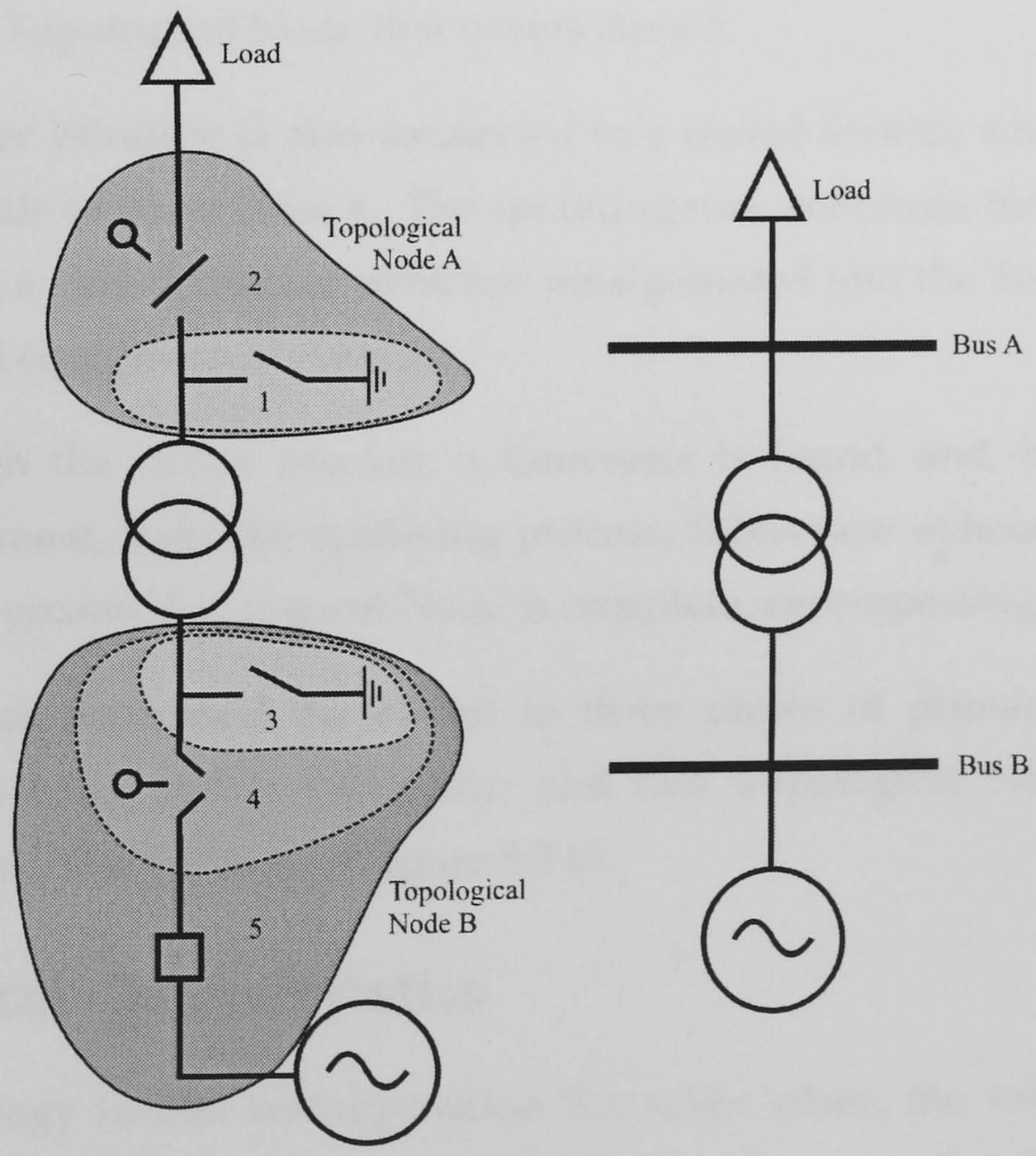


Figure 5.2 a) Schematic of the stages for Topological Node creation on a sample network. b) The resulting Bus Branch circuit

Taking the primary Transformer Winding of the Power Transformer, the initial piece of Primary Equipment, as the starting point, the process spiders out until the first piece of connected equipment, a Ground Disconnector, is found. This is not a piece of Primary Equipment, and so is amalgamated into the Topological Node, which is illustrated as Area 1 in the diagram.

The processing application then finds the other piece of equipment connected to the primary Transformer Winding, an Isolator. This is closed, and as such, the process continues, incorporating the Isolator into the Topological Node. This node now contains all equipment in Area 2.

The spidering process continues through the closed Isolator and finds that it is connected to a Load, which is a piece of Primary Equipment. As such, the process halts its progress in that direction. With all other possible routes exhausted it finishes, and the Topological Node is complete, containing all equipment in Area 2.

A new Topological Node is now created, and the secondary Transformer Winding of the Power Transformer is selected as the starting point. Its first connection is to a Ground Disconnector and since this is not a piece of Primary Equipment, it is included in the Topological Node that covers Area 3.

The Transformer Winding is also connected to a closed Switch, which enlarges the Topological Node to cover Area 4. The spidering now continues through the closed Switch, finding a closed Breaker, which is amalgamated into the Topological Node. It now holds all equipment in Area 5.

Passing through the circuit breaker, a Generator is found, and, being a piece of Primary Equipment, halts the spidering process. It has now exhausted all possible routes, and the second Topological Node is complete, encompassing Area 5.

This process has simplified the circuit to three pieces of primary equipment; a Transformer, a Load and a Generator; and two Topological Nodes, which are equivalent to the Buses shown in Figure 5.2 b).

5.3.3 Physical Characteristics

Once the topology format transformation has taken place, the values quantifying the physical characteristics of the network must be extracted from the values available in the CIM XML file. This can pose problems because CIM XML represents the physical characteristics of power system components in literal values such as MW, MVAr, Ohms and Amperes. Many proprietary data formats use ratio,

per unit and percentage values to represent the power system components. These values must be calculated from the available CIM data.

An example of this problem is calculating the per-unit voltage of a piece of equipment when given only the literal value. Calculating a per-unit voltage requires two values: the literal value of the voltage of the equipment itself; and the base voltage in that section of the network. In CIM, the equipment's data node contains only its own voltage. The base voltage of the network section is stored in the Connectivity Node. To find the base voltage the following steps must be taken using data from the CIM XML document:

1. Find the Terminal that the equipment is connected to
2. Find the Connectivity Node that this Terminal connects to
3. Find the Voltage Level that the Connectivity Node is in
4. Find the Base Voltage Level of the Voltage Level

Each of these stages involves searching and comparing different classes of node in the CIM XML file then extracting the appropriate information. For large networks this can involve searching several thousand nodes at a time, which can become a time consuming process. Under some circumstances the proprietary format may require data that cannot be extracted directly or derived from the CIM XML file, in which case default values are needed. These values can be taken from the default values used in the destination program or, alternatively, defined in a configuration file that can be altered by the operator before each execution.

When default values are used, it would be useful for the user to identify the attributes they were applied to, so the values can be adjusted for future conversions if necessary. This would be facilitated through a log file generated during the conversion process that would flag when a default had been used, and show its value.

5.3.4 Identifying a Specific Equipment Property

Problems also arise when an analysis program requires the identification of a specific equipment property for which CIM does not contain information for. For example, network load flow calculations require one of the generator buses to be defined as the swing bus, which is adjusted to balance the power flow. As mentioned previously, CIM XML does not contain bus information and translation

to Bus-Branch data formats requires buses to be generated by combining many CIM components into a single bus. Since all bus data must be generated prior to analysis there is nothing to define which bus should be the swing bus.

This problem can be solved by prompting the user to select a swing bus from a list of the buses generated, or automatically selecting one based on a set of criteria (e.g. the largest generator).

5.4 CIM XML to PSS/E Data Format Translation

PSS/E is a widely used power system simulation and analysis software package. Tools have been developed by the author for the translation of CIM XML data to the PSS/E format. In PSS/E data, each bus has an individual number associated with it but, since the buses themselves are not always defined in the CIM data, the algorithm detailed above was used to provide unique bus objects, which were assigned incremental identifiers.

To perform the necessary manipulations of XML files, there are two main options: The eXtensible Stylesheet Language Transform (XSLT) is a recommended language for converting XML documents into simple text, or other vocabularies of XML and could be used to convert XML to a plain text file; or create a module for the *Mercury* CIM Java Framework to translate and export a PSS/E compatible input file from the memory resident CIM Java objects.

5.4.1 Extensible Stylesheet Language Transform

An XSLT solution has the potential of providing a quick, and simple solution to the problem requiring only an XSLT parser (the software that applies an XSLT to an XML file) to operate instead of the full *Mercury* framework. Although XSLT is used for defining transforms to XML data, it is limited in many ways and lacks many features available in popular programming languages, such as arrays and the ability to change a variable once it has been created. When an attempt was made to implement the above algorithms in XSLT it was found that the language was unable to cope with the complexity required, and could only implement a reduced process for use on very small networks. For very large networks, the XSLT script was unable to finish processing the model file. This is due to XSLT's limited variable handling abilities, which prevented it from marking when nodes had previously been processed, and thus resulted in infinite loops occurring within the program as it repeatedly processed the same section of the network.

PSS/E requires a swing bus to be selected from the generation buses. When using XSLT to do this, a rule must be created because there is no facility to state that the swing bus has already been set. The simplest rule is to define the swing bus as being the bus connected to the first power generation device found. This could cause a problem, however, if the generator selected has insufficient capacity to secure the system in a simulation. In Java this is not a problem: the swing bus is assigned, and can be changed if required. A flag is set when a swing bus is assigned, thus preventing more than one swing bus in the network.

XSLT is better suited for simpler translations, where the data is moved directly, or requires very little alteration, merely a reformatting. This is not the case with the majority of the translations required to convert data into PSS/E input format. For these more complex conversions, Java, or other object-oriented languages such as C++, are more suitable. When the algorithms are implemented in one of these languages, it allows the conversion of much larger and more complex CIM XML files than is possible with an XSLT-based translation. This is primarily due to the limited variable handling in XSLT in comparison to mainstream programming languages.

5.4.2 Mercury Translation Module

Since the XSLT solution proved unviable, the algorithms were implemented as an export module for the *Mercury* framework. Since the CIM objects are stored in memory within the *Mercury* model library, the time taken to traverse the network increases linearly as the network size grows (as was discussed in section 4.4.2). The direct memory references between each objects removes the need to search and index to locate the next network component. By implementing a reverse-propagation system into the *Mercury* import module, to ensure that references are bi-directional (i.e. if a Terminal is associated with a Connectivity Node, the Connectivity node in return is associated with the Terminal), the resulting network is fully interconnected, allowing the algorithm to step through from Conducting Equipment to Terminal to Connectivity Node etc. without requiring any searches, and the ensuing time penalties. This provides a fast, efficient method for traversing the network using the functions already present in the CIM Java Framework, plus some simple logic to locate the next element in the sequence.

During the process, the Topological Nodes created are inserted into the model since they are standard CIM objects. Since the creation process can be accomplished in a fraction of a second, even for large models, it is generally a good idea to delete all

existing Topological Nodes prior to executing the Topological Nodes creation process. This, however, may not be ideal if additional information is included within the Topological Node, added after the process has run. As such, it is possible to store the model's original Topological Node instances prior to the process beginning, then, after the process has run, compare the newly created objects with the originals. If any original Topological Nodes contain a matching list of associations to any of the newly created objects, it can be assumed that the two represent the same bus, and the original Topological Node is retained in place of the newly generated instance.

5.4.2.1 The Bus and Branch Classes

The export module creates a *Bus* object for every Topological Node in the network. This *Bus* class provides functionality to obtain the required values from the Topological Node by transparently extracting values from the other CIM components connected to the node and returning them upon request. For example, the *getPg()* function for the *Bus* returns the total amount of real power generated at that bus. This function locates all the Synchronous Generators connected to the assigned Topological Node, locates their associated Generating Unit objects, extracts the *initialMW* attribute and returns the total of these values.

Similar functions are put in place for the reactive power generated, the real and reactive power demands, and the voltage level. This *Bus* class contains no values itself, but instead calculates them transparently from the model each time the function is called.

A similar class is created for each *Branch*, containing an association to either a transformer or a line. This class is more complex than the *Bus* class, since it must cope with different classes of associated CIM object. The transformer-associated instance, for example, must contain functionality to convert the multiple winding resistance and reactance values into a single transformer value. This requires a more complex formula than a simple addition but, as with the *Bus* class, each function returns a value "on-the-fly", calculating it from the values contained within the CIM object.

The PSS/E output file is created by looping through every *Bus* and *Branch* instance, extracting the appropriate values and inserting them into the correct position in a text file.

5.5 Example of CIM XML to PSS/E data Translation

To test the output generated by the CIM XML to PSS/E translation program when implemented as a *Mercury* module, a sample circuit was needed in CIM/XML format. The sample circuit was designed to contain a variety of components, including transformers; generators; lines; switches; breakers; and loads. The sample circuit created to test the translation script is illustrated in Figure 5.3.

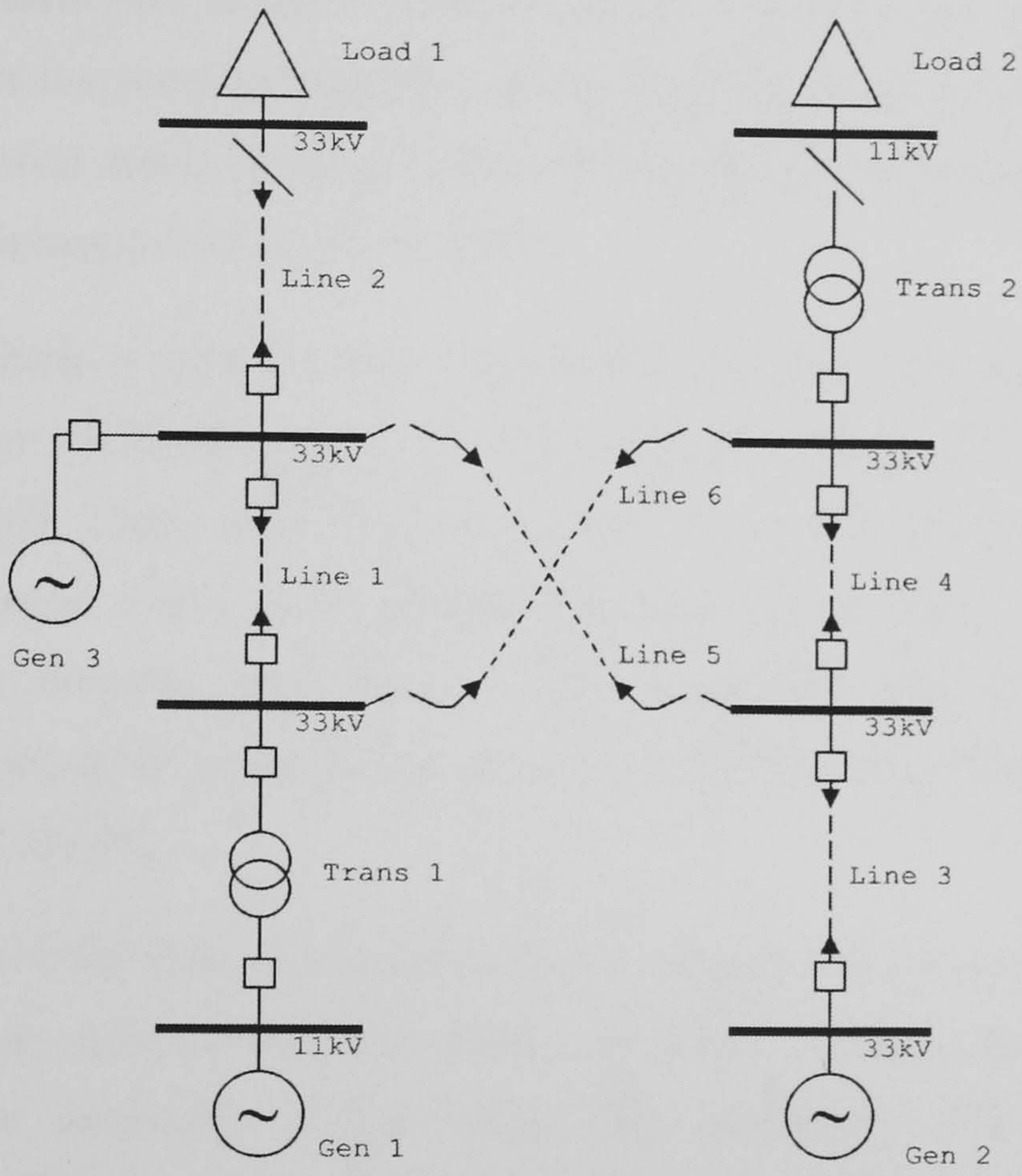


Figure 5.3 Schematic of case study network in Node-Breaker format.

From this diagram the CIM classes for each component can be identified, and the circuit converted into a CIM format model. This CIM data was then expressed as CIM XML to create a valid test file for the translation program. All the switches were closed to create a fully interconnected network.

The resulting file was run through the translation program. A PSS/E native output file was successfully produced by the program and then used as simulation input data for PSS/E. A power flow simulation was run using this input data and completed successfully. A valid simulation result for the circuit was produced.

The test proved that a valid CIM XML file can be interpreted and translated successfully, producing a valid PSS/E input file.

5.6 Chapter Summary

The ability to convert data in the CIM format into proprietary formats for power system applications is important, since many existing applications do not have native support for this relatively new format. With increasing numbers of power system models being stored in CIM XML, the ability to load and perform analysis on these models using existing applications will aid in the transition to open model standards throughout the industry.

For those applications that work at a higher level of abstraction, the high detail level of CIM data allows for accurate conversion to a less detailed format. This allows for successful conversion from Node-Breaker to Bus-Branch topologies, by combining several network elements into a single Bus.

CIM does not include a specific Bus class; however, the Topological Node class is analogous to a Bus. When Topological Node information is not present within the provided CIM XML data, they can be created dynamically using the network's present configuration, and if it is already present, the existing Topological Nodes can be converted directly into Buses. This approach also allows results from analysis applications to be returned as attributes that can be used to directly update objects in the CIM model.

While much of the CIM data is stored as literal values, destination applications can require these values as percentages or ratios. As such the data must be converted by extracting all the required factors from the available CIM data and using predetermined mathematical formulae to express the value in the required format.

Implementing the conversion algorithms for converting to PSS/E format highlighted the limitations of XSLT for performing complex transformations, and it was deemed unsuitable for the task. The module built on the CIM Java object storage framework, however, performed as expected and was able to cope with complex network configurations whilst producing valid output. This CIM Java Object storage system provides the ability to quickly traverse a network's topology allowing node-breaker to bus-branch conversions to take place in under a second for even large power system models comprising tens of thousands of CIM Objects.

6 Validation of CIM XML Data

6.1 Chapter Summary

The validation of CIM data is an important issue for network operators when exchanging power system data. When encoded as XML the validation of the resulting CIM XML data should take place on at least two levels during any exchange. It must check

- The syntax of the RDF formatted XML is correct
- The data encoded within the XML conforms to the CIM standard

A third level of validation is also required so that the CIM data describing specific attributes of a power system (e.g., switch status, voltage level) should then be validated, whether it is obtained from a CIM XML file or from another source. This level of validation ensures that the attributes and associated objects of each CIM object meet the minimum data requirements for their class.

The first two levels of validation are already built into the Mercury CIM Toolkit Framework: the first through the use of the XML Parsing functionality within the Java API, the second by the Mercury import module.

The third level of validation uses the Common Power System Modelling (CPSM)[34] minimum data requirements for the CIM to define the required attributes and associations for each class. Since the CPSM standard is a document listing the data requirements for a power system model in the CIM format, these requirements must be converted into logical rules and software written to test CIM data against these rules. All three levels of validation have been successfully demonstrated and are in use in international CIM Working Group activities.

6.2 XML Syntax Validation

The DocumentBuilder class, part of the standard Java API, validates the CIM XML document's syntax automatically during the parsing procedure. This checks that the XML is correctly formatted and throws an exception if the document is malformed. The XML validation provided by the DocumentBuilder class ensures that

- The file has an XML declaration to define the XML version and character encoding
- Each document has a single root element
- All elements are properly closed
- All namespaces used within the document have an entry in the root tag
- All tags are correctly formatted
- All attributes are enclosed within quotes
- Child Elements are correctly nested within their parent element

This level of validation ensures that the file provided is valid XML. However, this does not ensure that it is valid CIM data.

6.3 CIM Data Validation

Since CIM XML is, at the most basic level, an XML representation of CIM Objects as described in Section 2.5.3, it must validate against the CIM standard to ensure that the data is valid CIM. This requires each CIM XML element to be an instance of a valid CIM Class and for its child elements to correspond to the attributes and associations of that class.

6.3.1 Transformer Winding CIM XML Element Example

Below is an example of a CIM XML element for a transformer winding

```
<cim:TransformerWinding rdf:ID="_6E4BEB0E885452C9FC6868C0D46A3FA">
  <cim:TransformerWinding.b>0.0</cim:TransformerWinding.b>
  <cim:TransformerWinding.r>0.01904</cim:TransformerWinding.r>
<cim:TransformerWinding.ratedMVA>75.0</cim:TransformerWinding.ratedM
VA>
  <cim:TransformerWinding.windingType
rdf:resource="http://iec.ch/TC57/2003/CIM-schema-
cim10#WindingType.primary"/>
  <cim:TransformerWinding.x>7.37004</cim:TransformerWinding.x>
  <cim:TransformerWinding.MemberOf_PowerTransformer
rdf:resource="#_AF0E4452890842B8BDB07A9C2F2765AA"/>
  <cim:ConductingEquipment.Terminals
rdf:resource="#_22488CBA8340489EB4D6C48EE2F754EF"/>
  <cim:Naming.name>High Winding</cim:Naming.name>
</cim:TransformerWinding>
```

For this CIM XML element to be valid the CIM standard must have a Transformer Winding class, with the following attributes and associations:

- An attribute b which is a floating point number

- An attribute r which is a floating point number
- An attribute ratedMVA which is a floating point number
- An attribute windingType which is of type WindingType
- An attribute x which is a floating point number
- An association to another CIM Object (or Objects) called MemberOf_PowerTransformer
- An association to another CIM Object (or Objects) called Terminals
- An attribute name which is a string

The CIM Java class for TransformerWinding contains all of these attributes and associations either directly or by inheritance. However, for the element to be valid the associated objects must also be verified.

The MemberOf_PowerTransformer association within the TransformerWinding class requires the association to be with a PowerTransformer object but, since the XML element provides a reference to another XML element, this element must be verified as an instance of the PowerTransformer class or one of its subclasses.

Similarly, the Terminals association is a 0..n association with objects of the Terminal class, and so the XML element given in that reference must similarly be checked to ensure that it is of the Terminal class or one of its subclasses.

The problem with this validation is that, since the XML importation is looping through the XML elements one by one converting them to CIM Java Objects, it cannot guarantee that an object will not reference another XML element that has yet to be imported. To solve this problem, the import process uses a two-pass approach.

6.3.2 CIM Java Object Creation

The first pass converts all XML elements into their corresponding CIM Java Objects and puts an entry into the reference index that stores the original XML rdf:ID for each object together with a reference to the CIM Java object itself.

The attributes are converted into their corresponding types obtained from the class file itself using Java's reflection API. This allows an object to examine itself and extract its own attributes and methods along with their appropriate types.

For the TransformerWinding element shown above, the newly created CIM Java Object will know that its r, x, b and ratedMVA attributes are of type Double (double precision floating point). Since all of the XML child elements are imported as plain text (Strings) initially and stored in a temporary hash table, the attributes can be converted to their appropriate type by simply creating new instances of their type class (e.g. Double, Integer) with the string representation as the constructor field. For example:

```
r = new Double("0.01904");
```

This provides another level of validation to the import, since the object creation will report an error if the original XML element value is not valid. For a Double value this will catch non-numeric input, but if an attribute is of type Integer and the XML element for that attribute has the value "1.23" then it would be imported into the object using the code shown below:

```
attribute = new Integer("1.23");
```

This will report an error since "1.23" is not a valid integer. By catching this error and processing it correctly, the importation can continue but the final model will contain a report of the error within the data.

6.3.3 Reference Propagation

The associations must remain as string values until all the XML elements have been imported and converted to CIM Java Objects. When this first-pass is complete, the second-pass through the objects uses the reference index to locate the corresponding CIM Java object for that association and adds it to the object.

As with the attribute conversion, should the reference be to an object whose class is not the same as the association's class (or a subclass of it), then it will report an error. This prevents a TransformerWinding's MemberOf_PowerTransformer association from pointing to anything other than an instance of the PowerTransformer class (or one of its subclasses).

If an rdf:ID reference does not exist within the reference index then this indicates that the CIM object contains an association to another CIM object that was not contained within the CIM XML file. This too reports an error that can be caught and processed accordingly.

6.3.4 'CIMValidate' Validation Tool

This level of validation and the checking of the XML syntax itself can be accomplished using the CIMValidate Tool[35] in combination with a CIM RDF Schema file generated from the CIM Rational Rose UML file. This tool, developed by Langdale Consultants, can validate any XML file against a corresponding RDF Schema and reports any errors in the CIM data.

This tool is available as open source, and as such could have been integrated into the Mercury toolkit. The decision was taken to omit it from the Mercury Framework for two reasons:

- The model importation, using the process described, already validates the CIM XML file to the same level as the CIMValidate tool due to the architecture of the framework
- The CIMValidate tool, when used with large models took over twice as long to process on single processor computers, and, unlike the Mercury framework, it is not multi-threaded, and so does not take advantage of computers with multiple processors. This resulted in the CIMValidate tool taking 5 minutes 32 seconds to validate a CIM XML file of 60,025 elements compared with 42 seconds to import, validate and create 60,025 CIM Java Objects for the Mercury Toolkit[5].

Any file that is found to be free from errors either by the Mercury import module or the CIMValidate tool can be considered valid CIM data in terms of structure and syntax. This in itself, however, does not mean that the data contained within the file represents a valid power system.

6.4 Minimum Data Requirements

6.4.1 Validation of Empty Objects

The IEC 61970-301 standard defines the class hierarchy for the CIM with attributes and associations for each class, but does not define which attributes and associations are required and which are optional for each class. This means that a CIM Object

---

[5] Tests were undertaken on a 2.5Ghz Quad core G5 with 4.5Gb of memory server running Apple OS X 10.4.4 and using Java 5 version 1.5. The test file was a CIM XML file of 60,025 CIM Objects representing a large, real-world power transmission network.

can lack the attributes required for it to accurately represent a component within the system yet still validate.

Using the same transformer-winding example from the previous section, by removing all attributes and associations except *name*, we are left with:

```
<cim:TransformerWinding rdf:ID="_6E4BEB0E885452C9FC6868C0D46A3FA">
   <cim:Naming.name>High Winding</cim:Naming.name>
</cim:TransformerWinding>
```

It is obvious that this XML element contains no data about the winding's electrical properties or place within the network, yet both the Mercury import module and the CIMValidate tool find no errors with it. This is because there is nothing within the IEC 61970-301 standard to say that an instance of the TransformerWinding must contain any attributes or associations at all. As such, even an empty (or nearly empty) TransformerWinding object can be considered valid CIM data.

6.4.2 CPSM Minimum Data Requirements for the CIM

The CPSM Minimum Data Requirements for the CIM document contains general requirements and notes describing specific restrictions on object relationships. A validation tool must be capable of checking each CIM Object for the minimum attributes and associations required by the specification and for the requirements given in the general requirements and notes for each class which are not written as a series of logical rules.

6.4.3 Creating Minimum Data Requirement Rules

When using the Mercury CIM Java Framework, checking for the required attributes and associations within each CIM Object is a fairly simple process requiring an object to check that the defined attributes and associations are not *null*, which would indicate that the attribute does not have a value assigned or that the association does not reference another object. Similarly, checking that numeric values are within a specific range (e.g. greater than 0 or less than 100) can be accomplished with simple rules using basic comparison operators.

Interpreting the notes and general requirements, however, is more problematic. For the TapChanger class, which can 'be fixed or it may regulate voltage, phase angle, or both', the notes state:

> 'If the control mode is voltage, phase angle or both, the attributes "highStep", "lowStep", "neutralStep", and "normalStep" are all required. If voltage control is possible, the attributes "neutralkV" and "stepVoltageIncrement" are

also required. If phase angle control is possible, the attribute "stepPhaseShiftIncrement" is required'

The tculControlMode attribute within the TapChanger class does not actually state whether a Transformer is capable of voltage and/or phase control - this is defined in the PowerTransformer class's transformerType attribute. As such, when a TapChanger object is checking itself it must know the value of this attribute in the PowerTransformer it is contained within. To do this it must navigate to the TransformerWinding instance it is within and from there to the PowerTransformer class. Then, depending on the value of this attribute it must check its own minimum attributes.

This can written logically as:

If any of the below rules are true then the object is valid:

```
Start Rule: Given all of the below
     TransformerWinding.MemberOf_PowerTransformer.transformerType
   must be equal to "phaseControl"
   stepPhaseShiftIncrement must not be equal to <null>
   highStep must not be equal to <null>
   lowStep must not be equal to <null>
   neutralStep must not be equal to <null>
   normalStep must not be equal to <null>
End Rule

Start Rule: Given all of the below
     TransformerWinding.MemberOf_PowerTransformer.transformerType
   must be equal to "voltageControl"
   stepVoltageIncrement must not be equal to <null>
   neutralKV must not be equal to <null>
   highStep must not be equal to <null>
   lowStep must not be equal to <null>
   neutralStep must not be equal to <null>
   normalStep must not be equal to <null>
End Rule

Start Rule: Given all of the below
     TransformerWinding.MemberOf_PowerTransformer.transformerType
   must be equal to "voltageAndPhaseControl"
   stepVoltageIncrement must not be equal to <null>
   neutralKV must not be equal to <null>)
   stepPhaseShiftIncrement must not be equal to <null>
   highStep must not be equal to <null>
   lowStep must not be equal to <null>
   neutralStep must not be equal to <null>
   normalStep must not be equal to <null>
End Rule
```

The TransformerWinding.MemberOf_PowerTransformer.transformerType attribute means the object must navigate through its own TransformerWinding association to the corresponding PowerTransformer object and use its transformerType attribute

in the comparison operation. This allows rules for a particular class to be written so that the result is dependent on the value of one or more other attributes from associated objects.

6.4.4 Vendor Interpretations

An additional problem occurs when vendors interpret the CIM differently and produce different CIM representations for the same underlying data. An example of this issue is the containment of a PowerTransformer object. As discussed in Chapter 2, the PowerTransformer class is used as a container for one or more TransformerWinding objects. Each winding will be associated with (i.e., contained within) a different VoltageLevel. The PowerTransformer class has a MemberOf_EquipmentContainer association for representing its own containment, but from analysis of the test files from a number of utilities and vendors provided for the latest CIM Interoperability Tests, some have the PowerTransformer contained by an instance of the Substation class and other have it contained by a an instance of the VoltageLevel class.

Both of these representations are valid under the CIM standard since both the Substation and VoltageLevel class inherit from EquipmentContainer, and the PowerTransformer's MemberOf_EquipmentContainer association is to an object that is either an instance of EquipmentContainer or one of its subclasses. While both these representations are valid, it results in the same data being represented in two different ways, which can result in incompatibilities when exchanging data, the very issue the CIM was supposed to prevent.

This issue also requires more complex solutions for many of the general requirements such as:

- Each PowerTransformer and its associated TransformerWindings and TapChangers must be contained within one substation

This requires each TransformerWinding to ensure that the PowerTransformer it is contained within is also within the same Substation. However, since the PowerTransformer can either be contained within a Substation directly or within a VoltageLevel (which itself is contained within a Substation), the validation rules must check for both of these situations when checking that a TransformerWinding is contained within the same Substation as its parent PowerTransformer.

The TransformerWinding's own Substation can be found by a simple navigation through its VoltageLevel container to the Substation container. The PowerTransformer's MemberOf_EquipmentContainer can either be associated with a Substation or a VoltageLevel so the navigation can be either PowerTransformer->Substation or PowerTransformer->VoltageLevel->Substation. Therefore the original requirement for TransformerWinding containment is translated into two rules:

```
MemberOf_PowerTransformer.MemberOf_EquipmentContainer.MemberOf_Subst
ation is equal to all of
MemberOf_EquipmentContainer.MemberOf_Substation
Or
MemberOf_PowerTransformer.MemberOf_EquipmentContainer is equal to
all of MemberOf_EquipmentContainer.MemberOf_Substation
```

Since a PowerTransformer has only one MemberOf_EquipmentContainer association, only one rule can ever be met. If either of these rules is true for a TransformerWinding instance then its containment can be deemed valid.

6.4.5 Rule Inheritance

As with the class structure, by default a class inherits all the rules applied to its parent class. This can be seen in the CPSM Minimum Data Requirement document itself where the defined rules for an EquivalentLoad match those of an EnergyConsumer, its parent class. Similarly, all classes that contain a name attribute, and thus inherit from Naming, must contain a valid name attribute. By having this rule defined in the Name class it prevents the same rule being repeated in the rule set for every class that inherits from Naming.

This prevents the duplication of rules from parent class to sub-class and ensures that any additional classes added to the CIM standard can be integrated into the validation engine quickly and accurately. The option to prevent rule inheritance is maintained within the rule definitions for each class to allow for any special exceptions that may occur in future profiles.

6.4.6 Complex Rule Translation

Translating the entire CPSM Minimum Data Requirement document into a series of rules is a time-consuming manual task, resulting in 101 rules spread across 44 classes. These rules range from the very simple checks on a single attribute value to complex rules involving multiple sub-rules.

For example, the following rules check that a Compensator is either: operating in series with an r and x set; or shunt mode with maximumSections, mVArPerSection, nominalKV and normalSections set

```
Start Rule: Given any of the below
   Start Rule: Given all of the below
   compensatorType must be equal to CompensatorType.series
   r must not be equal to <null>
   x must not be equal to <null>
   End Rule

   Start Rule: Given all of the below
   compensatorType must be equal to CompensatorType.shunt
   maximumSections must not be equal to <null>
   mVARPerSection must not be equal to <null>
   nominalKV must not be equal to <null>
   normalSections must not be equal to <null>
   End Rule
End Rule
```

Sub-rules can be nested within additional sub-rules so that more complex requirements can be defined. For example, an Energy Consumer's requirements state that the object is valid if the conformingLoadFlag is true and energy is defined as any pfixed and qfixed with pfixedPct and qfixedPct as null, or true and energy is defined with pfixedPct and qfixedPct and the LoadArea has one or more AreaLoadCurves, or false and energy is defined using pfixed and qfixed.

To translate these into rules requires multiple, nested sub-rules:

```
Start Rule: Given any of the below
   Start Rule: Given all of the below
      conformingLoadFlat must be equal to true
      Start Rule: Given any of the below
         Start Rule: Given all of the below
            pfixed must not be equal to <null>
            qfixed must not be equal to <null>
            pfixedPct must be equal to <null>
            qfixedPct must be equal to <null>
         End Rule

         Start Rule: Given all of the below
            pfixedPct must not be equal to <null>
            qfixedPct must not be equal to <null>
            The number of LoadArea.AreaLoadCurves must be greater than
or equal to 1
         End Rule
      End Rule
   End Rule

   Start Rule: Given all of the below
      conformingLoadFlat must be equal to false
      pfixed must not be equal to <null>
      qfixed must not be equal to <null>
   End Rule
End Rule
```

Using this approach all the minimum data requirements from the CPSM document can be successfully translated into logical rules.

6.4.7 Applying the Minimum Data Requirement Rules

Now the CPSM document has been translated into logical rules a method of applying these rules to the CIM data must be found. There are three options available:

1. Translate the rules into an Resource Document Framework Schema (RDFS) or Web Ontology Language (OWL)[37] syntax and utilise an existing validation engine

2. Translate the rules into native Java code and implement them within each CIM class in the CIM Java Framework.

3. Specify the rules in RDF Schema, OWL or a custom format then create a separate engine within the CIM Java Framework to translate the rules into Java code on the fly.

Each of these options was considered.

6.4.7.1 Utilising an Existing Validation Engine

The CIM standard can be exported to RDFS automatically from the Rational Rose UML Model file using existing tools[36]. As described in Chapter 2, RDF is a data-model for objects that specifies the relationships between them using XML syntax, while RDF Schema is a vocabulary for describing the properties and classes of RDF resources. The CIMValidate tool uses this CIM RDF Schema to validate a CIM XML file against the CIM standard, but RDFS lacks the ability to fully express all the conditions set in the rules.

OWL adds additional vocabulary for describing classes and their properties, including: “relations between classes (e.g. disjointness), cardinality (e.g. "exactly one"), equality, richer typing of properties, characteristics of properties (e.g. symmetry), and enumerated classes”[37].

OWL offers the ability to check for the basic CPSM rules: that properties exist, are of a specific type and are within a given range. Study of the OWL vocabulary and semantics indicates that the complex rules described in 6.4.6 are outside the scope of OWL. The language is designed for expressing ontologies and has limited first

order logic abilities, but as noted in previous publications[38][39], OWL lacks the ability to fully express all logical rules without extensions.

The Eyeglass open-source validation engine is capable of validating a CIM XML file against an RDFS or OWL standard and flagging any errors in the CIM data. Ignoring the problems of converting every rule into valid OWL, the validation can only be undertaken on a complete CIM XML file. For stand-alone validation this is not a problem, but for integration with the CIM Java Framework it would require each model to be exported as a full CIM XML file to validate even a single object.

6.4.7.2 Translating the Rules to native Java code

Since each CIM class has a corresponding Java class in the CIM Java Framework, each class can be given a cpsmValidate() function that returns a true or false value for each CIM Object. The translation of the rules into native Java code is straightforward even for the complex rules described in 6.4.6.

The compensatorType rules become:

```
//Start Rule: Given any of the below
If
   (
   // Start Rule: Given all of the below
   (compensatorType.equals(CompensatorType.series) &&
     r!=null &&
     x!=null
   // End Rule
   )
   || // Or
   // Start Rule
   (compensatorType.equals(CompensatorType.shunt) &&
     maximumSections != null &&
     mVARPerSection != null &&
     nominalKV != null &&
     normalSections != null
   //End Rule
   )
//End Rule
) return true;
Else return false
```

While the more complex conformingLoadFlag rules in the EnergyConsumer become:

```
// Start Rule: Given any of the below
   if (
      //Start Rule: Given all of the below
      (conformingLoadFlat == true &&
        // Start Rule: Given any of the below
        (
          //Start Rule: Given all of the below
          (pfixed != null &&
           qfixed != null &&
```

```
        pfixedPct == null &&
        qfixedPct == null
       //End Rule
       )
    || // Or
       //Start Rule: Given all of the below
       (pfixedPct != null &&
        qfixedPct != null &&
        getLoadArea().getAreaLoadCurves().size() >= 1
       //End Rule
       )
    //End Rule
    )
//End Rule
)
      || // Or
    //Start Rule: Given all of the below
    (conformingLoadFlat == false &&
     pfixed != null &&
     qfixed != null
    //End Rule
    )
//End Rule
) return true;
else return false;
```

The simple true or false result, however, does not give any clues as to the location of the problem if an object fails the validation. To provide the user with additional feedback as to where the error has occurred, additional code can be added that provides notes for the user on the exact causes of any failure. This can either be inserted as an additional cpsmValidateErrors() function, or by modification of the existing function.

This increases the complexity of each rule by several orders of magnitude. The validation function for the compensatorType field alone now becomes:

```
//Create an empty ArrayList of Errors
ArrayList<String> Errors = new ArrayList<String>();


boolean subRule1 = false;
   // Start Rule: Given all of the below
   if (compensatorType.equals(CompensatorType.series) &&
      r!=null &&
      x!=null
   // End Rule
   ) subRule1 = true;
   else{
      if (!compensatorType.equals(CompensatorType.series))
         Errors.add("compensatorType is not CompensatorType.series");
      If (r==null)
         Errors.add("r is equal to null");
      If (x==null)
         Errors.add("x is equal to null");
   }

boolean subRule2 = false
```

```
// Start Rule
(compensatorType.equals(CompensatorType.shunt) &&
   maximumSections != null &&
   mVARPerSection != null &&
   nominalKV != null &&
   normalSections != null
//End Rule
) subRule2 = true;
else{
   if (!compensatorType.equals(CompensatorType.shunt))
      Errors.add("compensatorType is not CompensatorType.shunt");
   If (maximumSections ==null)
      Errors.add("maximumSections is equal to null");
   If (mVARPerSection ==null)
      Errors.add("mVARPerSection is equal to null");
   If (nominalKV ==null)
      Errors.add("nominalKV is equal to null");
   If (normalSections ==null)
      Errors.add("normalSections is equal to null");
  }

//Start Rule: Given either of the below
if (subRule1 == true ^ subRule2 == true) Errors.clear();

return Errors;
```

This method now checks each sub-rule individually. If they are found to be false it adds an entry to the list of Errors. At the very end, it checks if either of the sub-rules are valid. If so the object itself is valid and Errors that were flagged indicated that one of the sub-rules failed. This will always be the case, however, given that each sub-rule requires disparate values of compensatorType and the OR operator is replaced with an exclusive OR (XOR). If this XOR is true then the Errors list is cleared since the failure of one sub-rule is to be expected. However, if they both fail then the list of Errors is returned.

This method of applying the rules, however has two major drawbacks. Firstly the validation and error reporting methods are time consuming to code for each class and secondly the validation is performed by hard-coded rules that are compiled directly into the system. This reduces the flexibility of the validation engine, providing only a single validation standard that is hard to change and substantially increases the amount of work required to add any extensions to the system.

The approach does, however, offer three major benefits: all the CPSM rules can be successfully encoded; the native code allows even large scale models of 60,000+ objects to be full validated in under a second; and individual objects can now be validated within the CIM Java Framework without requiring the entire model to be exported.

6.4.7.3 Generating encoded rules at runtime

The benefits of the native Java rules approach detailed in the previous section indicate that the logical rules discussed in section 6.4.3 can be converted into Java code with relative ease. Combining the benefits of the Schema approach with the native Java methods would provide the advantage of a rule-set that can be updated without requiring the user to re-code or recompile any of the system while retaining the benefits of the native methods: low execution time; the ability to validate a single object; and the translation of all the CPSM Minimum Data Requirement class-specific rules and general requirements into valid rules.

Rule definition schema

Since the CIM Java Framework already validates the CIM XML data to a level equivalent to that provided by the RDF Schema based validation of the CIMValidate tool, there is little justification for trying to implement a full RDFS or OWL validation engine. Instead, a simple schema is created to define the logical rules created from the CPSM Document.

Using a few simple XML node types:

- Rule – Contains any number of the other nodes listed below, as well as containing other Rules. The node also has a condition attribute of "all", "none", "any" or "either" which correspond to the logical AND, NOT, OR and XOR operators. When the condition is "all", every child-node must be true for the rule to be valid; "none" requires that all of the child-nodes be invalid; "any" requires one or more of the child-nodes to be valid; and "either" requires one and only one of the child-nodes to be valid.

- Value – contains a value and an operator (equalTo by default). The value and operator are used to perform a comparison with the attribute. The Value node is also used as a child node of Condition in combination with a reference to another attribute either in the same object or in a remote object.

- Condition – Allows for a comparison to be made between an attribute or association (both local and remote) and a value or another association

- LocalAssoc – A child node of Condition that refers to another attribute of the same class, or can be chained to refer to attributes of other associated objects. This is expressed in the form *localAssoc.remoteAttribute*. Here, the period indicates that *remoteAttribute* is an attribute of the object that the *localAssoc*

association points to. The chain is not limited to a single link. A reference of the form *localAssoc.remoteAssoc.remoteAttribute* has the value of the attribute of an object that is not a direct association, but is referenced by navigating through another directly associated object.

- Class – Requires an association to be of a specific class (or subclass thereof)
- Number – Requires an association with 0..n multiplicity to have a certain number of associated objects (combines with an operator, equalTo by default)
- Comment – Used to add a user-readable comment to describe more complex rules.

When the logical rules are translated into an XML schema, the rules for the compensatorType attribute of the Compensator class become:

```
<cim:Compensator.compensatorType>

<!-- Start Rule: Given any of the below -->
<strath:Rule condition="either">
  <strath:Comment>Checks that the compensator is either: operating
in series with an r and x set; or shunt mode with maximumSections,
mVArPerSection, nominalKV and normalSections set</strath:Comment>

  <!-- Start Rule: Given all of the below -->
  <strath:Rule condition="all">
    <!-- compensatorType must be equal to CompensatorType.series --
>
    <strath:Value>CompensatorType.series</strath:Value>

    <!-- r must not be equal to null -->
    <strath:Condition operator="not">
      <strath:LocalAssoc>r</strath:LocalAssoc>
      <strath:Value>null</strath:Value>
    </strath:Condition>

    <!-- x must not be equal to null -->
    <strath:Condition operator="not">
      <strath:LocalAssoc>x</strath:LocalAssoc>
      <strath:Value>null</strath:Value>
    </strath:Condition>

  <!-- End Rule -->
  </strath:Rule>

  <!-- Start Rule: Given all of the below -->
  <strath:Rule condition="all">

    <!-- compensatorType must be equal to CompensatorType.shunt -->
    <strath:Value>CompensatorType.shunt</strath:Value>

    <!-- maximumSections must not be equal to null -->
```

```
<strath:Condition operator="not">
  <strath:LocalAssoc>maximumSections</strath:LocalAssoc>
  <strath:Value>null</strath:Value>
</strath:Condition>

<!-- mVARPerSection must not be equal to null -->
<strath:Condition operator="not">
  <strath:LocalAssoc>mVArPerSection</strath:LocalAssoc>
  <strath:Value>null</strath:Value>
</strath:Condition>

<!-- nominalKV must not be equal to null -->
<strath:Condition operator="not">
  <strath:LocalAssoc>nominalkV</strath:LocalAssoc>
  <strath:Value>null</strath:Value>
</strath:Condition>

<!-- normalSections must not be equal to null -->
<strath:Condition operator="not">
  <strath:LocalAssoc>normalSections</strath:LocalAssoc>
  <strath:Value>null</strath:Value>
</strath:Condition>

<!-- End Rule -->
</strath:Rule>

<!-- End Rule -->
</strath:Rule>
```

The Class and Number node types are not applied in this particular rule-set. An example of their usage can be shown when translating the CPSM requirements "A PowerTransformer may be contained by a Substation or a VoltageLevel" and "Each PowerTransformer must have two and only two TransformerWindings"[6] into rules.

For the class Power Transformer:

```
Start Rule: Given any of the below
  MemberOf_EquipmentContainer must be of class Substation
  MemberOf_EquipmentContainer must be of class VoltageLevel
End Rule

Start Rule: Given all of the below
  Contains_TransformerWindings must contain two associations
End Rule
```

These can be subsequently translated to:

```
<cim:PowerTransformer>
<cim:Equipment.MemberOf_EquipmentContainer>
  <strath:Rule condition="any">
    <strath:Class>Core.VoltageLevel</strath:Class>
```

---

[6] This requirement can be deemed controversial since it is not unusual for a transformer to contain Tertiary or Quartiary windings. The current revision of the CPSM Minimum Data Requirements document, however, states that a transformer may contain only two windings, so for a CIM power system model to be deemed valid it must meet these requirements.

```
    <strath:Class>Core.Substation</strath:Class>
  </strath:Rule>
</cim:Equipment.MemberOf_EquipmentContainer>

<cim:PowerTranformer.Contains_TransformerWindings>
  <strath:Rule condition="all">
    <strath:Number operator="equalTo">2</strath:Number>
  </strath:Rule>
</cim:PowerTranformer.Contains_TransformerWindings>
</cim:PowerTransformer>
```

These 6 simple node classes allow all the logical rules created from the CPSM to be defined as XML.

Applying the Rules to CIM Objects

Once the CPSM document has been converted to a series of logical rules and a simple XML Schema has been created to define the rules in a machine-readable format, a system for applying these rules to the CIM data is required.

This validation engine must translate the rule expressions into Java code similar to that produced manually in section 6.4.7.2 including the ability to provide the user with feedback on which part of each rule wasn't met.

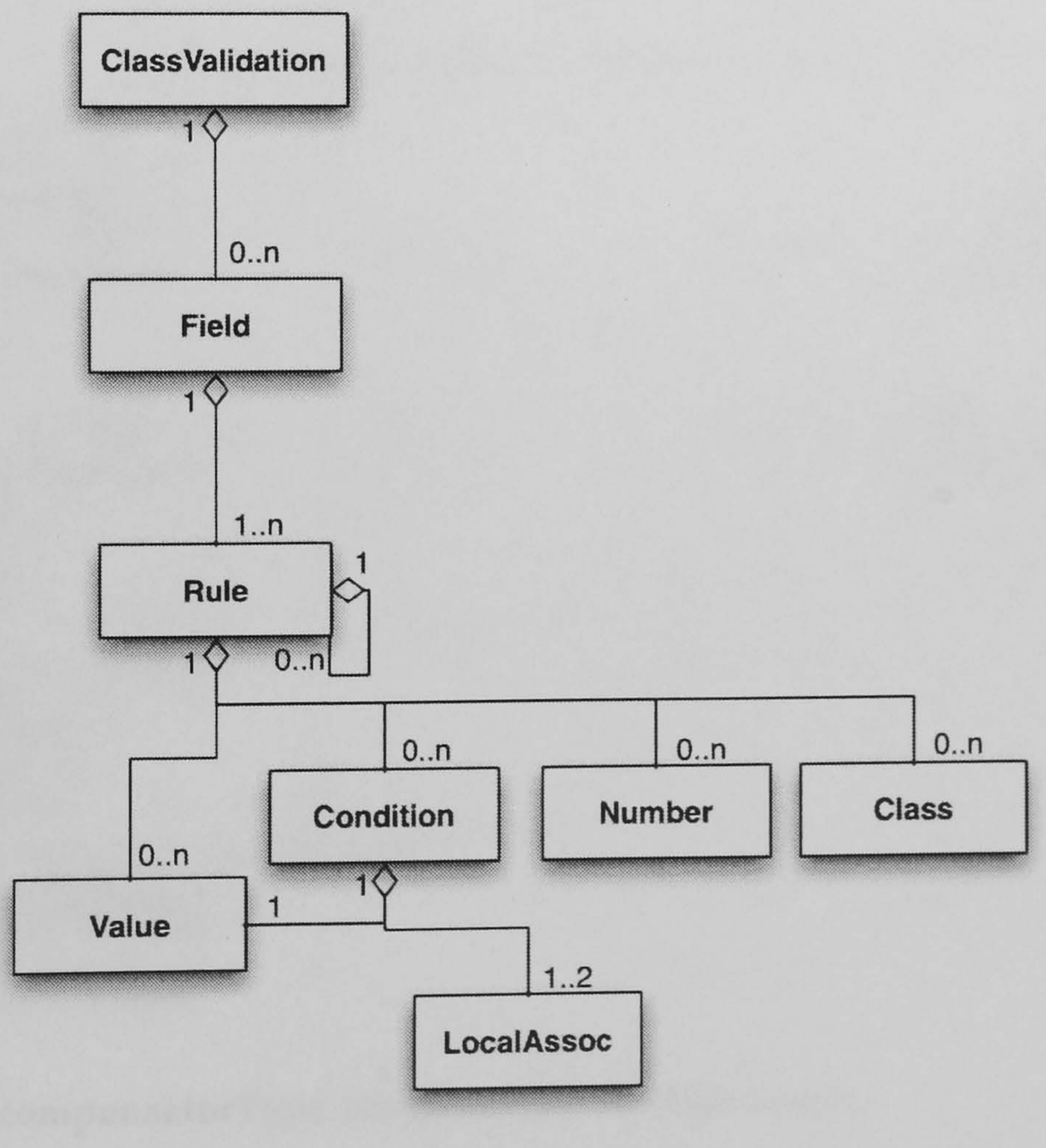


Figure 6.1 Validation Rules Class structure

The different types of nodes used to define the validation rules for each class can be expressed as classes as shown in Figure 6.1. Each class contains zero or more fields

that must be validated and each field in turn must have one or more rule defined. Each rule is comprised of any number of other Rules in the form of Value, Condition, Number and Class objects. Each Condition object is similarly made up of either one LocalAssoc and one Value object or two LocalAssoc objects.

Each class validates itself and passes the result up the tree to its parents, which uses the validation results from its children to in turn determine its own validation results. This continues until the top of the validation tree at which point the final validation result for each object is produced.

Figure 6.2 below illustrates this process for the compensatorType field of the Compensator class:

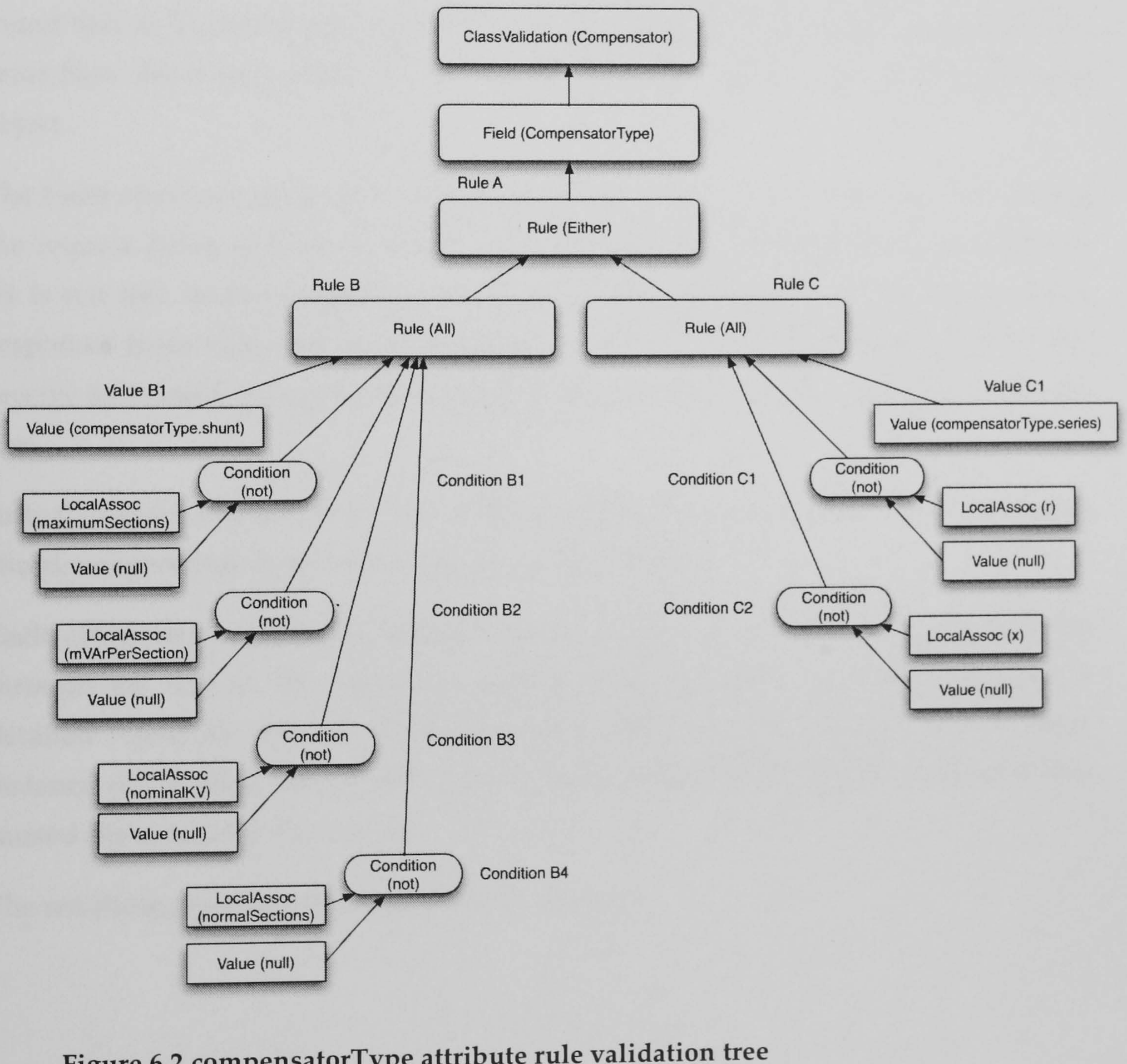


Figure 6.2 compensatorType attribute rule validation tree

For simplicity, this example assumes that no other fields in this class require validation. As shown in the diagram, the validation result of the Field class is dependent on the result of Rule A, which itself requires either Rule B or Rule C to be

valid. Rule B requires Value B1 and Conditions B1-4 to be valid while Rule C requires Value C1 and Conditions C1 & C2 to be valid.

The Value object uses its own value in combination with the value of the Field and the defined operator to determine its validity and passes the result back up to the Rule. Each Condition object has a LocalAssoc object that extracts the value for the specified local attribute, and a Value object that contains a single value. Each Condition object then uses these two values, along with its operator (equalTo, not, greaterThan etc.) to determine whether its result is valid or invalid.

The other benefit of this approach is that, should a Field declare itself as invalid, the point of error can be located by requesting each object in the tree to return the child objects that caused it to be invalid. For example, if an instance of Compensator found that its CompensatorType field was invalid it could request the reason for the error from the corresponding Field object within the Compensator ClassValidation object.

The Field object would in turn request the reason from Rule A, which in turn passes the request down to Rules B and C. If we assume the reason is because Condition B4 is not met because normalSections is not set, then Rule B would receive valid responses from Value B1 and Conditions B1 thru B3, but at Condition B4 it would receive an invalid response indicating that the error has occurred at this point in the tree.

Rule C, would also fail, since Value B1 and Value C1 are mutually exclusive, and a shunt compensator does not require an r and x value.

Each object that caused the failure (in this case Condition B4) is passed back up through the tree to the validation engine. This can then be used to produce a detailed report for the user listing the rules that were not met by that particular instance of the class, along with more a detailed list of the specific sub rules that caused the error and the reasons.

The resulting report is shown in Figure 6.3 below:

Compensator - CP-C1KV-345ST-HOLDENDV-ECARCO-ECAR

The following rules were not met by this instance

Rule - **Given either of the below**

Checks that the compensator is either: operating in series with an r and x set; or shunt mode with maximumSections, mVArPerSection, nominalKV and normalSections set

Rule - **Given all of the below**

[value] compensatorType is equal to CompensatorType.series

[condition] not (all of r (Value) must be equal to *<null>*)

[condition] not (all of x (Value) must be equal to *<null>*)

The rule failed because the conditions below were not met

[value] compensatorType is equal to CompensatorType.series

The rule failed because the conditions below were not met

[condition] not (all of r (Value) must be equal to *<null>*)

The rule failed because the conditions below were not met

[condition] not (all of x (Value) must be equal to *<null>*)

Rule - **Given all of the below**

[value] compensatorType is equal to CompensatorType.shunt

[condition] not (all of maximumSections (Value) must be equal to *<null>*)

[condition] not (all of mVArPerSection (Value) must be equal to *<null>*)

[condition] not (all of nominalkV (Value) must be equal to *<null>*)

[condition] not (all of normalSections (Value) must be equal to *<null>*)

The rule failed because the conditions below were not met

[condition] not (all of normalSections (Value) must be equal to *<null>*)

Figure 6.3 Validation output report from invalid Compensator object

This auto-generated report highlights the specific parts of each sub-rule that were found to be invalid, providing the user with the information required to locate the errors in the original model. This report is generated from the original machine-readable XML report, providing the ability for other software to validate CIM Objects and then automatically interpret the results accordingly.

Conversion of Rule Object Definitions to Java Methods

As with the CIM XML import interface described in Section 4.4.4, the Validation classes use Java's Reflection technology to convert the references in the XML data into the appropriate values or fields.

For the Value object this involves using the Field's type (e.g. String, Double, Integer) to cast the string value of the node to that type and comparing it to the value in the CIM object being validated.

A similar approach is used in the Class object, which attempts to find the CIM Class of that name using Class.forName() method. A comparison between the specific class and the class of the Field's associated object can then be undertaken.

The Number object's comparison is achieved by first casting the node value to an integer. Since all associations of 0..2-n in the framework use an ArrayList in the CIM Class, the ArrayList.size() method can be used to find the current multiplicity of that particular association and compare it to the node value.

The Condition object's LocalAssoc object splits its node value into separate parts using the period character as the separator. For example, in the TransformerWinding class rule definitions the MemberOf_PowerTransformer. MemberOf_EquipmentContainer.MemberOf_Substation is split into three parts: MemberOf_PowerTransformer,MemberOf_EquipmentContainer and MemberOf_Substation. The methods to access these associations can be found by prefixing "get" to the start of these field names and calling them in sequence.

For this example, the TransformeringWinding object's getMemberOf_PowerTransformer() method would be called first, returning its associated PowerTransformer object. The getMemberOf_EquipmentContainer() method would then be called on the PowerTransformer object returned by the first method, which then returns a VoltageLevel object. Finally the getMemberOf_Substation() method is called on the VoltageLevel object which returns a Substation. Since this is the final object returned at the end of the sequence this is the object that would be used in the final comparison. In this case, since it is a CIM Object and not an attribute, the comparison would be with another CIM Object, either of a direct local association or obtained in the same manner. Should the final object returned have been an attribute of a remote object then the comparison could have been with a specific value using the Value validation class.

For all of these rule objects, any generated error (known in Java as exceptions) must be handled correctly. An exception can indicate one of three things: the rules are malformed (e.g. A Field is a Double Attribute but the Comparison value is "XYZ"); the rule has not been met (e.g. in the example above, the PowerTransformer's getMemberOf_EquipmentContainer() returned a Substation or null object the subsequent method, getMemberOf_Substation() would fail); or that there was an

unexpected error in the program's execution. By dealing with the exceptions in the correct manner, the user is provided with the correct feedback as to the cause of any failure and the program's integrity is maintained.

Validation Class Initialisation

Each CIM Class that is detailed in the CPSM Minimum Data Requirements document has an entry in the Validation Rules XML file. Each entry in turn creates an instance of the Class Validation class that in turn creates an instance of the Field Validation for each field in the class that is to be validated. Each Field Validation object contains multiple instances of the Rule, Condition, Number, Value and Class objects in the same manner as the compensatorType example shown in Figure 6.2.

The Validation module accepts any CIM object, locates the appropriate Class Validation instance based on the class of the CIM object and passes the CIM object to the Class Validation object. The CIM object is then distributed through the tree to the other Validation objects that extract the values required to perform their validation operations. A validation process does not change the internal state of the Validation objects, allowing them to be used for multiple validations on any number of CIM objects.

The alternative, to create unique instances of each Validation object for every CIM object, would significantly increase the memory requirements of the system by adding tens or even of hundreds of additional objects onto every CIM object. For large models this could potentially result in millions of additional objects being added, significantly increasing the memory footprint of the model.

The importing and instantiation of the Validation rule XML file occurs when the system is initially started or whenever the XML file is updated. This removes the requirement of a validation process to import, parse and convert the rules entries every time a model of object is validated.

The validation process, however, still requires a significant amount of object casting and requires Java's Reflection technology to extract the data from each CIM object. This, combined with the overhead of generating errors descriptions and passing them up through the validation tree adds a significant overhead to the execution time compared with hard coded rules as described in Section 6.4.7.2. When validating 19,000 instances of the Compensator class, the hard-coded rules completed their validation (including errors reporting) in 0.024 seconds. The same test was undertaken using the Validation objects, which required 3.721 seconds to

complete the test. This indicates that the Validation objects system of validation is over 150 times slower than using hard coded rules. The time taken to validate large models, however, can be considered acceptable given that, even very large-scale CIM power system models of 500,000 to 1 million objects can be validated in two to three minutes. The software scales linearly with the size of the model and, as with the importation process, the validation engine benefits from multi-threading support, further reducing the execution time for large models.

6.5 Chapter Summary

The different methods of validation offer their own advantages and disadvantages. Utilising an existing RDFS/OWL validation engine along with an appropriate schema has the advantage of utilising existing tools and an open-standard schema. The major disadvantage of using an RDFS/OWL approach is its inability to successfully express every requirement of the CPSM standard at the current time.

Hard-encoded rules offer a significant speed advantage, allowing very large-scale models to be fully validated in a matter of seconds. There are, however, major disadvantages to this approach since hard coding the validation rules reduces the execution time at the expense of flexibility. Any changes to the CPSM standard requires significant re-programming of the software, while the addition of any new validation profiles require a significant amount of software development work.

The Validation Object tree approach allows all the CPSM rules to be expressed in a logical manner and the validation engine itself provides an automatic system of identifying the exact causes of any errors within an object. The integration with the CIM Java Framework also allows single objects to be validated in a fraction of a second, independent of the overall model size. While significantly slower than natively encoded rules, the execution time is still acceptable for large-scale models, and can be considered an acceptable trade-off given the advantages of the approach. This novel and powerful approach to validation builds on the openness and flexibility of the CIM Java Framework.

7 Automatic Network Integration

7.1 Chapter Introduction

Given the interconnected nature of the electricity grid, Distribution Network Operators (DNOs) and Generation Companies have to exchange power system data with each other and with the Transmission Network Operator (TNO) to ensure their network interoperate correctly. This process can involve the exchange of data in a number of different formats, often requiring manual translation into the company's own proprietary format before being integrated with their own power network model for interpretation and analysis.

For simple networks, manually joining the network models may seem to be the most obvious solution given the ease with which a trained network engineer could accurately identify the corresponding points of inter-connection. As the complexity of the networks increase, however, the number of points of inter-connection rises and the process becomes increasingly time consuming. With this manual integration, there is also an increased chance of errors being introduced to the output data, which could potentially lead to inaccurate data being introduced into an operational system. Such an event could result in a company incurring the additional expense to rectify these errors and any additional problems caused by the erroneous data. Taken from a TNO's perspective, an automated system for receiving network models from DNOs and Generators and automatically validating and combining them with a TNO's existing model would save time and significantly reduce the potential for human errors to be introduced into the data.

The adoption of the Common Information Model (CIM) for power systems by network operators provides a common, open format for representing power networks, but the problem of integrating network models from multiple sources still exists. This chapter proposes novel solutions to the problem of integrating network models in CIM format.

7.2 Representing Inter-Network Connections

When connecting two power system models together it is necessary to identify the points on each network that are to be electrically connected. This problem has been discussed in Chapter 3.5.5 and a solution proposed. By using the Network

Connection Point class described in 3.5.5, a network model in CIM format can define the points at which it connects to neighbouring networks.

7.3 Integrating Models of Identical Abstraction

With the exception of the simplest networks containing only a single Network Connection Point, specifying Network Connection Points for each network alone will not provide sufficient data to allow the accurate amalgamation of power system models. A method for automatically pairing Network Connection Points from separate models using (whenever possible) the existing data contained within the network model is therefore required.

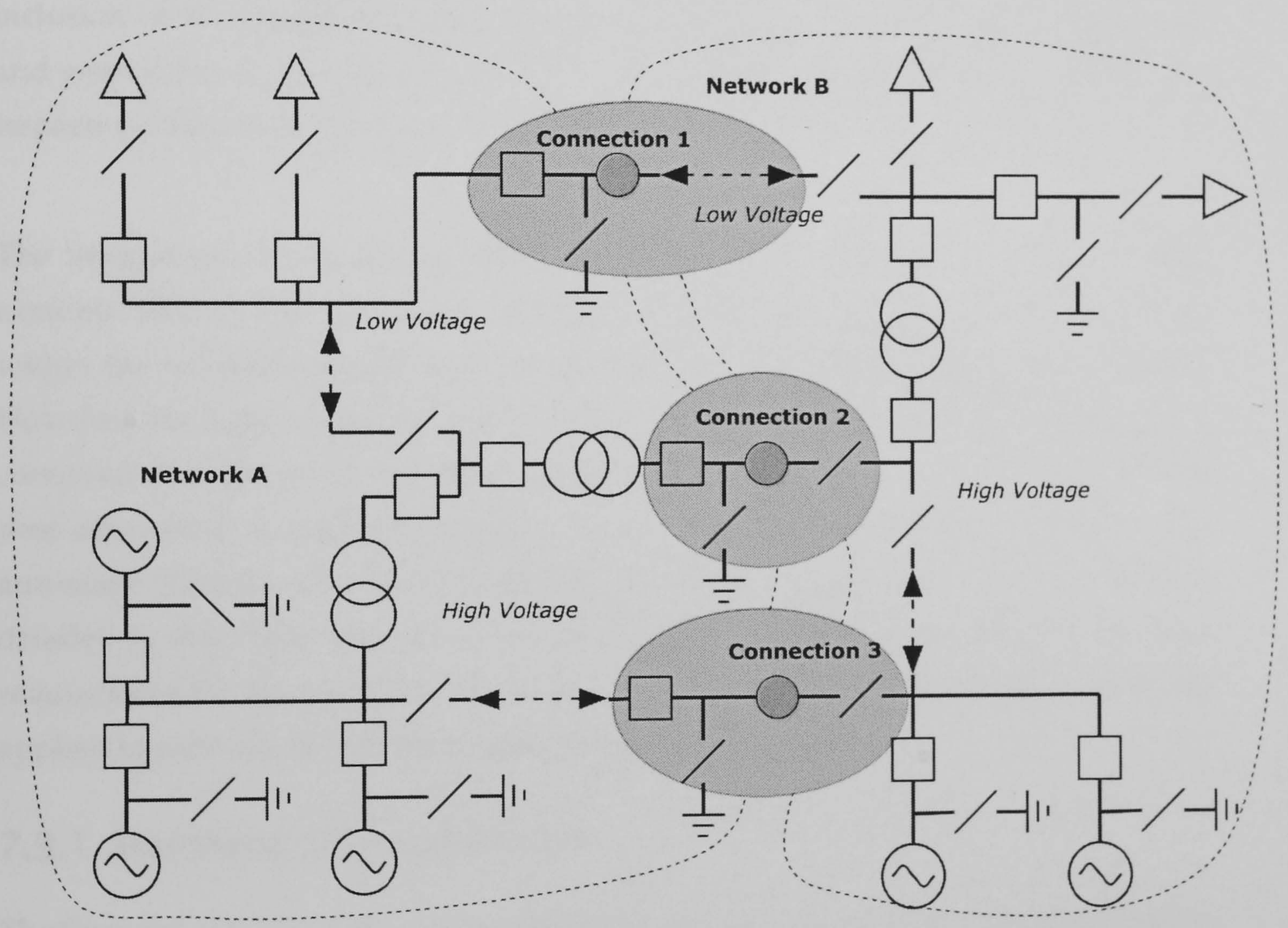


Figure 7.1 Network A and Network B with the inter-connection points marked

Figure 7.1 shows two sample networks, A and B, each containing multiple loads and generation sources. To integrate the two models, the process requires three stages:

1. The process must automatically pair the connection points from each network
2. It must then identify areas in the network models that overlap

3. Finally, the process resolves the issue of duplicate components in the overlapping sections

Upon successful completion of these three stages, the two network models will be combined to form a fully interconnected network model.

It is beneficial for a network operator's power system model to contain elements that represent portions of the other operators' connected networks, so that their system does not terminate at an unrealistic point such as a transformer winding or at the end of a line. Instead, network models will overlap at the points of interconnection, with sections of neighbouring networks or equivalent representations being included within a power system model. This allows the inclusion of all equipment within the same substation as well as equivalent loads and generation sources at the extremities to represent how this connected network impacts on the original network modelled at a high level of detail.

The integration can be accomplished manually with little difficulty for a simple example such as that shown in Figure 7.1 because the relative lack of complexity within the networks makes visual identification of overlapping areas a trivial task. However, for larger network models where each network will contain thousands of components and tens or hundreds of interconnections, manual integration is both time-consuming and prone to human-error. The methods developed to allow the automatic identification and integration of power system models in CIM format detailed in this thesis can be used with networks of varying complexity with little requirement for human intervention and have been successfully implemented and applied to network models of considerable size and complexity.

7.3.1 Matching Voltage Levels

The first step when locating the matching partner for a Network Connection Point is to identify all points within the other network with the same voltage level. This can be accomplished by locating the VoltageLevel associated with either the Connectivity Node or Terminal identified as the point of connection. From this VoltageLevel instance the nominal voltage of that portion of the network is obtained from the BaseVoltage associated with that VoltageLevel. There are, however, cases when the VoltageLevel association will not be present for the chosen component because the original model file only contains VoltageLevel associations for a minimal number of topological and conducting equipment components. In these

cases only core components such as Transformer Windings, Line Segments, Generators and Consumers will include associations to a VoltageLevel. Stepping outward through the network from the selected component until a piece of equipment with an associated VoltageLevel is found within the same topological node will provide the voltage for that particular section of interconnected equipment in the network.

7.3.2 Creating Component Identifiers

The object-oriented nature of the CIM representation of the power system allows each individual component of the CIM network to have an identifier created for it based on its attribute values, class type and associations. In this way, the probability of a Transformer Winding in one network being a representation of the same Transformer Winding contained in another network can be evaluated by comparing the two identifiers.

This comparison process makes use of the CIM Java objects architecture described in in Chapter 4 by integrating the comparison functionality into the object framework itself. Every object has a *compareTo* function that allows it to compare itself to any other object within the system. The default function, inherited by all the objects in a model, uses only an object's internal attributes and association relationships to compute the hash identifier and perform the comparison. This function is defined at the top level of the class hierarchy and then inherited by all subsequent child classes.

The *name* attribute of each component and other String values including *description, comment* and *aliasName* are ignored during the calculation given the likelihood that the internal naming conventions for equipment and user-readable descriptions and comments will differ depending on the source of the network model.

This method of calculating hash identifiers for each component, however, does not provide completely unique values. It is possible that simple components within a network model will produce identical hash identifiers when they have a small number of attributes with a limited number of different combinations. For example, a switch will always contain two Terminal associations and often only a single attribute to denote whether the switch is opened or closed. For a large model there will often be identical hash values across multiple instances of the same class when the name assigned to the component is ignored. It follows that comparing the

identifier of the first overlapping component of each network is not sufficient for matching connection points with the desired level of accuracy.

7.3.3 Creating Network Section Identifiers

A method of "spidering" through the network, combining hash identifiers from each component on each "step", allows for a comparison between network sections, or entire network models, decreasing the chance of a mismatch with each increment. This process reports the number of steps taken through each network from the common starting location until a mismatch, indicating a discrepancy between the networks, is found.

The spidering not only includes components that represent the physical network, it uses all the CIM objects associated with a particular object. These include objects that represent voltage levels, load curves, load areas, measurement devices, equipment containers (such as Substations) and various other objects that do not represent actual physical conducting equipment but still represent essential attributes of the power system model.

7.3.4 Weighting Connection Pair Matches

For the examples given in Figure 7.1, Connection 1's pair of Network Connection Points will be identified during the first stage since they are the only two Network Connection Points at the Low Voltage level. For the remaining two connection points, both in the High Voltage level, the process will then compare any overlapping network portions to find which provides the best match for each.

Ideally, after the process has compared each Network Connection Point with all other potential points in the connecting network, each point will have a unique partner in the other network. However, should a situation occur where no clear match is found for one or more pairs of connection points due to either a) there being no high-weight match or b) more than one high-weight match, then the process will be unable to automatically pair the connection points and thus require external input to continue.

The analysis of the best match requires the comparison of the weightings for each component in the network. By repeating this process for the other possible matching connection points a series of weightings can be calculated and then ranked to provide a best match. If one weighting is significantly higher than the others then it offers the best chance of a match.

The process involves comparing the two objects, one from each network, at each stage of the traversal and returning a *weighting* value to denote how accurately they match.

This weighting is calculated using the following algorithm:

```
If the two objects' classes do not match return 0
Otherwise
   Set weight to 1;
   For each non-String attribute
      If the attributes do not match
         If the attribute is numeric (Integer or Floating Point)
            difference = (lowest attribute / highest attribute)
            Set weight to weight*difference
         Otherwise if the attribute is boolean
            Set weight to weight*0.8
   For each association
      If the Association is 0..n
         If the number of associations doesn't match
            difference = (lowest number / highest number)
weight = weight * difference
   return weight
```

This algorithm provides a weighting of between 0 and 1 for each comparison, with 1 indicating a perfect match and 0 indicating they are either of a different class or contain no comparable attributes or associations.

These weightings can be combined for an increasing number of traversal steps by multiplying the individual weightings together to obtain a weighting for each connection point.

By repeating this process for all the available connection points a matrix is formed with the corresponding weightings for all possible matches of the available connections points in each network shown in Figure 7.1.

NCP	**A1**	**A2**	**A3**
B1	0.82	0.0	0.0
B2	0.0	0.92	0.65
B3	0.0	0.72	0.81

Table 7.1: Weighting distribution for Network Connection Point matching

The two networks A and B shown in Figure 7.1 each contain small network segments from the neighbouring network, resulting in an overlapping area of shared network segments highlighted in the diagram. Within the overlapping

sections of each network, minor discrepancies in component parameters and switch statuses occur due to the differences in networks produced by each source application. Although the overlapping sections differ by only a small margin, it is enough to prevent the components from being completely identical. The weightings in Table 7.1 are produced after a single step in the traversal from the potential connection points, including only those connected components one step into the neighbouring network.

Since Connection 1's surrounding components (ignoring any Connectivity Nodes and Terminals hidden from this diagram) consist of a Disconnector, Breaker and Line, this produces a class mismatch with points 2 and 3, both of which are surrounded by two Disconnectors and a Breaker. This results in a weighting of 0 between Connection 1 in network A and Connections 2 and 3 but a high weighting with its companion, B1. Connections A2 and A3, however, both have relatively high weightings with two possible connection points in the neighbouring network due to the similar network structure.

In this case the automatic matching process fails due to there being more than one possible match for one or more connection points. If the overlapping sections were larger, a second traversal step would remove this ambiguity, but since the overlapping sections contain only a single component per connection point from the neighbouring network the comparison process cannot proceed further.

The process will automatically match those points for which there exists only one possible match. For the remaining nodes, the user will be shown the calculated matches with the highest probability of being matched together with a small graphical diagram to represent the portions of the network. They will then be asked to either approve or alter the matching pairs before proceeding.

When each Network Connection Point has a valid partner in the connecting network, the areas from each model that represent portions of the neighouring network are deleted. The resulting two networks are then combined at the three connection points resulting in a fully contiguous interconnected network. The processes used to perform this join will be described further in Section 7.5.

7.4 Integrating Models at Different Levels of Abstraction

As discussed in the previous section, components within a network and even network sections can be compared for equivalence. When a mismatch occurs the process will compare the two components that produced the mismatch, but if the

weighting returned is too low then the process concludes that the networks are dissimilar.

However, instead of assuming that any mismatch between two network sections indicates a non-comparable network, it is possible to instead analyse the network at the point of divergence to ascertain whether the two networks represent the same section but at differing levels of detail.

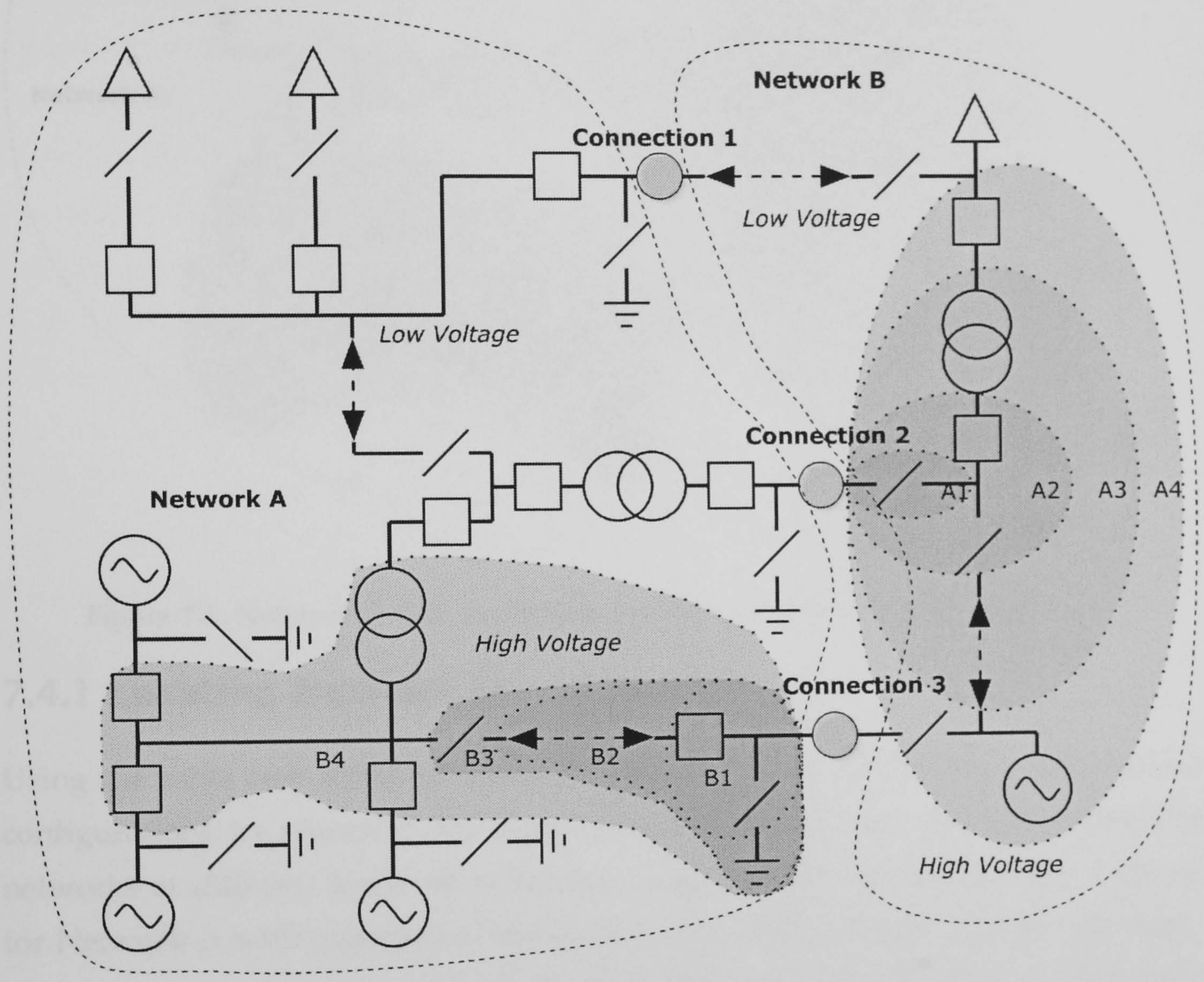


Figure 7.2: Network A with a simplified portion of Network B attached

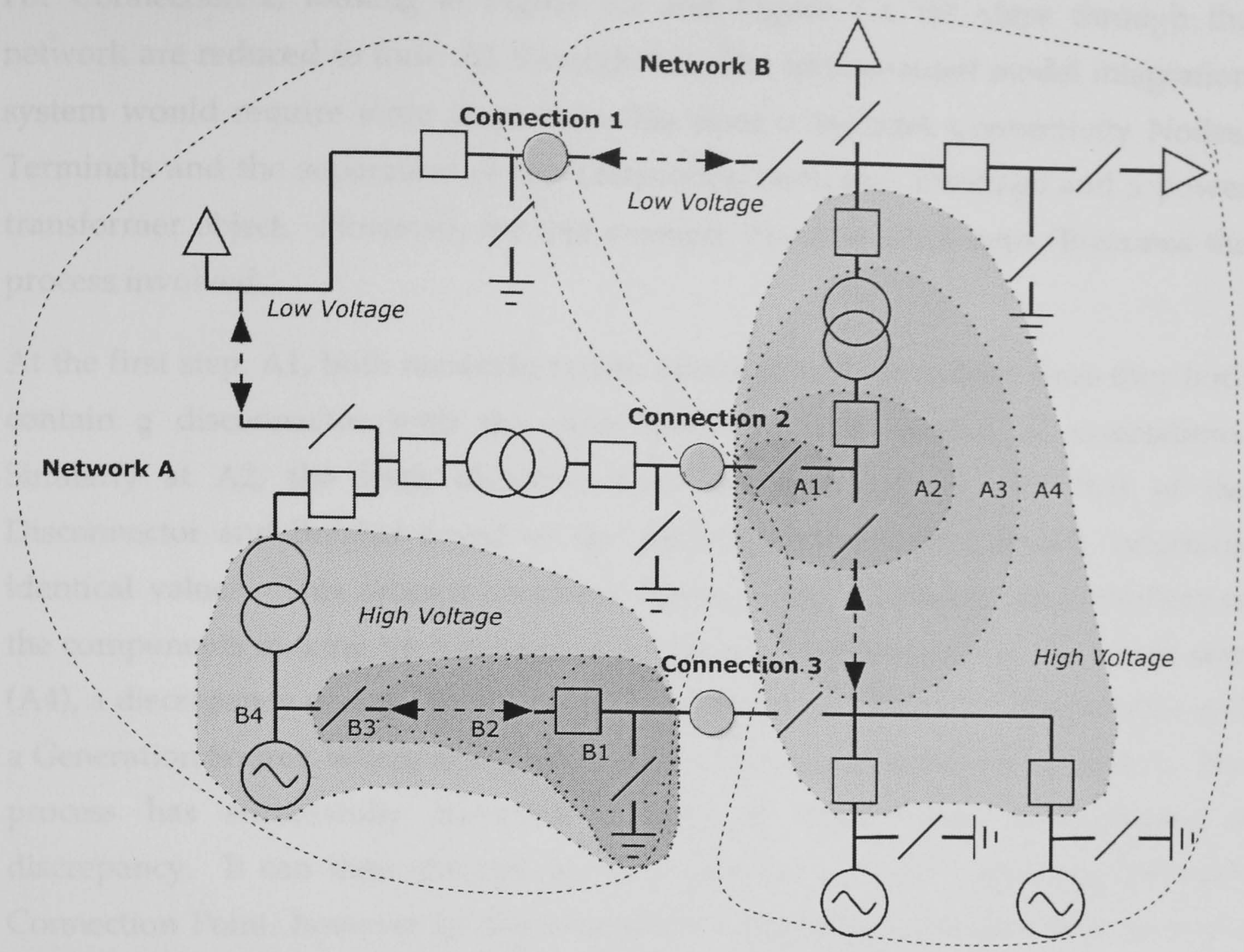


Figure 7.3: Network B with a simplified portion of Network A attached

7.4.1 Locating Network Discrepancies

Using the same two networks shown in Figure 7.1, we can produce two network configurations to represent the same overall network but with the connecting networks at different levels of abstraction. Figure 7.2 shows the network diagram for Network A with a portion of Network B included, but with a single load and a single source used in place of the multiple loads and generation sources. Similarly, Figure 7.3 represents the network diagram for Network B with a portion of Network A included with its loads and generation sources replaced with a single load and single generation source. These represent the network models each operator uses, containing a section of the neighbouring network that impacts on the functioning of their own system.

For networks such as those shown in Figure 7.2 and Figure 7.3, a discrepancy in network topology will occur when the spidering reaches the equivalent loads and sources. The process can then decide, depending on the level reached, whether this discrepancy indicates that the network sections are equivalent and that it has reached an equivalent load or source or whether it indicates the wrong connection point.

For Connection 2, looking at Figure 7.2 and Figure 7.3, the steps through the network are reduced to four: A1 through A4. The implemented model integration system would require more steps than this since it includes Connectivity Nodes, Terminals and the separation of the Transformer into two windings and a power transformer object. However, for this example the simplified steps illustrates the process involved.

At the first step, A1, both networks return identical hash identifiers since they both contain a disconnector with the same attributes and number of associations. Similarly at A2, the hash identifier from A1 is combined with that of the Disconnector and Breaker found as the process traverses the network, returning identical values. This process continues through A3, combining the identifiers of the components making up the line and transformer models until at the fourth step (A4), a discrepancy occurs. In Figure 7.2 the process encounters a Disconnector and a Generation Source, where as Figure 7.3 has a Disconnector and two Breakers. The process has successfully traversed through 3 steps before encountering a discrepancy. It can then attempt the same process with the remaining Network Connection Point, however in this example it would fail at the first step since the first component in Network B at Connection 3 is a Ground Disconnector, compared with a Breaker at the same point in Connection 2.

A similar traversal is shown when comparing the section of Network A connected to Network B, illustrated with steps B1 through B4 in Figure 7.2 and Figure 7.3. Here, as with the first example, after 3 steps through the network the hash identifier for the network sections are identical, then at the fourth step a mismatch is found as the traversal in Figure 7.2 reaches an equivalent generation source where as Figure 7.3 contains three Breakers.

7.4.2 Comparing Differing Levels of Abstraction

Instead of assuming that these two networks are incomparable, the process can instead check whether the level of detail for the components after this point has been lowered, or non-essential components that do no affect the electrical properties of that section (such as open ground disconnectors or closed switches) have been removed. In both cases the electrical properties of the remainder of the network have been maintained but as a limited number of equivalent components. It is possible to compare these two networks of differing size by converting the higher detailed model down to its electrical properties and comparing the two overlaps at this level.

A simple example is the different levels of detail that can be used to represent a line between two substations. At a high level of abstraction, a single AC Line Segment object provides a sufficient level of detail for performing a basic analysis of the network. However, the operator of the network may store the data for the same line at a much higher level of detail. Instead of amalgamating the line into a single line segment, their internal model may store data on each line segment that is connected in series to form the complete Line.

While both models would be valid CIM representations of a line, to the process described in the previous section, these two representations are different network configurations and as such it will report a mismatch at the first AC Line Segment in each, since its attributes (such as resistance, reactance and susceptance) will differ depending on whether it represents the entire line or just a section of it.

The method must therefore be extended so that the comparison works on multiple levels of abstraction. If a mismatch is found when comparing the hash identifiers of two object, instead of assuming this means the network differs from this point on the comparison will be performed on an amalgamation of an increasing number of components from that point on.

7.4.3 Incremental Bus-Branch Conversion and Comparison

Comparing two networks using this incremental Node-Breaker to Bus-Branch conversion proves involves the replacement of the *compareTo* function described in Section 7.3.2 in certain classes for which multiple pass comparisons are used to compare objects at different levels of abstraction. For the example above, the process need only request the comparison between two AC Line Segments once when there are differing levels of detail. If the simple hash identifier-comparison failed, the AC Line Segment object would perform the multi-pass comparison transparently by overriding the default *compareTo* function.

Similarly, any piece of Conducting Equipment contains a *compareTo* function that, as well as performing the simple hash-comparison between itself and the target object, can compare the target object to an amalgamation of itself and an increasing number of surrounding components. This is accomplished by performing the topological simplification of the network, one level at a time to create virtual components to represent multiple interconnected components using the same algorithms used in the Node Breaker to Bus Branch conversion described in Chapter 5.

By incrementally increasing the number of network components within each bus and checking for a match with the target object each time, it is possible to identify if they are comparable and how many components from the network are represented by the target component. The most simplified representation for a component would be a single piece of conducting equipment (to represent a bus), and an equivalent load and source component computed from the attached generating units and energy consumers.

This simplified representation provides sufficient data to conduct a steady-state load-flow analysis of the network. As such, when the network segments are found to have identical Bus-Branch representations, the *compareTo* function returns a positive result. Along with the positive result, the function includes additional data detailing the components included in the amalgamation. This informs any application utilizing the function that the result was due to an amalgamated match rather than from a direct one-to-one comparison.

7.5 Joining Power Network Models

The CIM Java Objects storage framework described in Chapter 4 is used as a long-term storage repository for multiple power systems models in CIM format. It is within this framework that the integration process has been implemented. The integration process, however, must be capable of integrating network models without destroying their original configuration. For this reason, three methods of integrating the models have been developed:

- Hard Join
- Copy Join
- Soft Join

7.5.1 Hard Join

The Hard Join method is used for making permanent joints that alter both networks to create a single network model using the algorithms described above. Components within the overlapping sections from each model that are from the neighouring network are removed, and the associations in each replaced so that the connectivity node and terminal associations at each interconnection point to an object in the connected model. Each model is still maintained as a separate entity, capable of being separated from the combined network, and topological processing

in either direction (from A to B or from B to A) produces identical network structures.

7.5.2 Copy Join

The Copy Join method uses the same process as the *Hard Join* method described above. However, instead of having separate models with interconnecting associations, the process creates a new model containing copies of all the components from the two networks as a new CIM power system model independent of the two parent networks.

7.5.3 Soft Join

A Soft Join is used in cases where the core network is being joined to satellite network that must maintain its original configuration after importation. This method of integration requires an additional level of complexity within the classes of CIM objects linked with the physical topology of the network (Conducting Equipment, Terminals, Connectivity Nodes). These objects will contain multiple paths and requires a system of flags to indicate when a topological association is with an external model. Without these flags to indicate when a path is an overlapping section on an adjoining network, during the topological analysis, the overlapping section could be incorrectly interpreted as being an additional branch.

To allow this one-way join, the core network will have its associations modified so that the satellite network becomes an integrated part of the overall network, and any topology analysis starting in the core network, at the point of interconnection with the satellite network, will proceed into this satellite network, treating it as an extension of the core network. However, this link is one way: if the topology analysis starts in the satellite network, it will either stop, or use the overlapping network section in the original model if it has no knowledge of the core network.

7.6 Validating Integration Output

There are a number of ways to validate the output from the integration process to ensure that the algorithms have performed as expected. Manual checking of the resulting CIM XML file is a tedious process and only possible on relatively small models. Instead, using the other applications created for exporting, viewing and visualising CIM network models can be used to verify the output.

7.6.1 Exporting the Output

The test models, A and B shown in FiguresFigure 7.2 and Figure 7.3, were manually created by splitting an existing model, as shown in Figure 7.1, into two. The CIM XML output from the integrator can be verified by using the PSS/E exporter described in Chapter 5 on the CIM XML original file then on the combined Network A plus Network B output. By comparing the load flow results from PSS/E on each of the two files produced by the exporter, the validity of the resulting combined model can be determined. In this case, since the user has the original model, the results should be identical. When this test is used in a real-world situation the load flow results will highlight any major errors in the resulting model, e.g. overloading of a branch or isolated buses.

7.6.2 Viewing the Model in the Mercury Library

While manually checking a CIM XML file is time-consuming and impractical for large files, using the Mercury Library's model viewer, a web front end used for viewing and editing CIM Java objects, the associations can be navigated with ease and the interconnections checked. For the test models, where the names of the objects that should be paired are known in advance, this provides a simple means of checking that the objects have the appropriate bi-directional associations. For models where there is only limited knowledge of what the resulting network should be in CIM format, the model viewer still provides a means of checking that duplicate objects have been deleted by checking for the existence of multiple instances of objects with the same name. and that, at the interconnection points, there are no

7.6.3 Graphically Checking the Network Structure

In Chapter 8 a means of generating network diagrams for CIM models will be described. These diagrams provide a means of graphically checking that the output from the model integration tool is consistent with the expected result. The diagrams allow the user to check that the two models have been fully integrated, since failure to create the proper associations would result in two islanded networks, clearly visible in the resulting diagrams or irregular connections between equipment showing on the diagram.

For example, during development and testing of the software that implements the algorithm, a small bug in the code prevented the overlapping section being deleted

from one of the models. While not obviously visible from the resulting CIM XML, the graphical view clearly showed the additional network section as an extra branch connected to the intersection point.

The graphical checking is the simplest means of ensuring that the resulting network model is correctly interconnected, and combined with the ability to export this file to PSS/E provides a means of ensuring that the integration has created a valid, interconnected model that is electrically robust.

7.7 Uses for the Model Integration Process

This process of network model integration has three primary applications, which will be discussed in the following sections.

7.7.1 Forming Regional or National Network Models

Network Operators, must exchange network data between themselves and and any generation companies then connect to their networks. Each company provides models of their own networks to their connecting partners who use them to form an overall regional or national network model.

This is an obvious application for an automatic model integration process, automating a procedure that is both time-consuming and prone to error when conducted manually. The level of network data provided by the generation companies and network operators is often defined by regulatory bodies and, as such the network data exchanged will often have lower detail, or the minimum required by the regulator.

It is in this situation that the process for integrating networks of differing levels of abstraction is beneficial. Given a scenario whereby network operator A must provide a neighbouring utility, network operator B, with a model for a segment of its own network, it may choose to provide the network segment model at the minimum level of detail set out under the regulatory codes. Network operator B will then integrate this network segment with its own network model for analysis and simulation purposes. When B then has to provide its own network model back to A, the segment of A's model already integrated into B's network model will be of a lower detail level than A's own, internal, network model. By using the integration process for differing levels of abstraction detailed previously, this discrepancy will not prevent the process from correctly matching the overlaps between the two network models.

7.7.2 Creation of new power system models in the CIM

The second application that will make major use of the model integration process is that of creating power system models in the CIM format from scratch.

Currently, the overwhelming majority of CIM data being produced is exported from existing applications or databases by either converting the application's own data format to CIM or mapping the database schema onto the CIM ontology. This is sufficient for exchanging pre-existing data but, given the increasing adoption of the CIM by a number of power system software vendors, the creation of new power system models in the CIM may prove to be the most universally acceptable format, and thus most compatible in the near future.

The network model integration process described above has been used to allow the rapid creation of test CIM networks by allowing the reuse of CIM network segments commonly duplicated throughout a typical power systems network.

By creating a number of basic substation models in the CIM format, either within a utility or provided by the substation supplier to represent the common layouts of substations at the distribution and transmission level voltages, large network configurations can be quickly constructed by joining these substations together. Adding energy consumers and generation equipment at the appropriate points of the network (either as stand alone objects or as pre-existing models) allows large networks to be formed quickly, requiring minimal changes to the properties of the substation components.

Wind farms, for example, contain a large number of identical, interconnected groups of equipment that represent the turbine and its associated switches, measurement devices, etc. The model integration process allows an operator to create a CIM representation for each turbine then quickly duplicate that section of the model multiple times and interconnect them all to provide a detailed model of the entire wind farm.

7.7.3 Creation of planning scenarios

The third application is for planning engineers who will, as described in the previous section, create new network section to describe a proposed connection to the main network, whether it be for a supply or demand point. By accessing a full model of the existing network in CIM format with multiple network connection points defined to indicate all points on the network suitable for connection by

another utility, the planner can automatically generate a number of different network configurations by integrating their proposed network at any number of locations.

Each of these power system network models in CIM format can then be exported to an analysis package, as described in chapter 5, allowing each scenario to be analysed individually for suitability.

By using the Mercury framework described in Section 4.7, this process can be accomplished remotely either using a web browser interface or with a web service. This allows the planner to examine a number of possible connection locations prior to the submission of a formal proposal to the main network operator.

7.8 Future Work

The extensions to the CIM proposed in this thesis, combined with the network integration method, provides the core element in an automatic network amalgamation process. The process, however, is limited to network models that contain both the new Network Connection Point class, and contain overlapping sections of network that are electrically identical.

Future avenues of research will focus on enhancing the automatic pairing of network connection points so that the lack of overlapping sections of network will not hinder the automatic integration of networks. The power system network attributes, combined with the use of intelligent techniques and the inclusion of GIS data has the potential to further aid the automatic integration of network models. This can be achieved by checking the validity of the resulting network model when applied to real-world situations and by using the physical location of network components and their proximity to potential connection points in satellite networks

These enhancements, introducing knowledge and rule-based reasoning processes into the application, pose significant software engineering challenges, both to design and integrate the decision making systems and additional network component data into the existing application, then to create suitable data-sets for training purposes.

An intelligent system capable of automatically identifying how separate networks with multiple connection points should be interconnected even when there are no overlapping sections present, would potentially remove the primary limitation of the existing process detailed in this chapter.

7.9 Chapter Summary

The previous work into creating a framework for storing CIM data as Java objects provides a powerful foundation for creating applications that can process CIM data directly. The network model integration process described in this chapter makes use of the CIM Java framework requiring only the addition of a single additional class, to automatically integrate connected network models.

Allowing each object that is either a piece of Conducting Equipment, or a direct subclass thereof, to compare itself with any other piece of conducting equipment on multiple levels of network abstraction allows the automatic overlap detection to cope with the differing levels of detail that can cause problems when exchanging network data between companies.

By creating libraries of typical substation, line and generation plant systems, the model integration process was used to create ad-hoc network configurations for the testing of other tools and applications based on the CIM Java Framework. This included a basic load flow simulation application written to accept native CIM data, as well as an evolution of the CIM XML to PSS/E native format conversion process described previously. The ability to quickly create test cases of varying size, complexity and configuration in native CIM format has allowed the development and testing of these applications to proceed faster than the reliance on existing CIM XML test data or the manual writing of CIM XML files would have allowed. The same applications provide major opportunities for planning engineers who wish to both create new network models and integrate them with existing power system models to analyse the impact of connections at different points of the network.

These additional applications, combined with the network model integration process's ability to compare, match and integrate complex network models in native CIM format, rather than translating to a proprietary format, demonstrates that the CIM standard can be used for more than simply data exchange. This application further demonstrates the benefits of the CIM Java object storage framework, providing the foundation for an application to quickly perform complex network interrogation and manipulation.

8 Visualisation of Network Topologies

8.1 Chapter Introduction

As an object-oriented data format, the CIM provides a means of representing the direct interconnections between components as associations between objects, thus allowing the topological analysis detailed previously to be performed on a network model. This system allows an application to accurately interpret the topology of a network but, for models without an accompanying graphical network schematic, the topology of the network is incomprehensible to an end user.

Providing the end user with a visual representation of a network's structure is important for two reasons:

1. Having a visual representation of the interconnections between components aids the end-user in quickly interpreting the existing data

2. When altering or adding components to a CIM network, having a visual reference and the ability to point and click on a network diagram is arguably more user-friendly than manually altering a data file without a visual reference.

What is required is an application that can produce a coherent visual representation of a CIM power system model's network's structure with minimal user input. This chapter shows how an open and flexible CIM toolkit architecture with an RDF XML output enables the adaptation of existing data graphing tools to provide a solution to the power system visualisation problem.

8.2 Automatic Graphing Tools

Since CIM XML data uses the Resource Document Framework (RDF) to define the relationships between each object, it is logical to investigate the use of existing RDF graphing utilities to organise the data. There are a number of RDF graphing applications available [40][41][42], all written in Java.

Of these three tools, the HP Labs tool is aimed more at visually navigating the data rather than organising it all onscreen. The Salzbery Research application is closed source and thus cannot be modified or integrated with the existing *Mercury* framework. MIT's Welkin application, however, is a Java applet that allows entire

data sets to be viewed and organised onscreen and is available under a Berkeley Software Distribution (BSD) license, allowing it to be modified and integrated without incurring any legal penalties.

Welkin is a graph-based RDF visualisation program developed to allow data analysts to visualise the overall shape and cluster characteristics of a set of data. The program was written as part of the Semantic Interoperability of Metadata and Information in unLike Environments (SIMILIE) project, a joint project between the W3C, MIT Libraries and MIT Computer Science and Artificial Intelligence Laboratories.

The software, written in Java, uses the relationships between XML nodes to cluster data, attracting nodes that are joined together, and forcing apart nodes that are not interconnected. This uses algorithms similar to those previously proposed by Yongli & Malik [43], and Ong, Gooi & Chan [44] for network layout generation, but the tool itself is domain-agnostic so can be used with any data expressed in an RDF format.

This clustering process takes place graphically in front of the user, who can alter various attributes within the program (mass, attraction, repulsion, acceleration etc.) to alter the behaviour of the nodes and speed up or slow down the clustering effect (thought it can result in an unstable state where the graph never condenses and behaves in an erratic manner). The user can also interactively move individual nodes on the screen to manually influence the final shape of the resulting graph

Welkin can read RDF, RDFS (RDF Schema), the OWL Web Ontology Language and TURTLE (a subset of the N3 textual notation for the RDF) formatted data regardless of the other ontologies used within the data. Since standard CIM XML data uses RDF to notate relationships between CIM objects, Welkin can read standard CIM XML data files without modification. A full power system model encapsulated in CIM XML, however, can exceed several megabytes in size and contain thousands of XML nodes. While Welkin can load files of this size, its ability to organise them is severely hampered by the large dataset.

For small CIM XML datasets, however, the automatic clustering and reorganisation of the data can produce graphs that begin to mirror the topological structure of the network.

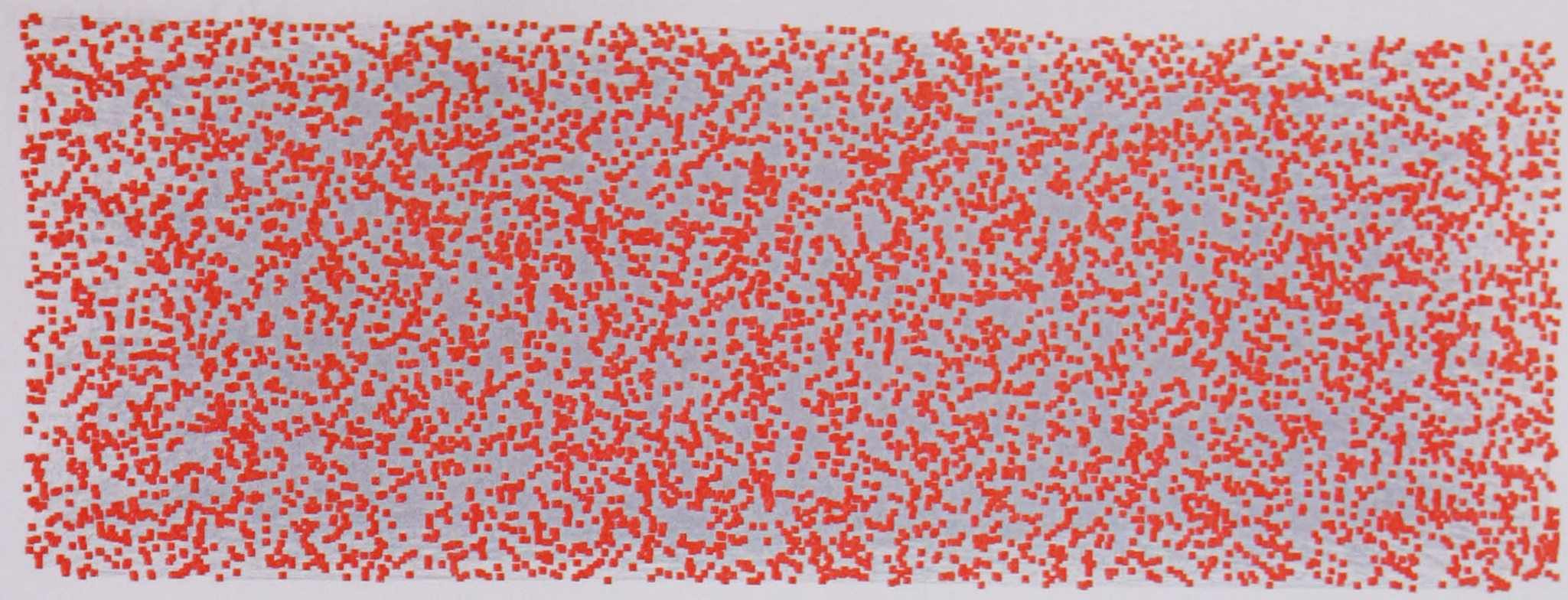


Figure 8.1 Welkin visualisation of Siemens 100 Bus Model before processing

8.2.1 Graphing Standard CIM XML data

Using an unmodified version of the Siemens 100 Bus CIM XML network file with Welkin requires several minutes to import the data into the application. Upon importation 6976 nodes are created and displayed on the screen to represent the network data as shown in Figure 8.1. The resulting graph displays all the objects within the CIM model and their associations, including all the components that define the network topology (Terminals, Connectivity Nodes and all classes that inherit from ConductingEquipment) and all other classes that represent non-topological components: voltage levels, measurement devices, substations, companies etc.

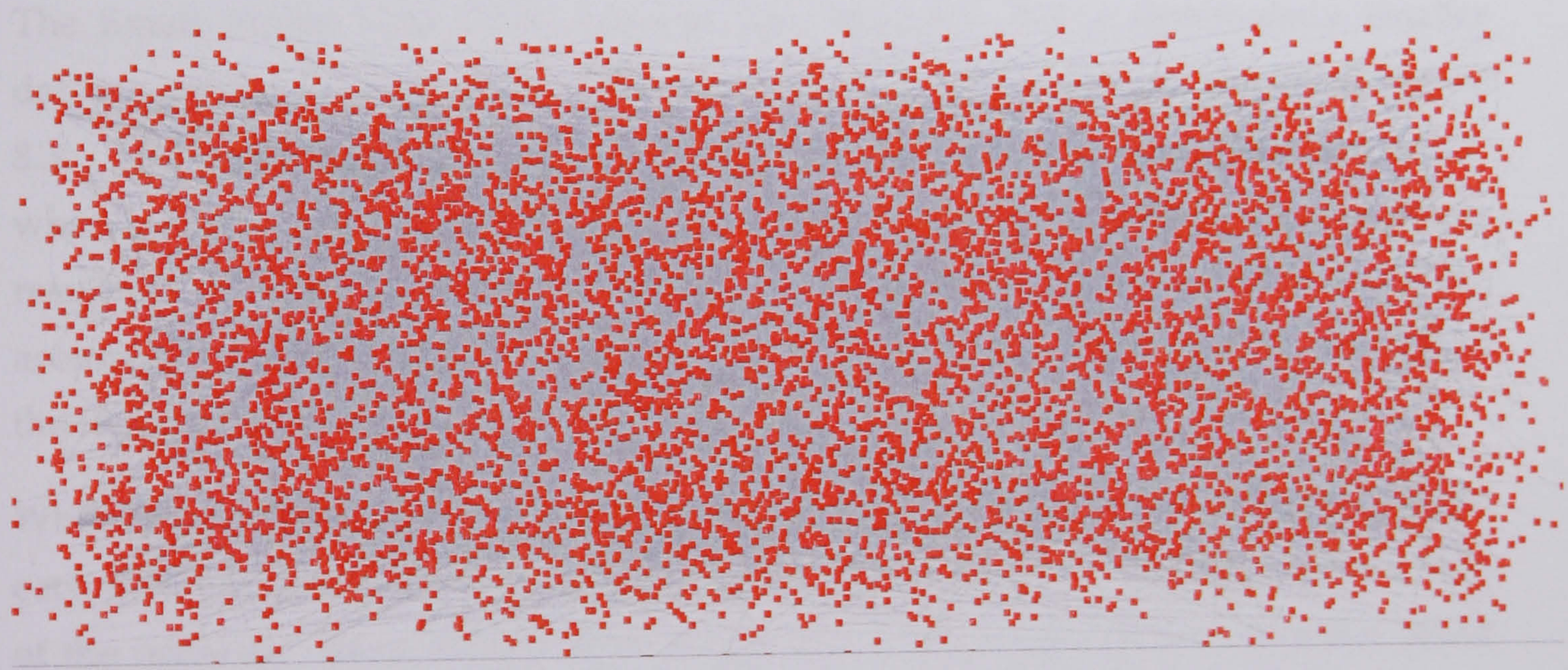


Figure 8.2 Welkin visualisation of Siemens 100 Bus Model after three minutes of processing

The graph of almost 7000 nodes, however, is too large to perform any meaningful analysis on. The algorithms for clustering the data fail to cope with such a large number of densely packed nodes and any attempt to ascertain the topological

structure of the network represented by this particular CIM XML files fails due to the large data set as shown in Figure 8.2.

Figure 8.3 Welkin visualisation of Small Model prior to processing

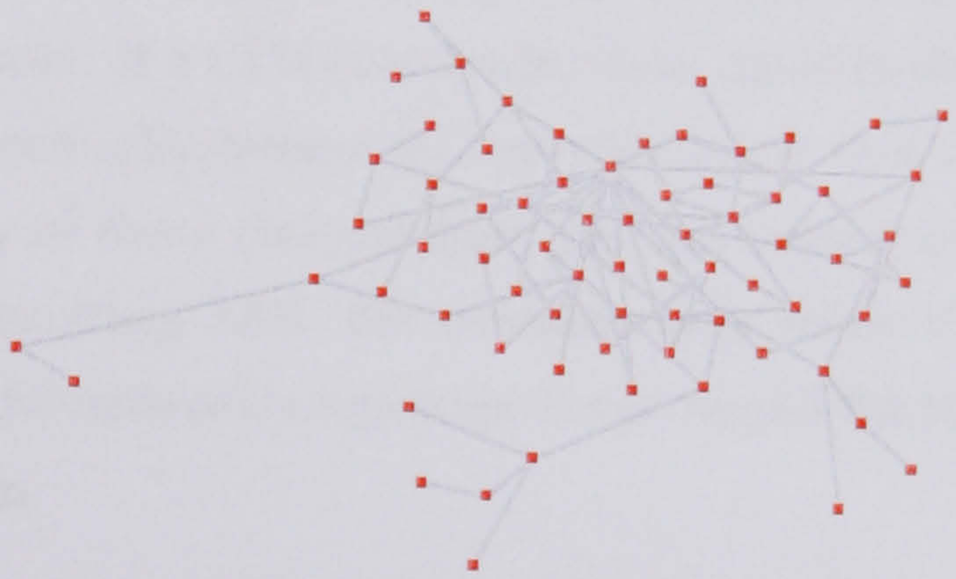


Figure 8.4 Welkin visualisation of Small Model after three minutes of processing

The Small Model CIM XML network file, however, has a significantly smaller dataset, producing only 70 nodes when imported into Welkin as shown in Figure 8.3. This produces a more manageable graph as shown in Figure 8.4. However, when using a full CIM XML file, the inclusion of non-topological components results in those nodes that are contained within a *VoltageLevel* container clustering around about their associated *VoltageLevel* and hence the diagram does not reflect the topological structure of the network

While this can be useful for identifying how many components are contained within each *VoltageLevel* it does not provide a means of visualising the topological structure of the network. Since Welkin does not provide a means of removing nodes based on their CIM class, the pruning of the XML file will have to be completed prior to importation.

8.2.2 Graphing Simplified CIM XML data

Since a full CIM XML network file has already proved to be overly complex for Welkin to analyse successfully, the dataset needs to be simplified.

8.2.2.1 Pruning a CIM XML Network Model

The first step is to prune the CIM XML file produce by omitting those components not directly related to the topological structure of the network. This can be accomplished using the *Mercury* software to export a limited set of CIM classes to XML. This set contains only those objects whose class inherits from *ConductingEquipment*, plus all the *ConnectivityNodes* and *Terminals*.

Using *Mercury* to export a reduced file prevents broken dependencies within the resulting XML file, since the export function checks the references within each object as it performs the export. If a CIM object references another object in the model that is not a piece of *ConductingEquipment*, a *ConnectivityNode* or a *Terminal* (or a subclass that inherits from any of these classes) then that reference is omitted from the XML file produced. The resulting XML file contains only those classes and subclasses required to describe the network's topology removing all the superfluous nodes and any references to them.

When importing this reduced CIM XML file into Welkin, the Siemens 100 Bus Model is still too large, with too many network components to provide a recognisable overview of the network's structure. The resulting graph, while several hundred components smaller than the full CIM XML file, is not discernibly different. The large number of densely packed, loosely connected components on screen once again prevents the automated clustering algorithm from condensing the network sufficiently to provide a useful representation of the structure.

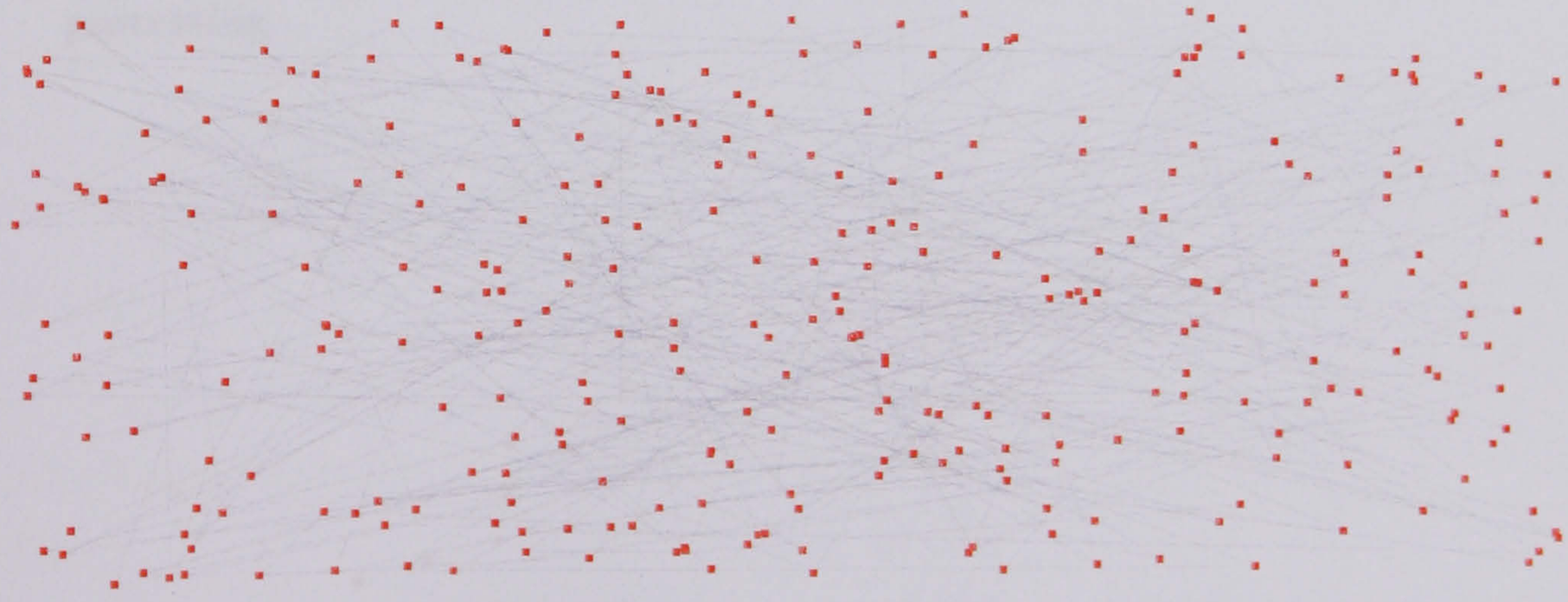


Figure 8.5 Welkin visualisation of the reduced format Langside & Cathcart model prior to processing

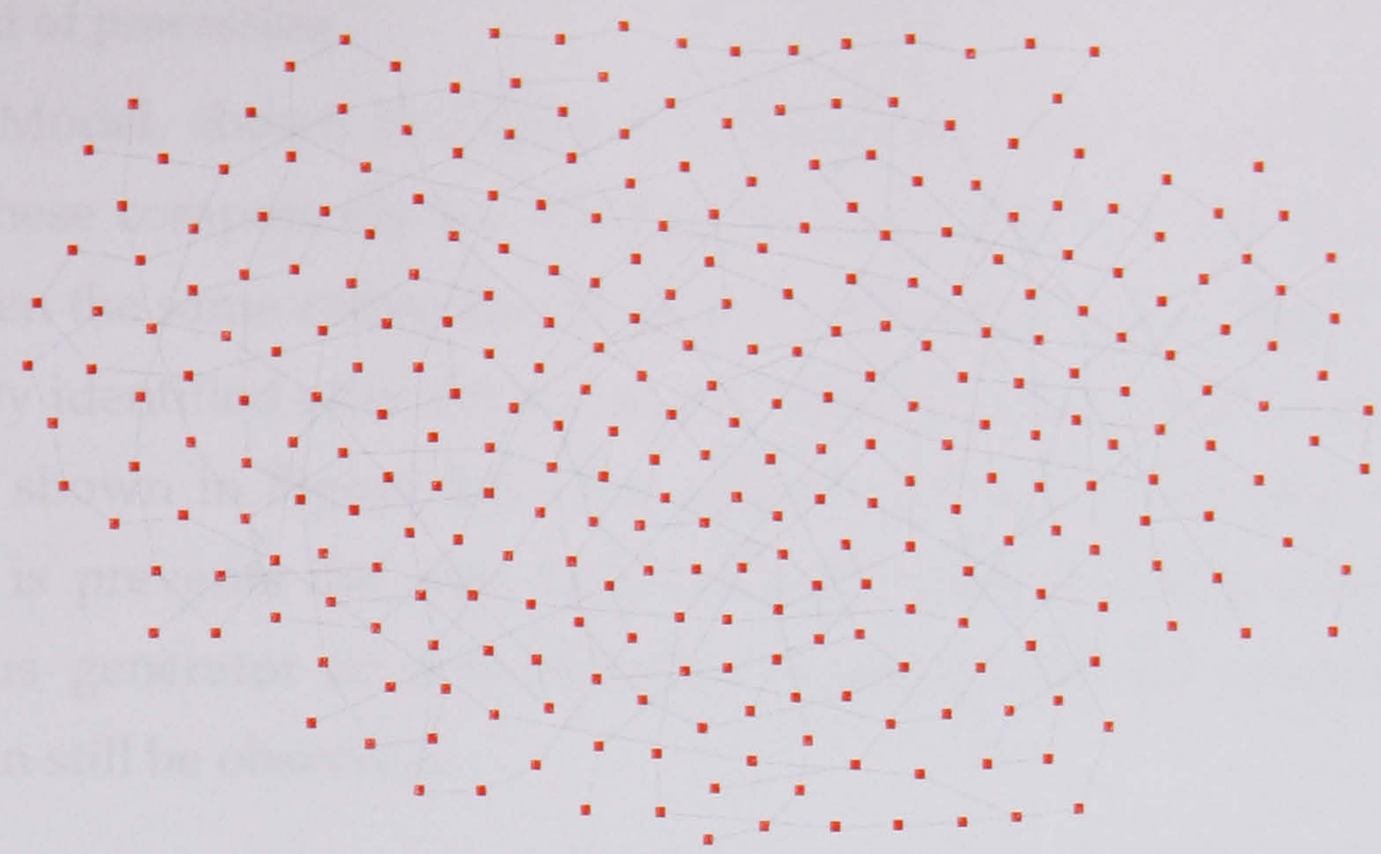


Figure 8.6 Welkin visualisation of the reduced format Langside & Cathcart model after three minutes of processing

The Langside and Cathcart network, a CIM power system of 307 objects, created for testing the *Mercury* software, was similarly pruned and imported into the Welkin application producing the graph shown in Figure 8.5. While significantly smaller than the Siemens model, the structure of the network is still not obvious even after the automatic clustering process has run for several minutes as shown in Figure 8.6.

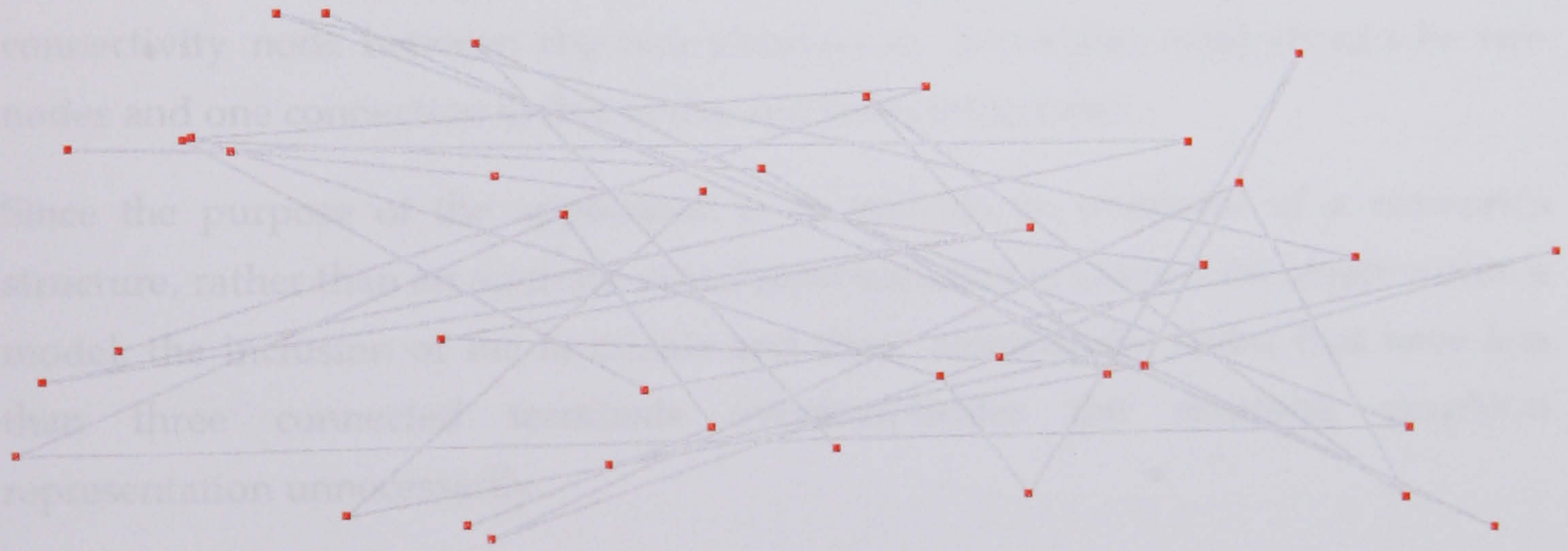


Figure 8.7 Welkin visualisation of the reduced format Small Model model prior to processing

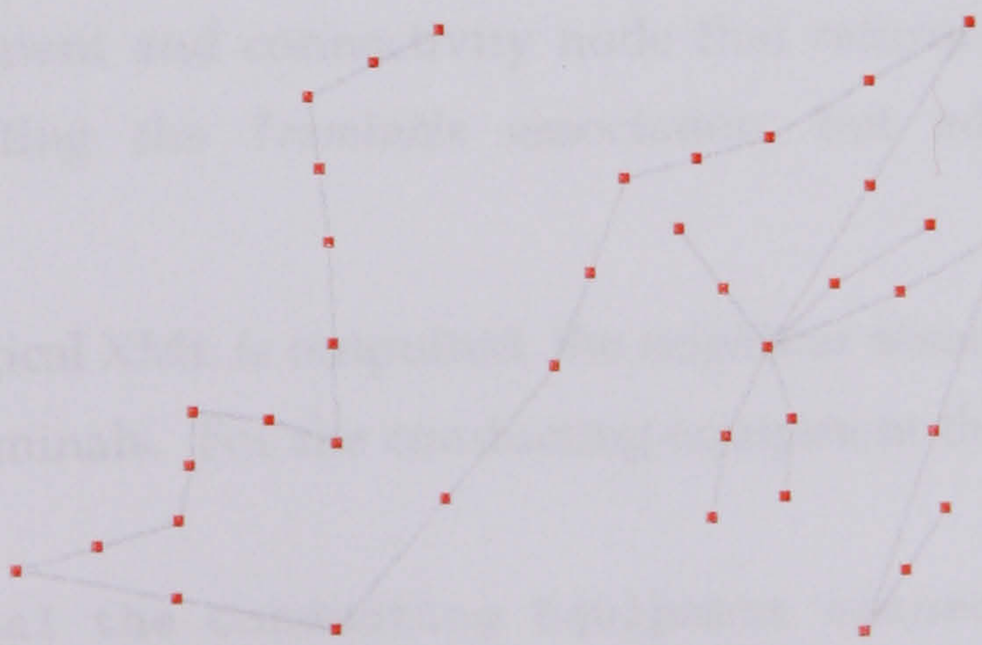


Figure 8.8 Welkin visualisation of the reduced format Small Model after thirty second of processing

The Small Model, shown in Figure 8.7, has fewer components than the previous models. These components are still loosely connected but less densely packed on screen. When the same conversion is applied to this model the shape of the network can be easily identified after the automatic clustering process has been run for thirty seconds as shown in Figure 8.8. The lack of a graphical notation to denote what each node is prevents the user from instantly differentiating between a breaker, synchronous generator or a transformer winding, but the overall layout of the network can still be observed.

From the diagram produced by the Small Model, it can be concluded that the inclusion of every terminal and connectivity node within the network complicates the on-screen view unnecessarily. Each physical component within the network is represented by at least two on-screen nodes representing the component itself and at least one terminal. Every connectivity node is similarly included on screen, further increasing the number of graphical nodes displayed. Since a connectivity node is required to join two terminals, the on-screen representation of a simple Conductor-Conductor join requires the inclusion of two terminals and a connectivity node between the two Conductors, increasing what should be two nodes and one connection to five nodes and four connections.

Since the purpose of the application is to provide an overview of a network's structure, rather than an accurate visual representation of every CIM object within a model, the inclusion of the terminals and those connectivity nodes that have less than three connected terminals overcomplicates the resulting graphical representation unnecessarily.

8.2.2.2 Simplified Topological Representation

Using the *Mercury* toolkit, an additional function was created for each piece of conducting equipment and connectivity node that returned a non-CIM compliant XML node, omitting the *Terminals* association, but adding a new *neighbour* association.

When the Topological XML is outputted, the *neighbour* association is computed from the connected Terminals. For the conducting equipment the following algorithm is used:

```
For each Terminal the Conducting Equipment connects to
   Retrieve the Terminal's Connectivity Node Association
   If the Connectivity Node has 3 or more connected Terminals
```

```
    Output the id of the Connectivity Node as a neighbour
  Else, if the Connectivity node has 2 connected Terminals
    Retrieve the name of the other Connected Terminal
    If the other Terminal has a connected piece of Conducting
Equipment
      Output the id of the Conducting Equipment as a neighbour
  Else, if the Connectivity Node has 1 connected Terminal
    Output nothing
```

Similarly, for the connectivity node:

```
If the Connectivity Node contains 3 or more Terminal Associations
  For each Terminal the Connectivity Node connects to
    If the other Terminal has a connected piece of Conducting
Equipment
      Output the id of the Conducting Equipment as a neighbour
Else if the Connectivity Node has less than 3 Terminal Associations
  Output nothing
```

A connectivity node will only produce an XML node for itself if it contains 3 or more terminal associations. This is because if it contains 2 or less terminal associations, none of the pieces of conducting equipment that connect to these terminals will identify themselves as being connected to that node (since the algorithm forces them to bypass the connectivity node and identify the other piece of conducting equipment on the other side as its neighbour). However, if a connectivity node has three or more terminals then it is a T or star point connection and an XML node is required to show an accurate network structure.

An exception to the rules occurs for *Transformer Windings*, since the default conducting equipment algorithm does not take account of the additional topological connections to the other windings within the *Power Transformer*. For transformer windings, an additional rule is appended to the algorithm included to include the other windings within the power transformer as *neighbours*.

When this Topological XML output is performed, the resulting XML file includes child nodes within the main component node of the form:

```
<strath:ConnectivityNode.neighbour rdf:resource="#_706cac32"/>
<strath:ConnectivityNode.neighbour rdf:resource="#_5eabdf21"/>
```

The *strath* prefix is used since the *cim* prefix is reserved for nodes that adhere to the CIM schema, which the *neighbour* association does not.

Figure 8.9 Welkin visualisation of the topological format Siemens 100 bus Model model prior to processing

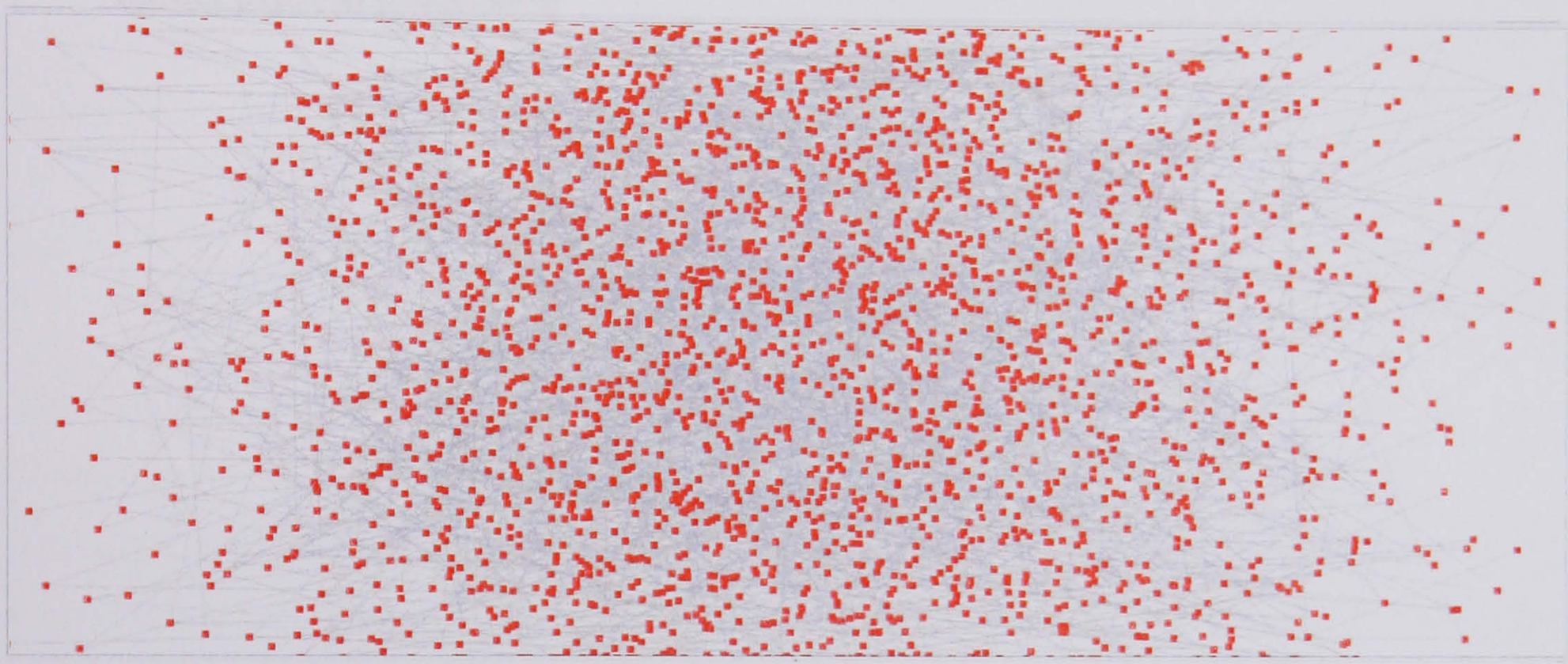


Figure 8.10 Welkin visualisation of the topological format Siemens 100 Bus model after three mintes of processing (border indicates edge of the applet's drawing canvas which nodes "bounce" off)

When the Siemens 100 Bus Model is exported as this Topological XML file and imported into Welkin, the number of nodes produces is reduced from 6976 to 2468, a reduction factor of 2.8 from the original model. Even with this reduction, however, the size of the network is still significant as shown in Figure 8.9. This prevents Welkin from condensing it sufficiently to provide a meaningful representation of the network structure as shown in Figure 8.10.

Figure 8.11 Welkin visualisation of the topological format Langside & Cathcart model prior to processing

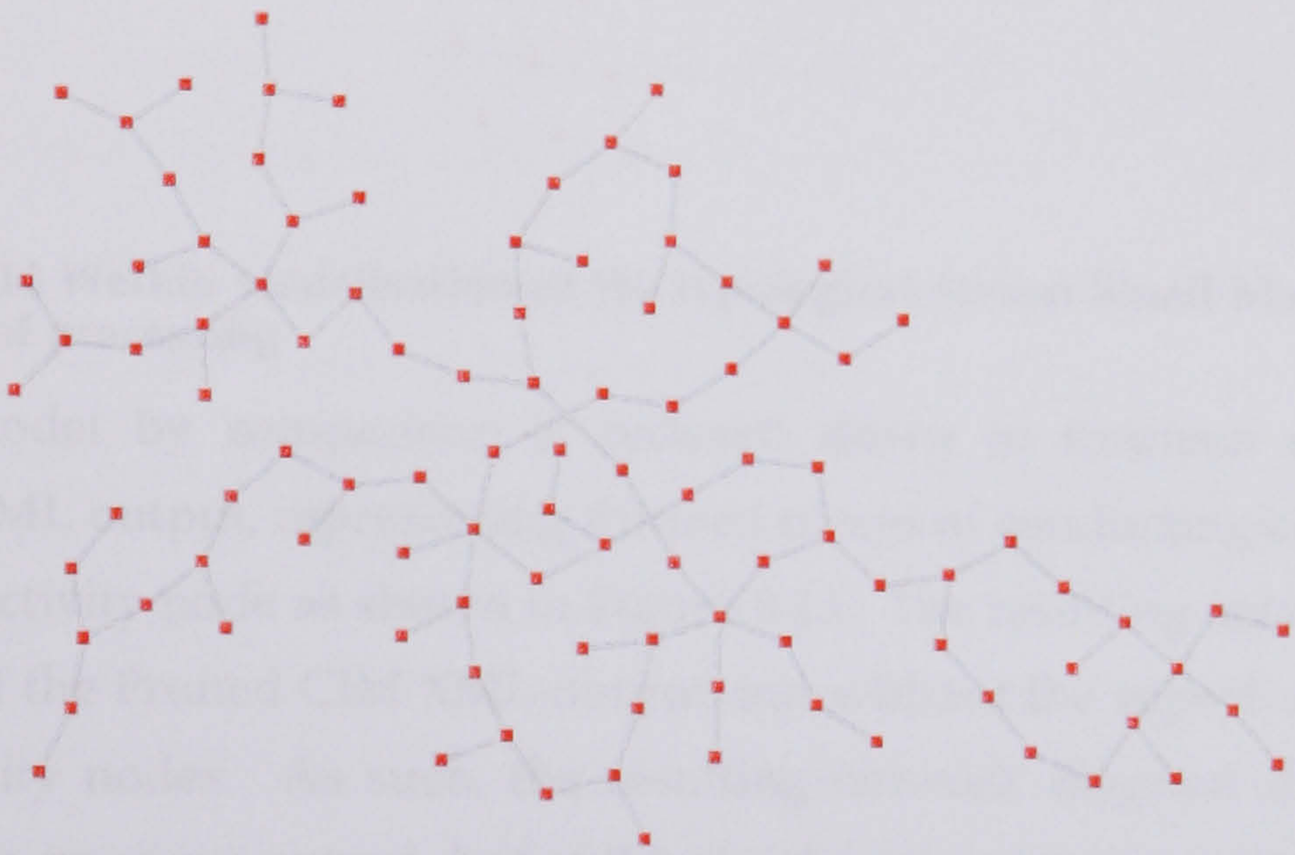


Figure 8.12 Welkin visualisation of the topological format Langside & Cathcart model after processing

When the same conversion is applied to the Langside and Cathcart model, it is reduced to 102 nodes as shown in Figure 8.11. This graph proves sufficiently compact to allow the network to cluster the nodes in such a manner as to provide a meaningful representation of the network structure within three minutes of the clustering process beginning. The graph shown in Figure 8.12 required only minor manual nudging of some key nodes to reduce overlapping.

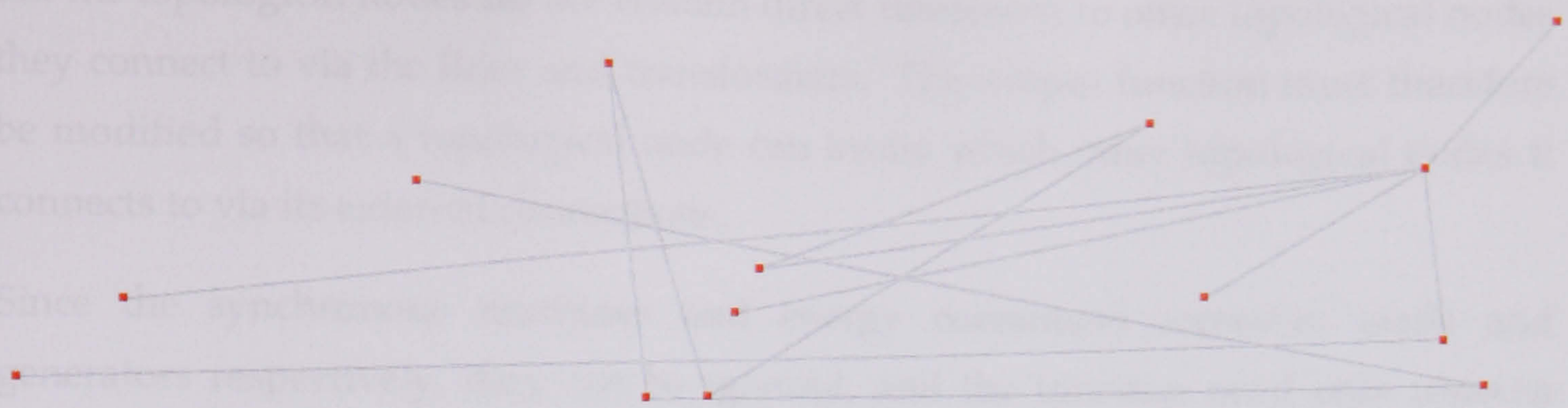


Figure 8.13 Welkin visualisation of the topological format Small Model prior to processing

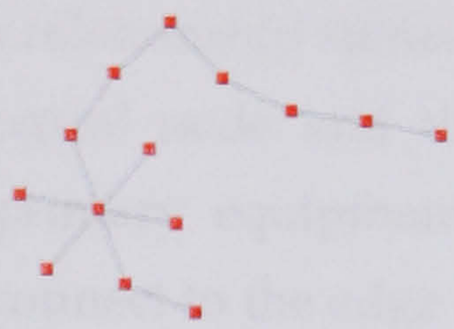


Figure 8.14 Welkin visualisation of the topological format Small Model after thirty seconds of processing

The Small Model by comparison is reduced down to fourteen nodes by the Topological XML output, representing thirteen pieces of conducting equipment and a single connectivity node as shown in Figure 8.13. The resulting network structure mirrors that of the Pruned CIM XML output, but without the superfluous terminals and connectivity nodes. As such, the resulting network diagram (Figure 8.14) is neater than the previous output, but still lacks the power system network diagram notation that would allow the user to know more about the components within the network.

8.2.2.3 Bus-Branch Notation

Since the large Siemens 100 Bus Model, even when reduced to almost a third of its original size, is still too complex to be automatically organised using the Welkin application, a method of further simplifying the topology is required.

The Node-Breaker to Bus-Branch conversion detailed previously converts a fully detailed CIM network model into a series of Toplogical Nodes (Buses), Lines and Transformers (Branches). This conversion process was initially created to allow CIM XML models to be used for the steady state load flow simulations and conversion to PSS/E format, but the same process can be used to create an XML file to represent the Bus-Branch configuration of the network.

Outputting only the *Topological Node* objects as XML nodes provides the bus data, but the topological nodes do not contain direct references to other topological nodes they connect to via the lines and transformers. The output function must therefore be modified so that a topological node can locate which other topological nodes it connects to via its external connections.

Since the synchronous machines and energy consumers represent loads and generators respectively, they can be ignored, and the function need only concern itself with any lines or transformer windings that connect to the edge of the topological node. The modifications detailed previously to convert CIM data into PSS/E format required the addition of the External_Terminals association to the *Topological Node* class. This 0..n relationship stores associations to the terminals that denote the edge of the topological node and the Terminals themselves contain associations to the pieces of primary equipment (loads, generators, transformer windings and generators) that connect to the edge of the topological node.

To identify the points at which a topological node connects to a branch, the function cycles through all the External Terminals of the topological node and identifies those that connect to either a line segment or a transformer winding. With these points identified it is a matter of locating the topological node connected to the other end of the branch.

For the transformer branches, the transformer winding contains an association to its parent power transformer. The other windings that are contained within the same power transformer can be identified from the *Contains_TransformerWinding* associations within the power transformer instance. It can be assumed the transformer contains more than one winding as, even with the *Auto Transformer* model proposed previously, the *Tap* class is an extension of the *Transformer Winding* class and as such would be included within the *Contains_TransformerWinding* association. Since every other topological node within the network will have a list of its own external terminals, by locating the terminals connected to the other windings within the transformer and comparing them to a list of external terminals from the other available topological nodes, the neighbouring topological nodes can be identified.

By adding another *neighbour* node to the outputted XML (as with the Topological XML output), the model is further simplified to a series of topological nodes which represent the buses of the network and the interconnections which represent the branches.

Figure 8.15 Welkin visualisation of the bus-branch format Siemens 100 bus Model prior to processing

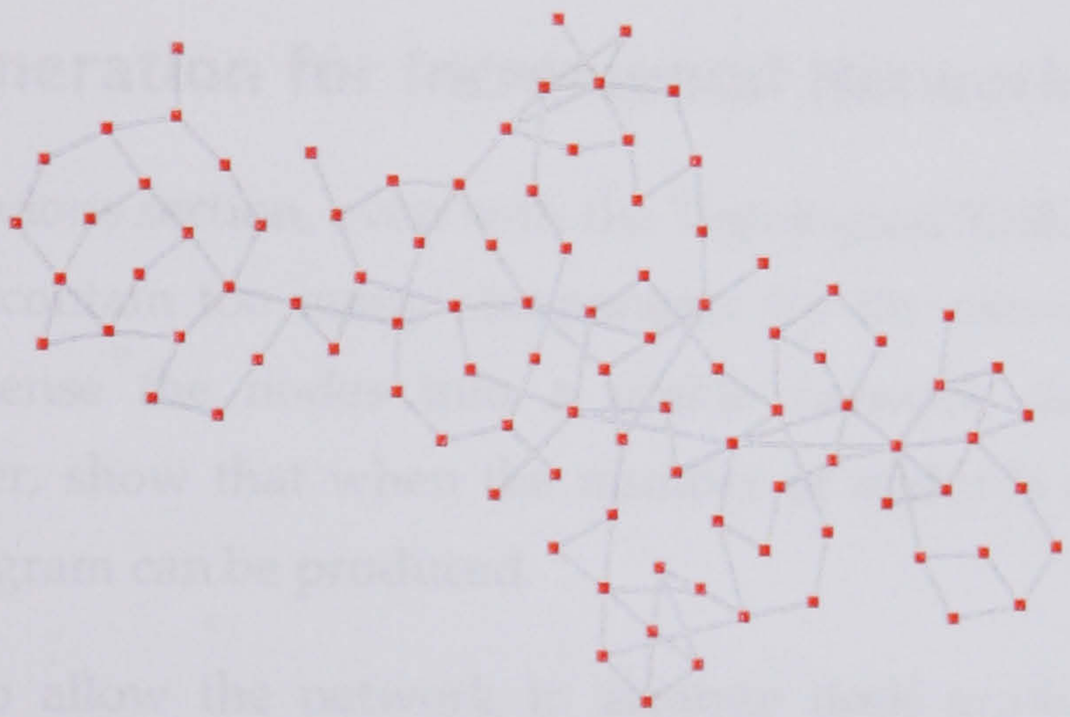


Figure 8.16 Welkin visualisation of the bus-branch format Siemens 100 Bus Model after two minutes of processing

When this conversion was applied to the Siemens 100 Bus model, the resulting XML file contained 99 Topological Nodes, indicating that the conversion algorithm used differs slightly to that used by Siemens, or that their 100 Bus model, does in fact contain data for only 99 buses. When this Bus-Branch XML file was fed into Welkin the graph shown in Figure 8.15 was produced. Upon execution, the clustering algorithm was able to successfully condense the network into a recognisable network structure as shown in Figure 8.16.

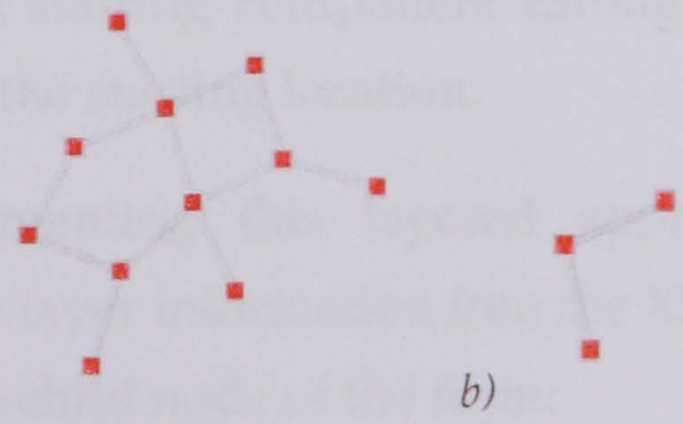


Figure 8.17 Welkin visualisation of the bus-branch format a) Langside & Cathcart model and b) Small Model after thirty seconds of processing

The Langside and Cathcart model and Small Model both produce simplified network structures containing only a few buses and branches as shown in Figure 8.17 a) and b). The reduction in detail for these models produces a simplified version of the network shape already produced by the Topological XML output.

While this Bus-Branch representation does not provide the user with details of the position of every component with the network, it does allow the user to locate buses and branches based on their overall network position. This is sufficient if the goal is to gauge the overall shape of a power system's configuration. If, however, the application is required to construct a usable network diagram containing all the network components at the Node-Breaker level then, for large networks, a different approach is required.

8.2.3 Path Generation for Incremental Network Visualisation

As seen in the previous section, even with the Topological XML output, large power system networks contain too many components for the network to automatically cluster and condense the nodes into a usable network diagram. The smaller networks, however, show that when the number of nodes is reduced sufficiently, then a useable diagram can be produced.

The solution is to allow the network to arrange itself gradually, increasing the number of components on the screen incrementally and fixing the positions of any on-screen nodes once their final position in the diagram has been decided. This way the user can build a network diagram in layers with the new components automatically arranging themselves around the previous layers.

The first step is to generate a path through the network using the traversal algorithm originally designed for the Node-Breaker to Bus-Branch conversion but modified to use the Topological XML system of bypassing Terminals and Connectivity Nodes with less than three connected terminals. Each step through the traversal becomes a new layer in the resulting network diagram, from layer 0 containing only the initial starting component through to layer n, containing the furthest components from the starting location.

The first step in implementing this layered approach to network diagram organisation is to insert the layer information into the XML nodes produced for each network component with a child node of the form:

```
<strath:layer strath:level="0"/>
```

Every CIM XML node in the exported Topological XML generated will contain a child node of this form to denote its layer in the network diagram.

The next modification requires changing the Welkin application itself to provide control over which layers are displayed and to allow all the visible, on-screen nodes to have their position fixed simultaneously.

8.2.3.1 Modifying the Graphing Tool to Interpret Layers

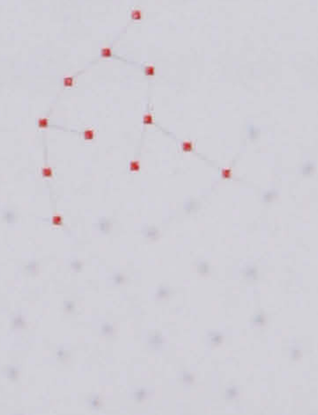
Since Welkin is released under the BSD license, the source code can be modified without breaching any copyright or licensing restrictions and the source is made freely available from the project's website. The software can therefore be modified to interpret the layer information embedded within the XML file.

During importation, if a strath:layer child-node is found within an XML node, the software stores the layer position for that node in its own internal layer-index. When the importation procedure is complete the layer-index contains n arrays of nodes (where n is the number of layers in the network).

a)

b)

c)



d)

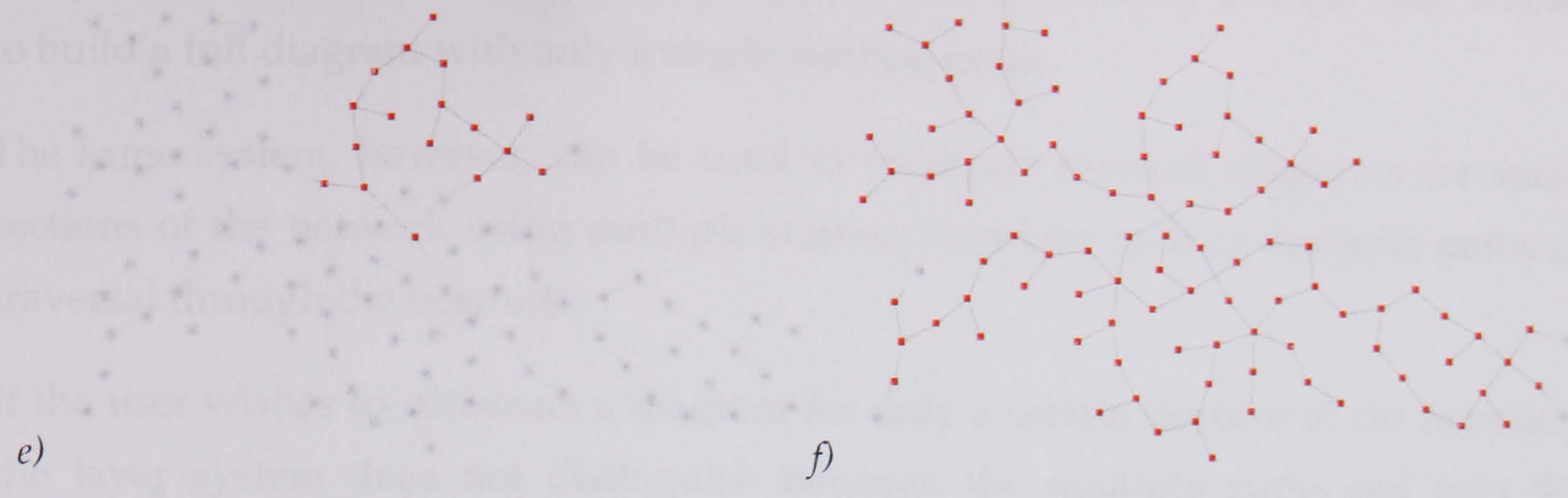


Figure 8.18 Langside & Cathcart network, multiple layers increasing incrementally from 2 visible layers (a), through 3 (b), 4(c), 5(d) to 6 (e) and 17 visible layers (f)

To begin with all nodes with a layer value greater than 0 are set as hidden, and thus ignored during the clustering process. The program is modified to provide a new panel on screen that contains a slider bar. By moving this slider, the number of visible layers is increased or decreased. This is shown in Figure 8.18, where the blurred nodes indicate those that are hidden, and the sharp nodes and connections indicate those that are visible. Figure 8.18 a) shows two visible layers, increasing to 3 visible layers at b) with the addition of two more nodes. This process continues through c) and d), from 4 to 5 visible layers, with e) showing the network visible with 6 exposed layers. Finally, f) shows the network with all bar the last layer visible.

Two additional buttons are added to the screen, which either fix or release all the visible nodes. To achieve this, the software loops through all the nodes in the graph and checks the state of their visibility. If they are set to be visible, it alters their on-screen position to be fixed or released, depending on the option chosen. If they are hidden, it leaves their position unchanged.

The purpose of the *Fix* and *Release All* buttons is so that during the network arrangement, fixing the positions of nodes in the visible layers does not affect the unorganised, hidden nodes. As a new layer is added, then organised it can be fixed, so that when the next layer of nodes is added to the screen, they do not alter the position of the nodes in the layers below them.

8.2.3.2 Multiple Paths and Multiple Layers

The Layer approach to network diagram construction works successfully for building full network diagrams for those networks too complex to be organised with all the components on-screen simultaneously. The number of onscreen nodes is limited only by the size of the on-screen workspace (itself a limitation of the

computer's screen resolution). This system, however, assumes that the user wishes to build a full diagram with only a single starting point.

The same system, however, can be used to construct network diagrams for small sections of the network using multiple starting locations and/or multiple paths of traversal through the network.

If the user wishes to construct a diagram for only a certain portion of the network, the layer system does not distinguish between the multiple paths out into the network from a single piece of conducting equipment (whether it be a single path from a generator or load, two paths, either side of a breaker, or any number of paths from a busbar section). The ability to expand the number of on-screen components based on a number of different paths and layers provides more flexibility in selecting only the desired network components.

Similarly, to construct a full network diagram, if the position of multiple pieces of equipment is known prior to the start of the process then, rather than draw the network diagram from a single starting location, multiple starting locations can be used. This is shown in Figure 8.19, where the network contains four separate starting locations.

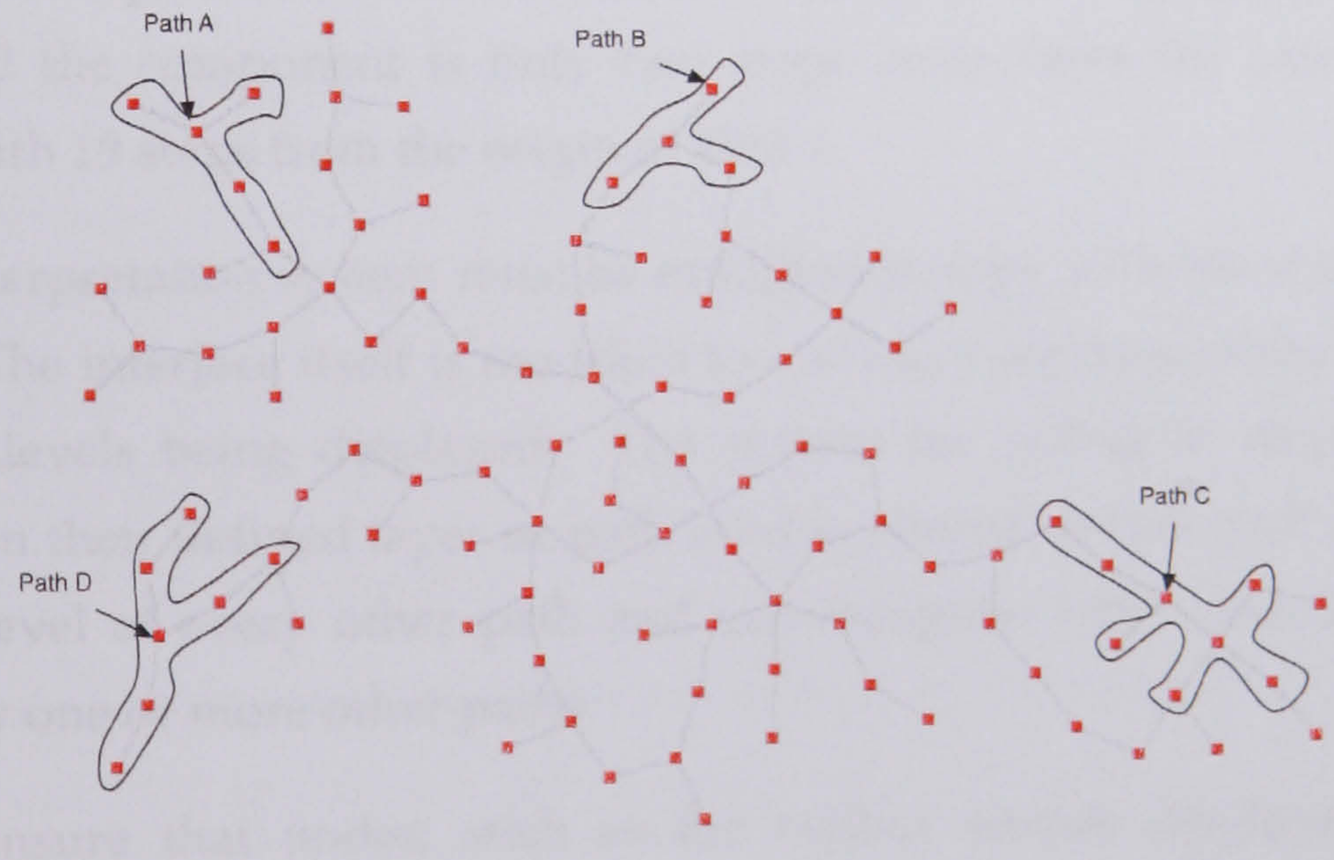


Figure 8.19 Langside & Cathcart Network with four paths, each with two layers visible. Starting locations indicated by the arrows.

Each of these starting locations has its own *path* into the network, with Layer 0 for each path being the chosen piece of starting equipment. From there, as with the basic Layer system, each component encountered in the traversal is assigned a level value for that path. Figure 8.19 shows each path with two layers visible.

Figure 8.20 Langside & Cathcart Network with Path A at Layer 9 and Path B at Layer 8. Shaded area indicates overlapping paths.

Since the multiple paths will inevitably meet, there will be overlapping components as shown in Figure 8.20, and a node may be assigned multiple level positions, one for each path. A typical XML node will be of the form:

```
<cim:BusbarSection rdf:ID="_1d154952">
  <strath:path strath:name="Path 0" strath:level="2"/>
  <strath:path strath:name="Path 1" strath:level="19"/>
</cim:BusbarSection>
```

This network component, a busbar section, is included in two paths, *Path 0* and *Path 1*. On *Path 0* the component is only two steps away from the starting location, compared with 19 steps from the origin of *Path 1*.

The layer interpretation system must be modified to cope with these multiple paths and layers. The interface itself is modified to include a separate slider for each path to alter the levels being displayed. The system for hiding or displaying nodes dependent on their defined layer or path level is altered so that each path modifier knows the level of every other path and can recognise when one of its nodes is controlled by one or more other paths.

This is to ensure that nodes, such as the busbar section displayed above, are displayed when either of the paths it is assigned to is visible at the desired level. For the node shown above the busbar will be visible if *Path 0* is greater than or equal to 2, or if *Path 1* is greater than or equal to 19. If *Path 1* had no knowledge of *Path 0*'s setting, and was unaware that one of its nodes was also contained in one or more additional paths, then even if *Path 0* was set to a value greater than 2, any change to the value of *Path 1* that resulted in it being less than 19 would hide the node.

These modifications to the *Mercury* code for XML generation, combined with the modifications to Welkin for interpreting this embedded data provide a fast and flexible system for generating a very basic power system network topology diagram. As mentioned previously though, the diagram lacks the power system iconography used for traditional power system diagrams. The direct point-to-point connections are also contrary to the traditional horizontal and vertical connections.

8.2.4 Modifying the Graphing Tool to Display Power System Model Information

Since Welkin uses the standard Java graphical libraries for producing the on-screen diagram, it is possible to modify the display depending on the type of component. Since the Topological XML nodes are using the basic CIM XML constructs, the CIM class can be extracted from the XML node during importation. Icons are defined for specific CIM classes using Java Graphics commands and the default icon; a square, red rectangle, is replaced with the Graphic instance assigned to each class.

For most equipment this process requires only an icon replacement, centred at the node's position, but there are special cases that require additional modifications to the Graphics code.

8.2.4.1 Drawing Transformer Diagrams

Since transformers are modelled as multiple windings, each winding is a separate node, and as mentioned previously, for the Topological XML, an connection is defined between these windings to maintain the topological integrity of the diagram. A winding centred at each node, however, would not produce a recognisable transformer diagram.

Instead, using a circle to represent each winding, the windings should be spaced in such a way so that, no matter the distance between the two winding nodes, the diagram produced shows a typical two winding transformer diagram in the centre of the connection line.

To do this, the graphics function is modified to include code that uses the following algorithm (where x and y are the two windings):

```
Compute the equation of the line between points x and y
Find the midpoint of that line

(For x)
  Move 5 pixels along the line closer to x
  Draw a circle of radius 10 pixels
```

```
Return to the midpoint of the line

(For y)
  Move 5 pixels along the line towards y
  Draw a circle of radius 10 pixels
```

This algorithm allows the transformer winding icons to move with the connecting line and highlight the location of transformers within the network.

8.2.4.2 Representing Line Segments

A line segment, like any other piece of conducting equipment in the diagram, is represented as a single node on the diagram. The segment normally has two connecting lines running from it to the other pieces of conducting equipment it connects to. To accurately depict the line in the diagram the connecting lines from the segment node are modified with the addition of an arrowhead at the end of the connecting line furthest from the segment node.

Using this method, a single line segment is represented as a normal connecting line, but with the addition of two black arrowheads at either end and the position of the actual segment node becomes a hinge point for the line.

The line segment nodes are altered further, with modifications to the clustering algorithm itself. Any node that is identified as being a line segment (whether it be AC or DC) is given a reduced attractiveness rating. This is to stop other nodes from clustering around the line segments to the same degree as the other components. This modification was implemented so that on the final diagram the clusters of components that are joined by lines (i.e. buses and branches) can still maintain a degree of separation.

Figure 8.21 Modified Welkin model diagram with power system icons to represent loads, generators, lines and transformers.

Figure 8.21 shows the results of these transformations, with the modified version of Welkin producing a diagram that instantly provides a recognisable overview of the power system network topology. The black squares indicate components for which no custom icon has been created, since at this stage, only icons for the primary pieces of equipment (loads, generators, lines and transformers) plus connectivity nodes (large black circles) have been created.

8.2.4.3 Elbow Connectors

Since traditional network diagrams do not use direct, diagonal, point to point connections between equipment, the ability to switch between the default straight connecting lines and elbow connections would provide the user with a means of producing a diagram more familiar to a power engineer.

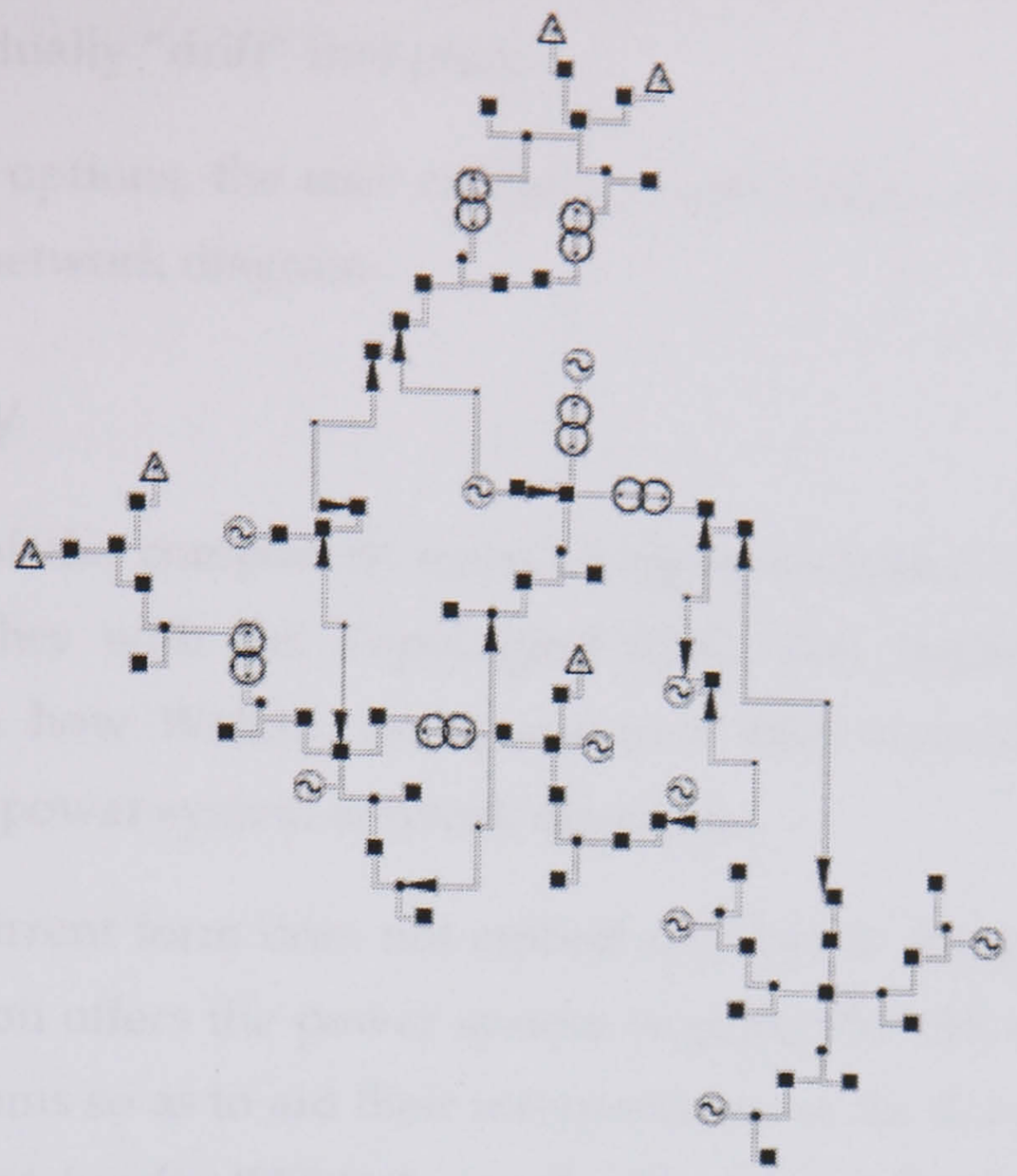


Figure 8.22 Modified Welkin diagram with elbow connectors between components

This modification requires the graphical engine to draw two lines, a horizontal and a vertical, rather than a single diagonal line. The vertical line attaches to the component at the top, and the horizontal line joins to the component below (with only a single horizontal or vertical line being drawn for those components with identical x or y coordinates). The results are shown in Figure 8.22, where all the diagonal connectors have been replaced with elbow connectors.

8.2.4.4 Align to Grid

The final modification to the Welkin program was the addition of an *Align to Grid* option. Since the graphics engine works at the pixel-level, few components lined up horizontally or vertically, as would be the case in a traditional diagram. To compensate for this, two methods for aligning to a grid were provided.

The first allows the clustering engine to run as normal, working out positions to the nearest pixel then provides the user with an *Align* button. This aligns all the onscreen nodes with the nearest grid point by rounding their x and y coordinates to the nearest 10.

The second option forces the clustering engine to align the nodes during the clustering process. This user-selectable option has the benefit of preventing the *rippling* effect that can take place even after the nodes have clustered and settled. It

can, however, prevent some clustering by preventing the small movements that allow nodes to gradually "drift" into place.

By providing both options, the user can use a combination of the two methods to create the desired network diagram.

8.2.5 Summary

The combination of the component icons, straight connectors and aligned screen components, together with the Topological XML data created in the previous section has shown how Welkin, an open-source RDF visualisation tool, can be enhanced to create power system network diagrams.

Since CIM in its current form does not embed any sort of diagram or visualisation data, this application offers the power system engineer the ability to quickly create recognisable diagrams so as to aid their interpretation of the data. The application's small code footprint (under 500kbytes with all required libraries) allows it to be used as an embedded applet within a web page and integrated with the other *Mercury* applications.

8.3 Rich Web Applications

The concept of using web browsers as the user-interface for computer applications has existed since the original NCSA Mosaic and Netscape Navigator browsers made their debut in 1993 and 1994 respectfully. It is only in the last few years, however, that the use of Asynchronous Javascript And XML (known as AJAX) has grown in popularity, allowing browser-based applications to take input from the user, request data from the server and update the browser window without a full refresh of the page.

Previously, any web-based applications required a full refresh of the page in order to send data to the server via the HTTP Post or Get protocols. The browser then redrew the entire screen when loading the new data from the server. The introduction of the XMLHttpRequest set of APIs as an ActiveX object in Microsoft's Internet Explorer 4.0, released in 1997 and subsequently adopted by the Mozilla family of browsers, Apple's Safari browser and Opera software's Opera browser, provided a means of sending and receiving data without refreshing the page. The XMLHttpRequest APIs, combined with the Javascript object-based scripting language provides a means of communicating with the server and then updating the browser window without requiring the entire page to be reloaded, instead the contents will change dynamically, like a normal desktop application.

The AJAX, Rich Web Application system offers several benefits over a standard, locally installed application:

- Cross-platform support. Any operating system with a compatible browser can utilise the application.
- Instant updates. Since the client's web browser reloads the Javascript code at the beginning of every session, there is no requirement to dispatch patches of updates to each user.
- Server-side processing. The processing requirements at the client side are minimal, with any complex operations being performed at the server-side. This can, however, be detrimental if the number of concurrent users strains the server.
- Remote access. If deployed on an internet-connected server, the application can be accessed from anywhere on the globe.

The system does, however, have a number of negative aspects that must be taken into consideration when writing a Rich Web Application:

- Network latency issues. The delays caused by the client-server communication can become and issue when the application is accessed over the Internet. Minimising the client-server communications and sending multiple commands as a single communiqué can reduce or eliminate any perceived delay.

- Browser interface limitations. The browser itself lacks the powerful graphical engine available for a desktop application. This can limit the type of applications suitable for deployment as a Rich Web Application. Complex CAD packages, photo manipulation software, or any programs requiring powerful 3D graphics support are not suited to the browser environment.

- Cross-browser support. While, in theory, any HTML and Javascript that is standards compliant should produce identical results across all browsers, the reality is that different browser platforms render the same code in different ways. This can hamper the development of web applications requiring either a limitation in supported browsers or custom, browser-specific code.

For the *Mercury* software, a Rich Web Application provides a means of directly accessing, modifying and creating server-stored CIM Java power system models via an interface that operates like a standard desktop application. The issues raised above regarding latency and the graphical limitations of the system, however, must be taken into consideration when designing any web applications.

8.3.1 Graphical Network Creation

The modified Welkin browser described previously allows a user to construct a network diagram for an existing CIM network model. It does not, however, provide a graphical interface to allow a user to create a new power system model in the CIM format.

Since an increasing number of EMS applications and power system analysis packages can import CIM data, usually in CIM XML format, it seems logical to create power system models in the CIM format rather than to create them in one proprietary format and then convert it to the CIM. A graphical interface, to allow CIM power system models to be constructed using a simple point and click interface

would provide a fast and efficient system for creating new power system models in the CIM format.

The two options available for this application are:

1) A standard desktop application to be installed and run on the user's computer, storing any created network model locally.

2) A rich web application, storing the model on the remote server.

The first option would provide the graphical network builder as a stand-alone application, lacking any integration with the existing *Mercury* software beyond exporting the resulting network in a format (CIM XML) that the *Mercury* software can then import.

The second option would allow the graphical network builder to become part of the *Mercury* toolkit, creating and saving CIM components within the *Mercury* model library as they are created.

Both options have their benefits. A stand-alone application running on the user's desktop has far less graphical limitations than the web application, and does not require a network connection to function. Applications to construct power system network models are already available from numerous power system software vendors, many of which can export CIM compatible models. A stand-alone application lacks any integration with the *Mercury* system beyond its CIM XML compatibility.

As well as allowing the creation of the power system network models from scratch, the web application provides another interface for then editing these models within the library. With a stand alone application, any editing would require the downloading, editing and then replacing of the existing model library entry instead of online editing of the library entry itself.

8.3.2 Interface Overview

The network creation interfaces uses a grid of individual cells, initially blank to form the *canvas* onto which the network diagram will be drawn. Each cell can contain either a piece of CIM Conducting Equipment or a Connectivity Node

A cell with a piece of Conducting Equipment in it can be joined to one or more neighbouring cells that contain Connectivity Nodes (up to a maximum of three, since cells cannot be joined diagonally). The number of adjacent cells a component

can connect to is defined by its class: Switches, Fuses and Rectifiers/Inverters for example, can be connected to two adjacent cells; Energy Consumers, Synchronous Generators and Earth points can be connected to only one adjacent cell; but Junctions can be connected to up to four adjacent cells.

Special cases exist for Line Segments and Bus Bar Sections, since they can span multiple cells.

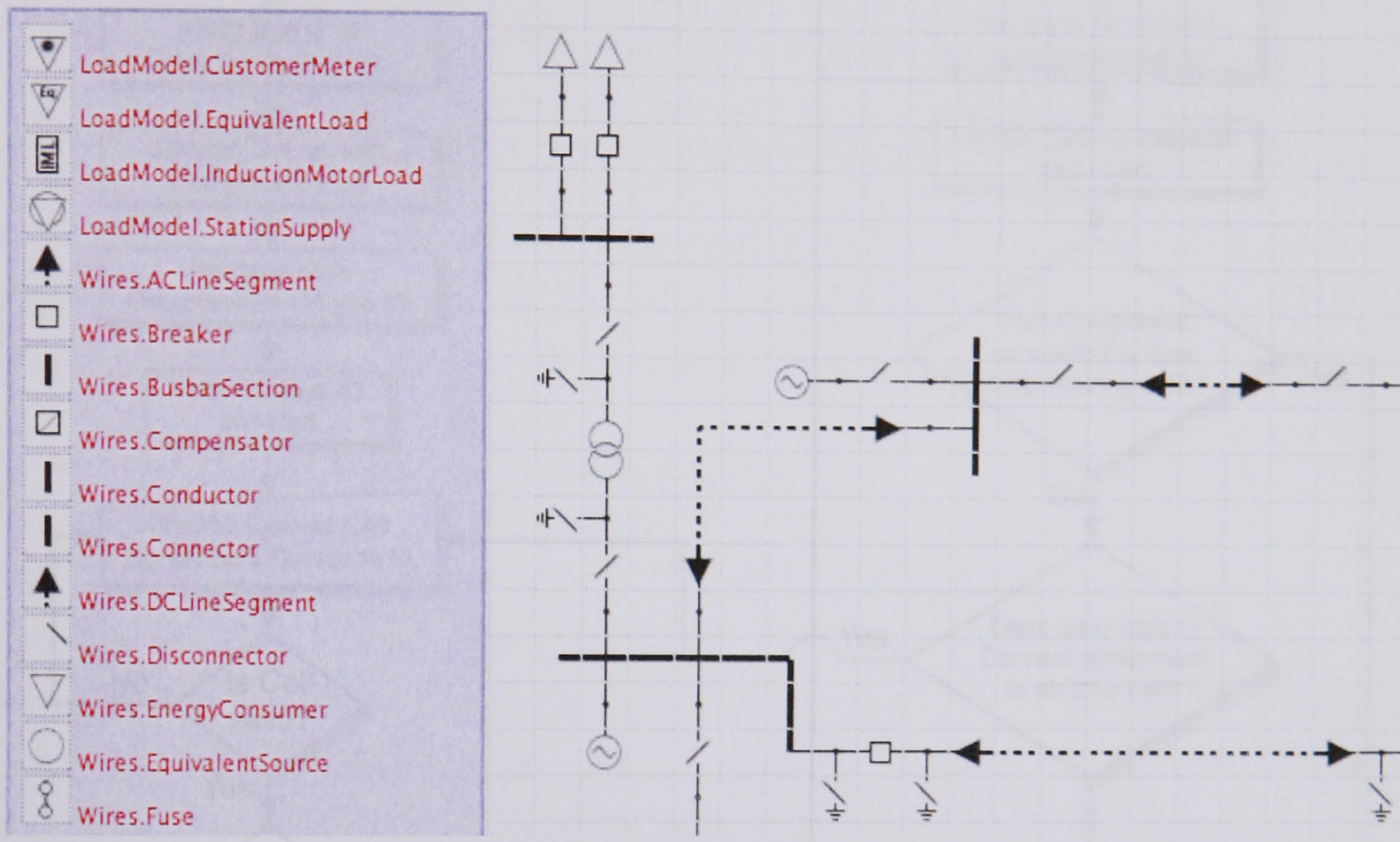


Figure 8.23 Screenshot of the Graphical Network Creator

A screenshot from the graphical network creator is shown in Figure 8.23, with a small section of the component selection menu and network diagram.

The process for adding a component to the network and defining its points of connection is shown in the flowchart in Figure 8.24. To add a component to a network the user chooses the type of component from a list of available Conducting Equipment classes. They then add this component to the desired place on the canvas either by first clicking on the component's icon or name then clicking on the desired cell or by dragging and dropping its icon onto the canvas. The browser updates the canvas by adding a small icon to the cell representing the type of equipment added.

During this process, the browser sends a command to the server notifying it of the creation of a new CIM component. The server returns the unique ID of the new component so that canvas cell can be assigned a unique ID attribute, allowing

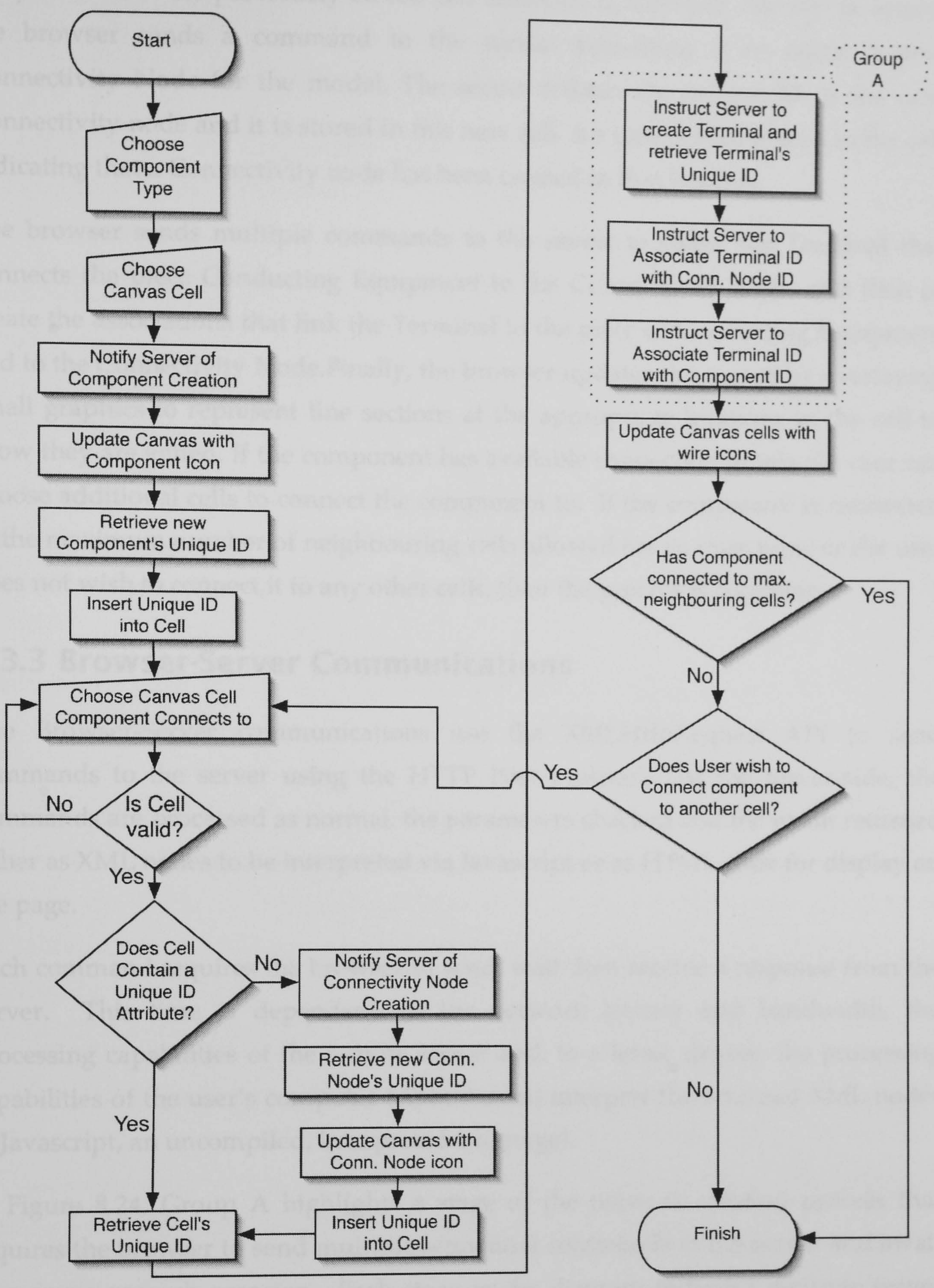


Figure 8.24 Graphical network interface component creation process

subsequent processes to equate a cell's component with its corresponding entry in the server's model library.

The user then clicks on a neighbouring cell to indicate where the component is to connect. If this cell already contains a Connectivity Node the browser acquires its

unique ID from the previously stored cell attribute. If, however, the cell is empty, the browser sends a command to the server instructing it to create a new Connectivity Node for the model. The server returns the unique ID of the new Connectivity node and it is stored in this new cell. An icon is then placed in the cell indicating that a Connectivity node has been created in that location.

The browser sends multiple commands to the server to create the Terminal that connects the piece Conducting Equipment to the Connectivity Node, and then to create the associations that link the Terminal to the piece of Conducting Equipment and to the Connectivity Node.Finally, the browser updates the canvas by overlaying small graphics to represent line sections at the appropriate locations in the cell to show they are joined. If the component has available connection points the user can choose additional cells to connect the component to. If the component is connected to the maximum number of neighbouring cells allowed for its class type, or the user does not wish to connect it to any other cells, then the process is complete.

8.3.3 Browser-Server Communications

The Browser-Server communications use the XMLHttpRequest API to send commands to the server using the HTTP Post protocol. At the server-side, the commands are processed as normal, the parameters checked and the result returned either as XML nodes to be interpreted via Javascript or as HTML code for display on the page.

Each command requires the browser to send, wait then receive a response from the server. This time is dependent on the network latency and bandwidth, the processing capabilities of the remote server and, to a lesser degree, the processing capabilities of the user's computer (since it must interpret the returned XML nodes in Javascript, an uncompiled, interpreted language).

In Figure 8.24, Group A highlights a stage of the network creation process that requires the browser to send multiple sequential commands to the server and await a response on each occasion. Each stage in the diagram indicates multiple server commands:

- The creation of the Terminal includes the Terminal creation command and an additional command to assign it a valid name based on its canvas location.

- The association command between the Connectivity Node and Terminal involves two commands, so as to associate the node with the terminal and the corresponding command to associate the terminal with the node.
- Similarly, the command to associate the Component with the Terminal includes the corresponding command to associate the Terminal with the Component.

Thus, this one group, performed sequentially without any user input, requires six commands to be sent to the server. If the previous section of the process requires the creation of a Connectivity Node in the connecting cell, this adds a further two commands for the creation and naming of the new node.

A single command can be sent, and a response received and interpreted without a noticeable delay on the user's side. When multiple commands are sent consecutively, a noticeable delay of close to two seconds can be introduced to the process.

Benchmarking the processing time at the server side indicated that the delay was due to the delays in sending and receiving the data. The processing time on the server side to create a new object was, on average 10 milliseconds, and between 20 and 60 milliseconds to add an association or update an attribute. With eight sequential commands, the server processing time, assuming the worst case scenario, accounts for 480ms. An average case, that of two object creations and an average of 40 milliseconds for the remaining six commands to update the attributes and modify the associations results in a processing time of 260ms. This indicates that almost 1.5 seconds is due to the delays in sending and receiving data.

Benchmarking single server commands found that the average time to send, receive and process a server call was 200ms. Since the processing at the server-side accounts for, on average, 40ms, this indicates that even on a local area network connection, a significant delays is introduced by the HTTP Post protocol.

For the Group of six commands, in three stages, shown above, the only response required by the browser itself is the unique ID of the newly created Terminal. The other five commands use either previously stored IDs, or browser-generated attributes for the name.

8.3.3.1 Multiple-Command Javascript Function

A method of invoking multiple server commands with a single XMLHttpRequest command would reduce the delay by requiring only a single HTTP Post. For a group of six sequential commands by reducing each transmission overhead of 160ms (a total of 960ms) to a single 160ms overhead this has the potential to remove 800ms of delay. The processing time at the server for each command will remain unchanged. However, an additional overhead for identifying and separating the multiple commands may be introduced.

An additional complication is introduced since commands may need to refer to the result of previous commands in the sequence, which will be known by the server when they are processed but, at the point of browser-side invocation, are unknown. A system must therefore be introduced that allows a command to refer to a previous command in the sequence.

The two options available are to modify the server-side code to include additional commands that perform more complex modifications to the model, or to modify the Javascript to transmit multiple commands in a single server call.

The addition of extra server-side commands has the benefit of reducing the processing burden on the client's computer and reducing the time required to perform complex operations on the network. The number of possible permutations of commands, however, requires a large and complex API, requiring modifications to both the Javascript and server-side code each time either the server API or Javscript code is modified.

A method of sending multiple server commands as a single XMLHttpRequest call, independent of which server commands are being called, would allow complex browser-side processes to be undertaken by combining server commands, removing the need for complex functions on the server-side. This can be accomplished by sending a series of commands using a pre-defined format that the server can then interpret and process sequentially, sending a response only after the sequence has completed processing.

A basic XML Schema is defined to contain the commands and their parameters, which can then be parsed at the server side and the corresponding server command can be found and executed. A single sequence of commands is contained within a *commandQueue* node of the format:

```
<commandQueue>
```

Within this parent node is contained multiple *command* tags representing each command being invoked on the server.

```
<command name="theServerSideCommandName">
```

The name of the command must match an available, public command on the server. To this node are added multiple child nodes to denote the parameters of the function, of the form:

```
<param id="p0">parameterValue</param>
```

Each parameter is given a sequential idea of the form *p0* to *pn*. The server-side XML parser will convert these parameters to the appropriate type, whether they be Strings, integers, floating point numbers or Booleans. If the parameter value is invalid then the function will fail to execute and the server will return an error.

The final child-node type within each command is a result identification tag of the form:

```
<result id="r0"/>
```

This provides a means of identifying the results from each command, and the resultant ids are numbered sequentially from *r0* to *rn*. By using these tags, the parameter tags can have an additional attribute added in place of the internal value:

```
<param id="p0" ref="r0"/>
```

This formatting of the *param* tag allows a parameter to define its value as being that of a previous command's result. The server executes the commands in order, and so any command can reference the result of a previous command as a parameter if it comes later in the sequence.

The server ensures that the parameter type and result type are compatible, and if not returns an error to the browser.

The final child node within each *commandQueue* is the *response*. This defines which results from the previously defined commands should be returned. This is included since many results will not be required at the browser side, and would previously only have been returned so that they could be used as a parameter in the next function call. The user can therefore define which results they wish to receive back with a node of the form:

```
<response ref=\"r0\"/>
```

The *ref* attribute denotes which result from the previous set of commands' *result* nodes should be returned.

The parsing of the XML commands and use of Reflection to compare the textual name of a command with the function itself on the server added a minimal overhead to the processing, adding less than 10ms to the entire process.

When used with a sequence of five sequential commands, the time taken to send, remotely process, receive and interpret was, on average, 250ms, rising to 280ms for six commands and dropping to 225ms for four commands. When used with a single command, the time taken was on average 12ms more than when used with the standard server call, indicating that as previously mentioned, the XML processing overheads and additional Javascript functionality themselves add a small delay to the procedure, thus making it efficient only for sending multiple commands simultaneously.

This system of transmitting multiple commands as a single server call removes the need for an overly complex server-side API by slightly increasing the complexity of the client-side Javascript code. This trade-off prevents the server-side API from becoming too large and complicated while maintaining the ability to perform more complex model operations as a series of events without introducing excessive amounts of latency into the system.

8.3.4 Inclusion of Network Data Overlays

The graphical Web Application interface offers the ability to include functionality from the main *Mercury* interface within the graphical editing environment. Using layers of cells, information about the components can be overlaid on their icons, and hidden with the click of a button. This provides a means of instantly identifying key attributes of any components

8.3.5 Integration of Rich Web Application with Graphing Tool

The models created by the graphical network creation interface have extra data saved with them in the library to describe their graphical layout in the creator interface. This is because the graphical interface for creating CIM power system models also provides a means of editing these models at a later time. This can involve either adding additional components to the model or modifying the attributes of existing components by selecting them from the diagram. Both of these options require the layout of the model to be saved into the library.

8.3.5.1 Saving the Network Creator Canvas

This requires saving the properties of the canvas and the attributes of every cell. By creating an XML file containing nodes representing every cell of the canvas all the required attributes are stored:

- The x and y coordinates of the cell
- The unique ID and class of the CIM component in that cell
- The graphic in the cell and its orientation
- The connections to neighbouring cells and any remaining connection points

From these attributes it is possible to recreate the canvas from the graphical network creation interface and continue creating or modifying an existing network.

Of these attributes, however, the only three that cannot be automatically extracted from an existing CIM network model, not created in the graphical creation interface, are the x and y coordinates and the orientation of the component's graphic. The orientation, however, can be calculated based on the component's x and y coordinates and that of its neighbouring cells.

8.3.5.2 Using the RDF Graphing Application to Generate Cell Positions

The modified version of Welkin described earlier can be used to create a graphical representation of a CIM network, automatically arranging the components onscreen. When this process has completed, each piece of conducting equipment has an x and y coordinate.

By transmitting these coordinates back to the server, along with the unique ID of each component as an XML stream, the server can parse the data and translate the Welkin x and y coordinates into cell positions for a graphical network creator canvas. Problems arise, however, for hidden Connectivity Nodes with only one or two pieces of Conducting Equipment connected. Since they are not included in the Topological XML file, they will have no coordinates associated with them from the transmitted Welkin file. This problem can be resolved by analysing the coordinates of the pieces of conducting equipment to which they connect and calculating the position of the Connectivity Nodes on the canvas.

By using the position of the surrounding connectivity nodes, a component's orientation (and thus graphic) can be calculated along with the position of the wire

graphics to show connectivity between cells. This allows a canvas to be created in the graphical network creator interface based on the layout produced in the modified Welkin application.

8.3.5.3 Limitations

The major limitation of this system is that while approximate coordinates can be created using Welkin, the rigid layout of the graphical network creation interface which allows only lines and bus bars to span more than one cell, can result in the canvas produced having small discrepancies compared with that produced in Welkin. Similarly, while in the graphical creation interface a bus bar can span multiple cells, in Welkin a bus bar is treated like any other component and exists as a single nodal point. This requires the conversion system to decide upon the size of bus bar to be created based on the number of pieces of equipment that connect to it, their disparity and how many cells are available.

While this system provides the ability to create a graphical network creation canvas for existing CIM power system models the discrepancies between the Welkin and graphical network creation layouts prevent a simple translation between the two.

8.4 Chapter Summary

The visualisation of network data for which no corresponding network diagram exists is not a new problem; however, the use of a modified version of an existing RDF XML graphing application and a modified, simplified version of a CIM XML representation of a power system provide an effective and novel system for generating network diagrams. Rather than relying on custom rules for placing the components, the software uses automatic clustering process to allow networks to organise themselves with only minor user-input. This, combined with the multiple levels of detail at which the network can be viewed (Full, Topologically reduced and Bus-Branch CIM XML) provides an extensible set of tools for viewing even large-scale systems. With the addition of basic icons to represent the type of component, and the elbow connectors for the interconnections between components, the graphic produced becomes a recognisable power system network diagram. The use of an existing graphing tool shows that an open CIM standard provides benefits in terms of access to existing data analysis tools, including those designed to interpret RDF XML data, which could bring significant benefits for power system operators.

The ability to create the multiple levels of detail dynamically from a single CIM power system network model is due to the use of the CIM Java object storage system. This allows the conversion algorithms to scale linearly with the model size, creating the required datasets in only a few seconds for even the largest of the test models.

A graphical model creation tool is useful in its own right, providing the user with a simple point and click interface on which they can construct power system network models in a familiar manner. By building this application on a native CIM structure, using the CIM Java object storage system to remotely store the data and a CIM-aware AJAX interface, the user is creating a power system network in a CIM format while not having to concern themselves with any of the peculiarities of the CIM's topological network representation.

By combining the automatic network diagram generation with the network creation interface a diagram in the RDF XML graphing application can be recreated in the network creation tool, allowing the graphical editing of pre-existing network models. This allows existing CIM models in XML format to be viewed and interpreted by engineers that are not familiar with the format. For the standard to be widely adopted, tools like these will be of paramount importance in aiding the transition process for engineers that wish to create and edit data in a familiar, graphical manner.

9 Conclusions & Future Work

9.1 Conclusions

The work presented in this thesis concerns the applicability of the CIM for use in the UK power industry for the exchange of operational and planning data. This has involved proposing extensions to the CIM to allow the power system to be represented at a sufficiently high level of detail, and the development of a number of novel methods for creating, processing, viewing and exporting data in a CIM format.

9.1.1 CIM Extensions

The extensions proposed for the CIM include a line model based on a previous proposal by Wang[7] but with a number of modifications that included the reintroduction of classes removed by Wang that fundamentally broke the standard IEC CIM standard, and the creation of a number of new classes to support the requirements of the UK utilities. The proposed line model used a number of Wang's proposed extensions while maintaining backwards compatibility with the standard CIM line model. The representation of ratings for components used classes that built on existing CIM classes, but whose omission from a power system model would remove a level of detail but not fundamentally alter the core data.

The new autotransformer model similarly built on the underlying CIM representation, extending it into new classes but providing a means of reverting back to an IEC standard representation to maintain compatibility.

In addition to these extensions to cover the representation of data required by the UK utilities, a further extension is proposed to support the exchange of data between utilities. This Network Connection Point extension is used to define points in the network that can be connected to neighbouring transmission or distribution networks, or to proposed power system models that represent planned points of generation or consumption. This extension is required by the model integration application that uses these points when identifying possible points of connection and overlap.

These extensions build upon existing work in extending the CIM. However, modifying the standard CIM classes creates major problems when exchanging data

between applications that are unable to interpret the extensions. For this reason it was of paramount importance that every extension proposed in this thesis can either revert back to an IEC CIM approved representation of the component with minimum transformation, or be omitted from the model without breaking the underlying power system model.

9.1.2 CIM Software Framework

While XML is the most common format for encapsulating CIM data, it must be converted to a suitable medium for medium term storage and processing. By using the CIM class structure as a software architecture, power system data could be stored in a native CIM format as memory resident objects, allowing both instant access and a means of medium to long term storage. The development of a generic import module has allowed the framework to cope with CIM extensions without requiring any rewriting of the underlying importation and management code.

This architecture has shown itself to be linearly scalable both in terms of memory usage and when used for processing network models. The use of memory resident objects has allowed functionality to be embedded within classes, including the ability for extended classes to export themselves as standard IEC CIM representations.

The object prevalent storage system, when combined with the Mercury model library provides a powerful platform for deploying online web applications. This provides instant, remote access to both the original models, and applications built on the CIM Java object framework.

This framework allows the rapid traversal of network topologies, providing a means of analysing and converting the network structure in seconds, even for large-scale models. This is used heavily in the export, integration and visualisation applications detailed in this thesis.

9.1.3 Translation and Conversion

The PSS/E export functionality, implemented as a module for the Mercury framework, fulfils a critical requirement of using CIM for planning: the ability to export power system models in a CIM format into a format compatible with an existing analysis package. The algorithms described use the CIM's own topological representation to identify the buses and branches then extract the required values from the native CIM data.

This approach relies on the object storage framework's ability to rapidly traverse entire network topologies, since the bus and branch data cannot be identified simply by mapping a bus or branch to every instance of one or more CIM classes. Instead, the algorithms discussed provide a means of dynamically generating this data by analysing the CIM's object connectivity to identify the buses and branches in the network.

9.1.4 Validation

While the importation system within the object prevalent storage framework performs a basic validation of the XML file and the correctness of the CIM representation, the development of a validation engine that can define more stringent rules on the data itself was required to ensure that the full power system models exchanged between companies were compatible. Since these rules were expressed as requirements in a standards document, not as a series of cardinality or attribute restrictions, it was decided that a method of expressing the rules as logical statements was most appropriate. This decision has since been tested by the development of a validation tool by one of the large power system vendors that used OWL schema to define the validation criteria.

The OWL based validation tool was unable to test for the conditional requirements expressed in the requirements document due to the limitations of the OWL schema and the validation engine used. This highlights the benefits of the approach described in this thesis: using logical rules and implementing an engine based on the object storage framework. This validation system was able to check a CIM power systems model against every requirement expressed in the standard, including those that were conditional.

9.1.5 Integration

Being able to store, exchange, validate and export networks in a CIM format allows operators and planners to share network information, ensure it is valid and use it with their existing analysis packages, for both planning and operational purposes. They must also integrate multiple models from multiple sources together to form a single, interconnected model of the power network. By doing this in a native CIM format, there is no loss of data (as can occur when importing and exporting to another application) and the CIM's topological representation provides a mechanism for identifying and removing overlaps between the network model. The application described in this thesis, when used with the newly defined Network

Connection Point class, has successfully integrated power system models with multiple connection points and overlapping sections.

9.1.6 Visualisation

Both planning and operational engineers must be able to visualise network topologies to aid their understanding of a network's design and operations. As such there was a requirement for the development of an application to dynamically create network diagrams from CIM data. By utilising an existing RDF data visualisation tool, the software for graphically displaying and organising the network could be created with simple extensions to an existing, open source, tool. The object storage framework's ability to quickly traverse network topologies allowed the visualisation of networks at different levels of abstraction to be accomplished by transforming the data presented to the application instead of transforming the data within the application itself.

The storage of these models in an object storage framework on a remote server allows the data for the visualisation application, embedded as an applet within a web page, to be dynamically generated on the server and then utilised by the applet. The applet then provides the user with an interface to view, and if required influence, the generation of the network diagram for the power system model. By adding functionality to return the applet's layout to the server as a series of coordinates for each component, this data can then be embedded within the CIM data stored on the server and used to generate diagrams in a number of formats (e.g. SVG, PNG).

9.1.7 Creation

The previous applications have provided mechanisms for storing, converting, validating, integrating and visualising existing CIM data. Engineers may wish to create new power system network models in a CIM format where either no existing model exists in any format, or where the existing format cannot be exported to CIM. This required the creation of a tool to allow the generation of new power system networks in a native CIM format using a familiar interface. By implementing this tool as a web application, the models are stored on the server as CIM objects, and the user requires only a standards-compliant web browser to visually create a power system network model. These models can then be shared, manipulated and used with any of the existing applications as normal, while embedding the schematic information within the CIM data on the server.

9.1.8 Using CIM Data for Operations and Planning

These tools together allow both operational and planning engineers to use native CIM data for the complete cycle of their work:

- For operational engineers, their EMS system can export their own power system network model in a CIM format, which can then be uploaded to a remote server, instantiated as CIM objects and made available to connection partners. The connection partners in turn can access this network, validate it against a pre-defined profile, export it to a format used by their analysis software and generate a schematic diagram to aid their understanding of the underlying topology. They can then upload their own network model, similarly generated from their own EMS system and automatically integrate it with the existing model to create a new, interconnected model to represent the entire network spanning their two areas of responsibility.

- Planning engineers can access the full power system network model uploaded by the main network operator, then create a proposed, addition to the network using the model creation tool. This new network proposal can then be validated against a profile supplied by the operator, and, if found to be valid, integrated with the main network. By choosing to create a number of new, interconnected network models by selecting multiple possible connection points on the main network and generating a new network model for each potential connection the planning engineer can create multiple network models to represent each possible scenario. By then exporting each of these model scenarios into the format used by their analysis software, they can decide upon the most suitable point for connection based on traditional power system analysis.

9.2 Future Work

While the applications described in thesis show that it is possible to use the CIM to both exchange data and develop tools for operational and planning purposes, there is scope to add a number of enhancements to the existing tools, and develop new applications based on the same CIM object framework.

9.2.1 Enhanced Validation

While the existing validation tool is capable of performing a level of validation beyond that of other OWL schema based tools, unless explicitly stated as a logical rule, it is ignorant of the basic requirements for a valid power system based on electrical properties. There is a need to extend this validation so that it checks the correctness of an object as well as its validity. For example, should a network contain 3 line segments of equal length, two with a resistance of 50 ohms, the other with a resistance of 5,000 ohms then unless a range has been specified for this attribute, the current validation engine will not flag this potential error.

A method of implementing such intelligent rules that would notice this discrepancy and warn the user, or that has knowledge of normal parameters for a component based on both its own internal attributes, and any associated components would allow the validation engine to check that the network model is both valid and sensible. This may require the integration of an expert system into the validation interface and the inclusion of a load flow application capable of quickly analysing the power system network model to locate any potential errors in the network representation or configuration.

9.2.2 Advanced Creation and Editing

The existing model creation interface is similarly ignorant of the structure of a power system network beyond the basic electrical connectivity. A means of automatically assigning hierarchy components (e.g. voltage level and substation containment) would further remove much of the CIM complexity from the user while ensuring the resulting file is both valid CIM and compliant with any pre-defined profiles.

9.2.3 Difference Models

While planning engineers can currently generate multiple scenario models by choosing to integrate their proposed network connections at a number of different locations, each creating a new power system model, this requires any changes made to the main power system model file to be mirrored across each scenario file. Instead, it would be beneficial to have the ability to store a single base file, then a number of different scenario files that contain instructions on how to modify this base file to add the new components and change its own, existing components to accommodate these additions.

This would allow the user to store multiple scenarios for a single base model without requiring either multiple files with duplicate data, or for the model integration procedure to be undertaken each time a scenario file is to be generated. Similarly, the user can make small modifications to an existing scenario model then save these changes as a second scenario that can be applied sequentially to the base model.

Such an approach, known as Difference Models, has been proposed[45] for use with the CIM. Using the object storage framework, it would be possible to both generate these difference models in the proposed format, and then subsequently apply them to a base CIM model. Such an application would be of benefit to planning engineers wishing to store a large number of different scenario iterations and a single base model file.

9.2.4 CIM Extensions for Distributed and Renewable Generation

The IEC 61970-301 is primarily concerned with the exchange of data at the transmission level, and the IEC 61968 is focussed at the distribution level. Neither standard contains classes to allow the detailed modelling of renewable forms of generation. Given the increasing amount of renewable generation in the UK, a significant portion of research and analysis will be required to add classes that will allow the detailed modelling of wind farms (requiring the introduction of a class to represent induction machines to complement the existing synchronous machine class), tidal and wave generators, micro-generation technologies (e.g. Combined Heat and Power Systems in residential locations) and any other forms of renewable or distributed generation with characteristics which are unlikely to be successfully modelled using the existing classes that are primarily aimed at modelling large scale hydro and thermal generating stations.

9.2.5 A Common Information Model for Energy Systems

While the CIM is being adopted by the electrical power industry, as yet there are no similar, open standards for modelling other systems within the energy domain. Since many electrical power companies also have interests in natural gas and oil production, extending the CIM to allow pipelines, refineries, storage facilities etc., to be modelled in a similar manner would facilitate the integrated modelling of a company's entire energy infrastructure.

This would allow, for example, the modelling of the gas pipeline infrastructure so that the electricity generation process can be modelled from the extraction of the natural gas, through a pipeline to a storage facility, then, via another pipeline, to a gas-fired power station where the gas is consumed and electricity is produced. By basing the pipeline infrastructure modelling on the same basic principles of the CIM's electrical network it would simplify the integration of the two models into a Common Information Model for Energy Systems.

This modelling effort would require significant research into how Gas and Oil systems are currently modelled and the requirements of the companies that operate them. The existing CIM standard has been developed over many years by a number of electrical utilities, software vendors and regulators, so it is envisioned that the creation of a sister standard for the Gas or Oil industry would require a comparable level of input from industrial partners should they be convinced that, in the long term, the effort would prove worthwhile.

9.2.6 Analysing CIM Models Natively

The network integration application described in Chapter 7 has shown that the CIM's class structure can be used to perform analysis of the network model. Further research is required to determine whether the CIM, by offering an object-based representation of a power system, would allow new, novel algorithms to be developed that would allow optimisation or probabilistic analysis of a power system network to be undertaken using the native CIM data. This would remove the need to convert the data and export it to an existing application for analysis. This work would investigate whether the use of object-oriented programming techniques in the application of such algorithms would prove faster or more flexible than traditional, procedural-based algorithms.

10References

[1] "IEC 61970 Energy management system application program interface (EMS-API) - Part 301: Common Information Model (CIM) Base", IEC, Edition 1.0, November 2003

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

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

[4] 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

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

[6] "Grid Code", National Grid, Issue 3, Revision 15, 1st April 2006, [Online] http://www.nationalgrid.com/uk/Electricity/Codes/gridcode/

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

[8] X. Wang, N. N. Schulz and S. Neumann, "CIM Extensions to Electrical Distribution and CIM XML for the IEEE Radial Test Feeders", IEEE Transactions on Power System, August 2003, Volume 18, Number 3 p.1021-1028

[9] "Common Graphics Exchange", Request for Proposal, CCAPI Task Force, October 2001. Available at http://cimuser.org/WG13/Documents/Common_Graphics_Display/CCAPI%2520Common%2520Graphics%2520Exchange%2520RFP%25202-01.pdf

[10] CIM User Group website [Online] http://www.cimuser.org

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

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

[13] Carson, J. R. "Wave propagation in overhead wires with ground return", 1926, Bell System Technical Journal, 5, pp.539-554

[14] Franklin, A.C., Franklin, D.P., "J&P Transformer Book, 11th Edition", 1983, ISBN 0-408-00494-0, pp. 162-164, 643-665

[15] Weedy, B.M., Cory, B.J., "Electric Power Systems, Fourth Edition", 1998, ISBN 0-471-97677-6, p. 71

[16] M.P. Selvan, K.S. Swarup, "Object Methodology", IEEE Power and Energy Magazine, January-February 2005, Volume 3, Issue 1, p.18-29

[17] (2001) Unified Modeling Language Specification. OMG [Online] http://cgi.omg.org/docs/formal/01-09-67.pdf

[18] Maths, Statistics and Computational Science, "SciMark 2.0", National Institute of Standards and Technology [Online] Available http://math.nist.gov/scimark2/

[19] Performance Comparison of Java/.NET Runtimes (SciMark 2.0 in Java, C# and C), Kazuyuki Shudo [Online] Available http://www.shudo.net/jit/perf/#scimark2

[20] Donald Knuth, "The Art of Computer Programming, Volume 3: Sorting and Searching, Third Edition", 1997, ISBN 0-201-89685-0, Section 6.2.1: Searching an Ordered Table, pp. 409-426.

[21] D. Barry, T. Stanienda, "Solving the Java object storage problem", IEEE Computer, November 1998, Volume 31, Issue 11, p.33-40 pp. 33-40

[22] Arnold deVos, S.E. Widergren, J Zhu, "XML for CIM Model Exchange", presented at Power Industry Computer Applications Conference, Sydney, Australia, 2001

[23] Prevalyer, [Online] http://www.prevayler.org/

[24] C.E. Villela, "An Introduction to Object Prevlance", [Online] http://www-106.ibm.com/developerworks/library/wa-objprev/

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

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

[27] Arnold deVos, "Simplified RDF Syntax for Power System Model Exchange", Langdale Consultants, available at http://www.langdale.com.au/DAF/PSModelExchange.pdf

[28] UMS Data Access Facility, OMG Open Technical Standard Candidate, formal/02-11-08, November 2002

[29] E.Z.Zhou, "XML and Data Exchange", IEEE Power Engineering Letters, April 2000, p.66-68

[30] Roman Kulyk, "Making Deregulated Markets Work: A Technical Roadmap", Version 1.1, SPI Group, 2nd October 2002

[31] Leila Schneberger, "CIM RDF Schema", Exported from CIM UML v10

[32] W3C Recommendation, "XSL Transformations (XSLT)" Version 1.0, November 1999, available at http://www.w3.org/TR/xslt

[33] Sun Microsystems, "Java Language Overview", available at http://java.sun.com/docs/overviews/java/java-overview-1.html"

[34] "IEC 61970 Energy management system application program interface (EMS-API) - Part 452: CIM Model Exchange", Draft, March 2006

[35] CIMValidate, Langdale Consultants [Online] http://www.langdale.com.au/validate/

[36] Xpetal, Langdale Consultants, [Online] http://www.langdale.com.au/styler/xpetal/

[37] W3C Recommendation, "OWL Web Ontology Language", 10th February 2004, [Online] http://www.w3.org/TR/owl-features/

[38] Cartik R. Kothari and David J. Russomanno, "Modeling Logic-Based Constraints In OWL", Proceedings of the IASTED International Conference on Artificial Intelligence and Applications, February 16-18 2004, Innsbruck, Austra

[39] Pascal Hitzler, Jurgen Angele, Boris Motik and Rudi Studer, "Bridging the Paradigm Gap with Rules for OWL", W3C Workshop on Rule Languages for Interoperability, 27-28 April 2005, Washington, DC, USA

[40] The SIMILE Project, Welkin Graph Based RDF Visualizer, Massachusetts Institute of Technology, [Online] http://simile.mit.edu/welkin/

[41] HP Labs, Experimental RDF Graph Visualizer, Hewlett Packard Development Company, [Online] http://www.hpl.hp.com/personal/Craig_Sayers/rdf/visual/

[42] Sunil Goyal, Rupert Westenthaler, RDF Gravity (RDF Graph Visualization Tool), Salzberg Research [Online] http://semweb.salzburgresearch.at/apps/rdf-gravity/index.html

[43] Zhu Yongli, O.P. Malik, "Intelligent Automatic Generation of Graphical One-Line Substation Arrangment Diagrams", IEEE Transactions on Power Delivery, July 2003, Volume 18, Number 3, p.729-735

[44] Y.S. Ong, H.B. Gooi, C.K.Chan, "Algorithms for Automatic Generation of One-Line Diagrams", IEE Proceedings. Generation, Transmission and Distribution, September 2000, Volume 147, Issue 5, p.292 – 298

[45] A. deVos, "RDF Difference Models", Langdale Consulatants, [Online] http://www.langdale.com.au/CIMXML/DifferenceModelsR05.pdf