МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к выполнению практических работ
по дисциплине
«Системы управления и сбора данных в электроэнергетике»
для студентов направления
13.03.02 «Электроэнергетика и электротехника»

Ставрополь 2025

СОДЕРЖАНИЕ

ПРАКТИЧЕСКАЯ РАБОТА №1

Тема: *Основные принципы построения общей информационной модели.*

Общие сведения о CIM

С развитием цифровых технологий возрастает потребность энергокомпаний в обмене на постоянной основе данными для обеспечения надежности работы объединенных энергосистем, принадлежащих и управляемых различными предприятиями.

Энергокомпании используют большое число различных форматов для хранения данных. Это информация по имуществу и планированию работы, хранимая в базе данных собственной разработки, топология сети в управляющей системе и обычные файлы, используемые расчетными программами.

Наряду с тем, что большинство этих данных необходимы внутри компании, часто необходимо обмениваться ими как внутри компании между различными приложениями, так и с другими компаниями. Большое число собственных форматов, используемых этими приложениями, требует множество промежуточных программ-конвертеров для импорта и экспорта данных между множеством систем. Этот экспоненциальный рост сложности, когда интегрируется возрастающее число приложений и увеличиваются обмены между множеством компаний, выдвинул требование для общего формата, который снимет все вопросы обмена данными в области электроэнергетики.

Этот формат данных получил название Общая информационная модель (Common Information Model, сокращенно CIM) для энергосистем и в настоящее время имеет два основных использования:

– облегчение обмена данными об энергосистеме между энергокомпаниями;

– обеспечение возможности обмена данными между приложениями внутри компании.

С момента появления современных компьютеров инженеры в энергосистемах использовали возможности этого инструмента в различных

областях и для выполнения сложных вычислений в энергосистеме и для управления ее работой в режиме реального времени. Все эти приложения требуют возможности хранения и обмена данными о системе.

Крупномасштабные систему управления энергопотреблением (англ. аббревиатура Energy Management Systems, сокр. EMS) и системы управления собственностью используют специальные схемы для определения структуры данных в хранилище данных, зачастую собственной разработки для отражения специфических требований системы. В программах расчета потокораспределения и уровня токов короткого замыкания используются специфичные для приложения форматы, которые содержат данные, требуемые каждым приложением. Для того, чтобы все эти программы работали вместе, в прошлом для обеспечения совместимости при интеграции часто приходилось покупать каждую часть программного комплекса предприятия у одного поставщика.

Дерегулирование энергетической отрасли, однако, потребовало множество программ от большого числа разных поставщиков, создавая при этом необходимость обмена большими наборами данных на регулярной основе. Использование форматов собственной разработки усложняет этот обмен, требует сложной трансляции между каждым форматом.

Аналогично, приложения, работающие не в темпе реального времени, традиционно используют жесткий собственный формат, содержащий только данные, требуемые для этой конкретной версии приложения. Когда последующая версия программы требует дополнительных деталей, файловый формат меняется, приводя к множеству форматов для одного приложения. Конечно, такой сценарий не ограничивается программами для энергетической отрасли. Изменение формата для каждой новой версии программы является общей практикой в индустрии программного обеспечения и обычно вызывает только незначительные неудовольствия, т.к. каждая новая версия программы содержит возможности импорта для конвертации файла в формате предыдущей версии в файл новой версии.

Проблемы происходят в случае, если компании требуется обмен данными между программами различных поставщиков и/или при наличии нескольких

версий одной и той же программы, работающей в одной и той же компании. Это требует от компании выполнения одного из приведенных ниже действия:

1. Поддерживать множество копий одинаковых данных во множестве форматов

2. Хранить данные в формате, совместимом с каждой программой, требуя удаления специфичных для приложения данных и соответственно теряя точность

3. Хранить данные в одном, очень детальном формате и создать программу для перевода из этого очень детального формата в требуемые для приложения файловые форматы

4. Использовать очень детальный формат, который совместим с любым приложением и содержит базовые данные для представления энергосистемы, позволяя в то же время сохранять дополнительные, детальные, специфические для приложения данные без нарушения формата в целом.

Третья возможность требует от компании разработки дополнительной программы для конвертации, но позволяет поддерживать только один формат, содержащий все необходимые данные. Четвертая возможность представляет идеальное решение, позволяя компаниям поддерживать один, очень подробный формат, который совместим с любыми их программами.

Однако эта возможность требует трех вещей:

1. Очень детальную модель для описания энергосистемы

2. Файловый формат, способный хранить расширяемые данные без воздействия на основные данные

3. Поставщики программного обеспечения и предприятия должны одобрить и внедрить эту модель данных и формат по экономическим или по административным причинам.

Common Information Model (CIM) для энергосистем потенциально отвечает первому из представленного списка требованию, а eXtensible Markup Language (XML), совместно с Resource Description Framework (RDF) предлагает средство для выполнения второго требования. Оставшееся требование может рассматриваться более как коммерческая задача, нежели техническая. Всеобщее принятие этого формата требует как от предприятий, так и от поставщиков

осознание выгод от принятия стандарта. В настоящее время все основные поставщики программного обеспечения для энергосистем активно участвуют в тестах на совместимость с CIM и популярность формата растет.

Объектно-ориентированный формат данных

Формат данных CIM построен на современном объектно-ориентированном подходе. Для описания формата данных, построенного на объектно-ориентированном подходе общепринятым является язык UML. UML (англ. Unified Modeling Language – унифицированный язык моделирования) – язык графического описания для объектного моделирования в области разработки программного обеспечения, для моделирования бизнес-процессов, системного проектирования и отображения организационных структур.

UML является языком широкого профиля, это – открытый стандарт, использующий графические обозначения для создания абстрактной модели системы, называемой UML-моделью. UML был создан для определения, визуализации, проектирования и документирования, в основном, программных систем. UML не является языком программирования, но на основании UML-моделей возможна генерация кода под любой язык программирования.

Основными понятиями UML являются классы, объекты и отношениям между ними.

Класс представляет специфический тип моделируемого объекта. Иерархия классов это абстрактная модель системы, определяющая каждый тип компонента в рамках системы как отдельный класс. Иерархия классов должна отражать реально существующую структуру системы.

UML диаграммы классов дают полезное средство для визуального представления иерархии классов.

Классы и объекты

Для того, чтобы понять, что такое классы и для чего они нужны, проведём аналогию с каким-нибудь объектом из повседневной жизни, например, с велосипедом. Велосипед — это объект, который был построен согласно чертежам. Так вот, эти самые чертежи играют роль классов в ООП. Таким

образом классы – это некоторые описания существенно важных свойств, схемы, чертежи по которым создаются объекты.

Каждый класс может иметь свои внутренние атрибуты и связи с другими классами. Каждый класс может быть воспроизведен много раз в отдельном экземпляре, называемом объектом, каждый из которых содержит одинаковое число и тип атрибутов, а так же связей. Однако каждый объект обладает собственными значениями атрибутов. На рисунке 1.1 приведен пример класса «Person» (человек) на языке UML. Класс обозначается с помощью прямоугольника, в заголовке которого указывается его название, а затем перечисляются его атрибуты.

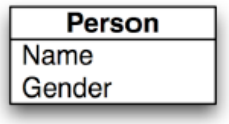


Рисунок 1.1 – Класс «Person»

Простой пример класса – это класс «Person», показанный на рисунке 1.1. Класс «Person» содержит два базовых атрибута: Name (имя) и Gender (пол). Например, если создаваемая система должна представлять каждого человека в университете, то достаточно только этого одного класса, т.к. каждый человек в университете мог бы быть представлен на самом базовом уровне с помощью определенных в классе Person атрибутов.

Для университета, содержащего 10000 студентов и персонала, система должна создать 10000 отдельных объектов Person, каждый содержащий значения для атрибутов Name и Gender независимо от других 9999 объектов (хотя они не обязательно должны быть уникальными).

Если от системы требуется хранить больше информации, чем только имя и пол, а также еще и отличать персонал от студентов, и кроме того различать их типы, то диаграмма классов должна быть более сложной.

Наследование (генерализация)

Наследование (также известное как генерализация) определяет класс как подкласс другого класса. Как подкласс он наследует все атрибуты своего

родителя, но может так же иметь и свои собственные атрибуты (рисунок 1.2).

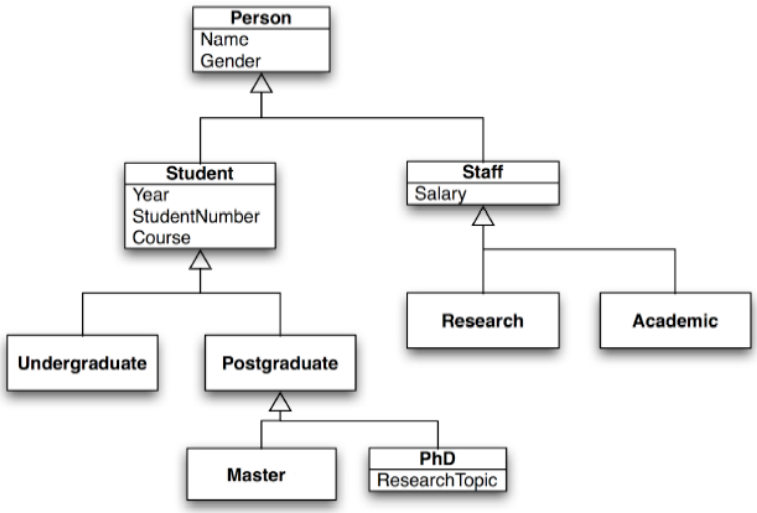


Рисунок 1.2 – Иерархия классов людей в университете

Рисунок 1.2 показывает иерархию классов для представления различных типов людей, существующих в университете. Эта диаграмма, как и все последующие диаграммы классов, использует стандартные UML символы. Student (студент) и Staff (персонал) являются подклассами класса Person. Student – это так же Person и соответственно имеет атрибуты Name и Gender, но он так же имеет дополнительные атрибуты для обозначения года поступления, студенческого номера и изучаемого курса. Аналогично, Staff также является Person, но имеет новый атрибут для указания оклада.

Класс Student так же имеет два подкласса: Undergraduate (учащийся) и Postgraduate (аспирант), оба наследуют все атрибуты от класса Student (и соответственно от класса Person), но не зависят друг от друга. Класс Postgraduate так же имеет два подкласса: Master (магистр) и PhD (соискатель степени). Класс PhD – это и Postgraduate, и Student, и Person со всеми их атрибутами, а так же с дополнительным собственным атрибутом ReseachTopic (исследуемая тема).

Класс Staff так же имеет два подкласса: Reseach (исследователь) и Academic (преподаватель), которые так же сохраняют все атрибуты классов Staff и Person.

Таким образом, хотя PhD является Postgraduate, Student и Person, не все объекты Studen являются объектами PhD. Аналогично, Academic – это Staff и

Person, но не каждый объект Staff является объектом Academic.

Ассоциация

Выше показано, как иерархия классов может быть сформирована описанием подклассов Person, но в классах на рисунке 1.2 никаких кроме наследования связей между классами нет.

На рисунке 1.3 представлено два дополнительных класса: Subject (предмет) и Period (период). Ни один из них не является типом класса Person, и таким образом не наследуется из класса Person. Однако, класс Subject имеет связь с классами Undergraduate и Academic.

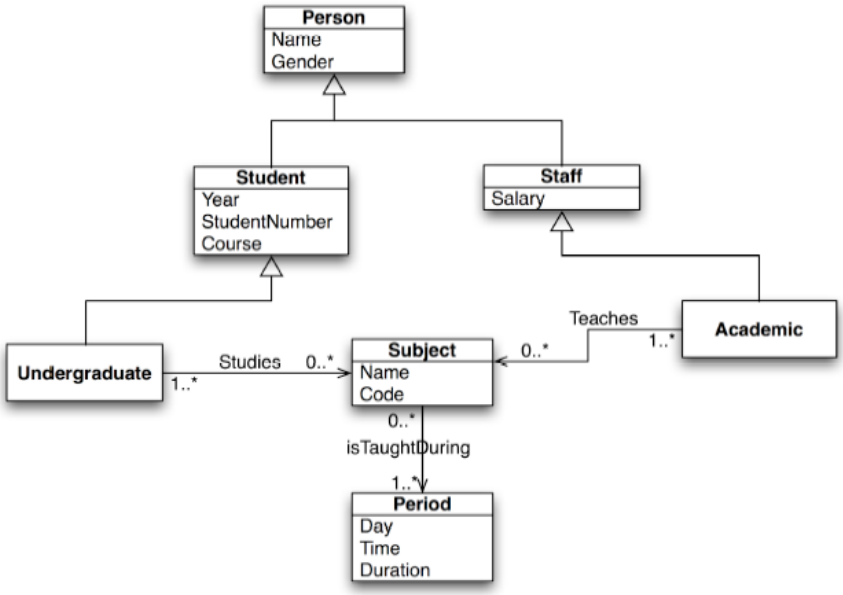


Рисунок 1.3 – Иерархия классов студентов, персонала и предметов

На рисунке 1.3 представлено два дополнительных класса: Subject (предмет) и Period (период). Ни один из них не является типом класса Person, и таким образом не наследуется из класса Person. Однако, класс Subject имеет связь с классами Undergraduate и Academic.

Объект Undergraduate может изучать несколько предметов и объект Academic может вести обучение нескольким предметам. Эти связи показаны на диаграмме как ассоциации между классами.

Для ассоциации Undergraduate-Subject дана роль «Studies» и расположение стрелки указывает, что это объект Undergraduate тот, который *Studies* (изучает) объект Subject (если бы стрелка указывала в обратном

направлении, то это означало бы, что объект Subject изучает объект Undergraduate).

На каждом конце ассоциативной связи указывается множественность. Для ассоциации Undergraduate-Subject указано, что объект Subject должен иметь от 1 и более (1..*) объектов Undergraduate, но объект Undergraduate может иметь от 0 и более (0..*) предметов (т.е. возможно, что некоторые объекты Undergraduate могут не изучать ни одного предмета из-за различных причин. Предполагается, что предмет не будет существовать в системе, если ни один студент не выбрал его для изучения).

Аналогично, объект Subject будет иметь от 1 и более объектов Academic, которые обучают этому предмету, но объект Academic может вести от 0 и более объектов Subject (т.е. не все объекты Academic должны вести обучение).

Другой дополнительный класс Period с атрибутами Day (день), Time (время) и Duration (продолжительность) отображает индивидуальное расписание и таким образом класс Subject имеет связь с классом Period. Как и с другими ассоциациями, ассоциация Subject-Period имеет роль *isTaughtDuring* (продолжительность обучения) и множественность которая указывает, что обучение объекту Subject будет проходить во время 1 и более периодов и что объект Period будет иметь от 0 и более объектов Subject, изучаемых во время этого периода.

Таким образом, на базовом уровне показаны связи между классами и как UML диаграмма классов предоставляет графическое средство для отображения этих связей.

Агрегация

Связь Агрегация определяет специальный тип ассоциации между классами, указывающую, что один из них является классом-контейнером для другого. В приведенном на рисунке 1.4 примере представлены два новых очень простых класса: University (университет) и Building (здание), содержащих только один атрибут, указывающий их имена. Множественность на диаграмме используется таким же образом, как и при ассоциации, указывая, что объект Building может быть частью 0 и более объектов University (предполагается, что

некоторые университеты функционируют во взаимодействии со школами и не каждое здание в рамках системы будет обязательно частью университета). Вторая множественность указывает, что объект University может содержать 0 или более зданий (0 если он, например, работает исключительно с помощью удаленного обучения).

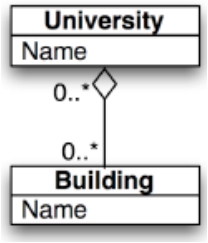


Рисунок 1.4 – Иерархия классов University и Building

В отличие от простой ассоциации, линия, указывающая связь на диаграмме содержит ромб вместо стрелки. Это говорит о том, что два класса имеют связь агрегация. Это может быть пересказано следующим образом: «Университет состоит из 0 или более Зданий», указывая, что связь сильнее, чем простая ассоциация. Однако, не закрашенный ромб указывает, что эти два класса не полностью взаимосвязаны, и что если университет будет закрыт, здание будет продолжать существовать (закрытие в смысле, что университет перестал существовать).

Композиция

Композиция – это специализированная форма агрегации, где «вкладываемый» объект является фундаментальной частью объекта-контейнера, и что если объект-контейнер разрушен, то все объекты, которые связаны с ним с помощью композиции, так же разрушены. Пример этого приведен на рисунке 1.5 между новым классом Room (комната) и классом Building.

Линия здесь имеет заштрихованный ромб, указывающий, что связь является композицией. Множественность устанавливает, что здание будет иметь 1 или более комнат (даже если здание пустое, то оно может рассматриваться как одна гигантская комната) и что комната будет содержаться только в 1 здании. Это отражает реальное взаимоотношения комнат и зданий. Если здание

разрушено, тогда комнаты в нем также разрушены.

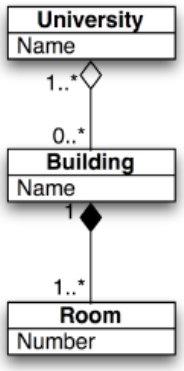


Рисунок 1.5 – Иерархия классов University, Building и Room

Любая система, которая осуществит эту схему, будет знать, что если объект Building разрушен, то любые объекты Room, которые содержались в этом конкретном объекте Building, будут так же разрушены.

Итог

Из предыдущих разделов должны быть понятны базовые принципы иерархии классов и как они представляются на диаграмме классов.

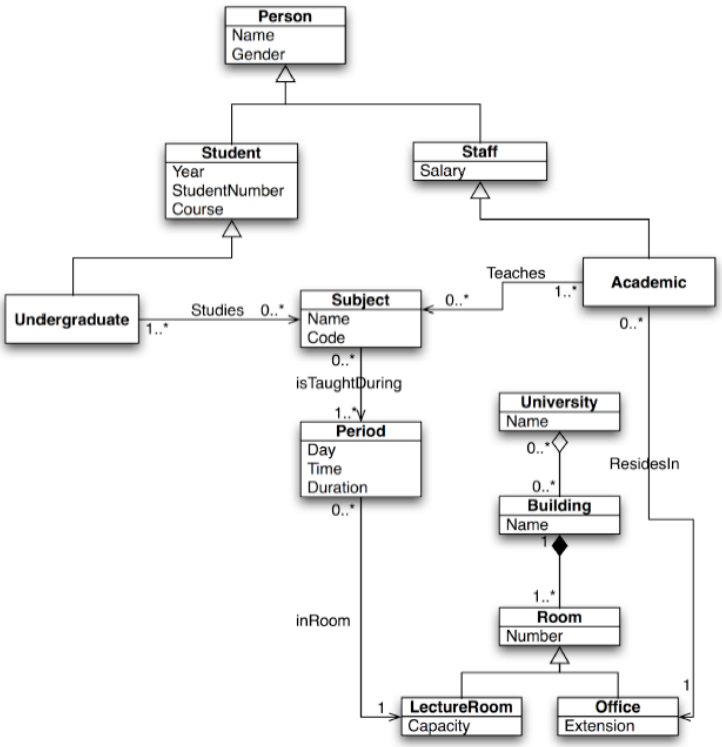


Рисунок 1.6 – Диаграмма классов, показывающая некоторые предыдущие классы и их связи

Рисунок 1.6 показывает некоторые классы из предыдущих разделов (вместе с двумя дополнительными подклассами класса Room), и как отдельные диаграммы на рисунках 1.3, 1.4 и 1.5 связаны между собой. Должно быть ясно, что система может быть развернута дальше для включения большего числа деталей из описания внутренней структуры университета: расписание для студентов и персонала или вычислительные возможности, доступные в каждой комнате. Для понимания CIM крайне необходимо понимание основ системы классов и объектно-ориентированного подхода к моделированию систем.

Задание

1. Составьте краткий конспект, ответьте на контрольные вопросы.
2. Продумайте и разработайте иерархию классов для моделирования системы, согласно вашему варианту. Опишите ее с помощью графических элементов языка UML. Для удобства составления, воспользуйтесь программным комплексом Visio (можно использовать шаблон «Схема модели UML»)

Таблица 1.1 – Варианты для задания №3

№ варианта	Система	№ варианта	Система
1	Школа	6	Банк
2	Аэропорт	7	Фитнесс-центр
3	Кинотеатр	8	Интернет-магазин
4	Автосервис	9	Автомобильный завод
5	Страховая компания	10	Книжное издательство

Контрольные вопросы

1. Что такое CIM?
2. Каковы причины и предпосылки возникновения CIM?
3. Какие подходы для решения проблемы обмены данными между приложениями могут быть предложены?
4. Какие три основных требования должен обеспечивать общий формат обмена данными?
5. Что такое UML?
6. Какие основные понятия UML вы знаете?
7. Что такое класс? Какие отношения между классами вы знаете?

ПРАКТИЧЕСКАЯ РАБОТА №2

Тема: *Основные классы общей информационной модели*

При использовании собственных форматов данных обмен информацией об энергосистеме между компаниями всегда является сложным. Раньше, компании традиционно использовали бы одно программное обеспечение или собственного производства или купленное у большой компьютерной компании, и потребовалось бы использовать только один собственный формат данных. С развитием информационных технологий и конкуренции между компаниями, появилась большая потребность в обеспечении возможности обмениваться информацией об энергосистеме между компаниями. Сложность для обмена данными добавляет рост возможностей выбора, вызванного наличием нескольких поставщиков программного обеспечения для энергосистем и доступность различных программных пакетов и архитектур. Эти вопросы обозначили потребность в едином, открытом стандарте для описания данных об энергосистеме, и способствующем совместимости между программными пакетами, а также обмену данными как в рамках одной компании, так и между компаниями.

Common Information Model (CIM) является открытым стандартом для описания компонентов энергосистемы. Он разработан Electric Power Research Institute (EPRI) в США. Стандарт был разработан как часть стандарта МЭК TC57 WG13 при разработке Control Center Application Programming Interface (CCAPI) с целью предоставить общую модель для описания компонентов в энергосистемах для использования в едином Energy Management System (EMS) Application Programming Interface (API). Формат был одобрен большинством поставщиков EMS для обеспечения обмена данными между их приложениями независимо от внутренней программной архитектуры или рабочей платформы.

Модель данных сама по себе является независимой от языка, определяя компоненты энергосистемы как классы вместе с взаимосвязями между этими классами: наследование, ассоциация и агрегация; так же определены параметры в каждом классе. Это дает основу для общей модели для представления всех

аспектов энергосистемы независимо от любого частного стандарта или формата. Это упрощает совместимость между программными комплексами, поскольку необходимо наличие только одного транслятора для конвертации из/в данные основанные на CIM, в то время как раньше потребовались бы трансляторы для конвертации из/в форматы всех других компаний.

На первый взгляд для инженера формат Common Information Model (CIM) может показаться сложным в сравнении с неструктурированным файловым форматом. Однако такой подход имеет ряд преимуществ, которые будут рассмотрены далее.

Иерархия CIM в настоящее время не имеет официального супер-класса (т.е. класса из которого наследуется каждый компонент). Однако основные CIM классы наследованы из класса IdentifiedObject, поэтому в настоящей работе он будет рассматриваться как базовый класс в иерархии.

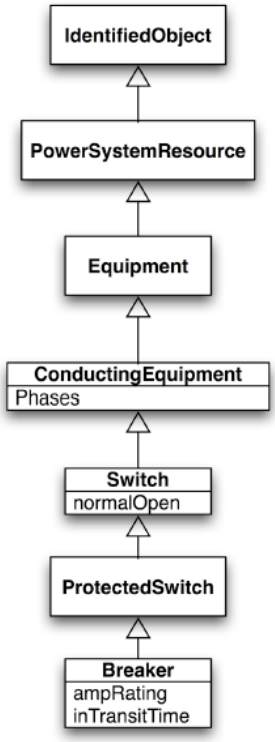
Для объяснения, почему в CIM выгоднее использовать структуру классов для определения компонентов вместо простого описания атрибутов для каждого отдельного типа компонента как независимой сущности будет использоваться простой пример.

Breaker (выключатель) – один из наиболее общих компонентов в энергосистеме – описан как «механическое переключающее устройство способное создавать, нести и прерывать токи при нормальных условиях цепи и так же создавать, нести в течение определенного времени и прерывать ток при определенных ненормальных условиях цепи». Для того, чтобы понять как это соотносится с иерархией классов CIM, Breaker может быть рассмотрен с различных уровней абстракции.

С наиболее детального уровня – это есть Выключатель, но т.к. наиболее базовая функциональность выключателя это способность быть отключенным или включенным, он может быть описан как специализированный тип переключателя. В рамках энергосистемы переключатель это часть физической сети, которая проводит электричество, и таким образом может быть рассмотрена как тип проводящего оборудования. Поскольку энергосистема может содержать оборудование, которое не проводит электричество напрямую, проводящее оборудование может быть рассмотрено как тип базового

оборудования. Часть оборудования может аналогично быть рассмотрено как ресурс энергосистемы.

Breaker может, таким образом, быть Power System Resource (ресурс энергосистемы), типом Equipment (оборудования), типом Conducting Equipment (проводящее оборудование) и типом Switch (переключатель). Это соответствует структуре иерархии классов, показанной на рисунке 2.1 ниже.



IdentifiedObject	Идентифицируемый объект
PowerSystemResource	Ресурс энергосистемы
Equipment	Оборудование
ConductingEquipment	Проводящее оборудование
Phases	Фазы
Switch	Переключатель
normalOpen	Нормально отключен
ProtectedSwitch	Защитный переключатель
Breaker	Выключатель
ampRating	Номинальный ток
inTransitTime	Время переключения

Рисунок 2.1 – Наследование иерархии классов Breaker

Класс IdentifiedObject является корневым классом для этой конкретной иерархии классов CIM. Другие классы CIM в иерархии классов Breaker:

– PowerSystemResource (ресурс энергосистемы), используется для описания любых ресурсов в энергосистеме, это может быть как физическое оборудование, такое как Switch (переключатель), так и организационная сущность, такая как SubControlArea (контролируемая подобласть);

– Equipment (оборудование), к которому относится любая часть энергосистемы, представляющая из себя физическое устройство не зависимо от того оно электрическое или механическое;

– ConductingEquipment (проводящее оборудование), используется для определения типов Equipment, которое разработано для несения тока или

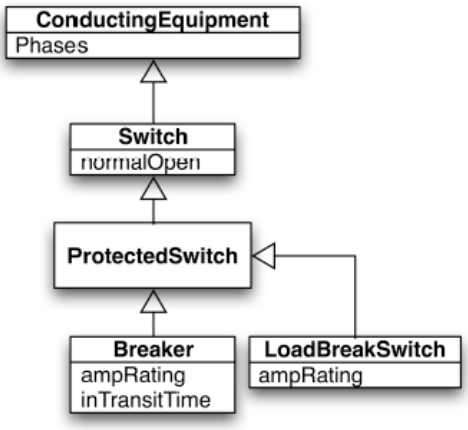
которое электрически присоединено к сети и содержит атрибуты для обозначения фаз (A, B, C, N или их любые комбинации);

– Switch (переключатель) – общий класс для любого проводящего оборудования, которое работает как переключатель в сети и следовательно имеет атрибут для определения нормального состояния переключателя: отключен или включен;

– ProtectedSwitch (защитный переключатель) – переключатель, который может управляться защитным оборудованием;

– Breaker (выключатель) – специальный подтип ProtectedSwitch с дополнительными атрибутами для определения номинального тока и времени переключения.

Как и с примером внутренней структуры университета в предыдущей работе, все подклассы наследуют атрибуты от их родительского класса, и по существу Breaker будет содержать атрибут normalOpen из класса Switch и атрибут phases из класса ConductingEquipment наравне со своими собственными атрибутами. Помимо класса Breaker, CIM стандарт содержит множество подклассов класса Switch, включая Jumper (перемычка), Fuse (предохранитель), Disconnector (разъединитель), LoadBreakSwitch (выключатель нагрузки) и GroundDisconnector (заземлитель).



ConductingEquipment	Проводящее оборудование
Phases	Фазы
Switch	Переключатель
normalOpen	Нормально отключен
ProtectedSwitch	Защитный переключатель
Breaker	Выключатель
ampRating	Номинальный ток
inTransitTime	Время переключения
LoadBreakerSwitch	Выключатель нагрузки

Рисунок 2.2 – Класс Switch с подклассами Breaker и LoadBreakSwitch

На рисунке 2.2 приведен пример, показывающий как класс LoadBraekSwitch, подкласс класса ProtectedSwitch вписывается в иерархию классов. Оба класса Breaker и LoadBreakSwitch наследованы из класса ProtectedSwitch, который в свою очередь наследован из класса Switch, таким

образом, они оба содержат атрибут normalOpen и поддерживают свои собственные атрибуты.

Так же, рассматривая их как обычный класс, система может рассматривать компоненты Breaker или LoadBreakSwitch как ProtectedSwitch, Switch, часть Condacting Equipment, часть Equipment, Power System Resorce или прямо IdentifiedObject.

Преимущества объектной модели по сравнению с традиционными форматами данных

Для объяснения, почему в качестве стандарта обмена данными был выбран именно объектно-ориентированный подход, рассмотрим пример, приведенный на рисунке 2.3.

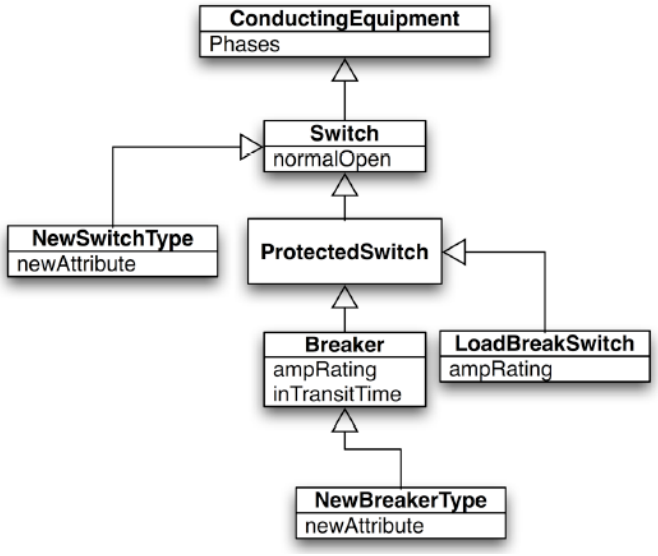


Рисунок 2.3 – Диаграмма класса Switch с новыми подклассами классов Switch и Breaker

При выполнении каким-либо блоком программы топологического анализа сети энергосистемы, необходимо знать состояние каждого переключателя: включен или отключен. Программе не нужно знать является ли переключатель выключателем или выключателем нагрузки или любым другим подтипом переключателя, т.к. интересуемый атрибут normalOpen существует в любом классе, который наследован от класса Switch. Если во время обработки модели будет найден компонента из класса Switch или любого из его подклассов, то

программа извлечет значение атрибута normalOpen и продолжит работу в соответствие с его значением.

Если новый тип класса Switch – NewSwitchType – будет добавлен в стандарт в последующем, как показано на рисунке 2.3, предполагая, что исходный класс Switch не будет изменен, тогда программа при выполнении анализа будет все равно способна обрабатывать этот NewSwitchType как будто бы он является переключателем. Даже если класс не существует на момент написания программы, она будет искать все компоненты, которые наследованы от класса Switch.

Аналогично, если новый подкласс класса Breaker – NewBreakerType – будет добавлен (как показано на рисунке 2.3), он все равно будет принадлежать типу класса Switch (т.к. его родительский класс – Breaker – является подклассом класса ProtectedSwitch, который сам является подклассом класса Switch) и может рассматриваться программой как Switch, ProtectedSwitch или Breaker.

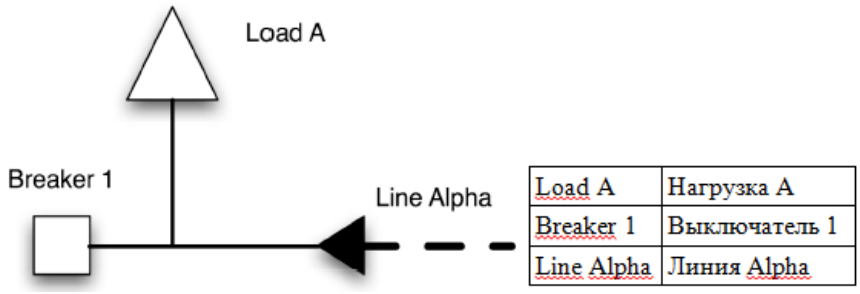
Как было показано, такое использование иерархии наследования для определения компонентов позволяет классам в рамках системы быть определенными как специализированные подклассы общего родительского класса до тех пор, пока не будет достигнут желаемый уровень детализации – от общего класса PowerSystemResource прямо до классов Breaker или LoadBreakSwitch.

Такое использование иерархии классов так же позволяет делать расширения для стандарта путем расширения существующих классов вместо введения абсолютно новых, независимых сущностей. Такой подход, как показано, может позволить существующим программам интерпретировать новые данные, хотя и на более высоком уровне абстракции, без необходимости существенной модификации.

Задание связей между элементами сети

Для определения, каким образом компоненты энергосистемы объединены вместо описания прямого соединения между компонентами в CIM используется понятия Terminal (терминал) и Connectivity Node (соединительный узел).

Для понимания, почему принят этот подход рассмотрим очень простой пример – схему, показанную на рисунке 2.4 ниже.



Load A	Нагрузка A
Breaker 1	Выключатель 1
Line Alpha	Линия Alpha

Рисунок 2.4 – Пример схемы для иллюстрации связности

Эта схема, содержащая Breaker (выключатель), Load (нагрузка) и Line (линия), потребовала бы три CIM объекта для представления блоков физического оборудования: Energy Consumer (для представления нагрузки), Breaker и AC или DC Line Segment для линии.

CIM не моделирует связанность с помощью ассоциации каждого компонента с каждым компонентом с которым он связан, т.к. если бы Breaker 1 содержал ассоциацию с Load A и Line Alpha; Load A содержала бы ассоциацию с Line Alpha и Breaker 1; и Line Alpha содержала бы ассоциацию с Breaker 1 и Load A, то результат соединения был бы определен так как показано на рисунке 2.5.

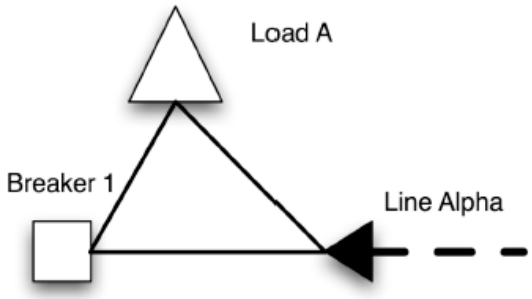
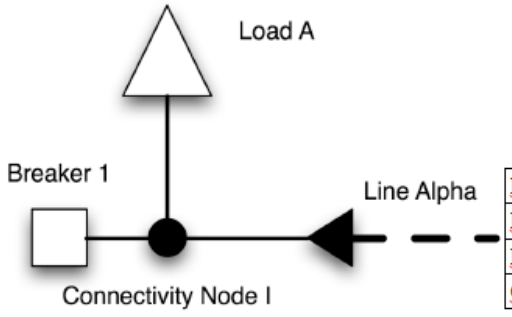


Рисунок 2.5 – Пример связности схемы с прямыми ассоциациями

Вместо этого CIM использует Connectivity Node для соединения оборудования, так что три или более элемента должны соединяться в T- или звездообразной точке. На рисунке 2.6 связность представлена корректно.

В CIM, однако, проводящее оборудование не ассоциируется напрямую с Connectivity Node. Проводящее оборудование будет иметь одну или более

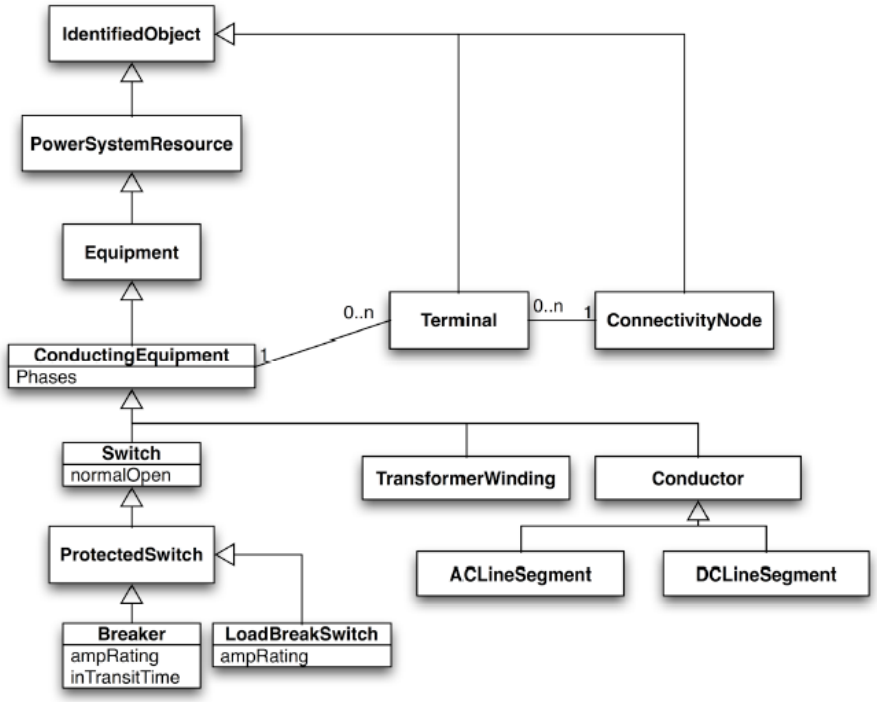
ассоциаций с объектами Terminal, которые в свою очередь ассоциированы с одним объектом ConnectivityNode.



Load A	Нагрузка A
Breaker 1	Выключатель 1
Line Alpha	Линия Alpha
Connectivity Node 1	Соединительный узел 1

Рисунок 2.6 – Пример связности схемы с использованием Connectivity Node

Взаимосвязи между классами Terminal, ConnectivityNode и ConductingEquipment показаны на рисунке 2.7.



IdentifiedObject – Идентифицируемый объект, PowerSystemResource –Ресурс энергосистемы, Equipment – Оборудование, ConductingEquipment – Проводящее оборудование, Phases – Фазы, Switch – Переключатель, normalOpen – Нормально отключен, ProtectedSwitch – Защитный переключатель, Breaker – Выключатель, ampRating – Номинальный ток, inTransitTime – Время переключения, LoadBreakSwitch – Выключатель нагрузки, Terminal – Терминал, ConnectivityNode – Соединительный узел, TransformerWinding – Обмотка трансформатора, Conductor – Проводник, ACLineSegment – Сегмент линии переменного тока, DCLineSegment – Сегмент линии постоянного тока

Рисунок 2.7 – Диаграмма классов проводящего оборудования и связности

Поскольку только проводящее оборудование проводит ток в сети, то ассоциация к классу Terminal есть только из класса ConductingEquipment с

множественностью 0..n, т.к. проводящее оборудование может иметь ноль или более соединений с сетью. Соответствующая связь от Terminal к Conducting Equipment имеет множественность 1, т.к. объект Terminal может быть связан только с одним объектом ConnectivityNode. Поскольку класс Breaker (через родительский класс Switch), Energy Consumer и AC или DC Line Segment (через класс Conductor) все наследованы из Conducting Equipment, то они так же наследуют ассоциативную связь с классом Terminal.

На рисунке 2.8 a) ниже показана соединительная связь между терминалами, проводящим оборудованием и соединительными узлами.

Введение объектов Terminal изначально может показаться излишним, но совместно с описанием связности, они используются для определения мест измерений потоков мощности, токов, напряжений.

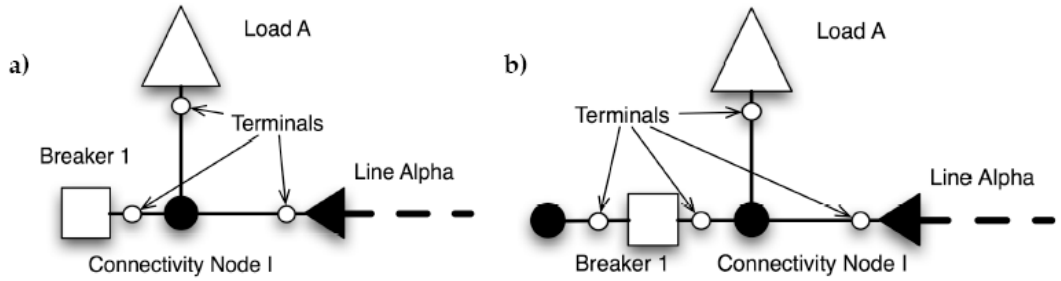


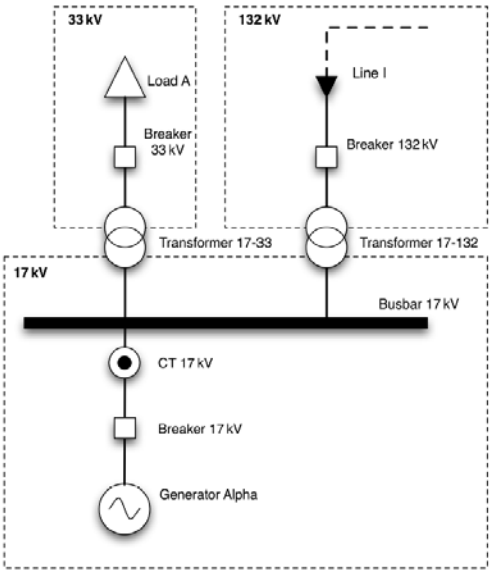
Рисунок 2.8 – Пример связности схемы с использованием объектом ConnectivityNode и Terminal

Важность возможности определения точки измерения с такой точностью может быть проиллюстрирована на рисунке 2.8 b). В этой диаграмме Breaker 1 ассоциирован с двумя Терминалами для описания фактических точек подключения, которые он имел бы в настоящей энергосистеме. Если Выключатель отключен, тогда измерения напряжения для Выключателя в этих точках будут различными. Определение измерения как части определенного компонента без указания места измерения, привело бы к неопределенности.

Конвертация схемы в CIM объекты

Выше была показана малая часть иерархии классов для описания CIM компонентов и показано как объекты Terminal и ConnectivityNode используются для определения связности компонентов в сети. Давайте рассмотрим более сложный пример, чтобы показать, как моделируются уровни напряжения, трансформаторы тока, силовые трансформаторы и генераторы путем конвертации стандартной однолинейной схемы в CIM объекты.

На рисунке 2.9 показана схема, содержащая один генерирующий источник, нагрузку, линию и шины. Схема так же содержит два силовых трансформатора и три уровня напряжения: 17 кВ, 33 кВ и 132 кВ. Нагрузка, линия и выключатели, как было сказано ранее, относятся соответственно к CIM классам EnergyConsumer, ACLineSegment и Breaker, а шины соответствуют классу BusbarSection. Генератор Alpha будет соответствовать одному из типов проводящего оборудования – SynchronousMachine – «электромеханическое устройство, которое работает синхронно с сетью». При работе в качестве генератора, объект SynchronousMachine должен иметь ассоциацию с экземпляром класса GeneratingUnit.



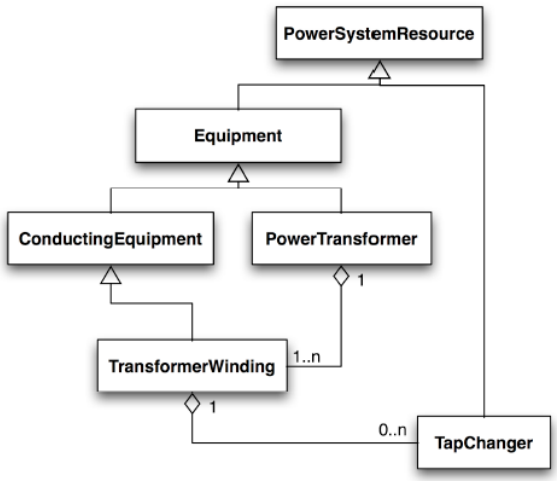
Load A – Нагрузка А; EnergyConsumer – Потребитель энергии; Breaker 17 kV, 33 kV, 132 kV – Выключатель 17 кВ, 33 кВ, 132 кВ; Line 1 – Линия 1; ACLineSegment – Сегмент линии переменного тока; Transformer 17-33, 17-132 – Трансформатор 17-33, 17-132; Busbar 17 kV – Система шин 17 кВ; BusbarSection – Секция системы шин; CT 17 kV – Трансформатор тока 17 кВ; Generator Alpha – Генератор Alpha; SynchronousMachine – Синхронная машина; GeneratingUnit – Генерирующая единица

Рисунок 2.9 – Пример схемы, части элементов которой поставлены в соответствие CIM классы

На рисунке 2.9 не указано соответствие CIM классам только для двух силовых трансформаторов и трансформатора тока.

Представление силовых трансформаторов как CIM объектов

Силовому трансформатору не поставлен в соответствие какой-либо один CIM класс, вместо этого он разделен на ряд компонентов с одним классом-контейнером PowerTransformer. Так, двухобмоточный трансформатор становится двумя объектами TransformerWinding, входящими в класс-контейнер PowerTransformer. Если есть устройство переключения анцапф для управления обмотками, тогда объект TapChanger ассоциируется с конкретной обмоткой и так же входит в состав объекта PowerTransformer. На рисунке 2.10 ниже показана UML диаграмма классов, формирующих трансформатор.



PowerSystemResource – Ресурс энергосистемы; Equipment – Оборудование; ConductingEquipment – Проводящее оборудование; PowerTransformer – Силовой трансформатор; TransformerWinding – Обмотка трансформатора; TapChanger – Переключатель анцапф

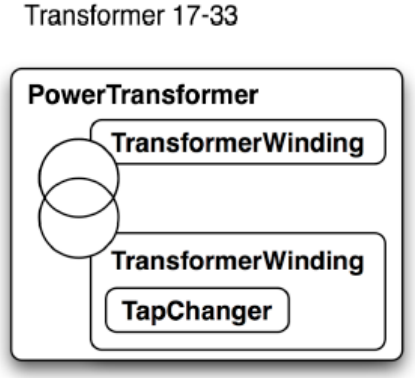
Рисунок 2.10 – Диаграмма классов модели трансформатора

Хотя класс PowerTransformer в системе является частью класса Equipment, он не проводит электричество сам по себе и потому не наследуется от класса ConductingEquipment, а наследуется от его родительского класса Equipment. Класс TransformerWinding, однако, наследуется от класса ConductingEquipment т.к. он физически подключен к сети и проводит электричество. Класс TapChanger является частью класса TransformerWinding и в связи с этим не может рассматриваться как отдельный тип оборудования со своими собственными правами и наследуется только от класса PowerSystemResource.

Классы PowerTransformer и TransformerWinding имеют связь агрегация, означающую, что объект PowerTransformer состоит из 1 или более объектов TransformerWindings, которые в свою очередь могут состоять из 0 или более объектов TapChanger.

Когда рассматривается физический трансформатор, расположенный на подстанции, класс-контейнер PowerTransformer может быть представлен как кожух трансформатора. Кожух сам по себе не проводит электричество в сети, а вместо этого содержит обмотки трансформатора, изоляционный материал, магнитопровод и все другие компоненты, которые составляют трансформатор.

Присоединение трансформатора к сети выполняют обмотки сами по себе. Эта связь отражена в CIM представлении, в котором есть класс TransformerWinding, наследуемый от класса ConductingEquipment. Таким образом, Трансформатор 17-33 на рисунке 2.9 может быть представлен как 4 CIM объекта: два объекта TransformerWinding, один объект TapChanger и один объект PowerTransformer. Такое представление приведено на рисунке 2.11.



PowerTransformer – Силовой трансформатор; TransformerWinding – Обмотка трансформатора; TapChanger – Переключатель анцапф

Рисунок 2.11 – CIM представление Трансформатора 17-33

Аналогично трансформатор с тремя обмотками может быть представлен как один экземпляр класса PowerTransformer, содержащий три экземпляра класса TransformerWinding.

Представление трансформатора тока как CIM объекта

Трансформатор тока CT 17 kV не соответствует напрямую проводящему оборудованию в CIM иерархии, как это могло бы ожидаться. Назначение

трансформатора тока это измерять ток в месте установки и таким образом при моделировании сети он является измерением с определенного места, а не оборудованием, делающим измерение.

Это приводит к созданию объекта Measurement для измерения тока в конкретном месте. Каждое отдельное проводящее оборудование имеет один или более терминалов для описания точек, которыми оно присоединено к сети. Если ассоциировать объект Measurement с определенным терминалом и указать, что это измерение является током, тогда объект Measurement будет отражать роль, играемую трансформатором тока.

Классификаторы (контейнеры)

Наряду с соединением компонентов, определяемого использованием ассоциаций ConductingEquipment-Terminal-ConnectivityNode, CIM содержит класс EquipmentContainer, который дает средства для группировки различного оборудования для отражения как электрического, так и неэлектрического содержимого.

Уровни напряжения

Проводящее оборудование не имеет атрибута voltage для определения напряжения как специфической величины, вместо этого они ассоциированы с объектом VoltageLevel, который является подклассом класса EquipmentContainer. Каждый объект VoltageLevel сам по себе ассоциирован с объектом BaseVoltage, который содержит один атрибут для определения номинального напряжения этой отдельной группы компонентов. Объект BaseVoltage может быть ассоциирован с более чем одним объектом VoltageLevel, т.к. стандартные уровни напряжения (например, 6, 10, 35, 110 кВ), будут существовать во всей сети. Каждый объект VoltageLevel, однако, содержит присоединенное оборудование только одного уровня напряжения. Таким образом может быть использован подкласс EquipmentContainer для представления электрического содержимого.

Подстанции

Класс Substation (подстанция) является подклассом класса EquipmentContainer, который может содержать множество объектов VoltageLevel и используется для определения коллекции оборудования «через которое электрическая энергия в большом количестве проходит для целей переключения или изменения ее характеристик».

В примере сети, показанном на рис. 2.9, трем различным уровням напряжения, отмеченным пунктирной линией, соответствует три объекта VoltageLevel, содержащихся в одном объекте Substation. Каждый экземпляр класса VoltageLevel также имеет ассоциированный с ним объект BaseVoltage с номинальным напряжением 17, 33 и 132 кВ.

Класс Substation, будучи подклассом класса EquipmentContainer может так же содержать другие объеты Equipment, такие как PowerTransformer, который как ранее было показано сам является контейнером, а не проводящим оборудованием. Класс Substation является примером подкласса класса EquipmentContainer для представления неэлектрического содержимого, т.к. он содержит оборудование, которое физически сгруппировано, но не обязательно соединено электрически.

Линии

Определение класса Line в CIM: «компонент разделяет систему, растянувшуюся между смежными подстанциями или от подстанции к смежной точке соединения». Линия может содержать множество сегментов линии постоянного (класс DCLineSegmen) или переменного тока (класс ACLineSegment), но сама не является физическим проводящим оборудованием. Объекты ACLineSegment и DCLineSegmen, не содержится внутри объекта VoltageLevel, вместо этого они содержатся в объекте Line (линия).

Поскольку сегмент линии используется для представления «провода или комбинации проводов … используемых для передачи переменного (или постоянного) тока между точками в энергосистеме», было бы некорректно определять его внутри определенного уровня напряжения в рамках подстанции.

В связи с этим классы ACLineSegmen и DCLineSegmen содержат прямую ассоциацию с классом BaseVoltage для определения их номинального уровня напряжения.

Эквивалентное CIM представление

После полной конвертации в CIM объекты, пример исходной схемы, показанной на рисунке 2.9, представлен в виде 45 CIM объектов, показанных на рисунке 2.12. Расположение экземпляра класса BusbarSection может показаться ошибочным, но в CIM для определения точки соединения оборудования используются объекты ConnectivityNodes. В связи с этим объектом BusbarSection используется в основном для предоставления точки для ассоциации (через свой терминал) объектов измерения, измеряющих напряжение на этой конкретной шине в системе. Это отражает расположение оборудование в физической системе, т.к. трансформатор напряжения часто измеряет напряжение на шинах подстанции.

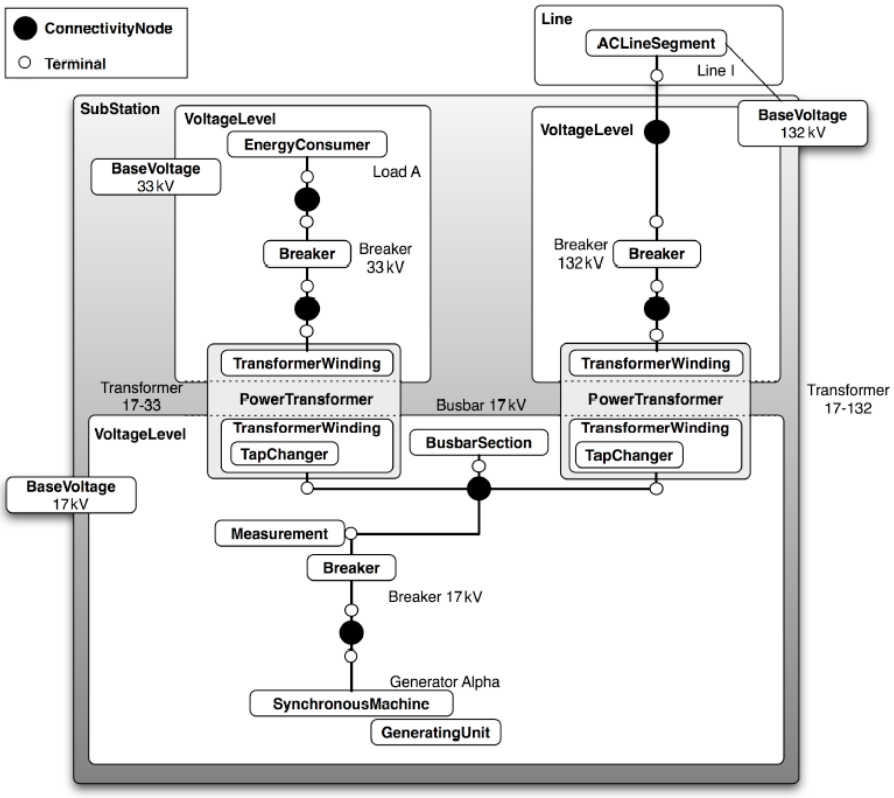


Рисунок 2.12 – Пример схемы, полностью сопоставленной с CIM

Этот пример схемы может быть продолжен далее с дополнением объектов

для представления областей управления, собственников оборудования, единиц измерения и графиков генерации и потребления, но для начала этого достаточно для понимания того, как существующее представление сети может быть сопоставлено CIM объектам.

Задание

1. Составьте краткий конспект, ответьте на контрольные вопросы.

2. Выполните сопоставление схемы сети с CIM объектами, согласно вашему номеру варианта. Для моделирования разъединителя используйте класс CIM Disconnector

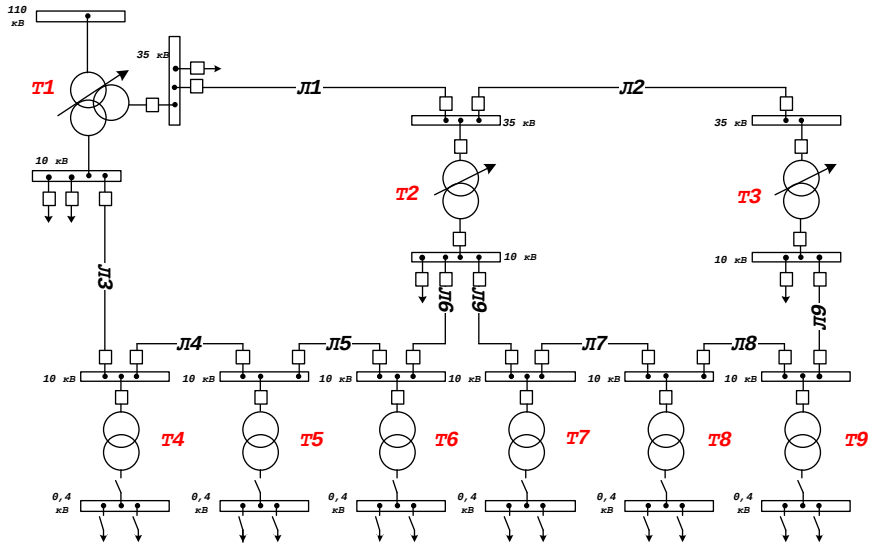


Рисунок 2.13 – Схема сети для сопоставления с CIM

Таблица 2.1 – Варианты заданий

№ варианта	Фрагмент сети	№ варианта	Фрагмент сети
1	T1,T2,T3,T9 + соединяющие линии	6	T2, T6, T7, T8+ соединяющие линии
2	T1,T4,T5,T6 + соединяющие линии	7	T1,T2,T3, T6 + соединяющие линии
3	T1,T2,T4,T5+ соединяющие линии	8	T1,T2,T3, T7 + соединяющие линии
4	T1,T3,T9,T8+ соединяющие линии	9	T6, T7, T8, T9 + соединяющие линии
5	T1,T4,T2,T6+ соединяющие линии	10	T2, T6, T7, T8 + соединяющие линии

3. Оформите отчет о выполненной работе.

Контрольные вопросы

1. Какие классы CIM вы знаете?
2. С помощью каких классов в ОИМ моделируются линии электропередач?
3. С помощью каких классов в ОИМ моделируются силовые трансформаторы?
4. Каким образом в ОИМ задаются связи между объектами?
5. Что такое терминал и соединительный узел?
6. Что такое контейнеры? Какие контейнеры вы знаете?

ПРАКТИЧЕСКАЯ РАБОТА №3

Тема: *CIM-XML формат представления общей информационной модели сети*

Общие сведения об XML и RDF

XML, eXtensible Markup Language это «универсальный формат для структурированных документов и данных», который быстро становится стандартом для хранения машинно-читабельных данных в структурированном, расширяемом формате, который доступен через интернет.

Фактически XML – это метаязык, который позволяет пользователю разрабатывать свой собственный маркированный язык для описания структуры данных.

XML синтаксис использует тэги для обозначения элементов внутри документа. Каждый элемент либо выражен как открывающийся и закрывающийся тэг, содержащий данные в виде:

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

Или в виде одного пустого тега с закрывающимся тэгом в конце

```
<tag/>
```

Элемент может так же содержать свои собственные атрибуты, которые выражаются в форме:

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

Когда элемент имеет начальный и конечный тэги, то любой другой элемент, содержащийся в этих тэгах, классифицируется как дочерний этого родительского элемента.

В качестве примера приведен простой XML тэг-синтаксис для хранения книги. Содержание и свойства книги затем могут быть выражены как XML, используя самоописывающиеся тэги в виде:

```
<book title="Introduction to XML" author="Alan McMorran">
    <revision number="2">
```

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

Здесь элемент book содержит собственные атрибуты для описания title и author, с дочерними элементами для описания revision книги, плюс несколько элементов chapter. Элементы chapter в свою очередь содержат элементы для каждого элемента paragraph, который сам содержит смешанные данные других элементов и текст. Хотя для любого со знанием английского языка имена этих тэгов семантически ясны, для корректной интерпретации их приложением синтаксис тэгов и семантика должны быть четко определены. Хотя XML сам по себе не содержит определений тэг-синтаксиса или семантики, с использованием XML нотации могут быть определены схемы для описания практически любого типа данных. Приложение, интерпретирующее XML данные должно знать используемые синтаксис и семантику, иначе оно будет иметь сложности с их интерпретацией. Это требует того, чтобы тэг-синтаксис и семантика XML были выражены в виде схемы, которая накладывает ограничения на структуру и содержание XML документа.

Набор правил использования тегов с определенными именами называется «XML схема». XML схема определяет элементы и атрибуты, которые могут быть в документе; какие элементы являются дочерними; число возможных дочерних элемента для каждого типа элементов; может ли элемент включать текст (например, является элемент пустым или с открывающимся и закрывающимся тэгами); тип данных элементов и атрибутов; являются ли их значения фиксированными; и имеют ли они значение по умолчанию.

В рамках базового XML документа не существует способа указать связь между двумя элементами, кроме как связей между родительскими и дочерними

элементами. Например, рассмотрим библиотечную систему, содержащую записи о множестве книг с информацией об их позиции на полке:

```
<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>
```

Каждый элемент book содержится в библиотеке как независимая запись, и если бы пользователь захотел добавить связь между книгами History of Glasgow, 1900-1950 и History of Glasgow, 1950-2000, для того, чтобы указать, что читатель может захотеть прочитать сначала более раннюю книгу, то с помощью базовых XML конструкций не существует стандартного способа сделать это. Resource Document Framework (RDF) это XML схема, используемая для обеспечения общего пространства данных в XML формате для обеспечения возможности устанавливать связи между XML узлами. Каждому элементу может быть присвоен уникальный атрибут ID в пространстве имен RDF (в котором используется префикс rdf). Добавление атрибута resource в элемент позволяет делать ссылки между элементами путем присваивания ему величины, указывающей на ID другого элемента. Для приведенного выше примера с библиотекой, присвоение каждой книге ID в соответствии с пространством имен RDF, позволяет включать дополнительные элементы sequel и sequelTo. Эти элементы содержат только единственный атрибут resource, который указывает на другой элемент в рамках этого документа путем ссылки на его ID.

```
<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"/>
    </lib: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>
```

Таким образом, с помощью XML и RDF можно описать любые CIM объекты, их свойства и связи между ними.

Упрощенные XML эквиваленты CIM объектов

Рассмотрим упрощенные XML эквиваленты рассмотренных ранее CIM объектов. Будем рассматривать эквиваленты в упрощенном виде, так как полное описание CIM объектов может быть довольно сложным и содержать десятки свойств каждого объекта, которые в учебных целях не имеют первостепенного значения. Также в целях упрощения не будет объединять элементы в контейнеры по номинальному напряжению. Полный список иерархии CIM-объектов и их свойств можно посмотреть по адресу: https://ontology.tno.nl/IEC_CIM/

ConnectivityNode

Класс «ConnectivityNode» наследуется от классов IdentifiedObject, и rdfs:Resource. От класса Resource наследуется существенное свойство ID – идентификатор объекта, от класса IdentifiedObject наследуется существенное свойство name – имя объекта, а сам класс ConnectivityNode содержит существенное свойство Terminals – список треминалов соединенных с узлом объектов. С учетом сказанного, упрощенный XML-эквивалент объекта класса «ConnectivityNode» будет выглядеть следующим образом:

```
<cim:ConnectivityNode rdf:ID="#1">
  <cim:IdentifiedObject.name>Узел1</cim:IdentifiedObject.name>
  <cim:ConnectivityNode.Terminals rdf:resource="#7"/>
</cim:ConnectivityNode>
```

BusbarSection

Класс «BusbarSection» наследуется от классов «Connector», «ConductingEquipment», «Equipment», «PowerSystemResource», «IdentifiedObject» и «rdfs:Resource». Перечисленные классы имеют множество свойств, но для упрощения мы оставим только свойства ID от класса Resource, свойство name от класса IdentifiedObject и свойство Terminals от класса ConductingEquipment. С учетом сказанного, упрощенный XML-эквивалент объекта класса «BusbarSection» будет выглядеть следующим образом:

```
<cim:BusbarSection rdf:ID="#18">
  <cim:IdentifiedObject.name>Шина ВН</cim:IdentifiedObject.name>
  <cim:ConductingEquipment.Terminals rdf:resource="#19"/>
</cim:BusbarSection>
```

Terminal

Класс «Terminal» наследуется от классов «ACDCTerminal», «IdentifiedObject» и «rdfs:Resource». Среди существенных свойств оставим ID (идентификатор), sequenceNumber (порядковый номер терминала), ConnectivityNode (соединительный узел, с которым он связан), ConductingEquipment (объект, к которому принадлежит терминал).

```
<cim:Terminal rdf:ID="#19">
  <cim:ACDCTerminal.sequenceNumber>1</cim:ACDCTerminal.sequenceNumber>
  <cim:Terminal.ConductingEquipment rdf:resource="#18"/>
  <cim:Terminal.ConnectivityNode rdf:resource="#3"/>
</cim:Terminal>
```

Breaker

Класс «Breaker» наследуется от классов «ProtectedSwitch», «Switch», «ConductingEquipment», «Equipment», «PowerSystemResource», «IdentifiedObject» и «rdfs:Resource». Среди существенных оставим свойство ID (идентификатор), name (имя), Terminals (терминалы) и open (вкл/выкл):

```
<cim:Breaker rdf:ID="#1">
    <cim:IdentifiedObject.name>В 1</cim:IdentifiedObject.name>
    <cim:ConductingEquipment.Terminals rdf:resource="#2"/>
    <cim:ConductingEquipment.Terminals rdf:resource="#3"/>
    <cim:Switch.open>true</cim:Switch.normalOpen>
</cim:Breaker>
```

EnergyConsumer

Класс «EnergyConsumer» наследуется от классов «ConductingEquipment», «Equipment», «PowerSystemResource», «IdentifiedObject» и «rdfs:Resource». Среди существенных оставим свойство ID (идентификатор), name (имя), Terminals (терминалы), pfixed (активная мощность нагрузки), qfixed (реактивная мощность нагрузки)

```
<cim:EnergyConsumer rdf:ID="#19">
  <cim:IdentifiedObject.name>Потребитель 1</cim:IdentifiedObject.name>
  <cim:ConductingEquipment.Terminals rdf:resource="#2"/>
  <cim:EnergyConsumer.pfixed>0.5</cim:EnergyConsumer.pfixed>
  <cim:EnergyConsumer.qfixed>0.3</cim:EnergyConsumer.qfixed>
</cim:EnergyConsumer>
```

Line

Класс «Line» наследуется от классов «EquipmentConatainer», «ConnectivityNodeContainer», «PowerSystemResource», «IdentifiedObject» и «rdfs:Resource». Среди существенных свойств оставим ID (идентификатор), name (имя), Equipments (оборудование – участки линии, которые образуют линию)

```
<cim:Line rdf:ID="#5">
  <cim:IdentifiedObject.name>Без имени</cim:IdentifiedObject.name>
  <cim:EquipmentContainer.Equipments rdf:resource="#6"/>
</cim:Line>
```

ACLineSegment

Класс «ACLineSegment» наследуется от классов «Conductor», «ConductingEquipment», «Equipment», «PowerSystemResource», «IdentifiedObject» и «rdfs:Resource». Среди существенных свойств оставим ID (идентификатор), name (название), Terminals (список терминалов), r (активное сопротивление), x (реактивное сопротивление), gch (активная проводимость), bch (реактивная проводимость).

```
<cim:ACLineSegment rdf:ID="#6">
  <cim:IdentifiedObject.name>Без имени</cim:IdentifiedObject.name>
  <cim:Equipment.EquipmentContainer rdf:resource="#5"/>
  <cim:ConductingEquipment.Terminals rdf:resource="#7"/>
  <cim:ConductingEquipment.Terminals rdf:resource="#8"/>
  <cim:ACLineSegment.r>0</cim:ACLineSegment.r>
  <cim:ACLineSegment.x>0</cim:ACLineSegment.x>
```

```
    <cim:ACLineSegment.gch>0</cim:ACLineSegment.gch>
    <cim:ACLineSegment.bch>0</cim:ACLineSegment.bch>
</cim:ACLineSegment>
```

PowerTransformer

Класс «PowerTransformer» наследуется от классов «ConductingEquipment», «Equipment», «PowerSystemResource», «IdentifiedObject» и «rdfs:Resource». Среди существенных свойств оставим ID (идентификатор), name (название), Terminals (список терминалов), PowerTransformerEnd (список обмоток трансформатора).

```
<cim:PowerTransformer rdf:ID="#15">
    <cim:IdentifiedObject.name>Без имени</cim:IdentifiedObject.name>
    <cim:ConductingEquipment.Terminals rdf:resource="#16"/>
    <cim:ConductingEquipment.Terminals rdf:resource="#17"/>
    <cim:PowerTransformer.PowerTransformerEnd rdf:resource="#13"/>
    <cim:PowerTransformer.PowerTransformerEnd rdf:resource="#14"/>
</cim:PowerTransformer>
```

PowerTransformerEnd

В последних версиях CIM модели, класс «TransformerWinding» заменен на «PowerTransformerEnd». Класс «PowerTransformerEnd» наследуется от классов «TransformerEnd», «IdentifiedObject» и «rdfs:Resource». Среди существенных свойств оставим ID (идентификатор), name (название), endNumber (номер обмотки), Terminal (терминал (клемма) обмотки), PowerTransformer (трансформатор, в составе которого находится обмотка), r (активное сопротивление обмотки), x (реактивное сопротивление обмотки), g (активная проводимость обмотки), b (реактивная проводимость обмотки), ratedU (номинальное напряжение обмотки).

```
<cim:PowerTransformerEnd rdf:ID="#13">
    <cim:IdentifiedObject.name>ВН</cim:IdentifiedObject.name>
    <cim:TransformerEnd.endNumber>1</cim:TransformerEnd.endNumber>
    <cim:TransformerEnd.Terminal rdf:resource="#16"/>
    <cim:PowerTransformerEnd.r>1</cim:PowerTransformerEnd.r>
    <cim:PowerTransformerEnd.x>2</cim:PowerTransformerEnd.x>
    <cim:PowerTransformerEnd.g>3</cim:PowerTransformerEnd.g>
    <cim:PowerTransformerEnd.b>4</cim:PowerTransformerEnd.b>
    <cim:PowerTransformerEnd.ratedU>10</cim:PowerTransformerEnd.ratedU>
    <cim:PowerTransformerEnd.PowerTransformer rdf:resource="#15"/>
```

```
    </cim:PowerTransformerEnd>
```

С учетом принятых к рассмотрению существенных свойств классов, получается следующая иерархия классов (рисунок 3.1).

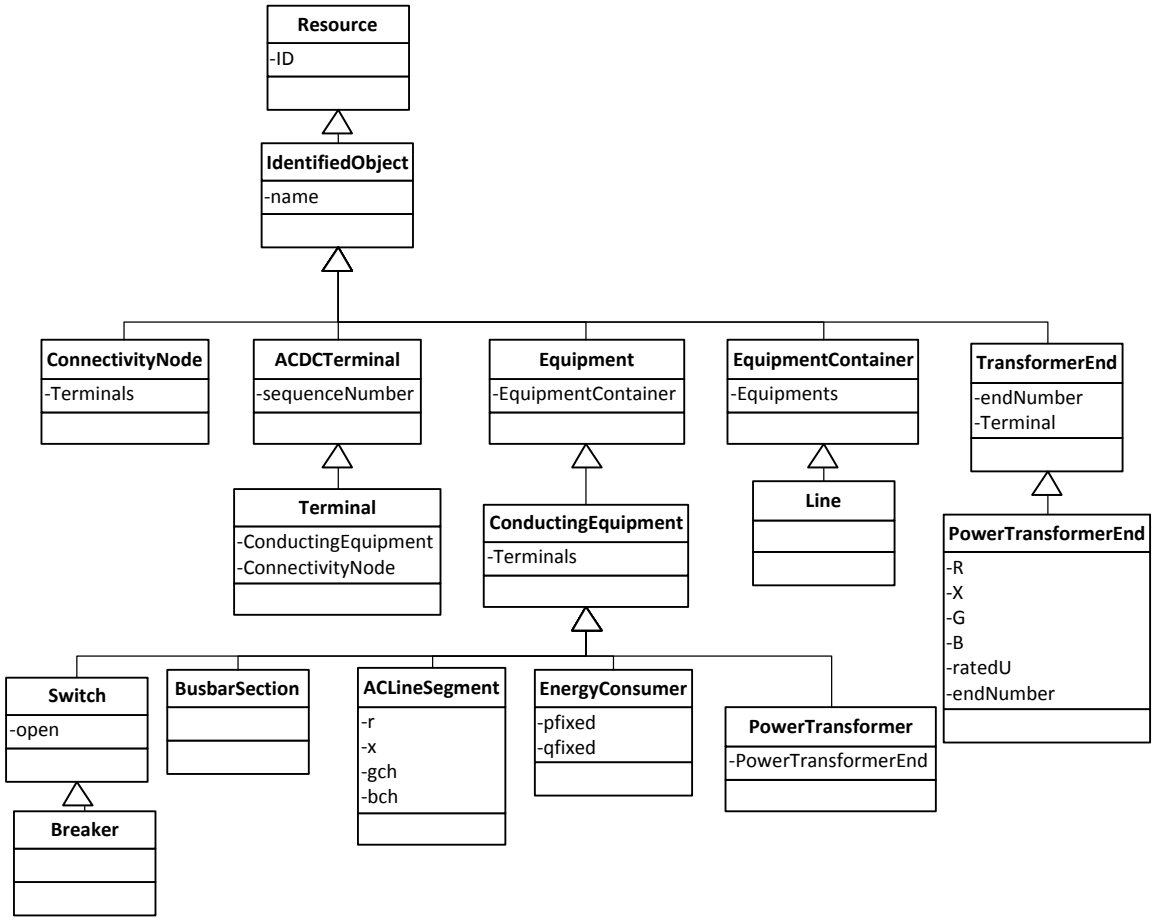


Рисунок 3.1 – Упрощенная иерархия CIM классов

Задание

1. Напишите краткий конспект, ответьте на контрольные вопросы.

2. Сформируйте XML-эквивалент разработанной в предыдущей работе общей информационной модели вашего варианта схемы сети.

Контрольный вопросы

1. Что такое XML? Для чего они применяется в CIM?
2. Что такое RDF? Для чего он применяется в CIM?

ПРАКТИЧЕСКАЯ РАБОТА №4

Тема: *Упрощенная цифровая модель сети*

Общие сведения о модели сети

В системах управления и сбора данных для организации процесса управления используются сложные математические вычисления, множество математических моделей реальных объектов, которые в совокупности формируют цифровую модель или цифровой двойник управляемого объекта.

В электроэнергетических системах цифровую модель управляемого объекта в самом упрощенном варианте можно представить в виде множества узлов (подстанции, опоры) и ветвей (линии электропередач, трансформаторы). Ветви разделяются на продольные и поперечные. Каждый узел связан с хотя бы одним другим узлом схемы сети посредством продольных ветвей. Узел описывается мощностью, проводимостью поперечной ветви и напряжением (рисунок 4.1).

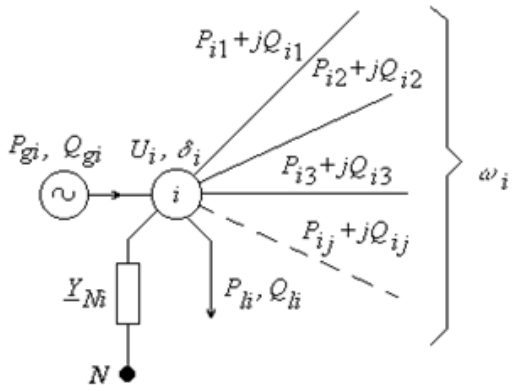


Рисунок 4.1 – Модель узла схемы сети

Для узла *i*:

P_{gi} – активная мощность генерации;
Q_{gi} – реактивная мощность генерации;
P_{li} – активная мощность нагрузки;
Q_{li} – реактивная мощность нагрузки;
$\underline{Y}_{Ni}$ – проводимость поперечной ветви;
U_i, δ_i – модуль и фаза напряжения;
ω_i – множество номеров узлов, смежных *i*-у.
Мощность нагрузки задается постоянной величиной или статическими

характеристиками.

Продольные ветви являются элементами схем замещения ЛЭП или трансформатора.

ЛЭП - моделируется симметричной П-образной схемой замещения, а трансформаторы – несимметричной, рисунок 4.2.

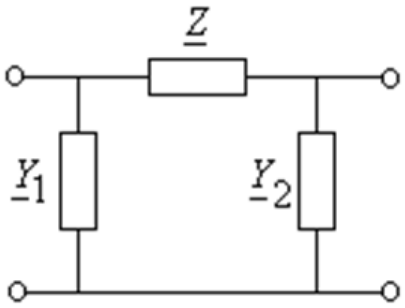


Рисунок 4.2 – Модель ветви схемы сети

Модель сети представляет собой узловые уравнения, представляющие собой комбинацию уравнений первого закона Кирхгофа и закона Ома.

В некоторых задачах кроме уравнений узловых напряжений используются уравнения для токов узлов или ветвей, модулей и фаз напряжений, а также потоков мощности по ветвям схемы сети.

Для формирования расчетной модели сети используется огромное количество информации об установленном на подстанции оборудовании, схем его соединения, поступающей информации от измерительных устройств, прогнозные данные, поступающие из внешних систем и так далее. Для организации сбора, хранения и обработки такого большого количества информации в SCADA-системах используются системы управления базами данных (СУБД).

В самом простейшем виде цифровую модели сети можно представить виде базы данных включающей в себя всего две взаимосвязанные таблицы: таблицу узлов и таблицу ветвей.

Таблица узлов будет включать в себя следующие параметры:
– номер узла;
– наименование узла;
– номинальное напряжение узла;
– активная мощность нагрузки;
– реактивная мощность нагрузки;

– расчетное напряжение.

Таблица ветвей будет включать в себя следующие параметры:
– номер узла начала;
– номер узла конца;
– состояние (вкл/выкл);
– активное сопротивление ветви;
– реактивное сопротивление ветви;
– активная проводимость ветви;
– реактивная проводимость ветви;
– коэффициент трансформации (для ЛЭП он равен нулю).

Примерно в таком виде база данных энергосистемы хранилась и обрабатывалась в 60-е – 80-е годы. Базы данных современных SCADA-систем, применяемых в электроэнергетике имеют гораздо более сложную структуру.

Задание

1. Сделайте краткий конспект, ответьте на контрольные вопросы.

2. Согласно вашему варианту, разработайте структуру базы данных для хранения цифровой модели сети. Продумайте тип данных для каждого свойства. Структуру представьте в табличном и графическом виде.

Таблица 4.1 – Варианты для задания №2

№ варианта	Таблица узлов	Таблица ветвей
1	Активная, реактивная мощность генерации	Марка провода
2	Тип узла (базовый, нагрузочный, генераторный)	Длина линии
3	Активная, реактивная мощность генерации	Мощность трансформатора
4	Тип узла (базовый, нагрузочный, генераторный)	Сечение провода
5	Активная, реактивная мощность генерации	Количество опор
6	Тип узла (базовый, нагрузочный, генераторный)	Потери мощности
7	Активная, реактивная мощность генерации	Падение напряжения
8	Тип узла (базовый, нагрузочный, генераторный)	Токовая загрузка
9	Активная, реактивная мощность генерации	Максимальный ток
10	Тип узла (базовый, нагрузочный, генераторный)	Зарядная мощность

3. Создайте запрос на языке SQL, соответствующий разработанной структуре БД.

Контрольные вопросы

1. Какими элементами можно описать простейшую цифровую модель сети?
2. Опишите модель узла схемы сети.
3. Опишите модель ветви схемы сети.
4. Каким образом организуется связь между таблицами узлов и ветвей?
5. Какие системы используются в современных SCADA для создания и хранения цифровой (информационной) модели сети?

ПРАКТИЧЕСКАЯ РАБОТА №5

Тема: *Создание простейшей базы данных упрощенной цифровой модели сети с помощью языке Python*

На сегодняшний день существует множество баз данных: реляционных и не реляционных, платных и бесплатных. Наиболее простейшей широко распространенной базой данных, применяемой, в том числе во многих мобильных телефонах, является база данных SQLite. SQLite – компактная встраиваемая СУБД. Слово «встраиваемый» (англ. embedded) означает, что SQLite не использует парадигмы клиент-сервер, то есть ядро SQLite не является отдельно работающим процессом, с которым взаимодействует программа, а представляет собой библиотеку, с которой программа компонуется, и ядро SQLite становится составной частью программы. Таким образом, в качестве протокола обмена используются вызовы функций библиотеки SQLite. Такой подход уменьшает накладные расходы, время отклика и упрощает программу. SQLite хранит всю базу данных (включая определения, таблицы, индексы и данные) в единственном стандартном файле на том компьютере, на котором исполняется программа. Простота реализации достигается за счёт того, что перед началом исполнения транзакции записи весь файл, хранящий базу данных, блокируется. Несколько процессов или потоков могут одновременно без каких-либо проблем читать данные из одной базы. Запись в базу можно осуществить только в том случае, если никаких других запросов в данный момент не обслуживается; в противном случае попытка записи оканчивается неудачей, и в программу возвращается код ошибки. Другим вариантом развития событий является автоматическое повторение попыток записи в течение заданного интервала времени.

Вот некоторые сценарии использования SQLite, упомянутые на официальном сайте этой системы:

– встраиваемые устройства и IoT;

– анализ данных;

– перенос данных из одной системы в другую;

– архивирование данных и (или) упаковка данных в контейнеры;

– хранение данных во внешней или временной БД;

– заменитель корпоративной БД, используемый в демонстрационных или испытательных целях;

– обучение, освоение начинающими практических приёмов работы с БД;

– прототипирование и исследование экспериментальных расширений языка SQL.

Важной особенностью СУБД SQLite является то, что она входит в стандартную библиотеку языка Python. То есть для работы с SQLite из Python-кода не нужно устанавливать некое клиент-серверное ПО, не нужно поддерживать работу какого-то сервиса, отвечающего за работу с СУБД, не нужно подключаться к дополнительным внешним серверам. Достаточно лишь импортировать модуль sqlite и приступить к его использованию в программе, получив в своё распоряжение систему управления реляционными базами данных. Для организации подключения к базе данных SQLite не нужно беспокоиться об установке драйверов, о подготовке строк подключения и о прочих подобных вещах.

Для подключения библиотеки SQLite необходимо открыть среду разработки IDLE, создать новый файл и в нем написать следующую строчку.

```
import sqlite3
```

Подключение к базе данных выполняется в одну строчку:

```
con.sqlite3.connect('my-test.db')
```

При первом запуска создаться новая пустая база данных и выполнится подключение к ней. При повторном запуске будет использоваться уже созданная раннее база данных.

Пустая база данных не представляет собой интереса. Поэтому необходимо создать в ней таблицу. Например, для создания таблицы узлов, включающей в себя поля номер узла и имя узла, используется следующая команда:

```
con.execute("""
CREATE TABLE NODES (
ID INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
NY INTEGER NOT NULL DEFAULT 0,
NAME TEXT NOT NULL DEFAULT '');""")
```

Здесь в тройные кавычки заключается запрос на языке SQL на создание новой таблицы. Используются тройные кавычки только по той причине, что мы пишем запрос на нескольких строчках. Мы могли бы записать этот же запрос в одну строку (в этом случае использовались бы одинарные кавычки), но такая длинная строка сложнее для восприятия, поэтому в данном случае мы использовали многострочное написание.

Помимо номера узла и его имени в таблице создается еще одно поле – ID. Это уникальный номер записи в таблице, который нам будет полезен при удалении и редактировании записи. Он имеет тип PRIMARY KEY, что означает, что значение этого поля будет уникальным для всей таблицы и AUTOINCREMENT, что означает, что оно будет автоматически генерироваться, если мы его явно не укажем. Остальные поля имеют атрибуты NOT NULL DEFAUL, которые задают значения полей по умолчанию.

При первом запуске указанный выше SQL запрос выполнится корректно. Однако при повторном запуске возникнет ошибка, поскольку таблица NODE уже будет существовать. Для предотвращения этой ошибки можно добавить фразу ‘IF NOT EXISTS’ после фразы ‘CREATE TABLE’.

Для добавления новой записи в таблицу необходимо выполнить следующий запрос:

```
con.execute("INSERT INTO NODES DEFAULT VALUES")
con.commit()
```

Первая строчка добавляет новую запись со значениями по умолчанию, которые были указаны при создании таблицы. Вторая строчка подтверждает произведенные изменения и сохраняет их на диск.

Для того, чтобы вывести на экран значения, хранящиеся в базе данных, можно использовать запрос типа «SELECT» и функцию print:

```
print(list(con.execute("SELECT * FROM NODES")))
```

Этой командой мы запрашиваем из базы данных все записи из таблицы «NODES», преобразуем полученный результат в список (команда list) и выводим его на экран. Результат будет следующий:

```
[(1, 0, ''), (2, 0, ''), (3, 0, ''), (4, 0, ''), (5, 0, '')]
```

Для того, чтобы получить более наглядный результат, можно применить следующий код:

```
print("ID\t|NY\t|NAME\t|")
for row in con.execute("SELECT * FROM NODES"):
    for col in row:
        print(col,end="\t|")
    print()
```

Здесь мы через цикл выводим каждую строку из результата запроса и каждый столбец из строки. Символ \t означает знак табуляции, то есть вставить 8 пробелов. Последний print нужен для перехода к новой строке. В результату получим следующее (рисунок 5.1):

```
ID      |NY      |NAME    |
1       |0       |        |
2       |0       |        |
3       |0       |        |
4       |0       |        |
5       |0       |        |
6       |0       |        |
7       |0       |        |
8       |0       |        |
9       |0       |        |
10      |0       |        |
11      |0       |        |
12      |0       |        |
```

Рисунок 5.1 – Вывод на экран таблицы из БД SQLite

Если нужно добавить запись с конкретными значениями, а не со значениями по умолчанию, необходимо это указать в запросе INSERT:

```
con.execute("INSERT INTO NODES (NY,NAME) VALUES('1','Узел№1')")
con.commit()
```

В том случае, если мы заранее не знаем, какие значения будут добавляться в таблицу (например, эти значения вводит пользователь с клавиатуры), конкретные значения можно заменить на символ «?», и передать их во втором параметре функции execute:

```
a=input("Введите номер узла: ")
b=input("Введите имя узла: ")
con.execute("INSERT INTO NODES (NY,NAME) VALUES(?,?)",(a,b))
con.commit()
```

SQLite автоматически подставит вместо вопросов переданные значения и сформирует нужный запрос.

Для удаления записи из таблицы используется запрос DELETE.

```
con.execute("DELETE FROM NODES")
con.commit()
```

Этот запрос удалит все записи из таблицы. Если нам нужно удалить конкретный узел, необходимо конкретизировать запрос с помощью ключевого слова WHERE. Например:

```
con.execute("DELETE FROM NODES WHERE ID=20")
con.commit()
```

В запросе на удаление можно точно так же использовать символ вопроса вместо конкретного числа и передать его значение вторым параметром:

```
con.execute("DELETE FROM NODES WHERE ID=?",(20,))
con.commit()
```

Обратите внимание, что, несмотря на то, что мы передаем всего один неизвестный параметр для формирования запроса, он все равно заключается в скобки и после него идет символ запятая. Это особенности языка Python.

Аналогичным образом можно выполнить запрос на изменения записи в таблице:

```
con.execute("UPDATE NODES SET NAME=?,NY=? WHERE ID=?",('Узел1',1,25))
con.commit()
```

Задание

1. Сделайте краткий конспект, ответьте на контрольные вопросы

2. Создайте на языке Python программу для работы с базой данных, разработанной в предыдущей работе. Программа должна позволять добавлять и удалять узлы и ветви цифровой модели сети.

3. Заполните базу данных данными, согласно вашему заданию из второй работы.

Контрольные вопросы

1. Какие базы данных вы знаете?
2. Чем SQLite отличается от других известных вам баз данных?
3. Как создать подключение к БД SQLite на языке Python?
4. Как добавить новую запись в БД SQLite на языке Python?
5. Как удалить запись в БД SQLite на языке Python?
6. Как изменить запись в БД SQLite на языке Python?

ПРАКТИЧЕСКАЯ РАБОТА №6 (4 ЧАСА)

Тема: *создание графического интерфейса для разработанной цифровой модели сети*

Язык Python позволяет создавать не только консольные (работающие в режиме командной строки) приложения, но и относительно просто разработать приложение с простейшим графическим пользовательским интерфейсом. Одной из самых простых библиотек, позволяющих быстро создать приложение с графическим интерфейсом является библиотека PySimpleGUI. Для ее использования сначала необходимо ее установить. Для этого в командной строке Windows необходимо набрать следующую команду:

```
pip install pysimplegui
```

Далее в коде нашей программы необходимо подключить эту библиотеку:

```
import PySimpleGUI as sg
```

Создание графического интерфейса происходит всего одной строчкой:

```
window = sg.Window('БД информационной модели сети', elements)
```

Первый аргумент функции – это заголовок окна, а второй – список элементов управления в окне.

Библиотека PySimpleGUI позволяет создавать большое количество элементов управления, поддерживаемых операционной системой. Вот некоторые из них:

Text – простое текстовое поле;

Input – однострочное поле ввода информации;

Combo – выпадающий список с заранее подготовленными значениями;

OptionMenu – выпадающий список как Combo, но позволяющий добавлять новые элементы;

Multiline – многострочное поле ввода;

Radio – переключатель между несколькими значениями;

Checkbox – флажок;

Spin – поле ввода со стрелочками увеличения и уменьшения значений;

Button – кнопка;

Image – изображение;

Canvas – панель рисования;

Tab – вкладка;

TabGroup – панель вкладок;

Table – таблица;

ProgressBar – индикатор прогресса;

и множество других.

Полный список элементов и их свойства приведен в официальной документации к библиотеке: https://www.pysimplegui.org/en/latest/ .

Каждый элемент обладает своим уникальным набором свойств. Но также есть и одной общее свойство – это его уникальный идентификатор, который нужен для взаимодействия с этим элементом. Это идентификатор называется key (ключ) и задается при создании объекта. Например, простейшая программа, которая выводит на экране слово «Hello, world» будет выглядеть следующим образом:

```
import PySimpleGUI as sg
Text1=sg.Text('Hello,world',key="text1")
elements=[[Text1]]
window = sg.Window('БД информационной модели сети', elements)
while True:
    event, values = window.read()
    if event == sg.WIN_CLOSED:
        window.close()
        break
```

В результате на экран будет выведено следующее окно (рисунок 6.1):

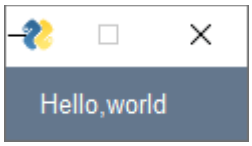


Рисунок 6.1 – Простейшее графическое приложение на языке Python

Здесь список элементов elements состоит только из одного элемента управления Text1, который имеет тип Text и ключ (уникальный идентификатор) «text1». Цикл while нужен для обработки событий окна. В данном случае мы используем обработку только одного события – закрытие окна (WIN_CLOSED).

Допустим, мы хотим обрабатывать еще одно событие: щелчок мышкой по текстовому полю. В этом случае, для текстового поля нужно добавить свойство

«enable_events=True» и добавить в обработчик событий соответствующее событие:

```
import PySimpleGUI as sg
Text1=sg.Text('Hello,world',key="text1",enable_events=True)
elements=[[Text1]]
window = sg.Window('БД информационной модели сети', elements)
while True:
    event, values = window.read()

    if event=='text1':
        window["text1"].update('clicked')

    if event == sg.WIN_CLOSED:
        window.close()
        break
```

Как видно из приведенного кода, для доступа к элементу управления используется конструкция вида window[key], где window – окно, которое мы создали, а key – уникальный идентификатор элемента, к которому хотим обратиться (в данном случае – text1). Функция update служит для обновления свойств выбранного элемента.

В данном примере мы рассмотрели окно, в котором всего один элемент управления. Обычно в приложении встречается сразу несколько элементов управления. В этом случае нужно понимать, по какому принципу библиотека PySimpleGUI располагает их в окне. В общем случае, переменная elements будет содержать список строк, внутри каждой из которых будет список столбцов. Каждый столбец при этом состоит из одного элемента (рисунок 6.2):

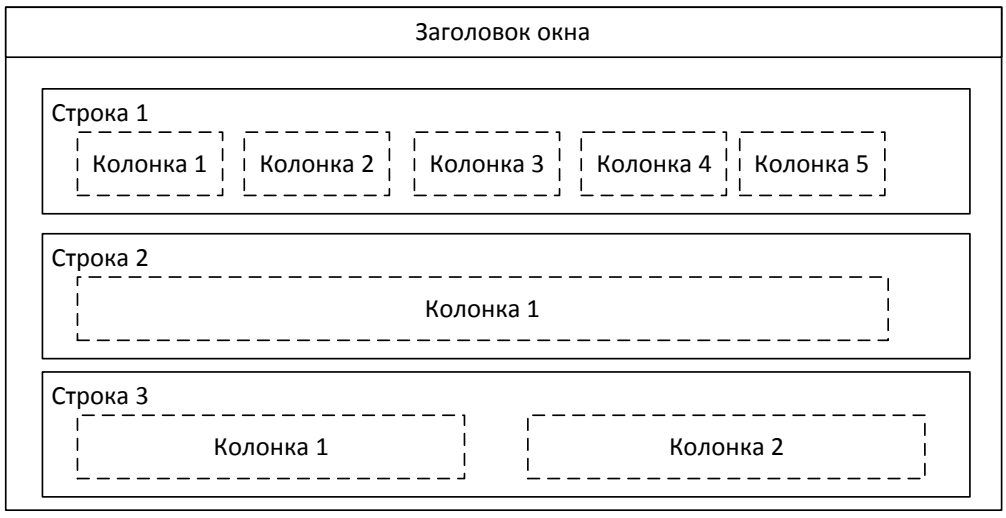


Рисунок 6.2 – Сетка размещения элементов управления в PySimpleGUI

Для размещения элементов управления в том порядке, в котором он приведен на рисунке, будет использоваться следующий код:

```
import PySimpleGUI as sg
Text1=sg.Text('T1',key="T1")
Text2=sg.Text('T2',key="T2")
Text3=sg.Text('T3',key="T3")
Text4=sg.Text('T4',key="T4")
Text5=sg.Text('T5',key="T5")
Text6=sg.Text('T6',key="T6")
Text7=sg.Text('T7',key="T7")
Text8=sg.Text('T8',key="T8")
Text8=sg.Text('T9',key="T9")
Row1=[Text1,Text2,Text3,Text4,Text5]
Row2=[Text6]
Row3=[Text7,Text8]
elements=[Row1,Row2,Row3]

```

В результате получится следующее окно (рисунок 6.3):

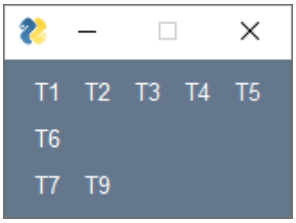


Рисунок 6.3 – Пример размещение элементов управления в сетке PySimpleGUI

Если же нужно создать более сложный столбец, который будет состоять из нескольких элементов, необходимо использовать служебный элемент управления «Column». Например (рисунок 6.4):

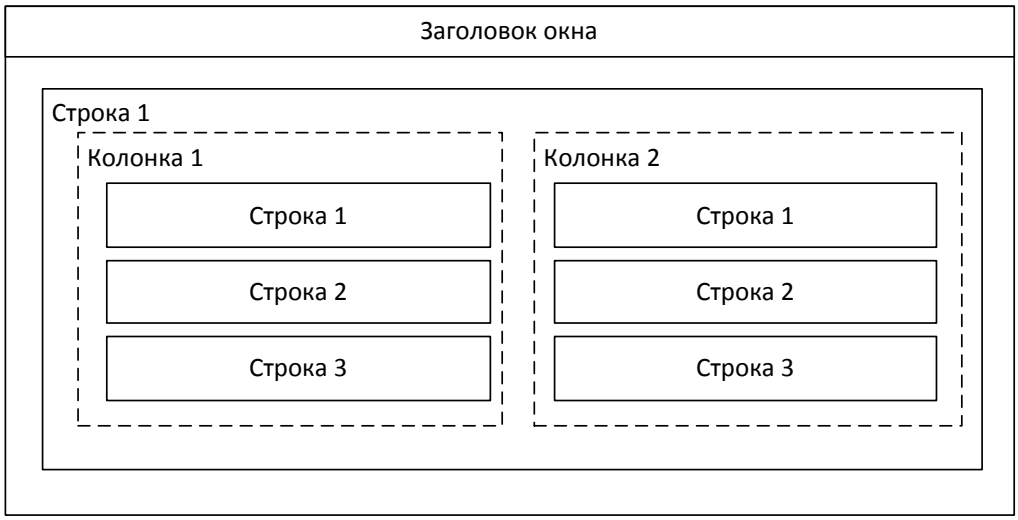


Рисунок 6.4 – Пример сложных колонок, состоящих их нескольких строк

```
import PySimpleGUI as sg
Text1=sg.Text('T1',key="T1")
Text2=sg.Text('T2',key="T2")
Text3=sg.Text('T3',key="T3")
Text4=sg.Text('T4',key="T4")
Text5=sg.Text('T5',key="T5")
Text6=sg.Text('T6',key="T6")
Text7=sg.Text('T7',key="T7")
Text8=sg.Text('T8',key="T8")
Text8=sg.Text('T9',key="T9")
Col1=[[Text1],[Text2],[Text3]]
Col2=[[Text4],[Text5],[Text6]]
elements=[[sg.Column(Col1),sg.Column(Col2)]]
```

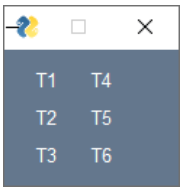


Рисунок 6.5 – Пример размещение элементов управления в сетке PySimpleGUI с использованием элемента «Column»

Как можно видеть из приведенных выше рисунков, цветовая схема оформления немного отличается от привычной. Библиотека PySimpleGUI позволяет выбрать одну из множества тем оформления. Полный список тем можно увидеть с помощью команды «sg.theme_previewer()». При этом откроется окно, изображенное на рисунке 6.6.

Рисунок 6.6 – Выбор темы оформления графического интерфейса

Как видно из приведенного рисунка, наиболее близкой к привычной для операционной системы Windows цветовой схеме является тема Default1. Для установки этой темы можно использовать команду

```
sg.theme('Default1')
```

Аналогичным образом, вместо Default1 можно выбрать любую другую цветовую тему.

Следующий пример демонстрирует основы работы с элементами «InputText» и «Button».

```
import PySimpleGUI as sg
sg.theme('Default1')
T1=sg.Text('Label',key="T1")
I1=sg.InputText('',key="IN1", enable_events = True)
B1=sg.Button('Button',key="BUTTON",enable_events = True)

elements=[[T1,I1,B1]]

window = sg.Window('БД информационной модели сети', elements)
while True:
    event, values = window.read()
    if event=='BUTTON':
        new_val=values['IN1']
        window["T1"].update(new_val)

    if event == sg.WIN_CLOSED:
        window.close()
        break
```

Приведенный код создаст следующее окно ():

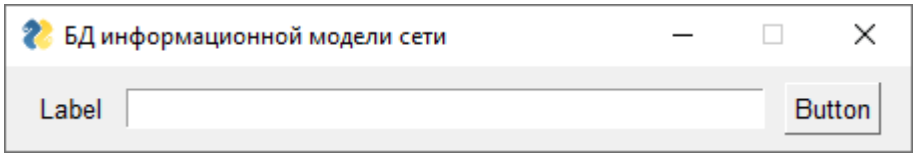


Рисунок 6.7 – Окно, демонстрирующее работу с элементами «Button» и «InputText»

Как видно из приведенных примеров, переменная event содержит идентификатор элемента, с которым в данный выполняет операции пользователь. В дополнении к этой переменной, переменная values содержит текущее значение элементов управления. В приведенном примере для

получения текущего значения текстового поля используется команда values[‘IN1’] (строка 12).

Рассмотрим более сложный пример, демонстрирующий работу с другими элементами управления и одновременно использующий уже рассмотренные ранее способы работать с базой данных:

```
import sqlite3 as sl
import PySimpleGUI as sg

con = sl.connect('my-test3.db')

con.execute("CREATE TABLE IF NOT EXISTS NODES"
            "(ID INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,"
            "NY INTEGER,"
            "NAME TEXT);")

COL1=[
    [sg.Text('Таблица узлов:')],
    [sg.Table(values        = list(con.execute("SELECT * FROM NODES")),
              headings      = ['ID','№','Название'],
              key           = 'NODES',
              enable_events = True,
              col_widths    = [5,10,30],
              auto_size_columns = False
              )]
    ]

COL2=[
    [sg.Text('ID:',size=10),                        sg.Text(key='N_ID')],
    [sg.Text('№:',size=10),                         sg.InputText(key='N_NY',enable_events=True,size=20)],
    [sg.Text('Название:',size=10),                  sg.InputText(key='N_NAME',enable_events=True,size=20)],
    [sg.Button('+',key='ADDN',enable_events=True),  sg.Button('-',key="DELN", enable_events=True)]
    ]

elements=[[sg.Column(COL1),sg.Column(COL2)]]
window = sg.Window('БД информационной модели сети', elements)

while True:
    event, values = window.read()
    ID=window['N_ID'].get()
    if event == sg.WIN_CLOSED:
        window.close()
        break

    if event=='NODES':
        if values['NODES']:
            data=window['NODES'].Values[values['NODES'][0]]
            window['N_ID'].Update(data[0])
            window['N_NY'].Update(data[1])
            window['N_NAME'].Update(data[2])


    if event=='ADDN':
        con.execute("INSERT INTO NODES DEFAULT VALUES")
        window['NODES'].update(list(con.execute("SELECT * FROM NODES")))


    if event=='DELN':
        con.execute("DELETE FROM NODES WHERE id=?",(ID,))
        window['NODES'].update(list(con.execute("SELECT * FROM NODES")))

    if "N_" in event:
        fld=event.replace("N_","")
        con.execute("UPDATE NODES SET "+fld+"=? WHERE ID=?",(values[event],ID))
        window['NODES'].update(list(con.execute("SELECT * FROM NODES")))

con.commit()
con.close()
```

Данный пример создает окно, содержащее в себе следующие элементы управления: таблицу узлов, поля для редактирования номера и имени узла и кнопки добавления и удаления записей:

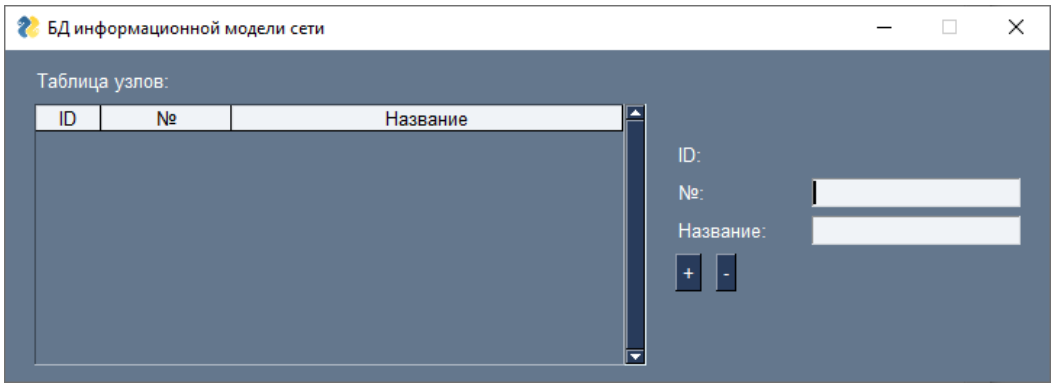


Рисунок 6.8 – Создание графического интерфейса для редактирования таблицы узлов

Для создания таблицы используется элемент управления «Table», который имеет следующие параметры:

– values – значения, которые необходимо отобразить в таблице;

– headings – заголовок таблицы;

– col_width – ширина каждого столбца;

– auto_size_columns – настройка, сообщающая о том, что не нужно автоматически изменять размер столбцов таблицы.

Для элементов типа «InputText» и «Text» также используется не рассмотренное раннее свойство size. Оно позволяет задать размер элемента, а не оставлять его по умолчанию.

Алгоритм работы программы следующий:

1. При загрузке программа подключается к базе данных, запрашивает имеющиеся записи в таблице узлов и заносит их в таблицу для отображения на экране.

2. При щелчке по строкам таблицы записи копируются в поля типа «InputText», а текущее значений ID выбранной записи копируется в поле типа «Text» (строки 39-44). Текущая выбранная строка находится в переменной values[‘NODES’][0]. Ноль в данном случае говорит о том, что мы будет отображать самую первую выбранную запись, поскольку элемент «Table»

позволяет выделять мышкой более одной записи. Предварительно выполняется проверка, действительно ли сейчас выбрана запись в таблице (строка 40).

3. При изменении поля типа «InputText» с клавиатуры выполняется запрос на изменение этого поля в базе данных (строка 58).

4. При нажатии на кнопку «-» выполняется удаление выбранной записи из базы данных.

5. При нажатии на кнопку «+» осуществляется добавление новой записи в базу.

Для обработки события «изменение значения поля» в данном случае используется небольшая хитрость. Вместо того, чтобы отдельно обрабатывать событие изменения поля N_NY и N_NAME используется обработка события поля, в котором присутствует префикс «N_» (строка 56). Далее этот префикс удаляется и обновляется поле в базе данных с именем без префикса.

Для сохранения изменений в базе данных, после закрытия окна вызываются методы commit() и close().

Задание

1. Создайте программу на языке Python с графическим интерфейсом согласно вашему варианту:

Таблица 6.1 – Варианты для задания 1

№ варианта	Задание
1	Программа решения квадратного уравнения. Коэффициенты квадратного уравнения пользователь должен вводить с клавиатуры, в поля типа «InputText». Корни уравнения необходимо вывести на экран в поля типа «Text» при нажатии на кнопку Button.
2	Программа определение гипотенузы прямоугольного треугольника. Катеты треугольника вводятся пользователем в поля типа «InputText». Гипотенузы выводится на экран в поле типа «Text» при нажатии на кнопку Button.
3	Сумма чисел. Пользователь вводит с клавиатуры 5 чисел в поля «InputText». Необходимо найти их произведений и вывести на экран в поле «Text».
4	Сумма квадратов. Пользователь вводит с клавиатуры 5 чисел в поля InputText. Рядом с каждым полем ввода необходимо расположить текстовое поле и вывести в него квадрат введенного значения. Ниже всех полей необходимо вывести сумму чисел и сумму квадратов чисел в поля типа «Text».
5	Четные и нечетные числа. Напишите следующую программу: пользователь вводит с клавиатуры 5 чисел в поля типа «InputText». Необходимо посчитать количество четных и нечетных чисел и вывести на экран это количество в поля типа «Text».

№ варианта	Задание
6	Четные и нечетные числа. Напишите следующую программу: пользователь вводит с клавиатуры 5 чисел в поля типа «InputText». Необходимо вывести в поле типа «Text» сообщение, показывающее, каких чисел оказалось больше: четных или нечетных.
7	Напишите следующую программу: пользователь вводит с клавиатуры 5 чисел в поля типа «InputText». Необходимо посчитать произведение четных и сумму нечетных чисел и вывести в поля типа «Text» полученные значения.
8	Угол между векторами. В поля типа «InputText» вводятся координаты x,y двух векторов. Необходимо вычислить угол между ними и вывести его на экран в поле типа «Text».
9	Прямоугольник. Напишите следующую программу: пользователь вводит с клавиатуры в поля типа «InputText» длину и ширину прямоугольника. Необходимо посчитать и вывести на экран в поля типа «Text»: периметр прямоугольника, его площадь и длину диагонали.
10	Окружность. Напишите следующую программу: пользователь вводит с клавиатуры в поле типа «InputText» радиус окружности. Необходимо посчитать и вывести на экран в поля типа «Text» длину окружности и площадь круга.

2. Создайте программу на языке Python, позволяющую работать с базой данных, хранящей цифровую модель сети, разработанную в предыдущей работе

Контрольные вопросы

1. Что такое PySimpleGUI?
2. Какие элементы управления в PySimpleGUI вы знаете?

ПРАКТИЧЕСКАЯ РАБОТА №7

Тема: *Экспорт цифровой модели сети в формате CIM-XML*

Создание классов в языке Python

Язык Python является современным объектно-ориентированным языком программирования. Это означает, что он имеет все возможности, позволяющие описывать классы и создавать на их основе объекты. Для создания класса в языке Python существует ключевое слово class. Например, для создания класса «Resource» необходимо записать:

```
class Resource:
```

Далее, после символа «:» необходимо записать свойства (параметры) этого класса и методы, которые он будет выполнять. Методы класса – это то же самое, что и функции, но которые могут быть выполнены только с объектами этого класса. Для этого в качестве первого параметра класса передается ключевое слово self. Параметры класса создаются в момент создания объекта класса с помощью специального метода – конструктора класса, который вызывается автоматически при создании класса. Конструктор любого класса всегда имеет название __init__. Например, описание класса «Resource» будет выглядеть следующим образом:

```
class Resource:
    def __init__(self):
        self.id=0
```

Здесь мы в конструкторе класса __init__ описали один параметр класса «Resource» - id, уникальный идентификатор объекта и присвоили ему значение по умолчанию - 0.

Для создания наследуемого класса необходимо в скобках указать название родительского класса, а при создании параметров этого класса необходимо записать строку super().__init__(), которая будет означать

«вызвать конструктор родительского класса». Например, описание класса «IdentifiedObject» будет выглядеть следующим образом.

```
class IdentifiedObject(Resource):
    def __init__(self):
        super().__init__()
        self.name=""
```

Аналогичным образом можно описать все остальные классы, приведенные на рисунке 3.1:

```
class ConnectivityNode(IdentifiedObject):
    def __init__(self):
        super().__init__()
        self.Terminals  = []

class ACDCTerminal(IdentifiedObject):
    def __init__(self):
        super().__init__()
        self.sequenceNumber = 0

class Terminal(ACDCTerminal):
    def __init__(self):
        super().__init__()
        self.ConductingEquipment = None
        self.ConnectivityNode    = None


class Equipment(IdentifiedObject):
    def __init__(self):
        super().__init__()
        self.EquipmentContainer=None

class ConductingEquipment(Equipment):
    def __init__(self):
        super().__init__()
        self.Terminals=[]


class ACLineSegment(ConductingEquipment):
    def __init__(self):
        super().__init__()
        self.r      = 0
        self.x      = 0
        self.gch    = 0
        self.bch    = 0
        self.Terminals=[]
        self.EquipmentContainer=None

class EquipmentContainer(IdentifiedObject):
    def __init__(self):
        super().__init__()
        self.Equipments=[]

class Line(EquipmentContainer):
    def __init__(self):
        super().__init__()
        self.Equipments=[]
```

```
class TransformerEnd(IdentifiedObject):
    def __init__(self):
        super().__init__()
        self.endNumber=0
        self.Terminal=None

class PowerTransformerEnd(TransformerEnd):
    def __init__(self):
        super().__init__()
        self.r=0
        self.x=0
        self.g=0
        self.b=0
        self.ratedU=0
        self.endNumber=0

class PowerTransformer(ConductingEquipment):
    def __init__(self):
        super().__init__()
        self.PowerTransformerEnd = None
```

Создание объекта любого класса происходит следующим образом:

```
N1= ConnectivityNode()
N2= ConnectivityNode()
N3= ConnectivityNode()
```

Как видно из приведенного примера, для создания объекта необходимо записать название класса и поставить две скобки. В данном случае создается 3 узла N1,N2,N3 класса «ConnectivityNode». Для обращения к свойствам каждого узла используется знак точка:

```
N1.name="Узел1"
N2.name="Узел2"
N3.name="Узел3"
```

Обратите внимание, что свойство name явно не определено в классе «ConnectivityNode». Но, поскольку класс «ConnectivityNode» наследуется от класса «IdentifiedObject», все свойства этого класса также доступны из объектов класса «ConnectivityNode».

Аналогичным образом объекты можно помещать не в отдельные переменные, а, например, в списки:

```
nodes=[ ConnectivityNode(),ConnectivityNode(),ConnectivityNode()]
```

Здесь мы создали список nodes, состоящий из трех объектов класса «ConnectivityNode». Объекты можно добавлять в список не только при создании списков, но и после, с помощью метода списка append.

В приведенном примере, после названия класса в скобках отсутствуют параметры, поэтому свойства объекта будут назначены по умолчанию. Это

не всегда удобно. Иногда, для сокращения записи, проще указать сразу конкретные параметры, вместо того, чтобы задавать их по умолчанию и потом менять. Например, если мы хотим, чтобы имена узлов задавались при их создании, можно записать следующий код:

```
nodes=[ConnectivityNode(”Узел1”),ConnectivityNode(”Узел2”),ConnectivityNode(”Узел3”)]
```

Чтобы такая запись работала, необходимо в конструктор узла после переменной self добавить еще один параметр, в котором будем передавать имя узла:

```
class ConnectivityNode(IdentifiedObject):
    def __init__(self,n):
        super().__init__()
        self.Terminals  = []
        self.name=n
```

С учетом этого, добавим для некоторых классов дополнительные параметры:

```
class Resource:
    def __init__(self):
        self.id=0

class IdentifiedObject(Resource):
    def __init__(self):
        super().__init__()
        self.name=""

class ConnectivityNode(IdentifiedObject):
    def __init__(self,name):
        super().__init__()
        self.name       = name
        self.Terminals  = []

class ACDCTerminal(IdentifiedObject):
    def __init__(self,num):
        super().__init__()
        self.sequenceNumber = num

class Terminal(ACDCTerminal):
    def __init__(self,eq,num):
        super().__init__(num)
        self.ConductingEquipment = eq
        self.ConnectivityNode    = None

class Equipment(IdentifiedObject):
    def __init__(self):
        super().__init__()
        self.EquipmentContainer=None

class ConductingEquipment(Equipment):
    def __init__(self):
        super().__init__()
        self.Terminals=[]
```

```
class ACLineSegment(ConductingEquipment):
    def __init__(self,L,R,X,G,B):
        super().__init__()
        self.r      = R
        self.x      = X
        self.gch    = G
        self.bch    = B
        self.Terminals=[Terminal(self,1),Terminal(self,2)]
        self.EquipmentContainer=L

class EquipmentContainer(IdentifiedObject):
    def __init__(self):
        super().__init__()
        self.Equipments=[]

class Line(EquipmentContainer):
    def __init__(self,R,X,G,B):
        super().__init__()
        self.Equipments=[ACLineSegment(self,R,X,G,B)]

class TransformerEnd(IdentifiedObject):
    def __init__(self):
        super().__init__()
        self.endNumber=0
        self.Terminal=None

class PowerTransformerEnd(TransformerEnd):
    def __init__(self,N,NAME,U,R,X,G,B):
        super().__init__()
        self.r=R
        self.x=X
        self.g=G
        self.b=B
        self.ratedU=U
        self.name=NAME
        self.endNumber=N

class PowerTransformer(ConductingEquipment):
    def __init__(self,windings):
        super().__init__()
        self.Terminals           = []
        self.PowerTransformerEnd = windings
        for t in windings:
            t.PowerTransformer  = self
            t.Terminal          = Terminal(self,t.endNumber)
            self.Terminals.append(t.Terminal)
```

С учетом произведенных изменений, создание объектов соединительного узла, линии и трансформатора будет выглядеть следующим образом:

```
N1=ConnectiveNode("Узел1")
N2=ConnectiveNode("Узел2")
N3=ConnectiveNode("Узел3")
L=Line(1.5,0.7,0,0)
T=PowerTransformer([PowerTransformerEnd(1,"Обмотка ВН",10,30,15,10,40),
PowerTransformerEnd(1,"Обмотка НН",0.4,30,15,10,40)])
```

Для соединения узлов и терминалов давайте добавим в класс узла функцию connect:

```
class ConnectivityNode(IdentifiedObject):
```

```
        def __init__(self,name):
            super().__init__()
            self.name       = name
            self.Terminals  = []
            self.busbars    = []

        def connect(self,T):
            self.Terminals.append(T)
            T.ConnectivityNode = self
```

Теперь соединить узлы N1 и N2 с линией L можно следующим образом:

```
N1.connect(L.Terminals[0])
N2.connect(L.Terminals[1])
```

Аналогичным образом можно соединить узлы N2 и N3 с трансформатором T:

```
N2.connect(T.Terminals[0])
N3.connect(T.Terminals[1])
```

Для перевода значений параметров классов в XML формат добавим в каждый класс метод XML и PRINT_XML. Эти методы будут просто выводить на экран с помощью метода print значения соответствующих свойств внутри соответствующих тегов XML:

```
class Resource:
    count = 1
    def __init__(self):
        self.id=Resource.count
        Resource.count +=1

    def PRINT_XML(self):
        print(f"<cim:{type(self).__name__} rdf:ID=\"#{self.id}\">")
        self.XML()
        print(f"</cim:{type(self).__name__}>")
        print()

    def link(self,CIM):
        return f"\t<cim:{CIM} rdf:resource=\"#{self.id}\"/>"
##########################################
class IdentifiedObject(Resource):
    def __init__(self):
        super().__init__()
        self.name="Без имени"

    def XML(self):

print(f"\t<cim:{__class__.__name__}.name>{self.name}</cim:{__class__.__name__
}.name>")
##########################################
class ConnectivityNode(IdentifiedObject):
    def __init__(self,name):
        super().__init__()
        self.name       = name
        self.Terminals  = []
        self.busbars    = []

    def connect(self,T):
        self.Terminals.append(T)
        T.ConnectivityNode = self
```

```
    def XML(self):
        super().XML()
        for t in self.Terminals:
            print(t.link(f"{__class__.__name__}.Terminals"))

    def PRINT_XML(self):
        super().PRINT_XML()
        for b in self.busbars:
            b.PRINT_XML()
##########################################
class ACDCTerminal(IdentifiedObject):
    def __init__(self,num):
        super().__init__()
        self.sequenceNumber = num

    def XML(self):

print(f"\t<cim:{__class__.__name__}.sequenceNumber>{self.sequenceNumber}</cim
:{__class__.__name__}.sequenceNumber>")
##########################################
class Terminal(ACDCTerminal):
    def __init__(self,eq,num):
        super().__init__(num)
        self.ConductingEquipment = eq
        self.ConnectivityNode    = None

    def XML(self):
        super().XML()

print(self.ConductingEquipment.link(f"{__class__.__name__}.ConductingEquipmen
t"))

print(self.ConnectivityNode.link(f"{__class__.__name__}.ConnectivityNode"))
##########################################
class Equipment(IdentifiedObject):
    def __init__(self):
        super().__init__()
        self.EquipmentContainer=None

    def XML(self):
        super().XML()
        if self.EquipmentContainer:

print(self.EquipmentContainer.link(f"{__class__.__name__}.EquipmentContainer"
))
##########################################
class ConductingEquipment(Equipment):
    def __init__(self):
        super().__init__()
        self.Terminals=[]

    def XML(self):
        super().XML()
        for t in self.Terminals:
            print(t.link(f"{__class__.__name__}.Terminals"))

    def PRINT_XML(self):
        super().PRINT_XML()
        for t in self.Terminals:
                t.PRINT_XML()
##########################################
class ACLineSegment(ConductingEquipment):
    def __init__(self,L,R,X,G,B):
```

```
        super().__init__()
        self.r      = R
        self.x      = X
        self.gch    = G
        self.bch    = B
        self.Terminals=[Terminal(self,1),Terminal(self,2)]
        self.EquipmentContainer=L

    def XML(self):
        super().XML()
        for i in ['r','x','gch','bch']:

print(f"\t<cim:{__class__.__name__}.{i}>{getattr(self,i)}</cim:{__class__.__n
ame__}.{i}>")
##########################################
class EquipmentContainer(IdentifiedObject):
    def __init__(self):
        super().__init__()
        self.Equipments=[]

    def XML(self):
        super().XML()
        for l in self.Equipments:
            print(l.link(f"{__class__.__name__}.Equipments"))
    def PRINT_XML(self):
        super().PRINT_XML()
        for l in self.Equipments:
            l.PRINT_XML()
##########################################
class Line(EquipmentContainer):
    def __init__(self,R,X,G,B):
        super().__init__()
        self.Equipments=[ACLineSegment(self,R,X,G,B)]
##########################################
class TransformerEnd(IdentifiedObject):
    def __init__(self):
        super().__init__()
        self.endNumber=0
        self.Terminal=None

    def XML(self):
        super().XML()

print(f"\t<cim:{__class__.__name__}.endNumber>{self.endNumber}</cim:{__class_
_.__name__}.endNumber>")
        print(self.Terminal.link(f"{__class__.__name__}.Terminal"))
##########################################
class PowerTransformerEnd(TransformerEnd):
    def __init__(self,N,NAME,U,R,X,G,B):
        super().__init__()
        self.r=R
        self.x=X
        self.g=G
        self.b=B
        self.ratedU=U
        self.name=NAME
        self.endNumber=N

    def XML(self):
        super().XML()
        for i in ['r','x','g','b','ratedU']:
```

```
print(f"\t<cim:{__class__.__name__}.{i}>{getattr(self,i)}</cim:{__class__.__n
ame__}.{i}>")

print(self.PowerTransformer.link(f"{__class__.__name__}.PowerTransformer"))
########################################
class PowerTransformer(ConductingEquipment):
    def __init__(self,windings):
        super().__init__()
        self.Terminals           = []
        self.PowerTransformerEnd = windings
        for t in windings:
            t.PowerTransformer  = self
            t.Terminal          = Terminal(self,t.endNumber)
            self.Terminals.append(t.Terminal)

    def XML(self):
        super().XML()
        for w in self.PowerTransformerEnd:
            print(w.link(f"{__class__.__name__}.PowerTransformerEnd"))

    def PRINT_XML(self):
        super().PRINT_XML()
        for w in self.PowerTransformerEnd:
            w.PRINT_XML()
########################################


nodes={}
nodes[1]=ConnectivityNode("Узел1")
nodes[2]=ConnectivityNode("Узел2")
nodes[3]=ConnectivityNode("Узел3")
nodes[4]=ConnectivityNode("Узел4")
lines=[Line(0,0,0,0),Line(0,0,0,0)]
trans=[PowerTransformer([PowerTransformerEnd(1,"ВН",10,1,2,3,4),
                         PowerTransformerEnd(2,"НН",0.4,1,2,3,4)])]

nodes[1].connect(lines[0].Equipments[0].Terminals[0])
nodes[2].connect(lines[0].Equipments[0].Terminals[1])
nodes[2].connect(lines[1].Equipments[0].Terminals[0])
nodes[3].connect(lines[1].Equipments[0].Terminals[1])
nodes[3].connect(trans[0].Terminals[0])
nodes[4].connect(trans[0].Terminals[1])

for n in nodes.values():
    n.PRINT_XML()

for l in lines:
    l.PRINT_XML()

for t in trans:
    t.PRINT_XML()
```

В приведенном примере конструкция вида {__class__.__name__} подставляет название текущего класса вместо того, чтобы его явно прописывать в функции print. В конструкторе класса «Resource» добавлено автоматическая генерация уникального номер объекта на основе счетчика

созданных объектов (при каждом создании нового объекта счетчик `count` увеличивается на единицу).

В результате выполнения приведенной выше программы будет сгенерирован следующий XML код:

```
<cim:ConnectivityNode rdf:ID="#1">
      <cim:IdentifiedObject.name>Узел1</cim:IdentifiedObject.name>
      <cim:ConnectivityNode.Terminals rdf:resource="#7"/>
</cim:ConnectivityNode>

<cim:ConnectivityNode rdf:ID="#2">
      <cim:IdentifiedObject.name>Узел2</cim:IdentifiedObject.name>
      <cim:ConnectivityNode.Terminals rdf:resource="#8"/>
      <cim:ConnectivityNode.Terminals rdf:resource="#11"/>
</cim:ConnectivityNode>

<cim:ConnectivityNode rdf:ID="#3">
      <cim:IdentifiedObject.name>Узел3</cim:IdentifiedObject.name>
      <cim:ConnectivityNode.Terminals rdf:resource="#12"/>
      <cim:ConnectivityNode.Terminals rdf:resource="#16"/>
</cim:ConnectivityNode>

<cim:ConnectivityNode rdf:ID="#4">
      <cim:IdentifiedObject.name>Узел4</cim:IdentifiedObject.name>
      <cim:ConnectivityNode.Terminals rdf:resource="#17"/>
</cim:ConnectivityNode>

<cim:Line rdf:ID="#5">
      <cim:IdentifiedObject.name>Без имени</cim:IdentifiedObject.name>
      <cim:EquipmentContainer.Equipments rdf:resource="#6"/>
</cim:Line>

<cim:ACLineSegment rdf:ID="#6">
      <cim:IdentifiedObject.name>Без имени</cim:IdentifiedObject.name>
      <cim:Equipment.EquipmentContainer rdf:resource="#5"/>
      <cim:ConductingEquipment.Terminals rdf:resource="#7"/>
      <cim:ConductingEquipment.Terminals rdf:resource="#8"/>
      <cim:ACLineSegment.r>0</cim:ACLineSegment.r>
      <cim:ACLineSegment.x>0</cim:ACLineSegment.x>
      <cim:ACLineSegment.gch>0</cim:ACLineSegment.gch>
      <cim:ACLineSegment.bch>0</cim:ACLineSegment.bch>
</cim:ACLineSegment>

<cim:Terminal rdf:ID="#7">
      <cim:ACDCTerminal.sequenceNumber>1</cim:ACDCTerminal.sequenceNumber>
      <cim:Terminal.ConductingEquipment rdf:resource="#6"/>
      <cim:Terminal.ConnectivityNode rdf:resource="#1"/>
</cim:Terminal>

<cim:Terminal rdf:ID="#8">
      <cim:ACDCTerminal.sequenceNumber>2</cim:ACDCTerminal.sequenceNumber>
      <cim:Terminal.ConductingEquipment rdf:resource="#6"/>
      <cim:Terminal.ConnectivityNode rdf:resource="#2"/>
</cim:Terminal>

<cim:Line rdf:ID="#9">
      <cim:IdentifiedObject.name>Без имени</cim:IdentifiedObject.name>
```

```
      <cim:EquipmentContainer.Equipments rdf:resource="#10"/>
</cim:Line>

<cim:ACLineSegment rdf:ID="#10">
      <cim:IdentifiedObject.name>Без имени</cim:IdentifiedObject.name>
      <cim:Equipment.EquipmentContainer rdf:resource="#9"/>
      <cim:ConductingEquipment.Terminals rdf:resource="#11"/>
      <cim:ConductingEquipment.Terminals rdf:resource="#12"/>
      <cim:ACLineSegment.r>0</cim:ACLineSegment.r>
      <cim:ACLineSegment.x>0</cim:ACLineSegment.x>
      <cim:ACLineSegment.gch>0</cim:ACLineSegment.gch>
      <cim:ACLineSegment.bch>0</cim:ACLineSegment.bch>
</cim:ACLineSegment>

<cim:Terminal rdf:ID="#11">
      <cim:ACDCTerminal.sequenceNumber>1</cim:ACDCTerminal.sequenceNumber>
      <cim:Terminal.ConductingEquipment rdf:resource="#10"/>
      <cim:Terminal.ConnectivityNode rdf:resource="#2"/>
</cim:Terminal>

<cim:Terminal rdf:ID="#12">
      <cim:ACDCTerminal.sequenceNumber>2</cim:ACDCTerminal.sequenceNumber>
      <cim:Terminal.ConductingEquipment rdf:resource="#10"/>
      <cim:Terminal.ConnectivityNode rdf:resource="#3"/>
</cim:Terminal>

<cim:PowerTransformer rdf:ID="#15">
      <cim:IdentifiedObject.name>Без имени</cim:IdentifiedObject.name>
      <cim:ConductingEquipment.Terminals rdf:resource="#16"/>
      <cim:ConductingEquipment.Terminals rdf:resource="#17"/>
      <cim:PowerTransformer.PowerTransformerEnd rdf:resource="#13"/>
      <cim:PowerTransformer.PowerTransformerEnd rdf:resource="#14"/>
</cim:PowerTransformer>

<cim:Terminal rdf:ID="#16">
      <cim:ACDCTerminal.sequenceNumber>1</cim:ACDCTerminal.sequenceNumber>
      <cim:Terminal.ConductingEquipment rdf:resource="#15"/>
      <cim:Terminal.ConnectivityNode rdf:resource="#3"/>
</cim:Terminal>

<cim:Terminal rdf:ID="#17">
      <cim:ACDCTerminal.sequenceNumber>2</cim:ACDCTerminal.sequenceNumber>
      <cim:Terminal.ConductingEquipment rdf:resource="#15"/>
      <cim:Terminal.ConnectivityNode rdf:resource="#4"/>
</cim:Terminal>

<cim:PowerTransformerEnd rdf:ID="#13">
      <cim:IdentifiedObject.name>BH</cim:IdentifiedObject.name>
      <cim:TransformerEnd.endNumber>1</cim:TransformerEnd.endNumber>
      <cim:TransformerEnd.Terminal rdf:resource="#16"/>
      <cim:PowerTransformerEnd.r>1</cim:PowerTransformerEnd.r>
      <cim:PowerTransformerEnd.x>2</cim:PowerTransformerEnd.x>
      <cim:PowerTransformerEnd.g>3</cim:PowerTransformerEnd.g>
      <cim:PowerTransformerEnd.b>4</cim:PowerTransformerEnd.b>
      <cim:PowerTransformerEnd.ratedU>10</cim:PowerTransformerEnd.ratedU>
      <cim:PowerTransformerEnd.PowerTransformer rdf:resource="#15"/>
</cim:PowerTransformerEnd>

<cim:PowerTransformerEnd rdf:ID="#14">
      <cim:IdentifiedObject.name>HH</cim:IdentifiedObject.name>
      <cim:TransformerEnd.endNumber>2</cim:TransformerEnd.endNumber>
      <cim:TransformerEnd.Terminal rdf:resource="#17"/>
      <cim:PowerTransformerEnd.r>1</cim:PowerTransformerEnd.r>
```

```
      <cim:PowerTransformerEnd.x>2</cim:PowerTransformerEnd.x>
      <cim:PowerTransformerEnd.g>3</cim:PowerTransformerEnd.g>
      <cim:PowerTransformerEnd.b>4</cim:PowerTransformerEnd.b>
      <cim:PowerTransformerEnd.ratedU>0.4</cim:PowerTransformerEnd.ratedU>
      <cim:PowerTransformerEnd.PowerTransformer rdf:resource="#15"/>
</cim:PowerTransformerEnd>
```

Задание

1. Изучите примеры кода программы на языке Python, предназначенной для сохранения цифровой модели схемы сети в формате CIM-xml.

2. Преобразуйте объекты, хранящиеся в вашей базе данных в эквивалентный CIM-XML формат.

3. Доработайте программу согласно вашему варианту:

Вариант 1: Добавьте класс и объекты типа EnergyConsumer.

Вариант 2: Добавьте класс и объекты типа Breaker.

Вариант 3: Добавьте класс и объекты типа BusbarSection.

4. Ответьте на контрольные вопросы.

Контрольные вопросы

1. Как на языке Python создать класс?

2. Как на языке Python добавить свойства (параметры) к классу?

3. Как на языке Python создать наследуемый класс?

4. Что такое конструктор класса?

5. Как на языке Python создать объект класса?

ПРАКТИЧЕСКАЯ РАБОТА №8

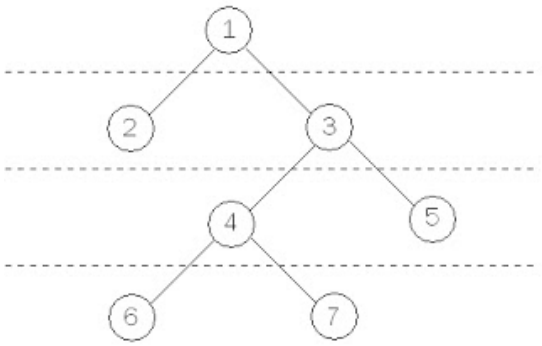
Тема: *Создание графического отображения схемы сети*

Общие сведения о создании графического отображения схемы сети

На сегодняшний день стандарт CIM не регламентирует правила построения графического изображения схемы сети. В большинстве случае схема сети создается пользователями вручную с помощью встроенных в SCADA специализированных векторных графических редакторов, похожих на MS Visio. Однако в некоторых случаях процесс создания графической схемы сети можно автоматизировать. Это возможно, например, в тех случаях, когда сеть имеет простую, разомкнутую топологию. В таких ситуациях можно использовать простой алгоритм построения древовидной схемы сети, двигаясь от центра питания к потребителям.

Основная сложность, которую приходится решать, при реализации данного алгоритма на ЭВМ, является определение нужной последовательности обхода узлов и ветвей сети. Если нарисовать схему сети, любой человек, интуитивно и безошибочно легко определит порядок обхода узлов и ветвей сети, необходимый для этого алгоритма. Однако у компьютера нет «глаз» и уж тем более интуиции. Все что у него есть – это таблица улов и таблица ветвей. И программисту необходимо четко задать последовательность действий, как, имея эти таблицы, определить порядок обхода узлов и ветвей. Данный алгоритм относится к так называемым «алгоритма на графах» и называется «алгоритм поиска в ширину». Для того чтобы объяснить в чем смысл этого алгоритма, давайте представим схему сети в виде ветвящегося дерева, изображенного на рисунке ниже.

Любую разомкнутую схему сети можно представить в виде такого дерева. Вершиной этого дерева является центр питания. Чтобы понять суть алгоритма поиска в ширину, давайте проведем имитацию «горения» графа. Представьте себе, что мы поджигаем самую первую вершину, дальше огонь перекидывается на смежные с ней вершины, с них на смежные с ними и т.д. Причем огонь идет только в одном направлении, от вершины – к концам.



Он не может вернуться обратно, так как предыдущие ветви и узлы уже сгорели. Имитируя такой процесс горения, можно определить в каком порядке будут «гореть» узлы и ветви. Запомнив этот порядок, мы получим последовательность прямого обхода графа.

Давайте подумаем, что нам необходимо, чтобы эмитировать такого «горение» графа. Прежде всего, нам необходима переменная, в которую мы будем сохранять порядок «сгорания» ветвей графа. В качестве такой переменной лучше всего использовать список. Назовем ее order (порядок).

Далее, для того, чтобы отмечать на схеме, какие узлы и ветви уже «сгорели», давайте введем для узлов и ветвей дополнительную переменную, которую назовем mark.

Тип этой переменной будет bool, так как она может принимать только два значения: false – еще не «сгорела» и «true» – уже «сгорела».

С учетом всего вышесказанного, программа построения схемы сети может выглядеть следующим образом:

```
from tkinter import *
# Класс узла
class Node:
    def __init__(self,t,n):
        self.ny     = n          # Номер узла
        self.x,self.y = 30,30    # Начальные координаты
        self.mark   = False      # Маркер для обозначения обработанных узлов
        self.idx    = 0           # Тип узла при ветвелении схемы: 2-первый, 1-
промежуточный, 0-последний
# Класс ветви
class Line:
    def __init__(self,ip,iq):
        self.ip,self.iq     = ip,iq
        self.mark   = False
# Найти все связанные с узлом ветви
def FindLines(ip=None,iq=None):
```

```
    ret=[]
    for l in lines:
        if ip and l.ip==ip: ret.append(l)
        if iq and l.iq==iq: ret.append(l)
    return ret
# Создаем узлы и ветви
nodes={}
nodes[1]=Node(0,1)
nodes[2]=Node(1,2)
nodes[3]=Node(1,3)
nodes[4]=Node(1,4)
nodes[5]=Node(1,5)
nodes[6]=Node(1,6)
nodes[7]=Node(1,7)
nodes[8]=Node(1,8)
nodes[9]=Node(1,9)
lines=[Line(1,2),Line(2,3),Line(2,6),Line(6,7),Line(3,4),Line(4,5),Line(4,8),
Line(6,9)]

stack=[nodes[1]]    # Узел, скоторого начинамем построение схемы
order=[]            # Порядок обхода узлов

while len(stack)>0:
    n=stack.pop()
    order.append(n)
    inc=FindLines(ip=n.ny)
    for l in inc:
        stack.append(nodes[l.iq])
        nodes[l.iq].idx       = 1
        nodes[inc[0].iq].idx  = 2
        nodes[inc[-1].iq].idx = 0

CX=30 # текущая позиция по X

for n in order:
    if n.mark==True: continue
    n.mark = True

    if n.idx!=0: CX +=80
    n.x=CX
    for l in FindLines(iq=n.ny):
        n.y=nodes[l.ip].y
        if n.idx!=2: n.y+=80

c = Canvas(Tk(), width=500, height=500)
c.pack()

for l in lines:
    c.create_line(nodes[l.ip].x,nodes[l.ip].y,nodes[l.iq].x,nodes[l.iq].y)

for n in nodes.values():
    c.create_oval(n.x-3,n.y-3,n.x+3,n.y+3)
    c.create_text(n.x+10,n.y+10,text=str(n.ny))
```

Здесь мы создали 2 класса: узлы и ветви. В каждом классе мы создали переменную mark. Узлы мы сохраняем в словаре, поэтому любой узел легко найти по его номеру. Поскольку у ветвей есть не один, а два номер (узел начала и узел конца), найти их через словарь не получится. Для быстрого поиска ветвей мы создали функцию «FindLines», которая будет искать в таблице ветви по

номеру узла начала или номеру узла конца. В переменную «stack» изначально мы добавляем узел, который будет «сгорать» первым. Далее из этого списка удаляется этот узел и добавляются следующие. И так далее, пока мы не обработаем все узлы. Для создания графических элементов используется объект Canvas библиотеки tkinter.

Объект Canvas позволяет создавать следующие примитивы:

– create_rectangle – создать прямоугольник;

– create_line – создать линию;

– create_oval – создать овал или окружность;

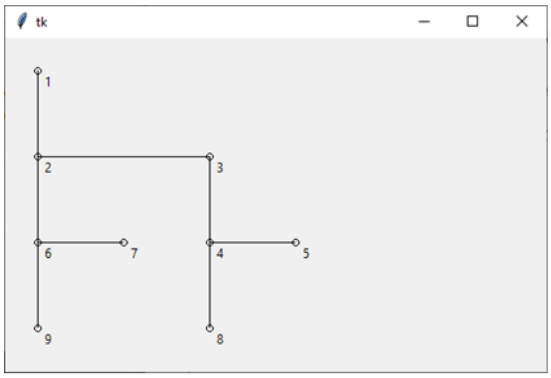
– create_text – создать надпись;

– create_arc – создать дугу

– create_polygon – создать многоугольник

Подробное описание упомянутых функций можно посмотреть в официальной документации к библиотеке, например здесь: https://metanit.com/python/tkinter/7.1.php

В результате выполнения приведенного выше кода получится следующее изображение схемы сети:



Задание

1. Доработайте приведенный пример программы для отображения схемы сети из базы данных, реализованной во 2-й работе.

2. Добавьте отображение трансформаторов.

3. Добавьте отображение выключателей.

4. Добавьте отображение нагрузок.

5. Добавьте отображение номинального напряжения

Контрольные вопросы

1. Для чего нужен «алгоритм поиска в ширину»?
2. Опишите основные этапы алгоритма поиска в ширину.
3. Какие параметры имеет функция create_rectangle?
4. Какие параметры имеет функция create_line?
5. Какие параметры имеет функция create_oval?
6. Какие параметры имеет функция create_text?
7. Какие параметры имеет функция create_arc?
8. Какие параметры имеет функция create_polygon?
9. Какую библиотеку для создания графических примитивов вы использовали?

СПИСОК РЕКОМЕНДОВАННОЙ ЛИТЕРАТУРЫ

1. Леоненков, А. Нотация и семантика языка UML / А. Леоненков. - 2-е изд., исправ. - Москва : Национальный Открытый Университет «ИНТУИТ», 2016. - 205 с. : ил. - (Основы информационных технологий). - http://biblioclub.ru/. - Библиогр. в кн. - ISBN 5-94774-408-2, экземпляров неограничено

2. Северенс, Ч. Введение в программирование на Python / Ч. Северенс. - 2-е изд., испр. - Москва : Национальный Открытый Университет «ИНТУИТ», 2016. - 231 с. : схем., ил. - http://biblioclub.ru/, экземпляров неограничено

3. Борзунов, С. В. Алгебра и геометрия с примерами на Python Электронный ресурс / Борзунов С. В., Кургалин С. Д. : учебное пособие для вузов. - Санкт-Петербург : Лань, 2020. - 444 с. - ISBN 978-5-8114-5489-1, экземпляров неограничено

4. Балджы, А. С. Математика на Python : учебно-методическое пособие / А.С. Балджы, М.Б. Хрипунова, И.А. Александрова ; Финансовый университет при Правительстве Российской Федерации, 1, Элементы линейной алгебры и аналитической геометрии. - Москва : Прометей, 2018. - 76 с. : табл. - http://biblioclub.ru/. - Библиогр. в кн. - ISBN 978-5-907003-86-6, экземпляров неограничено

5. Буч Г., Рамбо Д., Якобсон И. Язык UML. Руководство пользователя. 2-е изд.: Пер. с англ. Мухин Н. – М.: ДМК Пресс. – 496 с.: ил.

6. Кэгл К., Гиббонс Д., Хантер Д., Озу Н., Пиннок Д., Спенсер П., Введение в XML, Пер. с англ. Афанасьев И., Штерев И., − М.: Лори, 2006, 638 с.

7. ГОСТ Р 58651.2−2019, Национальный стандарт российской федерации, Единая энергетическая система и изолированно работающие энергосистемы, Информационная модель электроэнергетики, Базисный профиль информационной модели, Дата введения – 2020.01.01

8. Андреев Е.Б., Куцевич Н.А., Синенко О.В., SCADA-системы: взгляд изнутри / Е.Б. Андреев, Н.А. Куцевич, О.В. Синенко – М.: Издательство «РТСофт», 2004. - 176 с: ил.

9. Руденко Ю.Н. и Семенов В.А. Автоматизация диспетчерского управления в электроэнергетике/ под общей ред. Ю.Н. Руденко и В.А. Семенова. – М.: Издательство МЭИ, 2000. — 648 с.: ил.

10. Официальный сайт Python: https://www.python.org/

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

по организации самостоятельной работы студента
по дисциплине
«Системы управления и сбора данных в электроэнергетике**»**

Направление подготовки
13.03.02 «Электроэнергетика и электротехника»

Ставрополь, 2025

СОДЕРЖАНИЕ

Введение

Самостоятельная работа является внеаудиторной и предназначена для самостоятельного изучения определенных тем дисциплины по материалам, рекомендованным преподавателем, а также в порядке подготовки к практическим занятиям.

Цель дисциплины «Систему управления и сбора данных в электроэнергетике» – формирование согласно ООП ВО набора профессиональных компетенций будущего бакалавра по направлению 13.03.02 «Электроэнергетика и электротехника».

Для достижения этой цели в процессе изучения дисциплины решаются следующие задачи: ознакомление с основными понятиями и терминами изучаемой дисциплины, которыми будущий специалист будет оперировать в своей практической деятельности.

1 Общая характеристика самостоятельной работы студента при изучении дисциплины

Основная задача высшего образования заключается в формировании творческой личности специалиста, способного к саморазвитию, самообразованию, инновационной деятельности. Решение этой задачи вряд ли возможно только путем передачи знаний в готовом виде от преподавателя к студенту. Для решения этой задачи в учебные планы всех специальностей включена самостоятельная работа, составляющая 50% учебного времени студента.

В широком смысле под самостоятельной работой студентов следует понимать совокупность всей самостоятельной деятельности студентов, как в учебной аудитории, так и вне ее, в контакте с преподавателем и в его отсутствие.

В учебном процессе выделяют два вида самостоятельной работы:

– аудиторная;

– внеаудиторная.

Аудиторная самостоятельная работа по дисциплине выполняется на учебных занятиях под непосредственным руководством преподавателя и по его заданию.

Внеаудиторная самостоятельная работа выполняется студентом по заданию преподавателя, но без его непосредственного участия.

Содержание внеаудиторной самостоятельной работы определяется в соответствии с рекомендуемыми видами заданий согласно примерной и рабочей программ учебной дисциплины. Видами заданий для внеаудиторной самостоятельной работы по дисциплине «Систему управления и сбора данных» являются:

– для овладения знаниями: чтение текста (учебника, первоисточника, дополнительной литературы), составление плана текста, графическое изображение структуры текста, конспектирование текста, выписки из текста, работа со словарями и справочниками, ознакомление с нормативными

документами, учебно-исследовательская работа, использование аудио- и видеозаписей, компьютерной техники и Интернета и др.;

– для закрепления и систематизации знаний: обработка текста, повторная работа над учебным материалом (учебника, первоисточника, дополнительной литературы, аудио и видеозаписей, составление плана, составление таблиц для систематизации учебною материала, ответ на контрольные вопросы, заполнение рабочей тетради, аналитическая обработка текста (аннотирование, рецензирование, реферирование, конспект-анализ и др);

– для закрепления навыков – подготовка к лабораторным работам, оформление отчетов по лабораторным работам, выполнение дополнительных заданий, выданных преподавателем.

Самостоятельная работа может осуществляться индивидуально или группами студентов в зависимости от цели, объема, конкретной тематики самостоятельной работы, уровня сложности, уровня умений студентов. Контроль результатов внеаудиторной самостоятельной работы студентов может осуществляться в пределах времени, отведенного на обязательные учебные занятия по дисциплине и внеаудиторную самостоятельную работу студентов по дисциплине, может проходить в письменной, устной или смешанной форме.

Следует с первых дней обучения в вузе приучать себя к целесообразному распределению занятий по месяцам и неделям семестра. При этом очень важна равномерная работа как основное условие успешных занятий в вузе.

Высшей ступенью обучения является приобретение навыков самостоятельной творческой работы, а творческому применению знаний нужно учиться параллельно с их приобретением.

Задачами самостоятельной работы студентов по дисциплине «Систему управления и сбора данных в электроэнергетике» являются:

– формирования готовности студентов к поиску, обработке и применению информации для решения профессиональных задач;

– систематизация и закрепление полученных теоретических знаний и практических умений студентов;

– развитие познавательных способностей и активности студентов, творческой инициативы, самостоятельности, ответственности и организованности;

– формирование самостоятельности мышления, способностей к саморазвитию, самосовершенствованию и самореализации;

– выработка навыков эффективной самостоятельной профессиональной деятельности.

2 План-график выполнения самостоятельной работы

Общий объём самостоятельной работы в рамках изучения дисциплины «Систему управления и сбора данных в электроэнергетике» составляет 36 часов.

В структуре самостоятельной работы предусмотрено самостоятельное изучение теоретического материала, подготовка к лабораторным занятиям.

В таблице 2.1 приведен план график выполнения самостоятельной работы.

Таблица 2.1 – План график выполнения самостоятельной работы

Коды реализуемых компетенций, индикатора(ов)	Вид деятельности студентов	Средства и технологии оценки	Объем часов, в том числе		
			СРС	Контактная работа с преподавателем	Всего
3 семестр					
ИД-1$_{ПК-4}$	Подготовка к лекции	Собеседование	9		9
ИД-1$_{ПК-4}$	Подготовка к практическим занятиям	Собеседование	9		9
ИД-1$_{ПК-4}$	Самостоятельное изучение литературы	Контрольная работа	18		18
Итого за 4 семестр			36		36
Итого			36		36

3 Контрольные точки и виды отчетности по ним

Успеваемость студентов по дисциплине оценивается в ходе текущего и промежуточного контроля. Текущий контроль осуществляется в течение семестра на практических и лекционных занятиях.

В начале семестра студенческая группа получает график контрольных мероприятий с указанием форм и сроков их проведения, с указанием баллов за каждое контрольное мероприятие.

4 Методические рекомендации по изучению теоретического материала

Большая часть самостоятельной работы студента состоит в изучении литературы. Одна из задач студента – научиться самостоятельно работать с книгой, а это требует определенных затрат энергии и времени. Самостоятельная работа с книгой может быть успешной, если текст не только прочитан, но и законспектирован. Существует несколько форм записей, но любая форма записи не даст нужного результата, если не будет пробуждать мысли того, кто ее ведет, если отсутствует активная работа ума и формирование своих выводов из прочитанного.

Выбор формы записи зависит от индивидуальных особенностей человека, его образованности и опыта. При этом не меньшую роль играет назначение записей, то есть то, какие задачи ставит перед собой человек (для самообразования, для выступления на семинаре, для использования в будущем).

Введение записей мобилизует наряду со зрительной памятью, также и моторную память. Кроме того, у человека, систематически ведущего записи изучаемой литературы, создается свой фонд материалов для быстрого повторения и мобилизации накопленных знаний.

Все записи должны быть убористыми и компактными. Интервалы между строками должны быть достаточными, чтобы вписывать дополнения. Рекомендуется вести записи ручкой, а карандашом или ручкой другого цвета пользоваться для отметок и выделений при последующей работе. Полезно также датировать записи.

Записи могут носить различный характер: план, выписки, тезисы, аннотирование, конспектирование.

1. План - наиболее краткая формой записи. Это перечень вопросов, рассматриваемых в книге или статье. План обычно раскрывает структуру произведения, логику автора, способствует лучшей ориентации в содержании.

Так, составленным планом можно воспользоваться, чтобы вспомнить прочитанное или быстро отыскать в книге нужное место. Представление об основных пунктах плана дает оглавление книги, поэтому во многих случаях наименования глав и разделов можно использовать в качестве пунктов. Составление плана приучает логически мыслить, вырабатывать умение сжато и последовательно излагать суть вопроса в письменной и устной форме.

Существует два способа составления плана: работа над ним по ходу чтения и составление плана после ознакомления с произведением. При этом план получается более последовательным и стройным.

Трудность составления плана состоит в том, что надо выяснить для себя, прежде всего, построение изучаемого текста, ход мыслей автора и лишь затем изложить содержание работы кратко и ясно. При всей своей краткости план дает представление о содержании прочитанного. Следует учесть, что форма плана не исключает цитирования отдельных мет и обобщений. Различают простой и развернутый план. В отличие от простого плана развернутый план не только содержит перечисление вопросов, но и раскрывает основные идей произведения, может включать выдержки из него, схемы, таблицы. Планом, особенно развернутым, необходимо пользоваться при написании выступления или статьи.

В целом развернутый план дает гораздо большее представление о произведении, его основных идеях, задачах, которые в нем решаются. Он может включать положения, замечания, собственные мысли студента.

Важно знать, что составление планов помогает вырабатывать способность к отвлеченному, абстрактному мышлению, но наибольшую пользу составление плана даст подготовленным лицам, которые бывают достаточно лишь взглянуть

на перечень основных вопросов, чтобы воспроизвести содержание прочитанного.

2. Тезисы – более сложная и совершенная форма записи, чем составление плана.

Это сжатое изложение основных мыслей прочитанного произведения или подготовляемого вступления. Особенностью тезисов является их утвердительный характер.

В них сосредотачивается самое главное, только выводы и обобщения, в них меньше доказательств, иллюстрации и пояснений. Тезисы не должны повторять дословно текст, но в ряде мест могут быть близки к нему, воспроизводя некоторые характерные выражения автора, важные для понимания хода его мыслей. Составление тезисов помогает глубже понять основные идеи произведения, выделить главное в нем; приучают сжато, точно и четко сформулировать свои мысли, повышает культуру речи и письма. При составлении тезисов учитывают следующее. Прежде всего, если произведение небольшое, необходимо внимательно изучать его в целом, если большое – изучать по главам и разделам. Затем, когда будут ясны основные идеи, кратко и последовательно излагать их в виде пунктов.

Различают простые и сложные, развернутые тезисы. Если записывают только утверждение чего – либо, такой тезис называют простым, а сложным тезисом будет выражение главной мысли, содержащее, кроме утверждения, еще и краткое ее доказательство.

Часто тезисы формулируются самим автором как выводы и обобщения в заключении книги или разделах книги. Нередко тезисы выделяются в тексте другим шрифтом.

Рекомендуется делать тезисные записи своими словами, причем можно записывать один абзац за другим, учитывая смысловую связь между ними. Но в большинстве случаев следует составлять сводный тезис, сложный по форме. При этом объединяется несколько утверждений, тесно связанных между собой.

Тезисы по содержанию очень близки к конспекту, но конспект носит более описательный характер, и его положения не столь категоричны, как в тезисах. Кроме того, конспект представляет собой более полную форму записи.

Следует отметить, что различие между формами записей условно, но в любой форме запись – важнейшая часть самостоятельной работы с книгой.

3. Выписки. Это записи текста из книги: теоретических положений, статистических данных, имеющих по мнению читателя важное значение.

Достоинство выписок состоит в точности воспроизведения текста книги, удобстве пользования записями при последующей работе, в накоплении обобщений и фактического материала. Выписки полезны для повторения, освежения в памяти прочитанного, для быстрой мобилизации своих знаний, когда необходимо в короткий срок вспомнить материал. Выписки выделяют из текста самое главное и тем самым помогают глубже понять его. Без них трудно обойтись при подготовке доклада, реферата, выступления. Выписки следует рассматривать как составную часть тезисов и конспектов.

Выписывать текст можно и по ходу чтения и после его завершения. В последнем случае надо замечать места, которые потом будут выписаны. Необходимо каждую выписку снабжать ссылкой на источник с указанием соответствующей станицы. Это нужно, чтобы в последствии можно было быстро найти в книге соответствующее место. Целесообразно выписывать из текста только такие места, в которых содержится самое главное, суть вопроса. Выписки должны быть ориентированы на изучение произведения в целом, а не отдельных мест, поскольку положения, вырванные из общего контакта, понимаются нередко совсем не так, как этого хотел автор. Иначе говоря, отдельно взятые, лишенные пояснений выдержки могут быть не поняты или поняты неправильно.

Выписки бывают дословные (цитаты) и «свободные», когда мысли автора излагаются своими словами. Следует учесть, что большие отрывки, которые трудно цитировать, целесообразнее в краткой форме переложить своими словами, но «яркие» и важные места лучше выписывать дословно. Каждую цитату следует заключать в кавычки. Если ее берут из середины предложения,

то после вводных кавычек ставят три точки. Ставят их и в конце цитаты, если из предложения опущены последние слова.

Следует знать, что какого-либо единого метода выписок, годного для всех случаев, не существует, поскольку у каждого человека свои особенности мышления и восприятия, свои подход к теме. Все это влияет на содержание и характер выписок.

Выписки рекомендуется хранить в картотеке, конвертах или папках, на которых следует обозначить общую тему.

4. Аннотация – еще одна форма записи, являющаяся кратким обобщением содержания книги. Ею удобно пользоваться, если имеется намерение вернуться к изучаемому произведению. Аннотация может быть необходима и для того, чтобы не забыть о нем.

Для составления аннотации надо сначала полностью прочитать и глубоко продумать произведение. При всей своей краткости аннотация может содержать отдельные фрагменты авторского текста, а не только оценку книги или статьи.

5. Резюме очень близко к аннотации. Это запись, являющаяся краткой оценкой прочитанного материала. Различие между ними состоит в том, что аннотация сжато характеризует произведение в целом, а резюме концентрирует внимание на его выводах, главных итогах.

6. Конспект – наиболее совершенная и наиболее сложная форма записи. Слово «конспект» происходит от латинского «conspectus», что означает «обзор, изложение». В правильно составленном конспекте обычно выделено самое основное в изучаемом тексте, сосредоточено внимание на наиболее существенном, в кратких и четких формулировках обобщены важные теоретические положения.

Конспект представляет собой относительно подробное, последовательное изложение содержания прочитанного. На первых порах целесообразно в записях ближе держаться тексту, прибегая зачастую к прямому цитированию автора. В дальнейшем, по мере выработки навыков конспектирования, записи будут носить более свободный и сжатый характер.

Конспект книги обычно ведется в тетради. В самом начале конспекта указывается фамилия автора, полное название произведения, издательство, год и место издания. При цитировании обязательная ссылка на страницу книги. Если цитата взята из собрания сочинений, то необходимо указать соответствующий том. Следует помнить, что четкая ссылка на источник – непременное правило конспектирования. Если конспектируется статья, то указывается, где и когда она была напечатана.

Конспект подразделяется на части в соответствии с заранее продуманным планом. Пункты плана записываются в тексте или на полях конспекта. Писать его рекомендуется четко и разборчиво, так как небрежная запись с течением времени становиться малопонятной для ее автора. Существует правило: конспект, составленный для себя, должен быть по возможности написан так, чтобы его легко прочитал и кто-либо другой.

Формы конспекта могут быть разными и зависят от его целевого назначения (изучение материала в целом или под определенным углом зрения, подготовка к докладу, выступлению на занятии и т.д.), а также от характера произведения (монография, статья, документ и т.п.). Если речь идет просто об изложении содержания работы, текст конспекта может быть сплошным, с выделением особо важных положений подчеркиванием или различными значками.

В случае, когда не ограничиваются переложением содержания, а фиксируют в конспекте и свои собственные суждения по данному вопросу или дополняют конспект соответствующими материалами их других источников, следует отводить место для такого рода записей. Рекомендуется разделить страницы тетради пополам по вертикали и в левой части вести конспект произведения, а в правой свои дополнительные записи, совмещая их по содержанию.

Конспектирование в большей мере, чем другие виды записей, помогает вырабатывать навыки правильного изложения в письменной форме важные теоретических и практических вопросов, умение четко их формулировать и ясно излагать своими словами.

Таким образом, составление конспекта требует вдумчивой работы, затраты времени и труда. Зато во время конспектирования приобретаются знания, создается фонд записей.

Конспект может быть текстуальным или тематическим. В текстуальном конспекте сохраняется логика и структура изучаемого произведения, а запись ведется в соответствии с расположением материала в книге. За основу тематического конспекта берется не план произведения, а содержание какой-либо темы или проблемы.

Текстуальный конспект желательно начинать после того, как вся книга прочитана и продумана, но это, к сожалению, не всегда возможно. В первую очередь необходимо составить план произведения письменно или мысленно, поскольку в соответствии с этим планом строится дальнейшая работа. Конспект включает в себя тезисы, которые составляют его основу. Но, в отличие от тезисов, конспект содержит краткую запись не только выводов, но и доказательств, вплоть до фактического материала. Иначе говоря, конспект – это расширенные тезисы, дополненные рассуждениями и доказательствами, мыслями и соображениями составителя записи.

Как правило, конспект включает в себя и выписки, но в него могут войти отдельные места, цитируемые дословно, а также факты, примеры, цифры, таблицы и схемы, взятые из книги. Следует помнить, что работа над конспектом только тогда будет творческой, когда она не ограничена текстом изучаемого произведения. Нужно дополнять конспект данными из другими источников.

В конспекте необходимо выделять отдельные места текста в зависимости от их значимости. Можно пользоваться различными способами: подчеркиваниями, вопросительными и восклицательными знаками, репликами, краткими оценками, писать на полях своих конспектов слова: «важно», «очень важно», «верно», «характерно».

В конспект могут помещаться диаграммы, схемы, таблицы, которые придадут ему наглядность.

Составлению тематического конспекта предшествует тщательное изучение всей литературы, подобранной для раскрытия данной темы. Бывает,

что какая-либо тема рассматривается в нескольких главах или в разных местах книги. А в конспекте весь материал, относящийся к теме, будет сосредоточен в одном месте. В плане конспекта рекомендуется делать пометки, к каким источникам (вплоть до страницы) придется обратиться для раскрытия вопросов. Тематический конспект составляется обычно для того, чтобы глубже изучить определенный вопрос, подготовиться к докладу, лекции или выступлению на семинарском занятии. Такой конспект по содержанию приближается к реферату, докладу по избранной теме, особенно если включает и собственный вклад в изучение проблемы.

5 Методические указания по подготовке к практическим занятиям

В рамках самостоятельной работы предусмотрена подготовка студента к практическим занятиям. Практические занятия позволяют интегрировать теоретические знания и формировать практические умения и навыки студентов в процессе учебной деятельности.

Цели практических занятий по дисциплине «Системы управления и сбора данных в электроэнергетике»:

1) закрепление теоретического материала путем систематического контроля за самостоятельной работой студентов;

2) формирование умений использования теоретических знаний в процессе выполнения практических работ;

3) развитие аналитического мышления путем обобщения результатов практических работ;

4) формирование навыков оформления результатов практических работ в виде таблиц, графиков, выводов.

На практических занятиях осуществляются следующие формы работ со студентами: индивидуальная (оценка знаний, выполненных тестовых заданий, проверка рабочих тетрадей); групповая (выполнение заданий малыми группами по 2-4 человека); фронтальная (подведение итогов выполнения практических работ).

Список рекомендуемой литературы

Основная литература:

1. Леоненков, А. Нотация и семантика языка UML / А. Леоненков. - 2-е изд., исправ. - Москва : Национальный Открытый Университет «ИНТУИТ», 2016. - 205 с. : ил. - (Основы информационных технологий). - http://biblioclub.ru/. - Библиогр. в кн. - ISBN 5-94774-408-2, экземпляров неограничено

2. Северенс, Ч. Введение в программирование на Python / Ч. Северенс. - 2-е изд., испр. - Москва : Национальный Открытый Университет «ИНТУИТ», 2016. - 231 с. : схем., ил. - http://biblioclub.ru/, экземпляров неограничено

3. Борзунов, С. В. Алгебра и геометрия с примерами на Python Электронный ресурс / Борзунов С. В., Кургалин С. Д. : учебное пособие для вузов. - Санкт-Петербург : Лань, 2020. - 444 с. - ISBN 978-5-8114-5489-1, экземпляров неограничено

4. Балджы, А. С. Математика на Python : учебно-методическое пособие / А.С. Балджы, М.Б. Хрипунова, И.А. Александрова ; Финансовый университет при Правительстве Российской Федерации, 1, Элементы линейной алгебры и аналитической геометрии. - Москва : Прометей, 2018. - 76 с. : табл. - http://biblioclub.ru/. - Библиогр. в кн. - ISBN 978-5-907003-86-6, экземпляров неограничено

Дополнительная литература:

1. Буч Г., Рамбо Д., Якобсон И. Язык UML. Руководство пользователя. 2-е изд.: Пер. с англ. Мухин Н. – М.: ДМК Пресс. – 496 с.: ил.

2. Кэгл К., Гиббонс Д., Хантер Д., Озу Н., Пиннок Д., Спенсер П., Введение в XML, Пер. с англ. Афанасьев И., Штерев И., − М.: Лори, 2006, 638 с.

3. ГОСТ Р 58651.2−2019, Национальный стандарт российской федерации, Единая энергетическая система и изолированно работающие энергосистемы, Информационная модель электроэнергетики, Базисный профиль информационной модели, Дата введения – 2020.01.01

4. Андреев Е.Б., Куцевич Н.А., Синенко О.В., SCADA-системы: взгляд изнутри / Е.Б. Андреев, Н.А. Куцевич, О.В. Синенко – М.: Издательство «РТСофт», 2004. - 176 с: ил.

5. Руденко Ю.Н. и Семенов В.А. Автоматизация диспетчерского управления в электроэнергетике/ под общей ред. Ю.Н. Руденко и В.А. Семенова. – М.: Издательство МЭИ, 2000. — 648 с.: ил.

Учебно-методическая литература:

6. Системы управления и сбора данных в электроэнергетике. Методические указания по выполнению практических работ. Направление подготовки 13.03.02 «Электроэнергетика и электротехника», СКФУ, 2024 /сост. Зеленский Е.Г.