УДК 621.311-047.58:004.438(Java)

К.А. Один

Пермский национальный исследовательский
политехнический университет

КОНСТРУКТОР СХЕМ ДЛЯ МОДЕЛИРОВАНИЯ ЭЛЕКТРОЭНЕРГЕТИЧЕСКИХ СИСТЕМ

Рассматривается создание визуального конструктора схем на примере электроэнергетических систем. Описывается архитектура данной библиотеки, и даются рекомендуемые к использованию библиотеки для языка Java.

В программно-моделирующих комплексах электроэнергетических систем с произвольной структурой требуется визуальное создание схем этих систем из отдельных ее элементов (генераторов, двигателей, трансформаторов, нагрузок, газотурбинных установок, и т.д.). Все операции по созданию схемы удобно выполнять посредством манипулятора мыши и клавиатуры. К таким операциям, например, относятся: создание и удаление элементов, их соединение и редактирование параметров. Также в больших схемах отдельные логические блоки из набора элементов удобно отделять в подсистемы – специальные контейнеры с внутренней схемой, что прибавляет задач по взаимосвязи этих схем.

Для реализации такого конструктора схем необходимо написание специальной библиотеки, назовем ее проект-схемой. Проект-схема должна выполнять функции по хранению элементов, модификации структуры схемы, ее визуальному отображению и управлению событиями манипулятора мыши на визуальных панелях отображения схем.

Реализация конструктора схем

Реализуем библиотеку проект-схемы на языке программирования *Java*. И представим ее модель в виде диаграммы классов *UML* (рисунок). Отобразим на диаграмме лишь основные классы, атрибуты и методы.

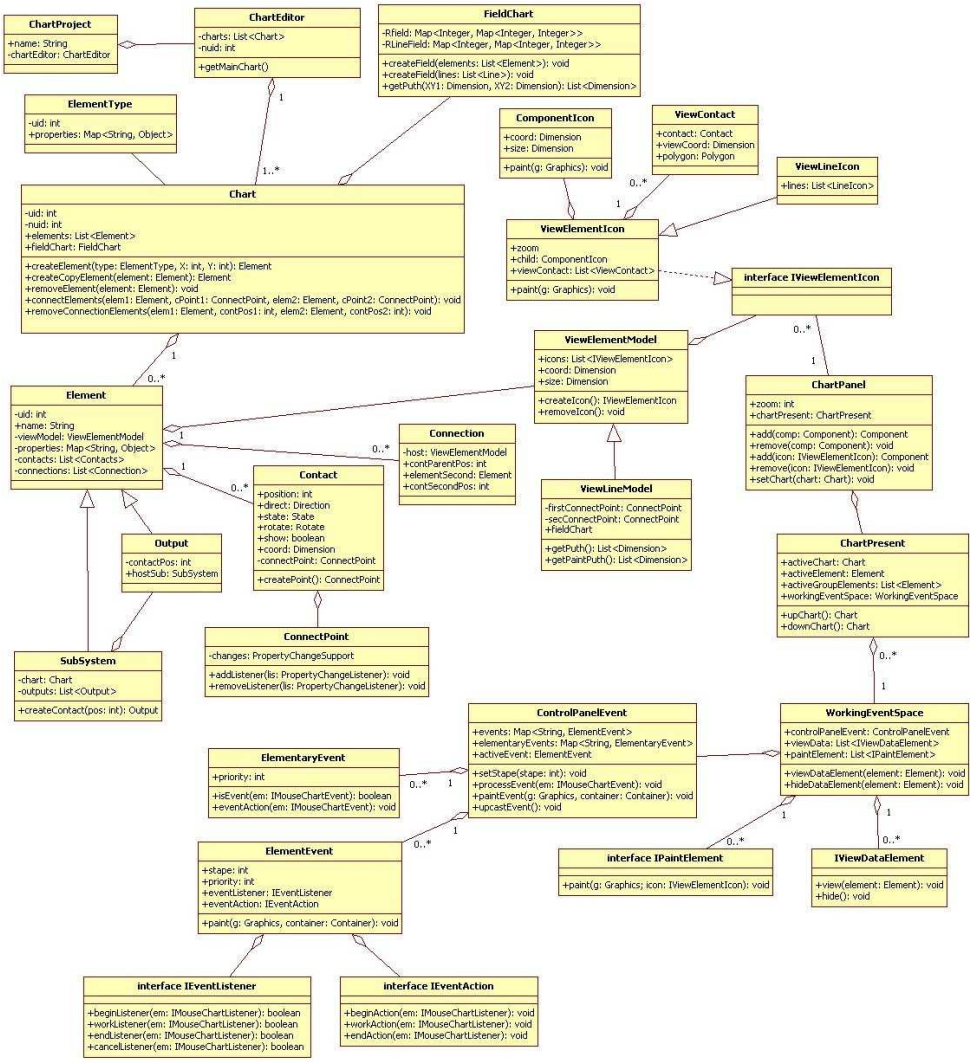

Рис. Диаграмма классов *UML*-библиотеки проекта-схемы

Хранение схемы. Хранение проекта-схемы представляется иерархически в объектах нескольких классов. *ChartProject*, *ChartEditor*, *Chart*, *ChartElement*.

Объект класса *ChartProject* хранит всю иерархическую структуру проекта со схемами, элементами и соединениями между элементами. Но фактически объект данного класса хранит в себе лишь название проекта, идентификационный номер и ссылку на объект класса *ChartEditor*.

Объект класса *ChartEditor* управляет всеми схемами проекта. А именно: присваивает уникальные идентификаторы; создает и удаляет схемы; предоставляет доступ к схемам.

Управление элементами осуществляет объект класса *Chart*, который хранит элементы, присваивая им уникальные идентификаторы, создает и удаляет. Создание элементов осуществляется по типу. Тип элементов содержит все необходимые параметры для создания элемента и реализован в классе *ElementType*. Основные параметры типа – это класс элемента и математическая модель. Сами элементы могут инициализироваться посредством вспомогательных классов и динамической загрузки с использованием пакета *java.reflection* [1].

Также объект класса *Chart* выполняет функцию соединения двух элементов. Соединение выполняется между двумя контактами первого и второго элемента (см. ниже).

Сами элементы схемы удобно выполнить по модели *MVC* (*model-view-controller*). Контроллером в данном случае является класс *ElementChart*, моделью отображения – класс *ViewElementModel*, а отображением – *ViewElementIcon*.

Хранение соединений осуществляется в самом элементе в виде списка. При этом у элемента существуют контакты класса *Contact*, которые и служат для соединения. Контакт содержит информацию о состоянии соединения, своих координатах, направлении вывода и другую информацию. При соединении контакт для инкапсуляции может выдавать специальный объект *ConnectPoint*, хранящий всегда актуальные координаты контакта и уведомляющий об их изменениях.

Для отделения частей схемы в независимые блоки предназначены подсистемы (класс *SubSystem*) – элемент схемы, имеющий внутри себя схему. Связь внешней схемы (в которой лежит подсистема) с внутренней осуществляется посредством контактов подсистемы и выводов (класс *Output*). Вывод – элемент схемы, ассоциированный с конкретным контактом подсистемы и лежащий в ее внутренней схеме.

Отображение схемы. Отображение схемы с ее элементами выполняется посредством трех компонентов: *ChartPresent*, *ChartPanel* и *ViewElementIcon*, в которой объект класса *ChartPresent* управляет размещением иконок элементов схемы в контейнере. Если необходимо перейти в схему подсистемы или подняться на уровень выше, обращение происходит именно к *ChartPresent*. Объект данного класса

привязан к конкретному контейнеру *ChartPanel* и может, например, дополнительно хранить активные элементы данного контейнера.

Объект класса *ChartPanel*, как было сказано, является контейнером, который управляет отображением иконок элементов схемы. Данный класс может быть создан на основе компонента *JPanel* библиотеки *Swing* [1, 2].

Сами иконки можно выполнить посредством интерфейса (*IViewIcon*), оставив его реализацию за элементами схемы. Элементы же позволяют создавать данные иконки для каждого контейнера, которые отображают схему с данным элементом.

Если присутствует необходимость представлять элементы различными картинками, то более легковесными решениями будет использование формата векторной графики *SVG*, либо использование своего облегченного векторного формата. Для обеспечения первого варианта существует, к примеру, библиотека *Batik SVG Toolkit* [3], которая имеет в своем составе необходимые компоненты для отображения данного формата. Но компоненты данной библиотеки не имеют свойства прозрачности. Поэтому если присутствует необходимость выполнять какое-либо отображение элемента за его пределами, например контактов, то лучше иконки выполнить двумя компонентами: внешним прозрачным на основе библиотеки *Swing* и внутренним – библиотеки *Batik SVG Toolkit*. Также следует быть внимательным с форматом отображаемой *SVG* картинки, не каждый формат правильно распознается библиотекой.

Отображение соединений между элементами осуществляется проводником, причем проводник должен проводиться по кратчайшему возможному маршруту между двумя элементами (см. ниже). Иконка соединения при этом должна выкладывать в контейнер набор небольших компонентов, следующих по маршруту соединения, а при удалении соединения – убирать из контейнера. Следовательно, в случае иконки на основе компонента *JComponent*:

```
addAncestorListener(new AncestorListener ()
{
        @Override
        public   void   ancestorAdded(AncestorEvent
event)
      {
```

```
            addLineIcons();
        }
        @Override
        public void ancestorRemoved(AncestorEvent
event)
        {
            removeLineIcons();
        }
});
```

Метод *addLineIcons* добавляет набор компонентов маршрута соединения, а метод *removeLineIcons* – удаляет. При этом набор этих компонентов должен уведомлять главную иконку соединения о событиях мыши.

Построение маршрутов соединительных линий. Для создания структуры схемы элементы необходимо соединять между собой, а отображать данные соединения – проводником между соединенными элементами. При этом маршрут этих соединений должен строиться автоматически, по кратчайшему пути.

Данная функция может быть поручена отдельному классу *Field*. Данный класс хранит в двумерном массиве занятые элементами ячейки и строит оптимальные маршруты соединительных линий между этими элементами. Двумерный массив удобнее представить хэш-таблицей, вложенной в другую хэш-таблицу (*Map<Integer, Map<Integer, Integer>>*). Тогда не придется изменять размер массива и заново его формировать при увеличении схемы.

Тогда двумерных массивов должно быть три. Первый, как было изложено, для хранения занятых элементами ячеек, второй – для поиска оптимального текущего маршрута между двумя заданными точками, что может быть выполнено, например, посредством волнового алгоритма, и третий – для хранения маршрутов соединительных линий. Данный массив нужен в тех случаях, когда необходимо при пересечении одной линией другую графически отображать обрывом ее в месте пересечения.

При этом алгоритм построения маршрутов с обрывами будет следующим. В массив заносится первая соединительная линия без изменений. Все остальные заносятся поочередно, отмечая, где каждой из них необходимо совершить обрыв.

Сами найденные маршруты хранятся в линиях (элемент схемы), в виде списка (*List<Dimension>*). В списке маршрута помимо двух крайних точек отмечаются только точки в местах изгиба маршрута, а обрывы, например, точкой (–1; –1).

События пользовательского интерфейса. Для конструирования схем с элементами необходимо выполнение различных событий посредством манипулятора мыши. Это создание и удаление элементов, копирование и перемещение, соединение элементов между собой, выделение/сброс активного элемента или группы элементов. Помимо этого могут быть следующие события: открытие всплывающих меню элементов с набором функций, создание контактов у подсистем и шин для соединения с другими элементами, изменение размеров элементов и т.д.

Причем данные события можно разделить на два типа. События, которые начинаются и заканчиваются мгновенно в ответ на действия мыши. Например, такие как выделение и сброс выделения активного элемента; открытие подсистемы. Второй тип – это события, выполнение которых делится на несколько этапов с помощью манипулятора мыши: от нажатия клавиши на мыши и перемещения при нажатой клавише до ее отпускания. Это соединение элементов, создание элементов, выделение групп элементов, перемещение элемента или группы элементов, открытие-закрытие всплывающих меню, создание/удаление контактов, изменение размеров элемента. Такие события назовем сложными.

Для исключения конфликта между событиями во время выполнения одного из сложных необходимо создание блокировки на все остальные. В данном случае уместным будет создание класса *ControlPanelEvents*, объект которого будет управлять выполнением всех загруженных в него правил обработки событий на всех панелях с элементами, т.е. события со всех панелей направляются в объект данного класса с целью обработки. Это позволит выполнять операции между панелями, например, копирование и перемещение элементов.

Загрузку правил обработки событий удобно выполнять посредством специальных интерфейсов. Для загрузки сложного события служат два интерфейса: слушателя этапов события (*IEventListener*) и обработки этих этапов (*IEventAction*). Первый интерфейс определяет

моменты начала события, переходов между этапами сложного события (начальный, рабочий и конечный этапы) и отмену события, второй интерфейс – обработку всех этапов.

Для загрузки элементарного события служит интерфейс *IElementaryEvent*, который определяет два метода: метод, определяющий начало события и метод, определяющий его обработку.

Итак, все сообщения о событиях, возникающих на панелях с элементами, будут направляться в *ControlPanelEvents*, где будут обрабатываться нужным образом. Причем удобно модифицировать стандартные сообщения о событиях *MouseEvent* библиотеки *java.awt.event* и внести информацию о месте события (элемент, пустое место контейнера, ресайзер элемента, контакт элемента), контейнере, контакте элемента. Таким образом, сообщение о событии будет иметь все необходимые данные для его быстрой обработки.

Интерфейс *IViewDataElement* в объекте класса *WorkingEventSpace* используется для отображения активного элемента схемы и имеет два метода *view(ChartElement element)* и *hide()*. Сюда может быть интегрирована работа с визуальным редактором свойств элемента [4].

Использование библиотеки проекта-схемы для расчета схем электроэнергетических систем. Для расчета схем электроэнергетической системы необходимо, во-первых, закрепить за ее элементами электрические модели. Сделать это гибче можно в отдельном модуле. Модель же электроэнергетической системы может быть построена при использовании векторно-матричного подхода и принципа структурного моделирования [5].

Таким образом, библиотека проекта-схемы может использоваться как графическое представление схемы, и интерактивная работа с ней осуществляется посредством манипулятора мыши. Также можно выполнить данную библиотеку универсально и использовать ее не только для построения электроэнергетических схем, но и различных диаграмм и блок-схем, где требуется соединение между элементами.

Библиографический список

1. Шилдт Герберт. Полный справочник по Java: пер. с англ. – 7-е изд. – М.: Вильямс, 2007. – 1040 с.

2. Шилдт Герберт. Swing: руководство для начинающих: пер. с англ. – М.: Вильямс, 2007. – 704 с.

3. Главная страница официального сайта Java библиотеки Batik SVG Toolkit [Электронный ресурс]. – URL: http://xmlgraphics.apache.org/batik/ (дата обращения: 27.04.2012).

4. Петроченков А.Б., Один К.А. Редактор свойств элемента электроэнергетической системы для программного моделирующего комплекса на языке Java // Энергетика. Инновационные направления в энергетике. CALS-технологии в энергетике: материалы IV Всерос. науч.-техн. интернет-конф. – Пермь: Изд-во Перм. нац. исслед. политехн. ун-та, 2011. – С. 188–198.

5. Кавалеров Б.В. Математическое моделирование в задачах автоматизации испытаний систем управления энергетических газотурбинных установок // Известия Юго-Западного государственного университета. – 2011. – № 1. – С. 74–83.

Получено 06.09.2012