Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут”
Кафедра обчислювальної техніки

Курсова робота

з дисципліни “Операційні системи”

Керівник роботи:
Проф. Симоненко В.П.

Виконавець роботи:
ст. Михайленко А.В.
ФІОТ гр. ІО-72

Допущена до захисту

_____ «___» 2010 р.

Захищена

_____ «___» 2010 р.

Київ — 2010

Зміст

1 Технічне завдання

Запрограмувати просторовий планувальник для розподілу завдань у багатопроцесорній системі і дослідити його роботу.

Топологія обчислювальної системи - ланцюг процесорів.

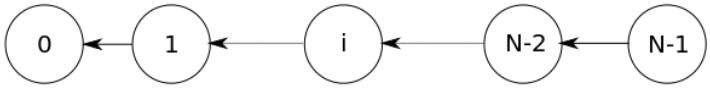


Рис. 1: Топологія обчислювальної системи

1.1 Аналіз топології

Лінійна топологія рідко використовується на сьогоднішній день, перш за все через те, що відмова одного компонента приводить до відмови усієї системи і через дуже низький коефіцієнт використання при сильній зв'язності задачі (що зумовлюється тим, що залежні задачі можна розміщувати лише на тих процесорах, куди можуть передати данні батьківські задачі). Але, теоретично це найдешевший із можливих варіантів топологій, який можна реалізувати на основі дешевих побутових або списаних комп'ютерів.

1.2 Допущення

У рамках цієї курсової роботи зробимо припущення:

1. Процесори можуть одночасно обчислювати задачу та передавати інформацію.
2. При транзитних пересилках процесор спочатку приймає усе повідомлення, а потім передає його наступній ланці.

З цих припущень випливає, що: $t_i = V * N$, де V - вага пересилки, N - кількість каналів між процесором-передавачем та процесором-приймачем.

2 Алгоритми просторового планування

Планування для ациклічних направлений графів - це NP-повна задача, яка оптимально вирішується лише для простих випадків. Оптимальні результати дають:

- Алгоритм Ху для однорідного дерева без вагів вузлів досягає оптимального планування за лінійний час.
- Алгоритм Кофмана та Грехема для однорідних вагів вершин досягає оптимальних результатів за квадратичний час.
- Алі та Ель-Ревіні розробили метод планування, який на відміну від попередніх двох варіантів працює з однорідними вагами пересилок та досягає оптимального результату за поліноміальний час.

2.1 Основні параметри, що використовуються при плануванні

Існують два основні підходи до планування на основі ациклічних орієнтованих графів:

- Планування на основі статичних списків. Алгоритм такого планування можна коротко описати так:
  1. Взяти із списку готових вершин першу.
  2. Розмістити її на процесорі, де вона може почати виконуватися через найменший час.
- Планування на основі динамічних списків:
  1. Визначити нові пріоритети усіх не розміщених вершин.
  2. Взяти вершину з найвищим пріоритетом.
  3. Розмістити її на процесорі, де вона може почати виконуватися через найменший час.

Під час планування ациклічних направлених графів частіше за все використовують 3 поняття:

t-level - це найдовший шлях від початкової вершини до даної вершини n_i, без врахування ваги n_i. Шлях рахується сумуванням ваги вершин і пересилок, які він охоплює. Цей параметр тісно пов'язаний з найменшим часом початку вершини.

b-level - це найдовший шлях від вершини n_i до вихідної вершини. Окремим видом цього параметру є static b-level (static level, sl), який відрізняється від традиційного тим, що при його підрахунку не враховується вага пересилок.

CP - критичний шлях графу - це найкоротший шлях від початкової до кінцевої вершини графу, визначає теоретичний найменший час виконання завдання.

У процесі планування t-level кожної вершини змінюється, а b-level частіше за все залишається незмінним до того моменту, як вершині буде виділено процесорний час.

2.2 Проблема маршрутизації повідомлень у неповнозв'язних системах

Для систем з обмеженими зв'язками виникає проблема маршрутизації та планування пересилки повідомлень.

Є два підходи до маршрутизації:

- Детерміністичний, при якому для кожної пари процесорів задано певні маршрути. Якщо вони зайняті, то повідомлення буде очікувати звільнення виділених маршрутів.
- Адаптивний, при якому система може динамічно обчислювати маршрути повідомлень між будь-якими двома процесорами.

При використанні адаптивної маршрутизації з'являються 2 типи блоків:

- Блокування ближнього повідомлення дальнім.
- Блокування дальнього повідомлення ближнім.

Перший варіант завжди набагато затратніший за часом. Ідея адаптивних алгоритмів полягає у використанні Алгоритмів Дейкстри для оптимізації знаходження найкоротших шляхів з ціллю уникнення першого типу блоків.

2.3 Короткий огляд алгоритмів планування

У нашому випадку ми маємо неповнозв'язну систему з обмеженою кількістю процесорів. Це звужує коло можливих варіантів алгоритмів до чотирьох:

Mapping Heuristic - алгоритм розроблений Ель-Ревіні та Льюісом у 1990 році. Він базується на встановленні пріоритетів згідно з статичним b-level.

Dynamic Level Scheduling - використовує показник, що називається динамічним рівнем (dynamic level). Динамічний рівень визначається як різниця статичного b-level та найменшого часу запуску вершини на процесорі. На кожній ітерації визначається максимальне значення динамічного рівня для всіх пар готових вершин і процесорів і обирається пара з найбільшим показником. Алгоритм був розроблений Сіхом та Лі у 1993 році.

Bottom-Up - алгоритм планує виконання критичного шляху на одному процесорі, інші вершини розподіляються так щоб збалансувати навантаження на процесори. Після цього додаються пересилки за допомогою специфічних для топології евристичних методів.

Bubble Scheduling and Allocation - алгоритм розроблений Квоком та Ахмадом у 1995 році. Спочатку усі вершини направляються на один процесор, який має найбільшу кількість зв'язків і стає основним. На другій фазі починається переміщення вершин на сусідні процесори, якщо при цьому покращується час запуску. Після цього основним стає один із сусідніх процесорів і алгоритм повторюється.

2.4 Детальний огляд алгоритму Mapping Heuristic

Алгоритм був уперше розроблений у 1990 році Ель-Ревіні та Льюісом (El-Rewini and Lewis).

Коротко, він може бути описаний наступним чином:

1. Обчислити всі статичні b-рівні для всіх вершин n_i у графі завдання.
2. Ініціалізувати список готових вершин усіма вхідними вершинами. Список повинен бути відсортований за спаданням пріоритетів.
3. Забрати першу вершину із списку.
4. Запланувати виконання цієї вершини на процесор, на якому вона зможе почати виконуватися якнайшвидше. Для визначення часу початку виконання необхідно маршрутизувати усі пересилки від батьківських вершин до виконуваної згідно з таблицями маршрутизації.

   Коментар: у зв'язку з особливостями топології, підходящими процессорами є лише ті, на які можна передати данні від усіх батьківських вершин.
5. Додати усі активовані вершини до списку готових вершин.
6. Повторювати пункти 3-5 до того часу як список готових вершин не стане порожнім.

На першому етапі алгоритму обчислюються статичний b-level для усіх вершин. Далі він використовується як критерій пріоритету при плануванні вершин.

До списку готових вершин додаються усі вхідні вершини у порядку спадання пріоритету.

Для визначення часу початку виконання вершини на процесорі використовується таблиця маршрутизації, згідно якої визначається час передачі від усіх батьківських вершин.

Очевидно, що найменший час початку виконання вершини на процесорі - це більша з двох величин:

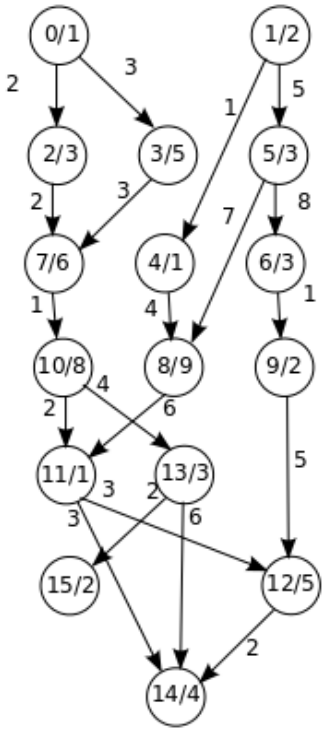
- час завершення попередньої задачі на цьому процесорі;
- максимальний час завершення батьківської задачі плюс час на пересилку.

Через те, що вершини приорітизуються за статичним b-рівнем, алгоритм спершу обирає важливі вершини (наприклад, вершини, що лежать на критичному шляху).

Часова складність алгоритму визначається як $O(v(p^3v+e))$, де p - кількість процесорів у системі.

3 Попередній аналіз тестових даних

Для аналізу якості планування у цій роботі ми будемо використовувати наступний приклад:



З графу видно, що максимальна ширина ярусу алгоритму - 3. Отже, необхідна кількість процесорів та максимальна степінь розпаралелювання також буде 3.

Критичний шлях графу:

№ Вершини	Вага
1	2
Зв'язок	5
5	3
Зв'язок	7
8	9
Зв'язок	6
11	1
Зв'язок	3
12	5
Зв'язок	2
14	4
Усього	47
Вершин	24

Табл. 1: Критичний шлях у графі алгоритму

Візуально видно, що граф сильно зв'язаний, з часто пересікаючимися гілками, що негативно вплине на загальне вирішення задачі планування.

З іншої сторони, ця особливість дасть змогу протестувати оптимізацію пересилок у планувальнику.

4 Аналіз результатів планування

Критеріями оптимізації у нашому випадку були загальний час виконання та час пересилок між процесорами.

Розглянемо результати роботи програми:

Діаграма Ганта розпланованої задачі:

№	P0	P1	P2
0	1	-1	0
1	1	-1	3
2	5	-1	3
3	5	2	3
4	5	2	3
5	4	2	3
6	8	-1	-1
7	8	-1	-1
8	8	-1	-1
9	8	7	-1
10	8	7	-1
11	8	7	-1
12	8	7	-1
13	8	7	-1
14	8	7	-1
15	6	10	-1
16	6	10	-1
17	6	10	-1
18	9	10	-1
19	9	10	-1
20	-1	10	-1
21	-1	10	-1
22	-1	10	-1
23	-1	13	-1
24	-1	13	-1
25	11	13	-1
26	12	15	-1
27	12	15	-1
28	12	-1	-1

29	12	-1	-1
30	12	-1	-1
31	-1	-1	-1
32	14	-1	-1
33	14	-1	-1
34	14	-1	-1
35	14	-1	-1

Інформація по кожній вершині:

№	Процесор	Час початку	Час кінця	Статичний b-рівень
0	2	0	1	30
1	0	0	2	24
2	1	3	6	27
3	2	1	6	29
4	0	5	6	20
5	0	2	5	22
6	0	15	18	14
7	1	9	15	24
8	0	6	15	19
9	0	18	20	11
10	1	15	23	18
11	0	25	26	10
12	0	26	31	9
13	1	23	26	7
14	0	32	36	4
15	1	26	28	2

Табл. 4: Статистика планування для кожної вершини

Отриманий варіант на 46% довший за оптимальний, що є нормальним. Така різниця спричинена як тим фактом, що евристичні алгоритми майже ніколи не дають оптимальних рішень, так і тим, що топологія системи зовсім не підходить для таких сильно зв'язаних задач.

5 Код програми

Listing 1: MHscheduler.java

```
1 package data;
2 import java.util.ArrayList;
3
4 public class MHscheduler {
5         private ArrayList<Node> nodes_;
6         private ArrayList<Node> entryNodes_=new ArrayList<Node>();
7         private ArrayList<Node> exitNodes_=new ArrayList<Node>();
```

```
8       private ArrayList<Node> readyNodes_=new ArrayList<Node>();
9       private boolean[] finished_;
10      private int[] procNextAvailible_;
11      private int[] num_of_proc_;
12      private int[] start_time_;
13      private int[] end_time_;
14      public MHscheduler(ArrayList<Node> nodes,int numProc){
15              nodes_=nodes;
16              for(int i=0;i<nodes_.size();i++){
17                      if(nodes_.get(i).getParents().size()==0){
18                              entryNodes_.add(nodes_.get(i));
19                              readyNodes_.add(nodes_.get(i));
20                      }else if(nodes_.get(i).getChildren().size()==0){
21                              exitNodes_.add(nodes_.get(i));
22                      }
23              }
24              procNextAvailible_=new int[numProc];
25              for(int i=0;i<numProc;i++){
26                      procNextAvailible_[i]=0;
27              }
28      }
29      public void createSchedule(){
30              start_time_=new int[nodes_.size()];
31              end_time_=new int[nodes_.size()];
32              num_of_proc_=new int[nodes_.size()];
33              finished_=new boolean[nodes_.size()];
34
35              for(int i=0;i<procNextAvailible_.length;i++){
36                      procNextAvailible_[i]=0;
37              }
38              for(int i=0;i<nodes_.size();i++){
39                      start_time_[i]=0;
40                      end_time_[i]=0;
41                      finished_[i]=false;
42                      nodes_.get(i).setSL(getSL(nodes_.get(i)));
43              }
44              while(readyNodes_.size()!=0){
45                      sortReadyNodes();
46                      Node cur=readyNodes_.get(0);
47                      System.out.println("Node:␣"+cur.getID());
48                      readyNodes_.remove(0);
49                      ArrayList<Connection> parents=cur.getParents();
50                      int edgeProc=procNextAvailible_.length-1;
51                      for(int i=0;i<parents.size();i++){
52                              if(num_of_proc_[nodes_.indexOf(parents.get(i).getNode())]<=
```

```
                    edgeProc){
53                      edgeProc=num_of_proc_[nodes_.indexOf(parents.get(i).
                            getNode())];
54                  }
55              }
56              int proc=0;
57              int time=Integer.MAX_VALUE;
58              for(int i=0;i<=edgeProc;i++){
59                  int est_time=procNextAvailible_[i];
60                  for(int j=0;j<parents.size();j++){
61                      int est=end_time_[nodes_.indexOf(parents.get(j).getNode
                            ())]+getRouteEstimate(i,num_of_proc_[nodes_.
                            indexOf(parents.get(j).getNode())],parents.get(j).
                            getWeight());
62                      if(est>=est_time){
63                          est_time=est;
64
65                      }
66                  }
67                  if(est_time<=time){
68                      time=est_time;
69                      proc=i;
70                  }
71              }
72              finished_[nodes_.indexOf(cur)]=true;
73              start_time_[nodes_.indexOf(cur)]=time;
74              end_time_[nodes_.indexOf(cur)]=time+cur.getWeight();
75              procNextAvailible_[proc]=time+cur.getWeight();
76              num_of_proc_[nodes_.indexOf(cur)]=proc;
77              ArrayList<Connection> children=cur.getChildren();
78
79              for(int i=0;i<children.size();i++){
80                  boolean ready=true;
81                  ArrayList<Connection> p2=children.get(i).getNode().getParents();
82                  for(int j=0;j<p2.size();j++){
83                      if(!finished_[nodes_.indexOf(p2.get(j).getNode())]){
84                          ready=false;
85                      }
86                  }
87                  if(ready==true){
88                      readyNodes_.add(children.get(i).getNode());
89                  }
90
91              }
92          }
```

```
93
94              for(int i=0;i<nodes_.size();i++){
95                      System.out.println(nodes_.get(i).getID()+" & "+num_of_proc_[i]+" &
                            "+start_time_[i]+" & "+end_time_[i]+" & "+nodes_.get(i).getSL()
                            +" \\\\ \\hline");
96              }
97
98      }
99
100     private void sortReadyNodes(){
101             ArrayList<Node> temp=new ArrayList<Node>();
102             while(readyNodes_.size()!=0){
103                     int max_priority=0;
104                     int pos=0;
105                     for(int i=0;i<readyNodes_.size();i++){
106                             if(readyNodes_.get(i).getSL()>=max_priority){
107                                     max_priority=readyNodes_.get(i).getSL();
108                                     pos=i;
109                             }
110                     }
111                     temp.add(readyNodes_.get(pos));
112                     readyNodes_.remove(pos);
113             }
114             readyNodes_=temp;
115     }
116
117     private int getSL(Node N){
118             if(exitNodes_.contains(N)==true){
119                     return N.getWeight();
120             }else{
121                     ArrayList<Connection> children_=N.getChildren();
122                     int res=0;
123                     for(int i=0;i<children_.size();i++){
124                             int temp=getSL(children_.get(i).getNode());
125                             if(temp>res)res=temp;
126                     }
127                     return res+N.getWeight();
128             }
129     }
130     public int getRouteEstimate(int start, int end,int size){
131             System.out.println("Start: "+start+"; End: "+end+"; Size: "+size+"; Length: "
                    +Math.abs(start-end)*size);
132                     return Math.abs(start-end)*size;//TODO fix
133     }
134
```

```
135         public void generateReport(){
136                 ArrayList<ArrayList<Integer>> timelines_=new ArrayList<ArrayList<Integer
                        >>();
137                 int timeLength=0;
138                 for(int i=0;i<end_time_.length;i++){
139                         if(end_time_[i]>timeLength)timeLength=end_time_[i];
140                 }
141                 for(int i=0;i<procNextAvailible_.length;i++){
142                         ArrayList<Integer> temp=new ArrayList<Integer>();
143                         timelines_.add(temp);
144                         for(int j=0;j<timeLength;j++){
145                                 temp.add(-1);
146                         }
147                 }
148                 for(int i=0;i<end_time_.length;i++){
149                         for(int j=start_time_[i];j<end_time_[i];j++){
150                                 timelines_.get(num_of_proc_[i]).set(j,nodes_.get(i).getID());
151                         }
152                 }
153                 System.out.println("&␣");
154                 for(int i=0;i<procNextAvailible_.length;i++){
155                         System.out.print("P"+i+"&");
156                 }
157                 System.out.println();
158                 for(int i=0;i<timeLength;i++){
159                         System.out.print(i);
160                         for(int j=0;j<procNextAvailible_.length;j++){
161                                 System.out.print("&"+timelines_.get(j).get(i));
162                         }
163                         System.out.println("\\\\␣\\hline");
164                 }
165         }
166
167 }
```

Listing 2: Node.java

```
1 package data;
2 import java.util.ArrayList;
3 import java.io.Serializable;
4
5 public class Node implements Serializable{
6         /**
7          *
8          */
9         private static final long serialVersionUID = -3388768794943281957L;
```

```
10          private int id_;
11          private int sl_;
12          private int weight_;
13          private ArrayList<Connection> in_=new ArrayList<Connection>();
14          private ArrayList<Connection> out_=new ArrayList<Connection>();
15          public Node(int weight,int id){
16                  weight_=weight;
17                  id_=id;
18          }
19          public void setSL(int sl){
20                  sl_=sl;
21          }
22          public void addParent(Connection c){
23                  in_.add(c);
24          }
25          public void addChild(Connection c){
26                  out_.add(c);
27          }
28          public ArrayList<Connection> getParents(){
29                  return in_;
30          }
31          public ArrayList<Connection> getChildren(){
32                  return out_;
33          }
34          public int getWeight(){
35                  return weight_;
36          }
37          public int getSL(){
38                  return sl_;
39          }
40          public int getID(){
41                  return id_;
42          }
43  }
```

Listing 3: Connection.java

```
1  package data;
2  import java.io.Serializable;
3  public class Connection implements Serializable{
4           private Node node_;
5           private int weight_;
6           public Connection(Node node,int weight){
7                   node_=node;
8                   weight_=weight;
9           }
```

```
10          public int getWeight(){
11                  return weight_;
12          }
13          public Node getNode(){
14                  return node_;
15          }
16 }
```

Listing 4: GraphConverter.java

```
 1 package data;
 2 import java.util.ArrayList;
 3 import java.util.Iterator;
 4
 5 import org.jgraph.JGraph;
 6 import org.jgraph.graph.CellView;
 7 import org.jgraph.graph.EdgeView;
 8 import org.jgraph.graph.GraphModel;
 9 import org.jgraph.graph.GraphCell;
10 import org.jgraph.graph.DefaultGraphCell;
11 import org.jgraph.graph.Edge;
12 import org.jgraph.graph.DefaultEdge;
13 import org.jgraph.graph.Port;
14
15 public class GraphConverter {
16          public static ArrayList<Node> convert(JGraph graph){
17                  GraphModel model=graph.getModel();
18                  ArrayList<Node> result=new ArrayList<Node>();
19                  ArrayList<GraphCell> cells=new ArrayList<GraphCell>();
20                  ArrayList<Edge> edges=new ArrayList<Edge>();
21                  for(int i=0;i<model.getRootCount();i++){
22                          Object temp=model.getRootAt(i);
23                          if(temp.getClass()==DefaultGraphCell.class){
24                                  cells.add((GraphCell)temp);
25                          } else if(temp.getClass()==DefaultEdge.class){
26                                  edges.add((Edge)temp);
27                          }
28                  }
29
30                  for(int i=0;i<cells.size();i++){
31                          String[] tokens=cells.get(i).toString().split("/");
32                          result.add(new Node(Integer.parseInt(tokens[1]),Integer.parseInt(tokens
                                [0])));
33                  }
34                  for(int i=0;i<edges.size();i++){
35                          GraphCell source=null;
```

```
36                          GraphCell destination=null;
37                          int weight=Integer.parseInt(edges.get(i).toString());
38                          int ready=0;
39                          for(int j=0;j<cells.size();j++){
40                                  if(((DefaultGraphCell)cells.get(j)).getChildren().contains(edges.
                                        get(i).getSource())){
41                                          source=cells.get(j);
42                                          ready++;
43                                  }
44                                  else if(((DefaultGraphCell)cells.get(j)).getChildren().contains(
                                        edges.get(i).getTarget())){
45                                          destination=cells.get(j);
46                                          ready++;
47                                  }
48                                  if(ready==2){
49                                          break;
50                                  }
51                          }
52                          if(source!=null&&destination!=null){
53                                  result.get(cells.indexOf(source)).addChild(new Connection(result
                                        .get(cells.indexOf(destination)),weight));
54                                  result.get(cells.indexOf(destination)).addParent(new Connection
                                        (result.get(cells.indexOf(source)),weight));
55                                  System.out.println("Connected␣"+result.get(cells.indexOf(source)).
                                        getID()+"␣and␣"+result.get(cells.indexOf(destination)).getID()
                                        );
56                          }
57                  }
58                  return result;
59          }
60 }
```

Listing 5: MainFrame.java

```
1 package gui;
2
3 import javax.swing.GroupLayout;
4 import javax.swing.JButton;
5 import javax.swing.JDialog;
6 import javax.swing.JFrame;
7 import javax.swing.JOptionPane;
8 import javax.swing.JPanel;
9 import javax.swing.JScrollPane;
10 import javax.swing.JTabbedPane;
11 import javax.swing.JTextField;
12 import javax.swing.event.MouseInputAdapter;
```

```
13
14
15 import data.GraphConverter;
16 import data.MHscheduler;
17 import data.Serializer;
18
19 import java.awt.Dimension;
20 import java.awt.FileDialog;
21 import java.awt.event.MouseEvent;
22 import java.io.File;
23 import data.Node;
24 import java.util.ArrayList;
25 public class MainFrame extends JFrame{
26         private static MainFrame frame_=new MainFrame();
27         private GraphEd graphEd_=new GraphEd();
28         private JButton convertButton_=new JButton("Convert graph");
29         private JButton scheduleButton_=new JButton("Start scheduling");
30         private JButton saveButton_=new JButton("Save");
31         private JButton loadButton_=new JButton("Load");
32         private JTextField procNum_=new JTextField();
33         private Serializer  serializer_ =new Serializer();
34         private ArrayList<Node> nodes_;
35         private MainFrame(){
36                 this.setSize(640,480);
37                 this.setDefaultCloseOperation(EXIT_ON_CLOSE);
38                 JTabbedPane tabs_=new JTabbedPane();
39                 tabs_.addTab("Editor", new JScrollPane(graphEd_,JScrollPane.
                       VERTICAL_SCROLLBAR_AS_NEEDED,JScrollPane.
                       HORIZONTAL_SCROLLBAR_AS_NEEDED));
40                 tabs_.addTab("Parameters",new JPanel().add(procNum_));
41                 JPanel buttonPanel=new JPanel();
42                 buttonPanel.add(convertButton_);
43                 buttonPanel.add(scheduleButton_);
44                 buttonPanel.add(saveButton_);
45                 buttonPanel.add(loadButton_);
46                 convertButton_.addMouseListener(new MouseInputAdapter(){
47                         @Override
48                         public void mousePressed(MouseEvent E){
49                                 MainFrame.getInstance().convert();
50                         }
51                 });
52                 scheduleButton_.addMouseListener(new MouseInputAdapter(){
53                         @Override
54                         public void mousePressed(MouseEvent E){
55                                 MainFrame.getInstance().startScheduling();
```

```
56                      }
57              });
58              saveButton_.addMouseListener(new MouseInputAdapter(){
59                      @Override
60                      public void mousePressed(MouseEvent E){
61                              MainFrame.getInstance().save();
62                      }
63              });
64              loadButton_.addMouseListener(new MouseInputAdapter(){
65                      @Override
66                      public void mousePressed(MouseEvent E){
67                              MainFrame.getInstance().load();
68                      }
69              });
70              GroupLayout g=new GroupLayout(this.getContentPane());
71              g.setVerticalGroup(g.createSequentialGroup()
72                              .addComponent(buttonPanel)
73                              .addComponent(tabs_));
74              g.setHorizontalGroup(g.createParallelGroup()
75                              .addComponent(buttonPanel)
76                              .addComponent(tabs_));
77              buttonPanel.setMaximumSize(new Dimension(this.getWidth(),this.getHeight()/10)
                    );
78              this.getContentPane().add(buttonPanel);
79              this.getContentPane().add(tabs_);
80              this.getContentPane().setLayout(g);
81              this.setVisible(true);
82      }
83      public static MainFrame getInstance(){
84              return frame_;
85      }
86      public void convert(){
87              nodes_=GraphConverter.convert(graphEd_.getGraph());
88      }
89      public void startScheduling(){
90              MHscheduler scheduler=new MHscheduler(nodes_,Integer.parseInt(procNum_.
                    getText()));
91              scheduler.createSchedule();
92              scheduler.generateReport();
93      }
94      public void save(){
95              File S;
96              FileDialog SDialog = new FileDialog(this);
97              SDialog.setMode(FileDialog.SAVE);
98              SDialog.setVisible(true);
```

```
 99                 if (SDialog.getFile() != null) {
100                         S = new File(SDialog.getFile());
101                         serializer_ . serialize (S, nodes_);
102                 }
103         }
104         public void load(){
105                 JOptionPane ConfirmPane = new JOptionPane(null,
106                                 JOptionPane.QUESTION_MESSAGE, JOptionPane.
                                        YES_NO_CANCEL_OPTION);
107                 ConfirmPane
108                                 .setMessage("Do␣you␣want␣to␣save␣current␣chart␣before␣loading␣
                                        new␣one?");
109                 JDialog dialog = ConfirmPane.createDialog(this, "Save");
110                 dialog.setVisible(true);
111                 try {
112                         if (Integer.parseInt(ConfirmPane.getValue().toString()) == JOptionPane.
                                CANCEL_OPTION) {
113                         } else {
114                                 if (Integer.parseInt(ConfirmPane.getValue().toString()) ==
                                        JOptionPane.YES_OPTION) {
115                                         this.save();
116                                 }
117                                 FileDialog FDialog = new FileDialog(this, "Open");
118                                 FDialog.setMode(FileDialog.LOAD);
119                                 FDialog.setVisible(true);
120                                 if (FDialog.getFile() != null) {
121                                         File F = new File(FDialog.getFile());
122                                         nodes_ = serializer_.deserialize(F);
123                                         this.repaint();
124                                 } else {
125                                         JOptionPane ErrorPane = new JOptionPane(null,
126                                                         JOptionPane.WARNING_MESSAGE,
                                                                JOptionPane.OK_OPTION);
127                                         ErrorPane.setMessage("Incorrect␣filename");
128                                         JDialog errorDialog = ErrorPane.createDialog(this,
129                                                         "Error");
130                                         errorDialog.setVisible(true);
131                                 }
132                         }
133
134                 } catch (NullPointerException NP) {
135                         JOptionPane ErrorPane = new JOptionPane(null,
136                                         JOptionPane.WARNING_MESSAGE, JOptionPane.
                                                OK_OPTION);
137                         ErrorPane.setMessage("Critical␣failure␣while␣loading.");
```

```
138                         JDialog errorDialog = ErrorPane.createDialog(this, "Error");
139                         errorDialog.setVisible(true);
140
141                     }
142                 }
143 }
```

Listing 6: GraphEd.java

```
 1 /*
 2  * @(#)GraphEd.java 3.3 23-APR-04
 3  *
 4  * Copyright (c) 2001-2004, Gaudenz Alder All rights reserved.
 5  *
 6  * This library is free software; you can redistribute it and/or
 7  * modify it under the terms of the GNU Lesser General Public
 8  * License as published by the Free Software Foundation; either
 9  * version 2.1 of the License, or (at your option) any later version.
10  *
11  * This library is distributed in the hope that it will be useful,
12  * but WITHOUT ANY WARRANTY; without even the implied warranty of
13  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
14  * Lesser General Public License for more details.
15  *
16  * You should have received a copy of the GNU Lesser General Public
17  * License along with this library; if not, write to the Free Software
18  * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
19  *
20  */
21 package gui;
22
23 import java.awt.BorderLayout;
24 import java.awt.Color;
25 import java.awt.Cursor;
26 import java.awt.Graphics;
27 import java.awt.Point;
28 import java.awt.Rectangle;
29 import java.awt.event.ActionEvent;
30 import java.awt.event.KeyEvent;
31 import java.awt.event.KeyListener;
32 import java.awt.event.MouseEvent;
33 import java.awt.geom.Point2D;
34 import java.awt.geom.Rectangle2D;
35 import java.io.Serializable;
36 import java.net.URL;
37 import java.util.Hashtable;
```

```
38 import java.util.Map;
39
40 import javax.swing.AbstractAction;
41 import javax.swing.Action;
42 import javax.swing.BorderFactory;
43 import javax.swing.ImageIcon;
44 import javax.swing.JApplet;
45 import javax.swing.JComponent;
46 import javax.swing.JPanel;
47 import javax.swing.JPopupMenu;
48 import javax.swing.JScrollPane;
49 import javax.swing.JToolBar;
50 import javax.swing.SwingUtilities;
51 import javax.swing.border.BevelBorder;
52 import javax.swing.event.UndoableEditEvent;
53
54 import org.jgraph.JGraph;
55 import org.jgraph.event.GraphSelectionEvent;
56 import org.jgraph.event.GraphSelectionListener;
57 import org.jgraph.graph.AbstractCellView;
58 import org.jgraph.graph.BasicMarqueeHandler;
59 import org.jgraph.graph.CellHandle;
60 import org.jgraph.graph.CellView;
61 import org.jgraph.graph.DefaultCellViewFactory;
62 import org.jgraph.graph.DefaultEdge;
63 import org.jgraph.graph.DefaultGraphCell;
64 import org.jgraph.graph.DefaultGraphCellEditor;
65 import org.jgraph.graph.DefaultGraphModel;
66 import org.jgraph.graph.Edge;
67 import org.jgraph.graph.EdgeRenderer;
68 import org.jgraph.graph.EdgeView;
69 import org.jgraph.graph.GraphConstants;
70 import org.jgraph.graph.GraphContext;
71 import org.jgraph.graph.GraphLayoutCache;
72 import org.jgraph.graph.GraphModel;
73 import org.jgraph.graph.GraphUndoManager;
74 import org.jgraph.graph.Port;
75 import org.jgraph.graph.PortRenderer;
76 import org.jgraph.graph.PortView;
77 import org.jgraph.graph.VertexRenderer;
78 import org.jgraph.graph.VertexView;
79
80 public class GraphEd extends JApplet implements GraphSelectionListener,
81                         KeyListener, Serializable  {
82
```

```
83
84          /**
85           *
86           */
87          private static final long serialVersionUID = 6109865275007699695L;
88
89          // JGraph instance
90          protected JGraph graph;
91
92          // Undo Manager
93          protected GraphUndoManager undoManager;
94
95          // Actions which Change State
96          protected Action undo, redo, remove, group, ungroup, tofront, toback, cut,
97                          copy, paste;
98
99          // cell count that gets put in cell label
100         protected int cellCount = 0;
101
102
103         // Construct an Editor Panel
104         public GraphEd() {
105                 // Construct the Graph
106                 graph = createGraph();
107                 // Use a Custom Marquee Handler
108                 graph.setMarqueeHandler(createMarqueeHandler());
109                 // Construct Command History
110                 //
111                 // Create a GraphUndoManager which also Updates the ToolBar
112                 undoManager = new GraphUndoManager() {
113                         // Override Superclass
114                         public void undoableEditHappened(UndoableEditEvent e) {
115                                 // First Invoke Superclass
116                                 super.undoableEditHappened(e);
117                                 // Then Update Undo/Redo Buttons
118                                 updateHistoryButtons();
119                         }
120                 };
121                 populateContentPane();
122
123                  installListeners (graph);
124         }
125
126         public void destroy() {
127                 super.destroy();
```

```
128             PortView.renderer = new PortRenderer();
129             EdgeView.renderer = new EdgeRenderer();
130             AbstractCellView.cellEditor = new DefaultGraphCellEditor();
131             VertexView.renderer = new VertexRenderer();
132         }
133
134
135         // Hook for subclassers
136         protected void populateContentPane() {
137             // Use Border Layout
138             getContentPane().setLayout(new BorderLayout());
139             // Add a ToolBar
140             getContentPane().add(createToolBar(), BorderLayout.NORTH);
141             // Add the Graph as Center Component
142             getContentPane().add(new JScrollPane(graph), BorderLayout.CENTER);
143         }
144
145         // Hook for subclassers
146         protected JGraph createGraph() {
147             JGraph graph = new MyGraph(new MyModel());
148             graph.getGraphLayoutCache().setFactory(new DefaultCellViewFactory() {
149
150                 // Override Superclass Method to Return Custom EdgeView
151                 protected EdgeView createEdgeView(Object cell) {
152
153                     // Return Custom EdgeView
154                     return new EdgeView(cell) {
155
156                         /**
157                          * Returns a cell handle for the view.
158                          */
159                         public CellHandle getHandle(GraphContext context) {
160                             return new MyEdgeHandle(this, context);
161                         }
162
163                     };
164                 }
165             });
166             return graph;
167         }
168
169         // Hook for subclassers
170         protected void installListeners(JGraph graph) {
171             // Add Listeners to Graph
172             //
```

```
173             // Register UndoManager with the Model
174             graph.getModel().addUndoableEditListener(undoManager);
175             // Update ToolBar based on Selection Changes
176             graph.getSelectionModel().addGraphSelectionListener(this);
177             // Listen for Delete Keystroke when the Graph has Focus
178             graph.addKeyListener(this);
179         }
180
181         // Hook for subclassers
182         protected void uninstallListeners(JGraph graph) {
183             graph.getModel().removeUndoableEditListener(undoManager);
184             graph.getSelectionModel().removeGraphSelectionListener(this);
185             graph.removeKeyListener(this);
186         }
187
188         // Hook for subclassers
189         protected BasicMarqueeHandler createMarqueeHandler() {
190             return new MyMarqueeHandler();
191         }
192
193         // Insert a new Vertex at point
194         public void insert(Point2D point) {
195             // Construct Vertex with no Label
196             DefaultGraphCell vertex = createDefaultGraphCell();
197             // Create a Map that holds the attributes for the Vertex
198             vertex.getAttributes().applyMap(createCellAttributes(point));
199             // Insert the Vertex (including child port and attributes)
200             graph.getGraphLayoutCache().insert(vertex);
201         }
202
203         // Hook for subclassers
204         public Map createCellAttributes(Point2D point) {
205             Map map = new Hashtable();
206             // Snap the Point to the Grid
207             if (graph != null) {
208                 point = graph.snap((Point2D) point.clone());
209             } else {
210                 point = (Point2D) point.clone();
211             }
212             // Add a Bounds Attribute to the Map
213             GraphConstants.setBounds(map, new Rectangle2D.Double(point.getX(),
214                     point.getY(), 0, 0));
215             // Make sure the cell is resized on insert
216             GraphConstants.setResize(map, true);
217             // Add a nice looking gradient background
```

```
218             GraphConstants.setGradientColor(map, Color.blue);
219             // Add a Border Color Attribute to the Map
220             GraphConstants.setBorderColor(map, Color.black);
221             // Add a White Background
222             GraphConstants.setBackground(map, Color.white);
223             // Make Vertex Opaque
224             GraphConstants.setOpaque(map, true);
225             return map;
226         }
227
228         // Hook for subclassers
229         protected DefaultGraphCell createDefaultGraphCell() {
230             DefaultGraphCell cell = new DefaultGraphCell("Cell␣"
231                     + new Integer(cellCount++));
232             // Add one Floating Port
233             cell.addPort();
234             return cell;
235         }
236
237         // Insert a new Edge between source and target
238         public void connect(Port source, Port target) {
239             // Construct Edge with no label
240             DefaultEdge edge = createDefaultEdge();
241             if (graph.getModel().acceptsSource(edge, source)
242                     && graph.getModel().acceptsTarget(edge, target)) {
243                 // Create a Map thath holds the attributes for the edge
244                 edge.getAttributes().applyMap(createEdgeAttributes());
245                 // Insert the Edge and its Attributes
246                 graph.getGraphLayoutCache().insertEdge(edge, source, target);
247             }
248         }
249
250         // Hook for subclassers
251         protected DefaultEdge createDefaultEdge() {
252             return new DefaultEdge();
253         }
254
255         // Hook for subclassers
256         public Map createEdgeAttributes() {
257             Map map = new Hashtable();
258             // Add a Line End Attribute
259             GraphConstants.setLineEnd(map, GraphConstants.ARROW_SIMPLE);
260             // Add a label along edge attribute
261             GraphConstants.setLabelAlongEdge(map, true);
262             return map;
```

```
263         }
264
265         // Create a Group that Contains the Cells
266         public void group(Object[] cells) {
267                 // Order Cells by Model Layering
268                 cells = graph.order(cells);
269                 // If Any Cells in View
270                 if (cells != null && cells.length > 0) {
271                         DefaultGraphCell group = createGroupCell();
272                         // Insert into model
273                         graph.getGraphLayoutCache().insertGroup(group, cells);
274                 }
275         }
276
277         // Hook for subclassers
278         protected DefaultGraphCell createGroupCell() {
279                 return new DefaultGraphCell();
280         }
281
282         // Returns the total number of cells in a graph
283         protected int getCellCount(JGraph graph) {
284                 Object[] cells = graph.getDescendants(graph.getRoots());
285                 return cells.length;
286         }
287
288         // Ungroup the Groups in Cells and Select the Children
289         public void ungroup(Object[] cells) {
290                 graph.getGraphLayoutCache().ungroup(cells);
291         }
292
293         // Determines if a Cell is a Group
294         public boolean isGroup(Object cell) {
295                 // Map the Cell to its View
296                 CellView view = graph.getGraphLayoutCache().getMapping(cell, false);
297                 if (view != null)
298                         return !view.isLeaf();
299                 return false;
300         }
301
302         // Brings the Specified Cells to Front
303         public void toFront(Object[] c) {
304                 graph.getGraphLayoutCache().toFront(c);
305         }
306
307         // Sends the Specified Cells to Back
```

```
308         public void toBack(Object[] c) {
309                 graph.getGraphLayoutCache().toBack(c);
310         }
311
312         // Undo the last Change to the Model or the View
313         public void undo() {
314                 try {
315                         undoManager.undo(graph.getGraphLayoutCache());
316                 } catch (Exception ex) {
317                         System.err.println(ex);
318                 } finally {
319                         updateHistoryButtons();
320                 }
321         }
322
323         // Redo the last Change to the Model or the View
324         public void redo() {
325                 try {
326                         undoManager.redo(graph.getGraphLayoutCache());
327                 } catch (Exception ex) {
328                         System.err.println(ex);
329                 } finally {
330                         updateHistoryButtons();
331                 }
332         }
333
334         // Update Undo/Redo Button State based on Undo Manager
335         protected void updateHistoryButtons() {
336                 // The View Argument Defines the Context
337                 undo.setEnabled(undoManager.canUndo(graph.getGraphLayoutCache()));
338                 redo.setEnabled(undoManager.canRedo(graph.getGraphLayoutCache()));
339         }
340
341         //
342         // Listeners
343         //
344
345         // From GraphSelectionListener Interface
346         public void valueChanged(GraphSelectionEvent e) {
347                 // Group Button only Enabled if more than One Cell Selected
348                 group.setEnabled(graph.getSelectionCount() > 1);
349                 // Update Button States based on Current Selection
350                 boolean enabled = !graph.isSelectionEmpty();
351                 remove.setEnabled(enabled);
352                 ungroup.setEnabled(enabled);
```

```
353                 tofront.setEnabled(enabled);
354                 toback.setEnabled(enabled);
355                 copy.setEnabled(enabled);
356                 cut.setEnabled(enabled);
357         }
358
359         //
360         // KeyListener for Delete KeyStroke
361         //
362         public void keyReleased(KeyEvent e) {
363         }
364
365         public void keyTyped(KeyEvent e) {
366         }
367
368         public void keyPressed(KeyEvent e) {
369                 // Listen for Delete Key Press
370                 if (e.getKeyCode() == KeyEvent.VK_DELETE)
371                         // Execute Remove Action on Delete Key Press
372                         remove.actionPerformed(null);
373         }
374
375         //
376         // Custom Graph
377         //
378
379         // Defines a Graph that uses the Shift-Button (Instead of the Right
380         // Mouse Button, which is Default) to add/remove point to/from an edge.
381         public static class MyGraph extends JGraph implements Serializable{
382
383                 /**
384                  *
385                  */
386                 private static final long serialVersionUID = -1138175044509316933L;
387
388                 // Construct the Graph using the Model as its Data Source
389                 public MyGraph(GraphModel model) {
390                         this(model, null);
391                 }
392
393                 // Construct the Graph using the Model as its Data Source
394                 public MyGraph(GraphModel model, GraphLayoutCache cache) {
395                         super(model, cache);
396                         // Make Ports Visible by Default
397                         setPortsVisible(true);
```

```
398                     // Use the Grid (but don't make it Visible)
399                     setGridEnabled(true);
400                     // Set the Grid Size to 10 Pixel
401                     setGridSize(6);
402                     // Set the Tolerance to 2 Pixel
403                     setTolerance(2);
404                     // Accept edits if click on background
405                     setInvokesStopCellEditing(true);
406                     // Allows control-drag
407                     setCloneable(true);
408                     // Jump to default port on connect
409                     setJumpToDefaultPort(true);
410             }
411
412     }
413
414     //
415     // Custom Edge Handle
416     //
417
418     // Defines a EdgeHandle that uses the Shift-Button (Instead of the Right
419     // Mouse Button, which is Default) to add/remove point to/from an edge.
420     public static class MyEdgeHandle extends EdgeView.EdgeHandle implements
            Serializable{
421
422             /**
423              *
424              */
425             private static final long serialVersionUID = -1156890233926707112L;
426
427             /**
428              * @param edge
429              * @param ctx
430              */
431             public MyEdgeHandle(EdgeView edge, GraphContext ctx) {
432                     super(edge, ctx);
433             }
434
435             // Override Superclass Method
436             public boolean isAddPointEvent(MouseEvent event) {
437                     // Points are Added using Shift-Click
438                     return event.isShiftDown();
439             }
440
441             // Override Superclass Method
```

```
442             public boolean isRemovePointEvent(MouseEvent event) {
443                     // Points are Removed using Shift-Click
444                     return event.isShiftDown();
445             }
446
447     }
448
449     //
450     // Custom Model
451     //
452
453     // A Custom Model that does not allow Self-References
454     public static class MyModel extends DefaultGraphModel implements Serializable{
455             /**
456              *
457              */
458             private static final long serialVersionUID = -4044961398639255380L;
459
460             // Override Superclass Method
461             public boolean acceptsSource(Object edge, Object port) {
462                     // Source only Valid if not Equal Target
463                     return (((Edge) edge).getTarget() != port);
464             }
465
466             // Override Superclass Method
467             public boolean acceptsTarget(Object edge, Object port) {
468                     // Target only Valid if not Equal Source
469                     return (((Edge) edge).getSource() != port);
470             }
471     }
472
473     //
474     // Custom MarqueeHandler
475
476     // MarqueeHandler that Connects Vertices and Displays PopupMenus
477     public class MyMarqueeHandler extends BasicMarqueeHandler implements Serializable
        {
478
479             // Holds the Start and the Current Point
480             protected Point2D start, current;
481
482             // Holds the First and the Current Port
483             protected PortView port, firstPort;
484
485             /**
```

```
486                 * Component that is used for highlighting  cells  if
487                 * the graph does not allow XOR painting.
488                 */
489                protected JComponent highlight = new JPanel();
490
491                public MyMarqueeHandler() {
492                        // Configures the panel for  highlighting  ports
493                        highlight  = createHighlight();
494                }
495
496                /**
497                 * Creates the component that is used for highlighting  cells  if
498                 * the graph does not allow XOR painting.
499                 */
500                protected JComponent createHighlight() {
501                        JPanel panel = new JPanel();
502                        panel.setBorder(BorderFactory.createBevelBorder(BevelBorder.RAISED));
503                        panel. setVisible (false);
504                        panel.setOpaque(false);
505
506                        return panel;
507                }
508
509                // Override to Gain Control (for PopupMenu and ConnectMode)
510                public boolean isForceMarqueeEvent(MouseEvent e) {
511                        if (e.isShiftDown())
512                                return false;
513                        // If  Right Mouse Button we want to Display the PopupMenu
514                        if (SwingUtilities.isRightMouseButton(e))
515                                // Return Immediately
516                                return true;
517                        // Find and Remember Port
518                        port = getSourcePortAt(e.getPoint());
519                        // If  Port Found and in ConnectMode (=Ports Visible)
520                        if (port != null && graph.isPortsVisible())
521                                return true;
522                        // Else Call Superclass
523                        return super.isForceMarqueeEvent(e);
524                }
525
526                // Display PopupMenu or Remember Start Location and First Port
527                public void mousePressed(final MouseEvent e) {
528                        // If  Right Mouse Button
529                        if (SwingUtilities.isRightMouseButton(e)) {
530                                // Find Cell in  Model Coordinates
```

```
531                         Object cell = graph.getFirstCellForLocation(e.getX(), e.getY());
532                         // Create PopupMenu for the Cell
533                         JPopupMenu menu = createPopupMenu(e.getPoint(), cell);
534                         // Display PopupMenu
535                         menu.show(graph, e.getX(), e.getY());
536                         // Else if in ConnectMode and Remembered Port is Valid
537                 } else if (port != null && graph.isPortsVisible()) {
538                         // Remember Start Location
539                         start = graph.toScreen(port.getLocation());
540                         // Remember First Port
541                         firstPort = port;
542                 } else {
543                         // Call Superclass
544                         super.mousePressed(e);
545                 }
546         }
547
548         // Find Port under Mouse and Repaint Connector
549         public void mouseDragged(MouseEvent e) {
550                 // If remembered Start Point is Valid
551                 if (start != null) {
552                         // Fetch Graphics from Graph
553                         Graphics g = graph.getGraphics();
554                         // Reset Remembered Port
555                         PortView newPort = getTargetPortAt(e.getPoint());
556                         // Do not flicker (repaint only on real changes)
557                         if (newPort == null || newPort != port) {
558                                 // Xor-Paint the old Connector (Hide old Connector)
559                                 paintConnector(Color.black, graph.getBackground(), g);
560                                 // If Port was found then Point to Port Location
561                                 port = newPort;
562                                 if (port != null)
563                                         current = graph.toScreen(port.getLocation());
564                                 // Else If no Port was found then Point to Mouse
                                        Location
565                                 else
566                                         current = graph.snap(e.getPoint());
567                                 // Xor-Paint the new Connector
568                                 paintConnector(graph.getBackground(), Color.black, g);
569                         }
570                 }
571                 // Call Superclass
572                 super.mouseDragged(e);
573         }
574
```

```
575             public PortView getSourcePortAt(Point2D point) {
576                     // Disable jumping
577                     graph.setJumpToDefaultPort(false);
578                     PortView result;
579                     try {
580                             // Find a Port View in Model Coordinates and Remember
581                             result = graph.getPortViewAt(point.getX(), point.getY());
582                     } finally {
583                             graph.setJumpToDefaultPort(true);
584                     }
585                     return result;
586             }
587
588             // Find a Cell at point and Return its first Port as a PortView
589             protected PortView getTargetPortAt(Point2D point) {
590                     // Find a Port View in Model Coordinates and Remember
591                     return graph.getPortViewAt(point.getX(), point.getY());
592             }
593
594             // Connect the First Port and the Current Port in the Graph or Repaint
595             public void mouseReleased(MouseEvent e) {
596                     highlight(graph, null);
597
598                     // If Valid Event, Current and First Port
599                     if (e != null && port != null && firstPort != null
600                                     && firstPort != port) {
601                             // Then Establish Connection
602                             connect((Port) firstPort.getCell(), (Port) port.getCell());
603                             e.consume();
604                             // Else Repaint the Graph
605                     } else
606                             graph.repaint();
607                     // Reset Global Vars
608                     firstPort = port = null;
609                     start = current = null;
610                     // Call Superclass
611                     super.mouseReleased(e);
612             }
613
614             // Show Special Cursor if Over Port
615             public void mouseMoved(MouseEvent e) {
616                     // Check Mode and Find Port
617                     if (e != null && getSourcePortAt(e.getPoint()) != null
618                                     && graph.isPortsVisible()) {
619                             // Set Cusor on Graph (Automatically Reset)
```

```
620                         graph.setCursor(new Cursor(Cursor.HAND_CURSOR));
621                         // Consume Event
622                         // Note: This is to signal the BasicGraphUI's
623                         // MouseHandle to stop further event processing.
624                         e.consume();
625                 } else
626                         // Call Superclass
627                         super.mouseMoved(e);
628         }
629
630         // Use Xor-Mode on Graphics to Paint Connector
631         protected void paintConnector(Color fg, Color bg, Graphics g) {
632                 if (graph.isXorEnabled()) {
633                         // Set Foreground
634                         g.setColor(fg);
635                         // Set Xor-Mode Color
636                         g.setXORMode(bg);
637                         // Highlight the Current Port
638                         paintPort(graph.getGraphics());
639
640                         drawConnectorLine(g);
641                 } else {
642                         Rectangle dirty = new Rectangle((int) start.getX(), (int) start.
                                getY(), 1, 1);
643
644                         if (current != null) {
645                                 dirty.add(current);
646                         }
647
648                         dirty.grow(1, 1);
649
650                         graph.repaint(dirty);
651                         highlight(graph, port);
652                 }
653         }
654
655         // Overrides parent method to paint connector if
656         // XOR painting is disabled in the graph
657         public void paint(JGraph graph, Graphics g)
658         {
659                 super.paint(graph, g);
660
661                 if (!graph.isXorEnabled())
662                 {
663                         g.setColor(Color.black);
```

```
664                     drawConnectorLine(g);
665             }
666     }
667
668     protected void drawConnectorLine(Graphics g) {
669             if ( firstPort  != null && start != null && current != null) {
670                     // Then Draw A Line From Start to Current Point
671                     g.drawLine((int) start.getX(), (int) start.getY(),
672                                     (int) current.getX(), (int) current.getY());
673             }
674     }
675
676     // Use the Preview Flag to Draw a Highlighted Port
677     protected void paintPort(Graphics g) {
678             // If Current Port is Valid
679             if (port != null) {
680                     // If Not Floating Port...
681                     boolean o = (GraphConstants.getOffset(port.getAllAttributes())
                            != null);
682                     //  ... Then use Parent's Bounds
683                     Rectangle2D r = (o) ? port.getBounds() : port.getParentView()
684                                     .getBounds();
685                     // Scale from Model to Screen
686                     r = graph.toScreen((Rectangle2D) r.clone());
687                     // Add Space For the Highlight Border
688                     r.setFrame(r.getX() - 3, r.getY() - 3, r.getWidth() + 6, r
689                                     .getHeight() + 6);
690                     // Paint Port in Preview (=Highlight) Mode
691                     graph.getUI().paintCell(g, port, r, true);
692             }
693     }
694
695     /**
696      * Highlights the given cell view or removes the highlight if
697      * no cell view is specified.
698      *
699      * @param graph
700      * @param cellView
701      */
702     protected void highlight(JGraph graph, CellView cellView)
703     {
704             if (cellView != null)
705             {
706                     highlight.setBounds(getHighlightBounds(graph, cellView));
707
```

```
708                             if (highlight.getParent() == null)
709                             {
710                                     graph.add(highlight);
711                                     highlight.setVisible(true);
712                             }
713                     }
714                     else
715                     {
716                             if (highlight.getParent() != null)
717                             {
718                                     highlight.setVisible(false);
719                                     highlight.getParent().remove(highlight);
720                             }
721                     }
722             }
723
724             /**
725              * Returns the bounds to be used to highlight the given cell view.
726              *
727              * @param graph
728              * @param cellView
729              * @return
730              */
731             protected Rectangle getHighlightBounds(JGraph graph, CellView cellView)
732             {
733                     boolean offset = (GraphConstants.getOffset(cellView.getAllAttributes())
                            != null);
734                     Rectangle2D r = (offset) ? cellView.getBounds() : cellView
735                                     .getParentView().getBounds();
736                     r = graph.toScreen((Rectangle2D) r.clone());
737                     int s = 3;
738
739                     return new Rectangle((int) (r.getX() - s), (int) (r.getY() - s),
740                                     (int) (r.getWidth() + 2 * s), (int) (r.getHeight() + 2 * s
                                        ));
741             }
742
743
744     } // End of Editor.MyMarqueeHandler
745
746     //
747     //
748     //
749
750     //
```

```
751         // PopupMenu and ToolBar
752         //
753
754         //
755         //
756         //
757
758         //
759         // PopupMenu
760         //
761         public JPopupMenu createPopupMenu(final Point pt, final Object cell) {
762                 JPopupMenu menu = new JPopupMenu();
763                 if (cell != null) {
764                         // Edit
765                         menu.add(new AbstractAction("Edit") {
766                                 public void actionPerformed(ActionEvent e) {
767                                         graph.startEditingAtCell(cell);
768                                 }
769                         });
770                 }
771                 // Remove
772                 if (!graph.isSelectionEmpty()) {
773                         menu.addSeparator();
774                         menu.add(new AbstractAction("Remove") {
775                                 public void actionPerformed(ActionEvent e) {
776                                         remove.actionPerformed(e);
777                                 }
778                         });
779                 }
780                 menu.addSeparator();
781                 // Insert
782                 menu.add(new AbstractAction("Insert") {
783                         public void actionPerformed(ActionEvent ev) {
784                                 insert(pt);
785                         }
786                 });
787                 return menu;
788         }
789
790         //
791         // ToolBar
792         //
793         public JToolBar createToolBar() {
794                 JToolBar toolbar = new JToolBar();
795                 toolbar.setFloatable(false);
```

```
796
797             // Insert
798             URL insertUrl = getClass().getClassLoader().getResource(
799                             "resources/insert.gif");
800             ImageIcon insertIcon = new ImageIcon(insertUrl);
801             toolbar.add(new AbstractAction("", insertIcon) {
802                     public void actionPerformed(ActionEvent e) {
803                             insert(new Point(10, 10));
804                     }
805             });
806
807             // Toggle Connect Mode
808             URL connectUrl = getClass().getClassLoader().getResource(
809                             "resources/connecton.gif");
810             ImageIcon connectIcon = new ImageIcon(connectUrl);
811             toolbar.add(new AbstractAction("", connectIcon) {
812                     public void actionPerformed(ActionEvent e) {
813                             graph.setPortsVisible(!graph.isPortsVisible());
814                             URL connectUrl;
815                             if (graph.isPortsVisible())
816                                     connectUrl = getClass().getClassLoader().getResource(
817                                                     "resources/connecton.gif");
818                             else
819                                     connectUrl = getClass().getClassLoader().getResource(
820                                                     "resources/connectoff.gif");
821                             ImageIcon connectIcon = new ImageIcon(connectUrl);
822                             putValue(SMALL_ICON, connectIcon);
823                     }
824             });
825
826             // Undo
827             toolbar.addSeparator();
828             URL undoUrl = getClass().getClassLoader().getResource(
829                             "resources/undo.gif");
830             ImageIcon undoIcon = new ImageIcon(undoUrl);
831             undo = new AbstractAction("", undoIcon) {
832                     public void actionPerformed(ActionEvent e) {
833                             undo();
834                     }
835             };
836             undo.setEnabled(false);
837             toolbar.add(undo);
838
839             // Redo
840             URL redoUrl = getClass().getClassLoader().getResource(
```

```
841                         "resources/redo.gif");
842             ImageIcon redoIcon = new ImageIcon(redoUrl);
843             redo = new AbstractAction("", redoIcon) {
844                     public void actionPerformed(ActionEvent e) {
845                             redo();
846                     }
847             };
848             redo.setEnabled(false);
849             toolbar.add(redo);
850
851             //
852             // Edit Block
853             //
854             toolbar.addSeparator();
855             Action action;
856             URL url;
857
858             // Copy
859             action = javax.swing.TransferHandler.getCopyAction();
860             url = getClass().getClassLoader().getResource(
861                             "resources/copy.gif");
862             toolbar.add(copy = new EventRedirector(action, new ImageIcon(url)));
863
864             // Paste
865             action = javax.swing.TransferHandler.getPasteAction();
866             url = getClass().getClassLoader().getResource(
867                             "resources/paste.gif");
868             toolbar.add(paste = new EventRedirector(action, new ImageIcon(url)));
869
870             // Cut
871             action = javax.swing.TransferHandler.getCutAction();
872             url = getClass().getClassLoader().getResource(
873                             "resources/cut.gif");
874             toolbar.add(cut = new EventRedirector(action, new ImageIcon(url)));
875
876             // Remove
877             URL removeUrl = getClass().getClassLoader().getResource(
878                             "resources/delete.gif");
879             ImageIcon removeIcon = new ImageIcon(removeUrl);
880             remove = new AbstractAction("", removeIcon) {
881                     public void actionPerformed(ActionEvent e) {
882                             if (!graph.isSelectionEmpty()) {
883                                     Object[] cells = graph.getSelectionCells();
884                                     cells = graph.getDescendants(cells);
885                                     graph.getModel().remove(cells);
```

```
886                         }
887                 }
888         };
889         remove.setEnabled(false);
890         toolbar.add(remove);
891
892         // To Front
893         toolbar.addSeparator();
894         URL toFrontUrl = getClass().getClassLoader().getResource(
895                         "resources/tofront.gif");
896         ImageIcon toFrontIcon = new ImageIcon(toFrontUrl);
897         tofront = new AbstractAction("", toFrontIcon) {
898                 public void actionPerformed(ActionEvent e) {
899                         if (!graph.isSelectionEmpty())
900                                 toFront(graph.getSelectionCells());
901                 }
902         };
903         tofront.setEnabled(false);
904         toolbar.add(tofront);
905
906         // To Back
907         URL toBackUrl = getClass().getClassLoader().getResource(
908                         "resources/toback.gif");
909         ImageIcon toBackIcon = new ImageIcon(toBackUrl);
910         toback = new AbstractAction("", toBackIcon) {
911                 public void actionPerformed(ActionEvent e) {
912                         if (!graph.isSelectionEmpty())
913                                 toBack(graph.getSelectionCells());
914                 }
915         };
916         toback.setEnabled(false);
917         toolbar.add(toback);
918
919         // Zoom Std
920         toolbar.addSeparator();
921         URL zoomUrl = getClass().getClassLoader().getResource(
922                         "resources/zoom.gif");
923         ImageIcon zoomIcon = new ImageIcon(zoomUrl);
924         toolbar.add(new AbstractAction("", zoomIcon) {
925                 public void actionPerformed(ActionEvent e) {
926                         graph.setScale(1.0);
927                 }
928         });
929         // Zoom In
930         URL zoomInUrl = getClass().getClassLoader().getResource(
```

```
931                         "resources/zoomin.gif");
932             ImageIcon zoomInIcon = new ImageIcon(zoomInUrl);
933             toolbar.add(new AbstractAction("", zoomInIcon) {
934                     public void actionPerformed(ActionEvent e) {
935                         graph.setScale(2 * graph.getScale());
936                     }
937             });
938             // Zoom Out
939             URL zoomOutUrl = getClass().getClassLoader().getResource(
940                         "resources/zoomout.gif");
941             ImageIcon zoomOutIcon = new ImageIcon(zoomOutUrl);
942             toolbar.add(new AbstractAction("", zoomOutIcon) {
943                     public void actionPerformed(ActionEvent e) {
944                         graph.setScale(graph.getScale() / 2);
945                     }
946             });
947
948             // Group
949             toolbar.addSeparator();
950             URL groupUrl = getClass().getClassLoader().getResource(
951                         "resources/group.gif");
952             ImageIcon groupIcon = new ImageIcon(groupUrl);
953             group = new AbstractAction("", groupIcon) {
954                     public void actionPerformed(ActionEvent e) {
955                         group(graph.getSelectionCells());
956                     }
957             };
958             group.setEnabled(false);
959             toolbar.add(group);
960
961             // Ungroup
962             URL ungroupUrl = getClass().getClassLoader().getResource(
963                         "resources/ungroup.gif");
964             ImageIcon ungroupIcon = new ImageIcon(ungroupUrl);
965             ungroup = new AbstractAction("", ungroupIcon) {
966                     public void actionPerformed(ActionEvent e) {
967                         ungroup(graph.getSelectionCells());
968                     }
969             };
970             ungroup.setEnabled(false);
971             toolbar.add(ungroup);
972
973             return toolbar;
974         }
975
```

```
976         /**
977          * @return Returns the graph.
978          */
979         public JGraph getGraph() {
980                 return graph;
981         }
982
983
984         /**
985          * @param graph
986          *                The graph to set.
987          */
988         public void setGraph(JGraph graph) {
989                 this.graph = graph;
990         }
991
992         // This will change the source of the actionevent to graph.
993         public class EventRedirector extends AbstractAction implements Serializable{
994
995                 protected Action action;
996
997                 // Construct the "Wrapper" Action
998                 public EventRedirector(Action a, ImageIcon icon) {
999                         super("", icon);
1000                        this.action = a;
1001                }
1002
1003                // Redirect the Actionevent
1004                public void actionPerformed(ActionEvent e) {
1005                        e = new ActionEvent(graph, e.getID(), e.getActionCommand(), e
1006                                        .getModifiers());
1007                        action.actionPerformed(e);
1008                }
1009        }
1010
1011
1012        /**
1013         *
1014         * @return a String representing the version of this application
1015         */
1016        protected String getVersion() {
1017                return JGraph.VERSION;
1018        }
1019
1020        /**
```

```
1021            * @return Returns the redo.
1022            */
1023           public Action getRedo() {
1024                   return redo;
1025           }
1026
1027           /**
1028            * @param redo
1029            *                 The redo to set.
1030            */
1031           public void setRedo(Action redo) {
1032                   this.redo = redo;
1033           }
1034
1035           /**
1036            * @return Returns the undo.
1037            */
1038           public Action getUndo() {
1039                   return undo;
1040           }
1041
1042           /**
1043            * @param undo
1044            *                 The undo to set.
1045            */
1046           public void setUndo(Action undo) {
1047                   this.undo = undo;
1048           }
1049
1050 }
```

Література

1. Конспект лекцій з дисципліни "Операційні системи".
2. Andrew S. Tanenbaum: Dristributed Systems: Principles and Paradigms 2nd Edition. Prentice Hall 2007. ISBN 0132392275.
3. Yu-Kwong Kwok: Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors. 1998
4. Yang, T: Scheduling parallel tasks on an unbounded number of processors 1994.
5. Sarkar,V: Partitioning and Scheduling Parallel Programs for Multiprocessors 1989.