Interactive graph visualization system and its application in network management

Mikhail Kryshen
kryshen@cs.petrsu.ru

Petrozavodsk State University
Department of Computer Science

Annual International Workshop
on Advances in Methods
of Information and Communication Technology

13.05.2015

Problem Domain

- Visualization of ICT infrastructure:
  - networking research and management,
  - testing of network topology discovery tools,
  - spatial and organizational mapping of network devices.
- Interactive graph visualization:
  - complex interactive vertices:
    - composable visual elements,
    - trigger changes to the graph and parameters of visualization;
  - dynamic graph layout.
- Different visual representations of the same data.
- Programmable interactive user interface.

Visualization Method

Source graph

Problem domain data

Mapping rules

Domain-specific language based on Clojure (LISP) using library of composable visual objects

Graph of interactive visual objects

Graph layout

Combination of force-based layout and a fast algorithm for predictable placement of new vertices

Interactive visualization of the graph

Change the mapping rules and update the graph in response to user actions

Components of the Visualization System

Generic

- Indyvon (**In**teractive **Dy**namic **V**isualizati**on**):
  - lightweight stateless context-dependent interactive visual objects.

- Pegla (**Pe**rsistent **G**raph **La**yout):
  - immutable persistent data structures,
  - graph layout algorithms.

ICT infrastructure

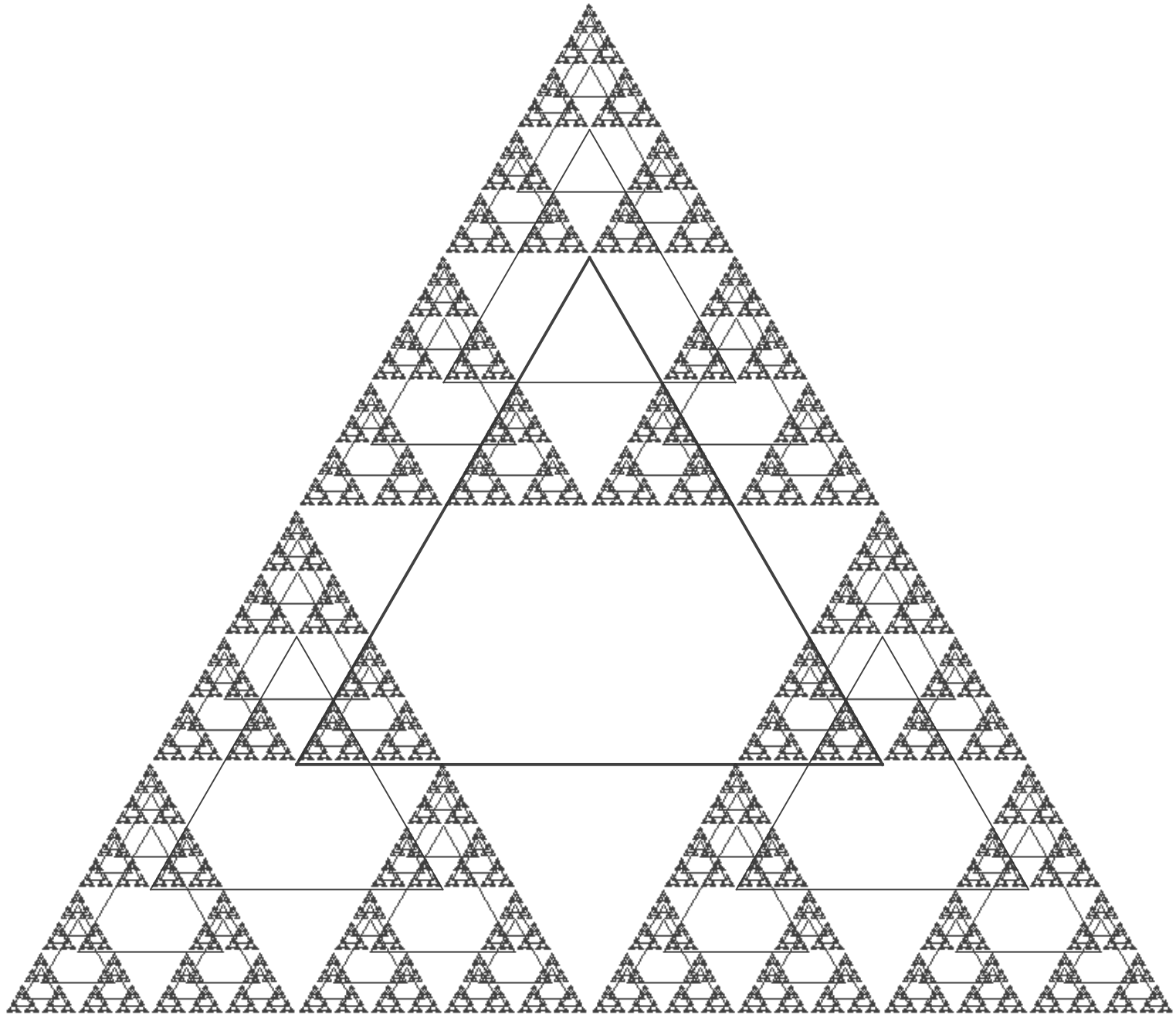
- SON (**S**patial, **O**rganizational, **N**etwork):
  - ICT infrastructure model,
  - database.

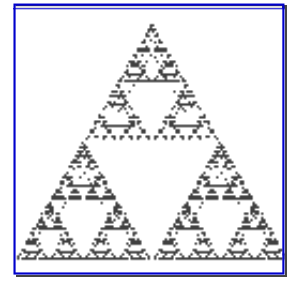
- ICT infrastructure graph visualizer:
  - visual objects for representing the ICT infrastructure elements,
  - mapping source SON graph to graph of visual objects.

Abstractions

- Simple abstract data types:
  - interactive visual object,
  - graph,
  - graph layout.
- Implemented as immutable data structures:
  - based on persistent data structures provided by Clojure,
  - support functional programming,
  - reduce mutable state that needs to be synchronized.
- Composability:

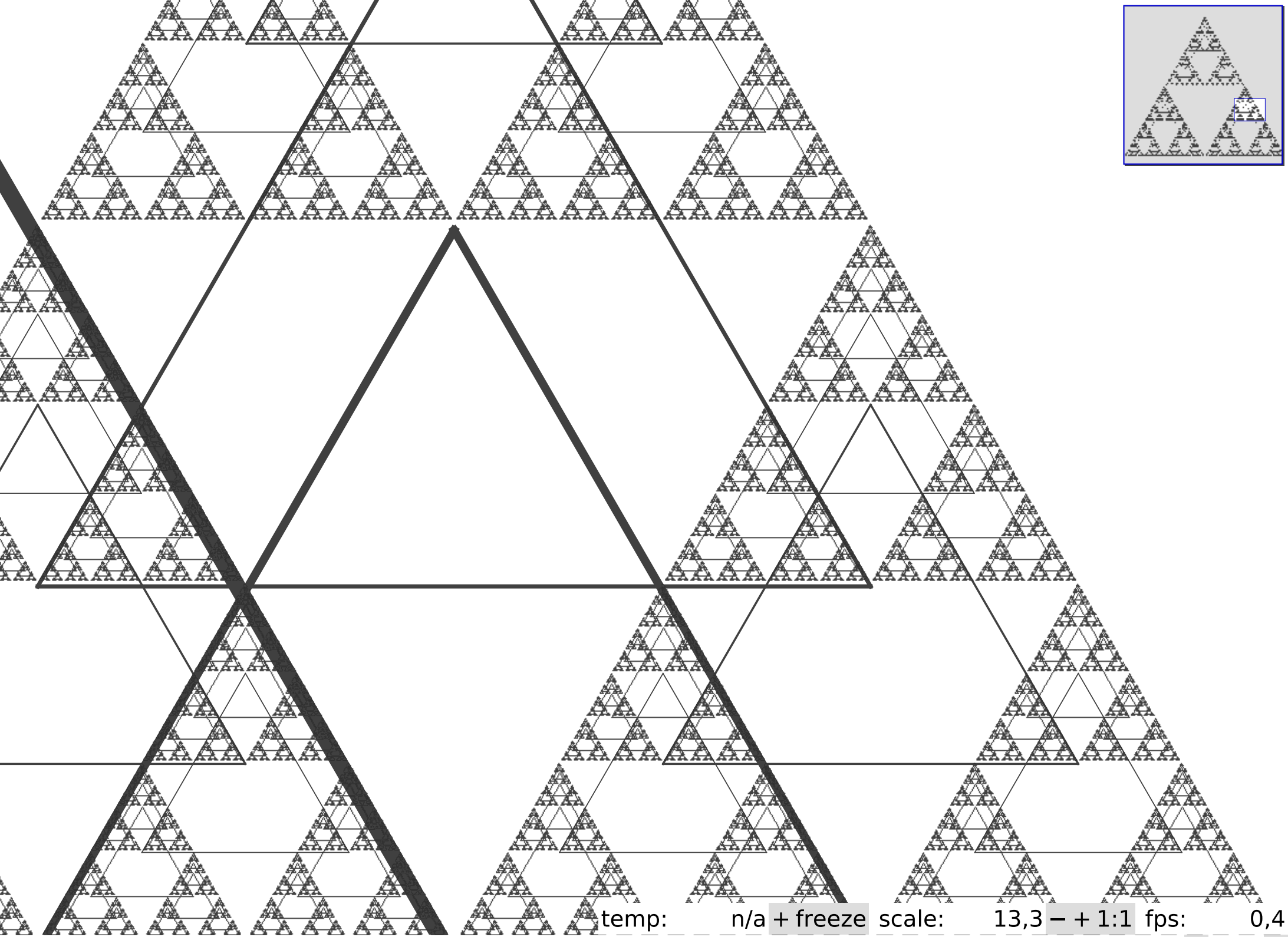
  a graph which vertices are visualizations of the graph itself...





temp: n/a + freeze scale: 1,8 − + 1:1 fps: 0,9

< 6/20 >


temp: n/a + freeze scale: 13,3 − + 1:1 fps: 0,4

Indyvon

- Interactive visual objects.
- Construction of visual objects:
  - immediate mode (IMGUI-like),
  - functional geometry,
  - interactive programming.
- Overcoming limitations of existing UI toolkits:
  - remove explicit scene graph or component hierarchy,
  - allow context-dependent reuse of visual objects (Flyweight pattern),
  - reduce state duplication between model and presentation.
  - allow multi-threading and asynchronous drawing.

View ADT

View

$$\text{render!} : \text{View} \times \text{Context} \to \text{Context}$$
$$\text{geometry} : \text{View} \times \text{Context} \to \text{Geometry}$$

Context:

- image buffer, clipping area, coordinate transform,
- registration of event handlers,
- thread-local mutable.

Look and behaviour of a View can be context-dependant.

Geometry provides dimensions and anchor points for alignment.

Interactive visual object (example)

Immediate mode:

```
(defn selectable [content]
  (reify View
    (render! [v]
      (with-handlers v
        (if (hovered? v)
          (draw! (border 2 content))
          (draw! (padding 2 content)))
        [:mouse-entered _ (repaint!)]
        [:mouse-exited _ (repaint!)]))
    (geometry [_]
      (geometry (padding 2 content)))))
```

Functional composition:

```
(let [n (atom 0)]
  (vbox (ref-view n label)
        (add-handlers (selectable (label "Increment"))
                      [:mouse-clicked _ (swap! n inc)])))
```

0
Increment

Interactive visual object (example)

Immediate mode:

```
(defn selectable [content]
  (reify View
    (render! [v]
      (with-handlers v
        (if (hovered? v)
          (draw! (border 2 content))
          (draw! (padding 2 content)))
        [:mouse-entered _ (repaint!)]
        [:mouse-exited _ (repaint!)]))
    (geometry [_]
      (geometry (padding 2 content)))))
```

Functional composition:

```
(let [n (atom 0)]
  (vbox (ref-view n label)
        (add-handlers (selectable (label "Increment"))
                      [:mouse-clicked _ (swap! n inc)])))
```

1
Increment

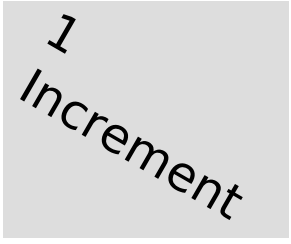
Interactive visual object (example)

Immediate mode:

```
(defn selectable [content]
  (reify View
    (render! [v]
      (with-handlers v
        (if (hovered? v)
          (draw! (border 2 content))
          (draw! (padding 2 content)))
        [:mouse-entered _ (repaint!)]
        [:mouse-exited _ (repaint!)]))
    (geometry [_]
      (geometry (padding 2 content)))))
```

Functional composition:

```
(let [n (atom 0)]
  (rotate 30
    (vbox (ref-view n label)
          (add-handlers (selectable (label "Increment"))
                        [:mouse-clicked _ (swap! n inc)]))))
```



Graph ADT

Graph$[V, A]$ — graph,
V — vertex type, A — associated data type.

vertices : Graph$[V, A] \to$ Set$[V]$

incoming : Graph$[V, A] \times V \to$ Set$[V]$

outgoing : Graph$[V, A] \times V \to$ Set$[V]$

data : Graph$[V, A] \times V \to A$

intrinsic? : Graph$[V, A] \times V \to \{0, 1\}$

Graph construction:

into : Graph$[V, A] \times$ Graph$[V, A] \to$ Graph$[V, A]$

remove : Graph$[V, A] \times$ Graph$[V, A] \to$ Graph$[V, A]$

Set$[E]$ — set of elements of type E.

Graph layout

Layout$[V, A]$ <: Graph$[V, A]$

- location : Layout $\times V \to$ Point
- bounds : Layout $\times V \to$ Interval
- edges : Layout $\to$ Set$[V \times V \times$ Point $\times$ Point$]$
- visible : Layout $\times$ Interval $\to$ Set$[V]$
- layout-bounds : Layout $\to$ Interval
- constraints : Layout $\to$ Constraints

IterativeLayout$[V, A]$ <: Layout$[V, A]$

- update : IterativeLayout $\to$ IterativeLayout
- complete? : IterativeLayout $\to \{0, 1\}$

Constraints specify edge attachment points,
vertex bounds, and constraint the vertex locations.

Layout algorithms

Force-based layout:

force-based-layout : Constraints $\rightarrow$ IterativeLayout

Fast initial layout:

initial-layout : Graph $\times$ Constraints $\rightarrow$ Layout

Composing layout algorithms:

$l_0 =$ force-based-layout(c)

$l_1 =$ into$(l_0,$ initial-layout$(g_1, c))$

$l_2 =$ into$(l_1,$ initial-layout$(g_2, c))$

Mapping rules

Define arbitrary filtering and aggregation of vertices.

S — type of vertices in the source graph,

Mapping Graph$[S]$ (problem domain model)
to Graph[View].

Rule — pair of functions p, f:

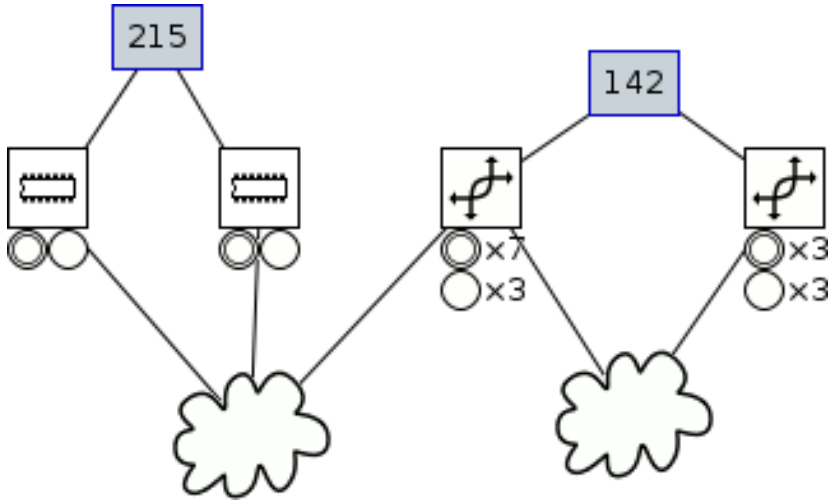
$p : S \to \text{List}[S]$
for a source graph vertex returns
a list of vertices that should be aggregated,

$f : \text{List}[S] \to \text{View}$
for the list of vertices returns
an interactive visual object.

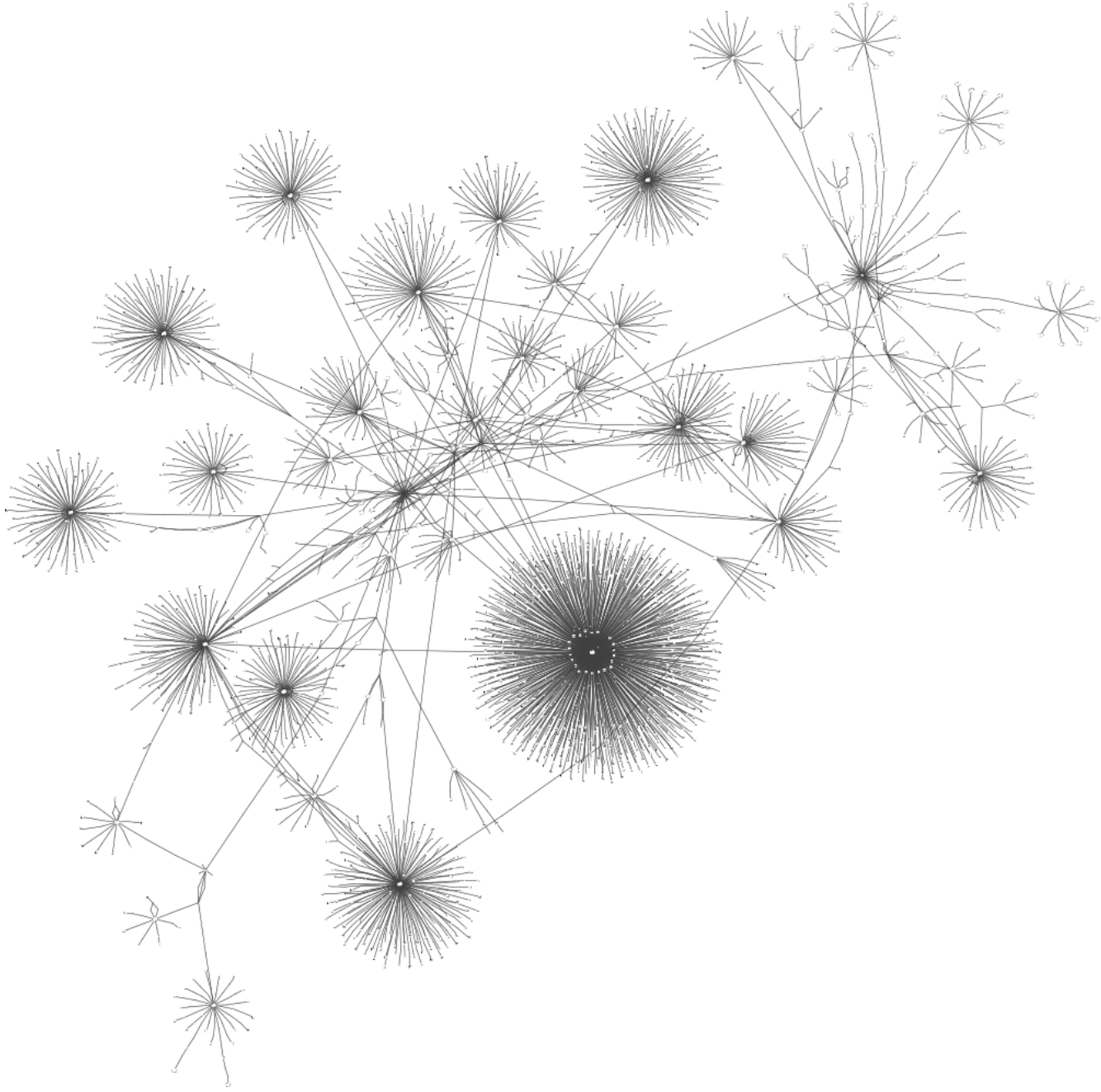
A rule is applicable if p returns a non-empty list.

Mapping rules (example)

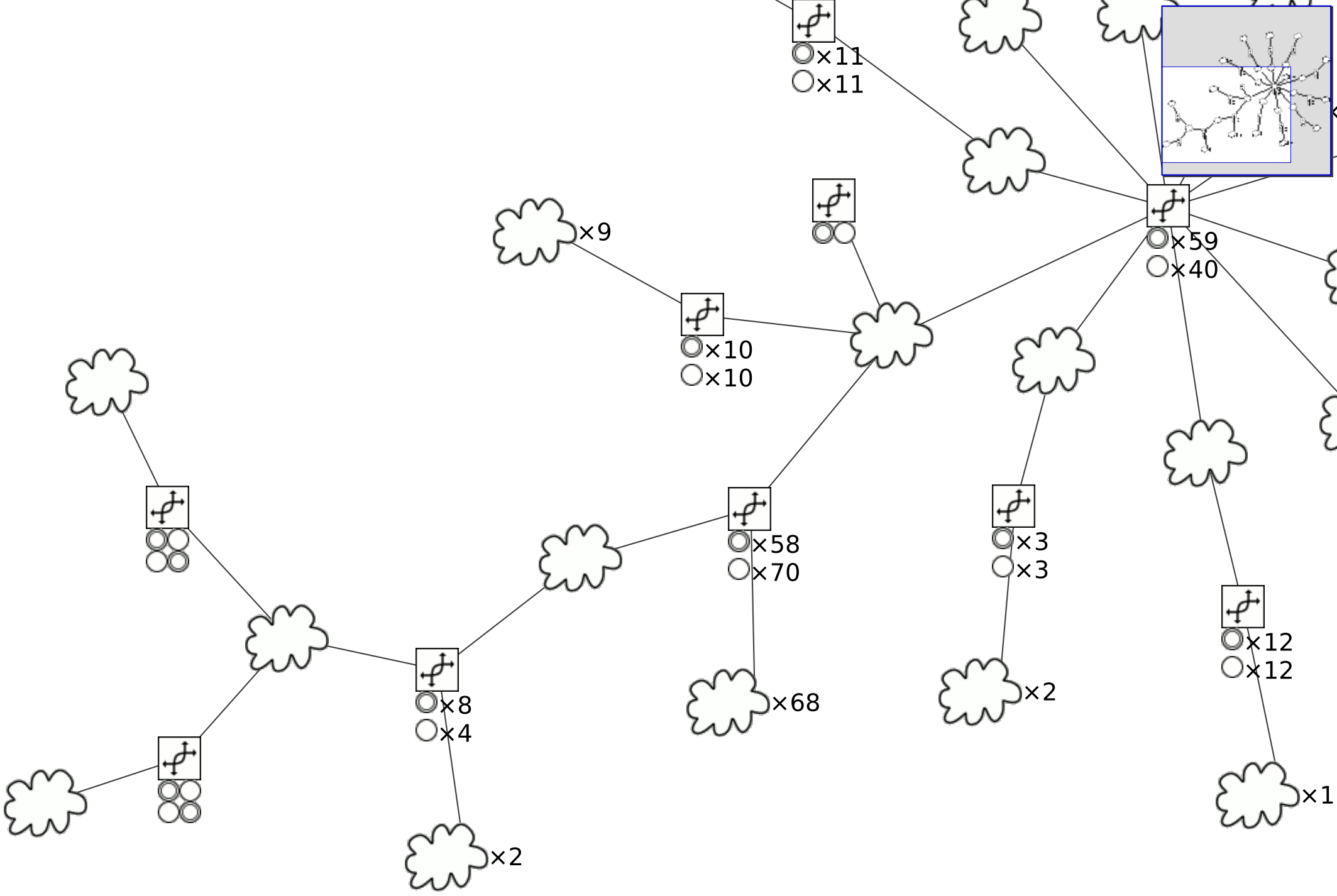
```
;; A room with all its divisions
(path Room Occupancy)
  ;; Room number in a rectangle
  (comp border (partial panel 5) label :number first)

;; A device and all its link and network interfaces
(path Device LinkInterface NetworkInterface)
  ;; Stack of elements, with constraint for y coordinate
  (comp (const-y 0) stack)

Network default
```



PetrSU Network

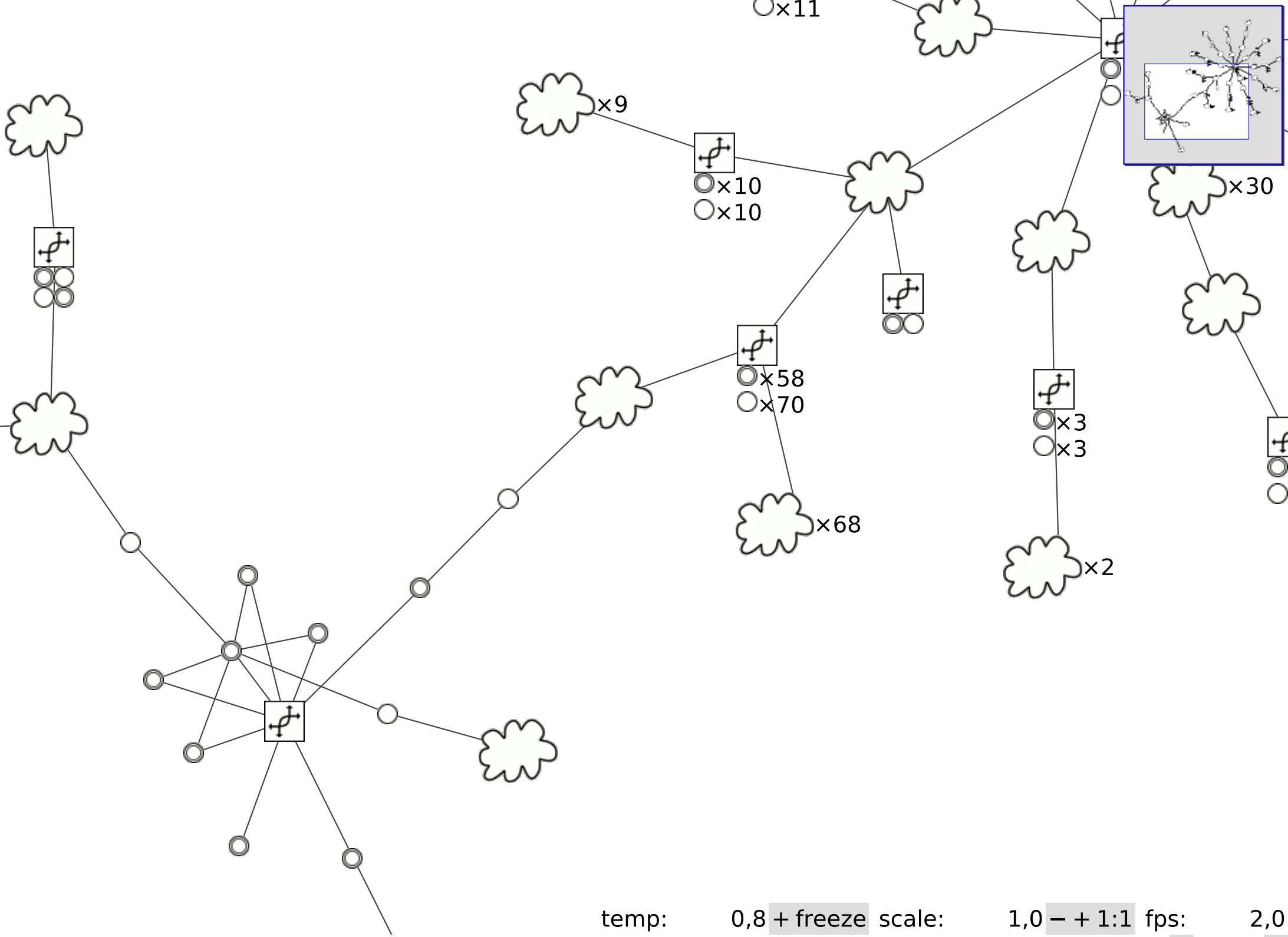


1571 device, 5377 vertices (subnets, devices and network interfaces)



temp: 0,8 + freeze scale: 1,0 − + 1:1 fps: 12,6

< 16/20 >



temp: 0,8 + freeze scale: 1,0 − + 1:1 fps: 2,0

< 16/20 >

Главный корпус

1

101 Серверная

Кафедра Информатики

×5

×3

Отдел телекоммуникаций

2

202 Кафедра

Кафедра Информатики

×7

201 Лаборатория

Кафедра Информатики

Учебный корпус

1

101 Серверная

Отдел телекоммуникаций

2

201

Отдел телекоммуникаций

×12

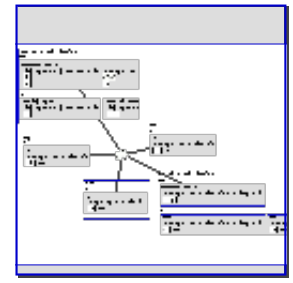
202

Отдел телекоммуникаций

×10

temp: 0,8 + freeze scale: 1/1,3 − + 1:1 fps: 5,5

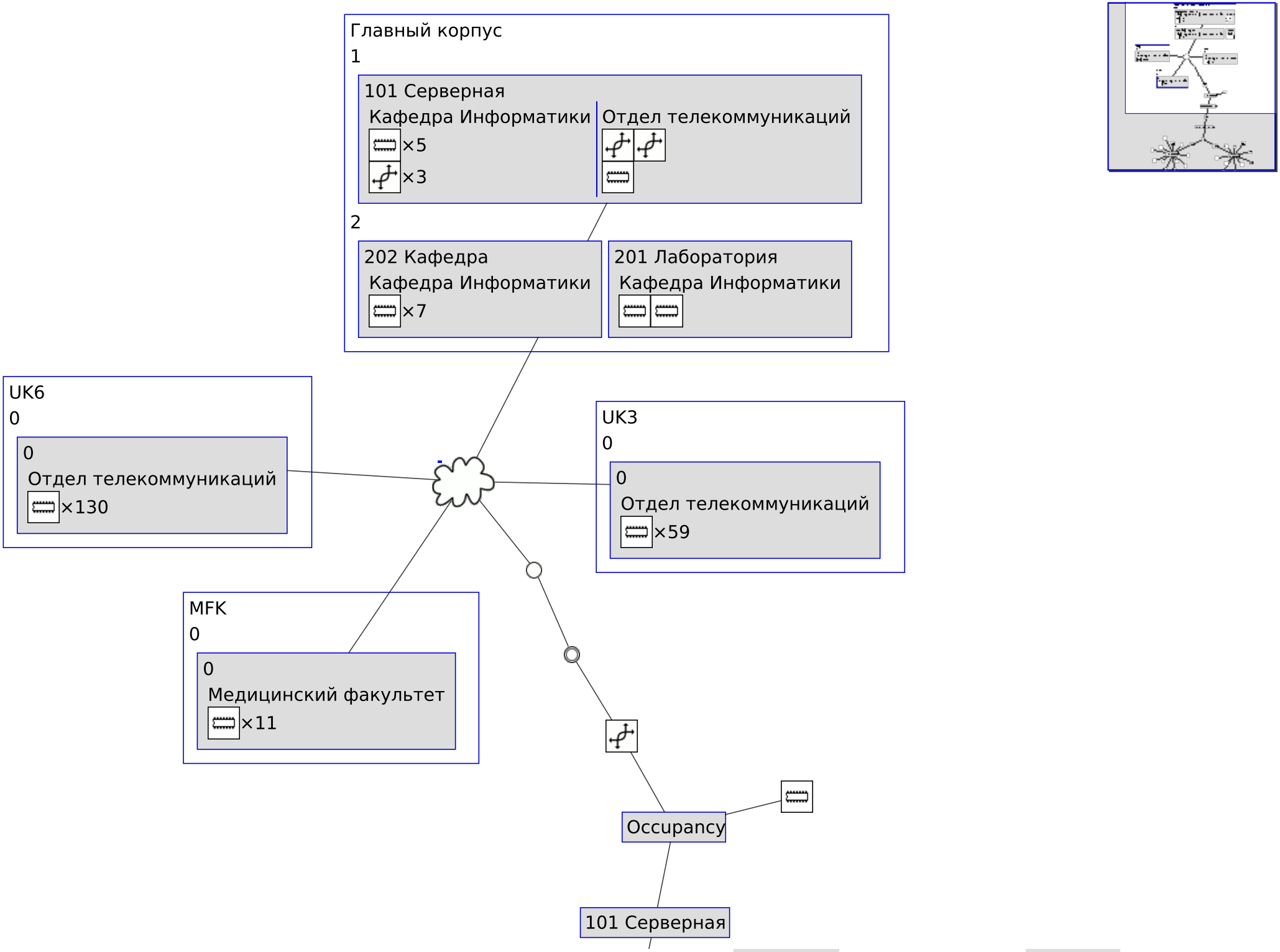
< 17/20 >



Главный корпус
1
101 Серверная
Кафедра Информатики
×5
×3
Отдел телекоммуникаций
2
202 Кафедра
Кафедра Информатики
×7
201 Лаборатория
Кафедра Информатики
UK3
0
0
Отдел телекоммуникаций
×59
UK6
0
0
Отдел телекоммуникаций
×130
MFK
0
0
Медицинский факультет
×11
Учебный корпус
1
101 Серверная
Отдел телекоммуникаций
2
201
Отдел телекоммуникаций
×12
202
Отдел телекоммуникаций
×10

temp: 0,8 + freeze scale: 1/1,3 − + 1:1 fps: 3,5

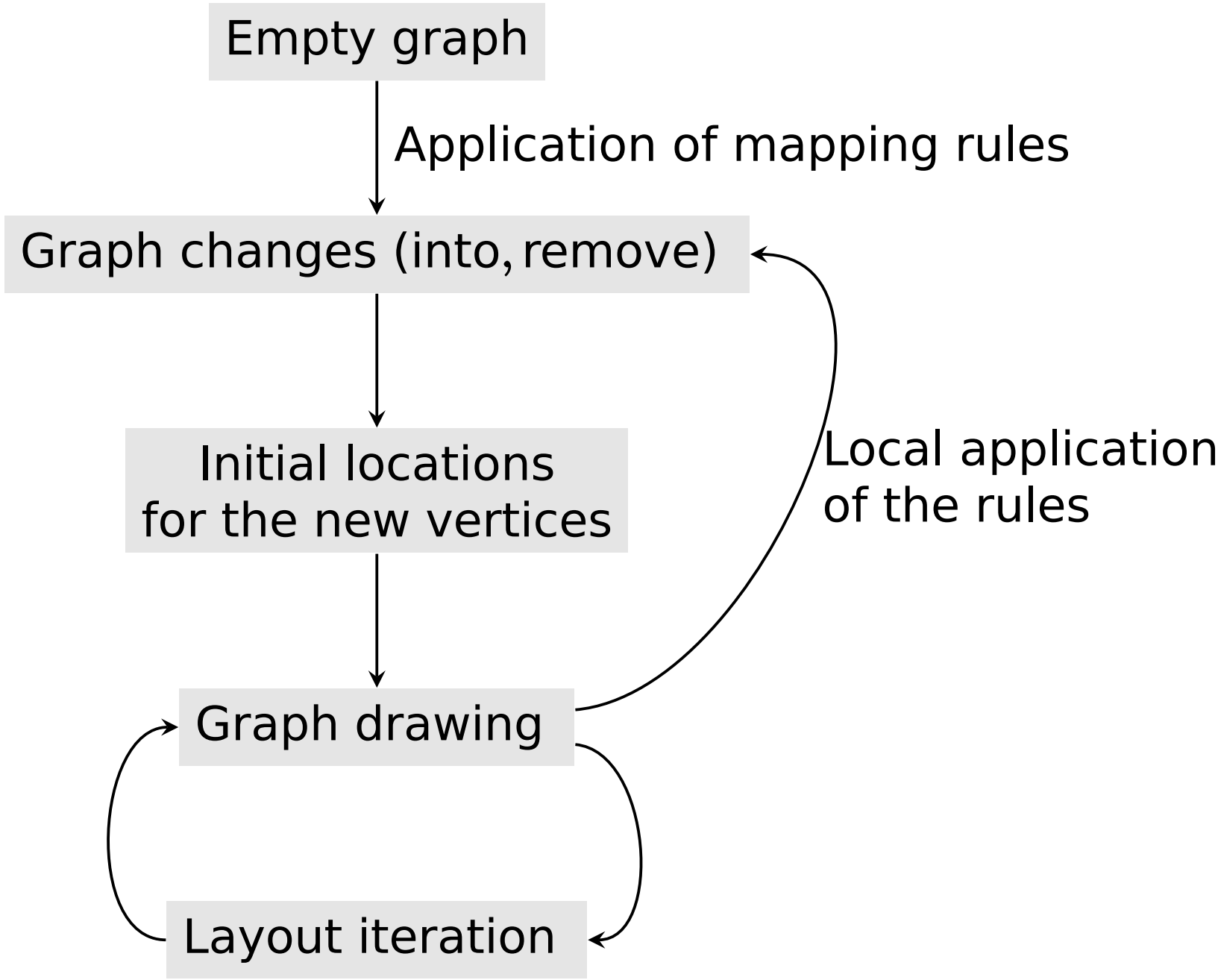
< 17/20 >



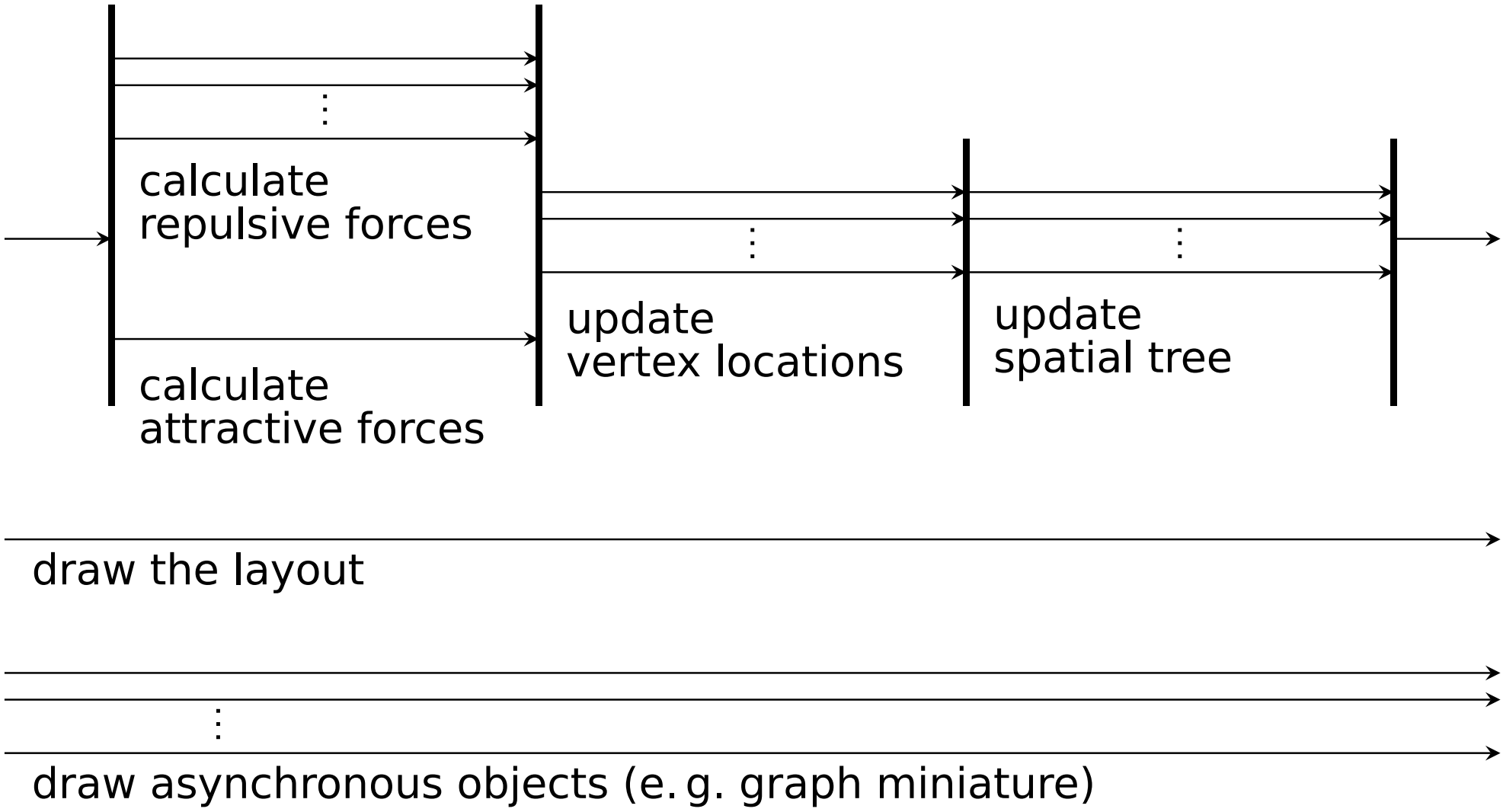
temp: 0,8 + freeze scale: 1/1,3 − + 1:1 fps: 5,6

< 17/20 >

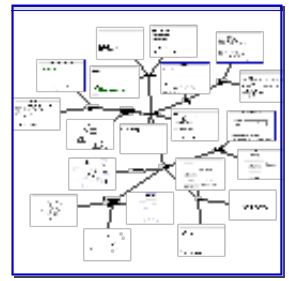
Updating the Graph



Parallel Implementation



Visualization of the PetrSU network graph (5377 vertices)
takes 11.5 seconds on Intel i5-3330 CPU (360 iterations 32 ms each).



View ADT

View
render! : View × Context → Context
geometry : View × Context → Geometry

Context:
- image buffer, clipping area, coordinate transform,
- registration of event handlers,
- thread-local mutable.

Look and behaviour of a View can be context-dependent.

Geometry provides dimensions and anchor points for alignment.

Indyvon

- Interactive visual objects.
- Construction of visual objects:
  - immediate mode (IMGUI-like),
  - functional geometry,
  - interactive programming.
- Overcoming limitations of existing UI toolkits:
  - remove explicit scene graph or component hierarchy,
  - allow context-dependent reuse of visual objects (Flyweight pattern),
  - reduce state duplication between model and presentation.
  - allow multi-threading and asynchronous drawing.

Graph layout

Layout$[V, A]$ <: Graph$[V, A]$
location : Layout × V → Point
bounds : Layout × V → Interval
edges : Layout → Set$[V \times V \times \text{Point} \times \text{Point}]$
visible : Layout × Interval → Set$[V]$
layout-bounds : Layout → Interval
constraints : Layout → Constraints

IterativeLayout$[V, A]$ <: Layout$[V, A]$
update : IterativeLayout → IterativeLayout
complete? : IterativeLayout → $\{0, 1\}$

Constraints specify edge attachment points, vertex bounds, and constraint the vertex locations.

Mapping rules (example)

```
;; A room with all its divisions
(path Room Occupancy)
  ;; Room number in a rectangle
  (comp border (partial panel 5) label (number first))
;; A device and all its link and network interfaces
(path Device Link0nterface NetworkInterface)
  ;; Stack of elements, with constraint for y coordinate
  (comp (const-y 0) stack)
Network default
```

Interactive visual object (example)

Immediate mode:

```
!defn pulsetable [content]
  !ready time
    !render ! [v]
      !with-handlers v
        !if !hovered? v!
          !draw! !border 2 content!!
          !draw! !padding 2 content!!!
        [:mouse-entered _ !repaint!!]
        [:mouse-exited _ !repaint!!]!
    !geometry [_]
      !geometry-!padding 2 content!!!!!

Functional composition:

  !let [n !atom 0!]
    !vbox !ref-view n !label
      !add-handlers !selectable !label "Increment"!!!
                    [:mouse-clicked _ !swap! n inc!]!!!

1
increment
```

Graph ADT

Graph$[V, A]$ — graph,
V — vertex type, A — associated data type.
vertices : Graph$[V, A] \to$ Set$[V]$
incoming : Graph$[V, A] \times V \to$ Set$[V]$
outgoing : Graph$[V, A] \times V \to$ Set$[V]$
data : Graph$[V, A] \times V \to A$
intrinsic? : Graph$[V, A] \times V \to \{0, 1\}$

Graph construction:
into : Graph$[V, A] \times$ Graph$[V, A] \to$ Graph$[V, A]$
remove : Graph$[V, A] \times$ Graph$[V, A] \to$ Graph$[V, A]$

Set$[E]$ — set of elements of type E.

Layout algorithms

Force-based layout:
force-based-layout : Constraints → IterativeLayout

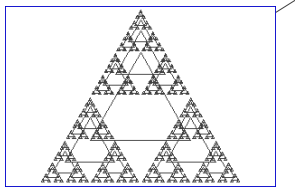
Fast initial layout:
initial-layout : Graph × Constraints → Layout

Composing layout algorithms:
l_0 = force-based-layout(c)
l_1 = into$(l_0$, initial-layout$(g_1, c))$
l_2 = into$(l_1$, initial-layout$(g_2, c))$

Mapping rules

Define arbitrary filtering and aggregation of vertices.
S — type of vertices in the source graph,
Mapping Graph$[S]$ (problem domain model) to Graph[View],
Rule — pair of functions p, f:
$p : S \to$ List$[S]$
for a source graph vertex returns a list of vertices that should be aggregated,
$f :$ List$[S] \to$ View
for the list of vertices returns an interactive visual object.

A rule is applicable if p returns a non-empty list.

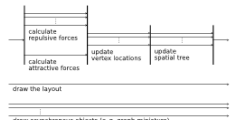


Components of the Visualization System

Generic
- Indyvon (Interactive Dynamic Visualization):
  - light-weight stateless context-dependent interactive visual objects.
- Pegla (Persistent Graph Layout):
  - immutable persistent data structures.
  - graph layout algorithms.

ICT infrastructure
- SDN (Spatial, Organizational, Network):
  - ICT infrastructure model.
  - database.
- ICT infrastructure graph visualizer
  - visual objects for representing the ICT infrastructure elements.
  - mapping source BDR graph to graph of visual objects.

Parallel Implementation



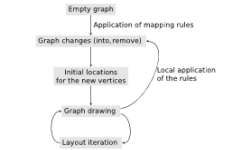
Visualization of the PetrSU network graph (5377 vertices) takes 11.5 seconds on Intel i5-3330 CPU (360 iterations 32 ms each).

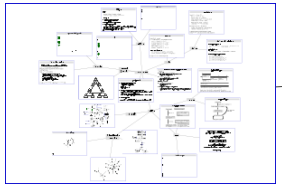
Abstractions

- Simple abstract data types:
  - interactive visual object,
  - graph,
  - graph layout.
- Implemented as immutable data structures:
  - based on persistent data structures provided by Clojure,
  - support functional programming,
  - reduce mutable state that needs to be synchronized.
- Composability:

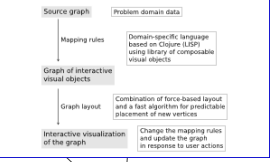
a graph which vertices are visualizations of the graph itself...

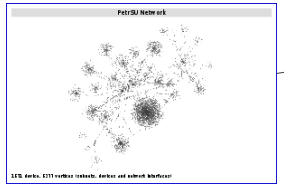
Updating the Graph

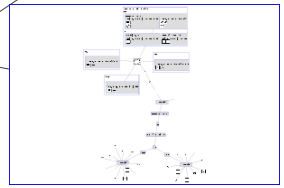


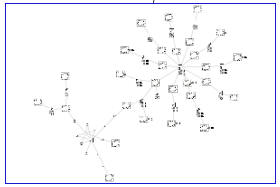


Visualization Method









Problem Domain

- Visualization of ICT infrastructure:
  - networking research and management,
  - testing of network topology discovery tools,
  - spatial and organizational mapping of network devices.
- Interactive graph visualization:
  - complex interactive vertices:
    - composable visual elements,
    - trigger changes to the graph and parameters of visualization;
  - dynamic graph layout.
- Different visual representations of the same data.
- Programmable interactive user interface.

Interactive graph visualization system and its application in network management

Mikhail Kryshen
kryshen@cs.petrsu.ru

Petrozavodsk State University
Department of Computer Science

Annual International Workshop on Advances in Methods of Information and Communication Technology

13.05.2015

Fractal example

Mapping

Components

Pegla

Layout

Process

Overview

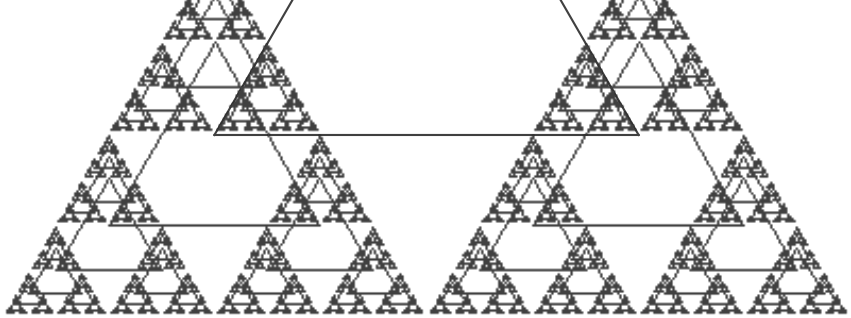
Slides

Visualization samples

Intro

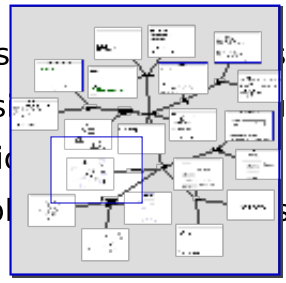
temp: 0,8 + freeze scale: 1/2,4 − + 1:1 fps: 0,8

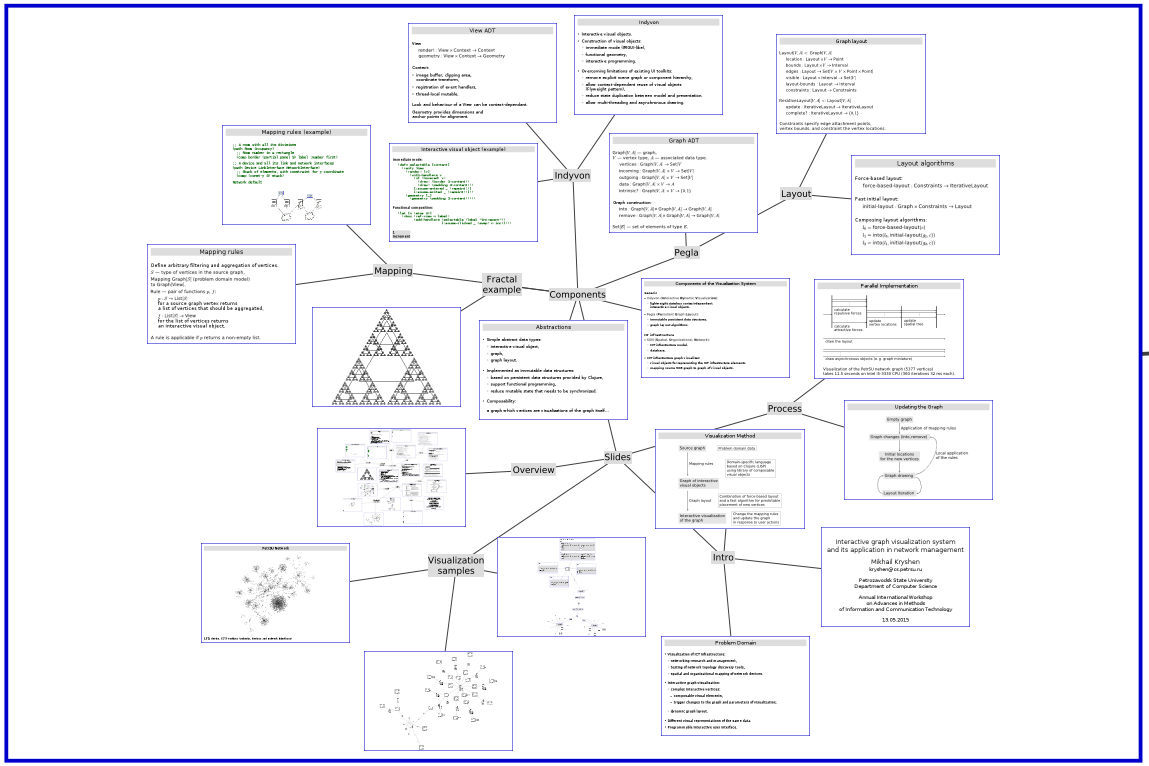
< 20/20 >



- Implemented as
  - based on pers
  - support functi
  - reduce mutabl
- Composability:

  a graph which vertices are visualiz





Overview

Visualization

temp: 0,8 + freeze scale: 1,8 − + 1:1 fps: 8,7

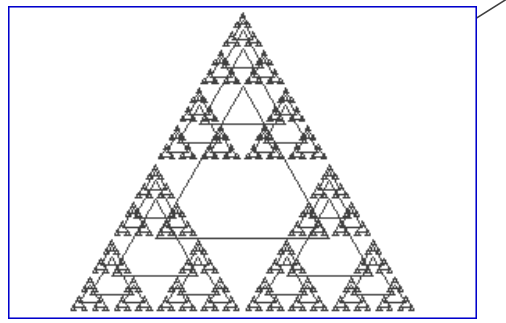
< 20/20 >

SU Network

a list of vertices that should be aggregated,

f : List[S] → View
for the list of vertices returns
an interactive visual object.

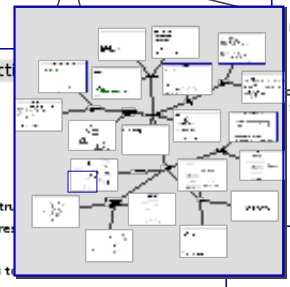
A rule is applicable if p returns a non-empty list.

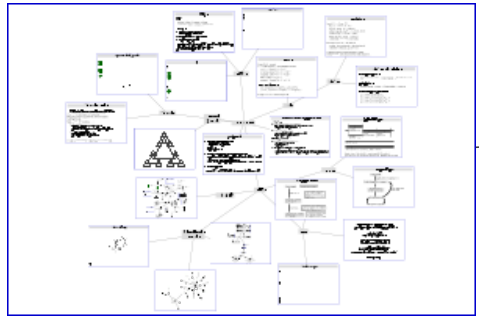


Abstract

- Simple abstract data types:
  - interactive visual object,
  - graph,
  - graph layout.
- Implemented as immutable data stru
  - based on persistent data structures
  - support functional programming,
  - reduce mutable state that needs t
- Composability:

  a graph which vertices are visualizations of the graph itself...

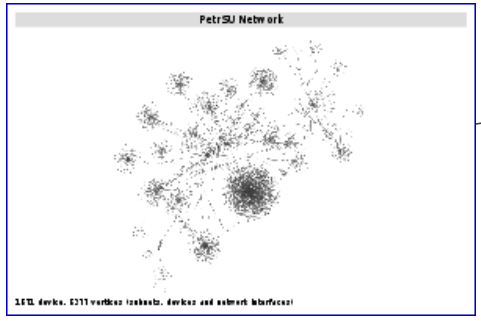


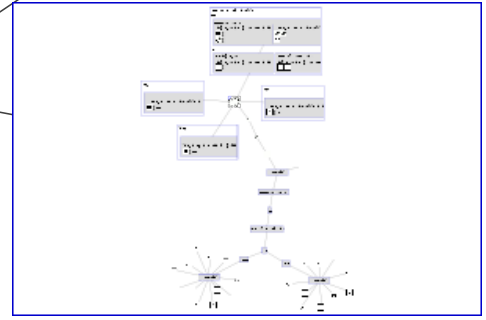


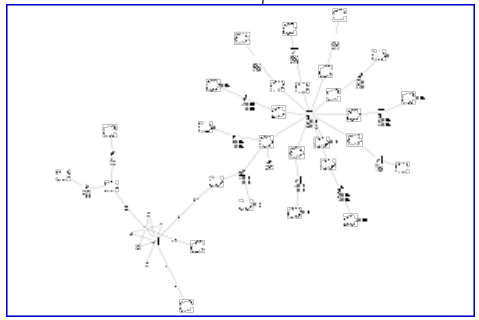
Slides

Overview

Visualization samples







temp: 0,8 + freeze scale: 5,6 − + 1:1 fps: 0,5

< 20/20 >