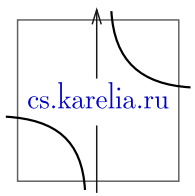
Петрозаводский Государственный Университет
Кафедра Информатики и математического обеспечения

Nest: экспериментальная платформа для исследования моделей и методов управления сетями локальных поставщиков услуг Интернета

Докладчик:

к. т. н., доцент Ю. А. Богоявленский, ybgv@cs.karelia.ru
http://cs.karelia.ru



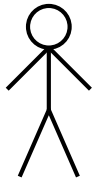
Актуальность

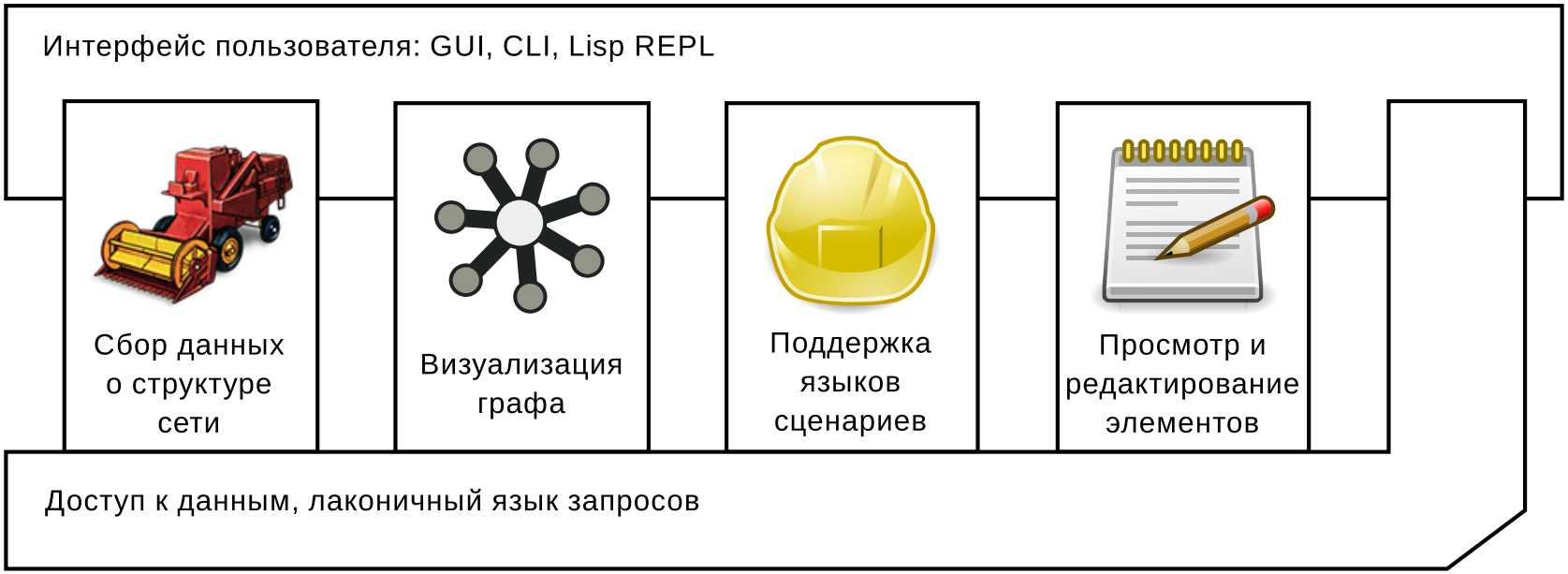
- Отчет организации FIND (Future Internet Design): «Архитектурные решения современного Интернета не предоставляют эффективных инструментов управления системами поставщиков сетевых услуг».

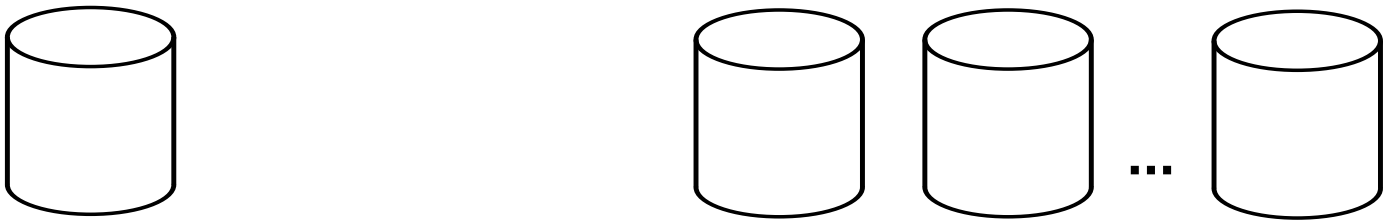
  Винтон Серф

- Сложность управления сетями локальных поставщиков сетевых услуг (лПСУ) и магистральными сетевыми системами сопоставима.
- ИКТ-инфраструктура лПСУ — Сеть.
- Необходима разработка моделей и методов управления.
- Необходима разработка экспериментальных платформ для выполнения эталонных тестов в реалистичных условиях.
- Необходима разработка универсального средства визуализации элементов и топологии Сети.

Архитектура системы Nest





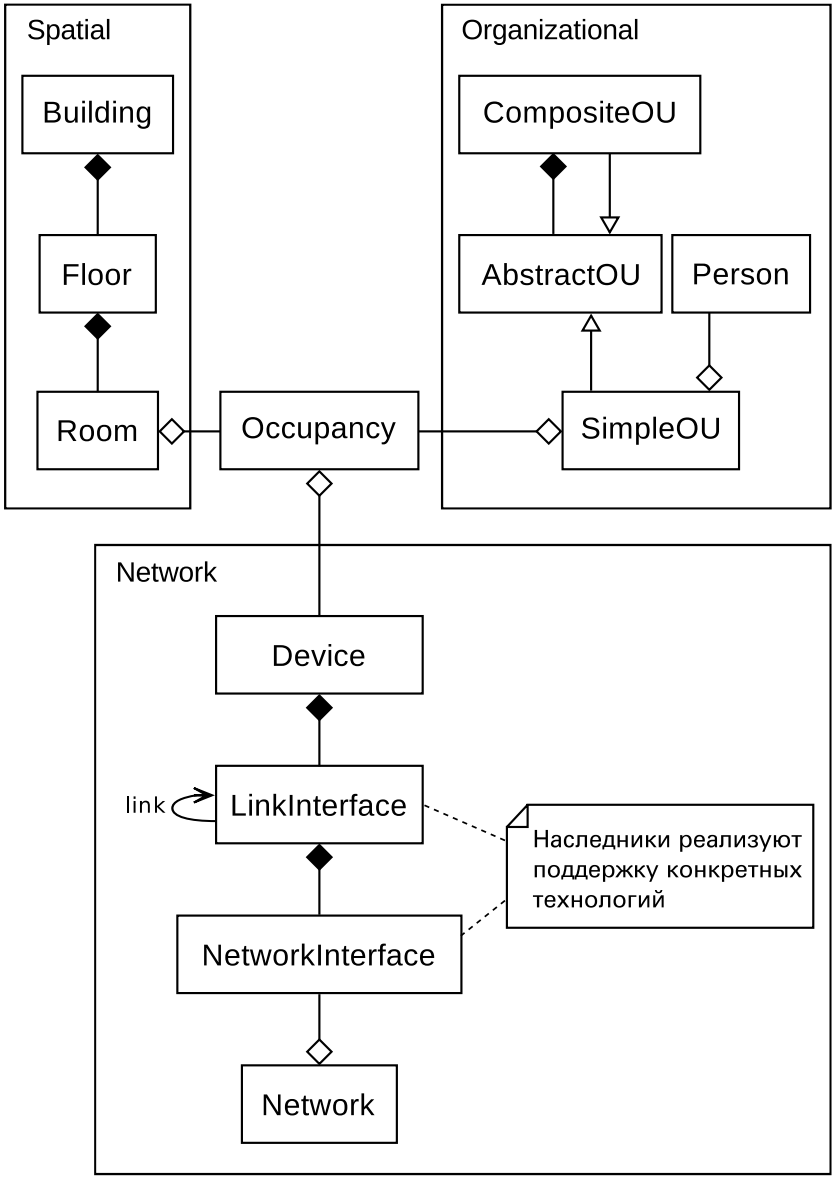


БД SON

Дополнительные источники (потоки Netflow, журналы сетевых служб)

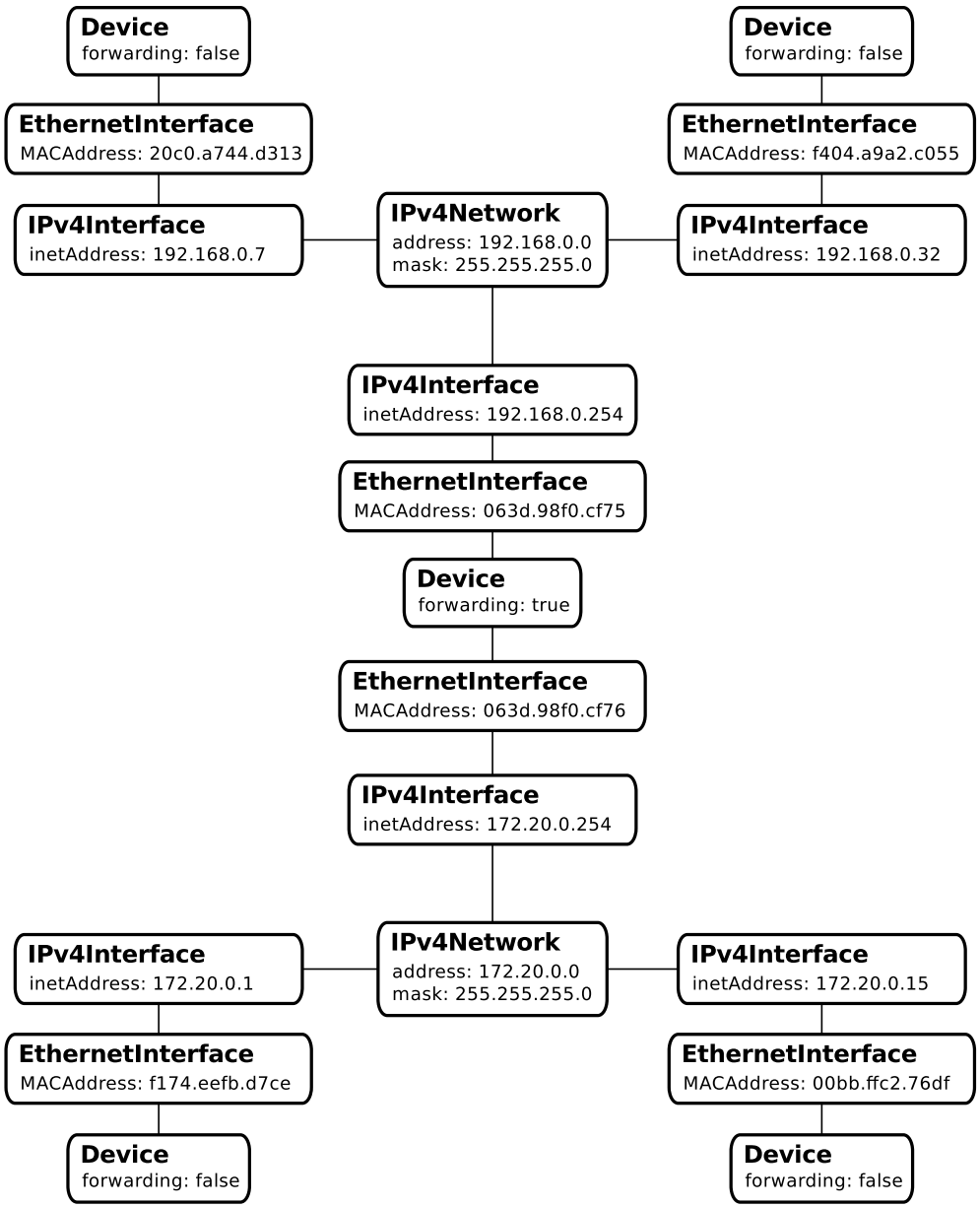
Среда выполнения — Java Runtime Environment 7.

Объектная модель SON



- Пространственная структура.
- Организационная структура.
- Сетевая структура:
  - представление сети на канальном и сетевом уровнях;
  - простота, расширяемость;
  - связь с моделями организационной и пространственной структур;
  - представление источников и приемников потоков данных как элементарных, так и агрегированных.

Объектный граф



- Основной источник данных системы Nest — ООБД, содержащая объектный граф, построенный в соответствии с моделью SON.
- Сетевая компонента графа поддерживается автоматизированно.
- Пространственная и организационная компоненты поддерживаются администратацией Сети.

Задача получения графа Сети

- Не существует стандартных средств для сбора информации о топологии.
- Закрытые решения, привязанные к производителю:
  - IBM Tivoli NetView
  - CiscoWorks
  - HP OpenView

Требования к подсистеме получения графа Сети:

- Использовать протокол SNMP для сбора информации с сетевых устройств.
- Использовать объектную модель SON для представления вершин и связей графа Сети.
- Не устанавливать дополнительное ПО на оконечные сетевые узлы.
- Использовать для построения графа только информацию от маршрутизаторов Сети.
- Получать граф с помощью параллельного алгоритма.

Исходные данные

- На вход алгоритма передается адрес произвольного маршрутизатора.
- Алгоритм построения графа Сети основан на анализе таблицы интерфейсов, таблицы маршрутизации, ARP-таблицы маршрутизаторов Сети.
- Необходимые данные запрашиваются из MIB маршрутизаторов по протоколу SNMP:
  - `ifTable`, `ifXTable` — таблица канальных интерфейсов.
  - `ipAddrTable` — соответствие интерфейсов и IP-адресов.
  - `ipNetToMediaTable` — содержимое ARP-кеша маршрутизатора.
  - `ipRouteTable`, `ipForwardTable`, `ipCidrRouteTable` — таблица маршрутизации.

Оценка алгоритма

- Предполагаемая временная сложность линейна по числу сетевых элементов.
- Построение графа Сети ПетрГУ (около 4 500 объектов) — 1 минута (Intel Xeon 2.5 ГГц, 8 ядер, ОЗУ 1 ГБ, ОС Linux 3.1.0).
- Исходящий трафик: 3.5 КБ. Входящий: 25 КБ (Пропускная способность — 100 Мбит/с).
- Полученный граф соответствует логической топологии сети с добавлением информации о физических интерфейсах оконечных СЭВМ и маршрутизаторов.
- Содержимое подсетей определяется с точностью до периода обновления ARP-кэша маршрутизатора.
- Графы, полученные для Сети ПетрГУ, совпадают с данными системных администраторов.

Параллельная процедура построения графа Сети

- Последовательная процедура не позволяет эффективно использовать процессор по причине периодического ожидания сетевого ввода/вывода.
- Независимые участки графа Сети могут строиться параллельно.
- В ходе реализации алгоритма был предложен подход, при котором построение отдельных узлов графа изолировано в специальных классах-поставщиках.
- Выполнение экземпляров классов-поставщиков может производиться параллельно, обеспечивая простой способ распараллеливания алгоритма.
- Реализовано средствами параллельного программирования, предоставляемых пакетом `java.util.concurrent` из состава JDK7.

Время построения графа Сети в зависимости от размера пула потоков и количества ядер

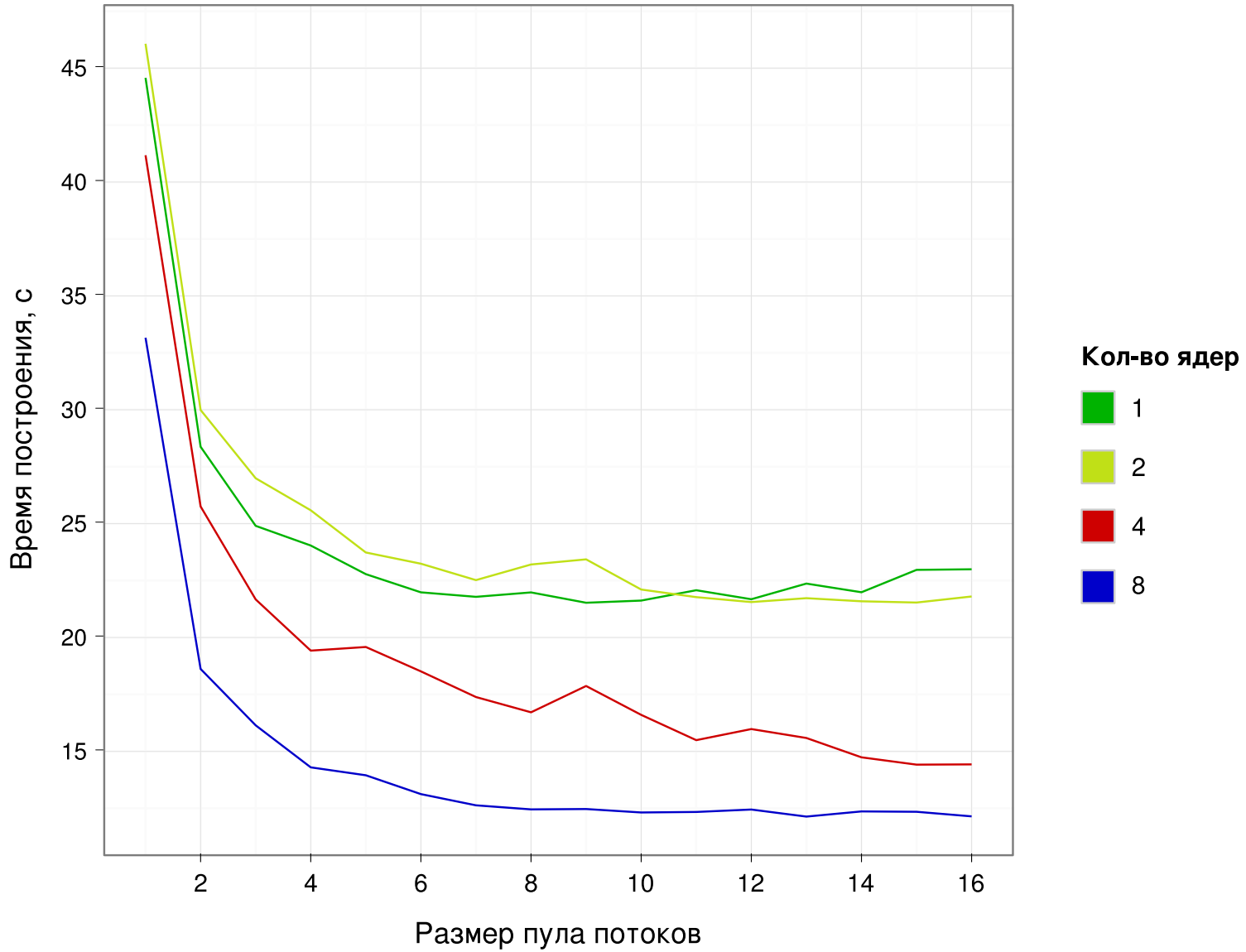


Иллюстрация работы подсистемы

Провайдеры

Граф сети

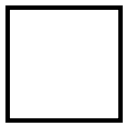


Иллюстрация работы подсистемы

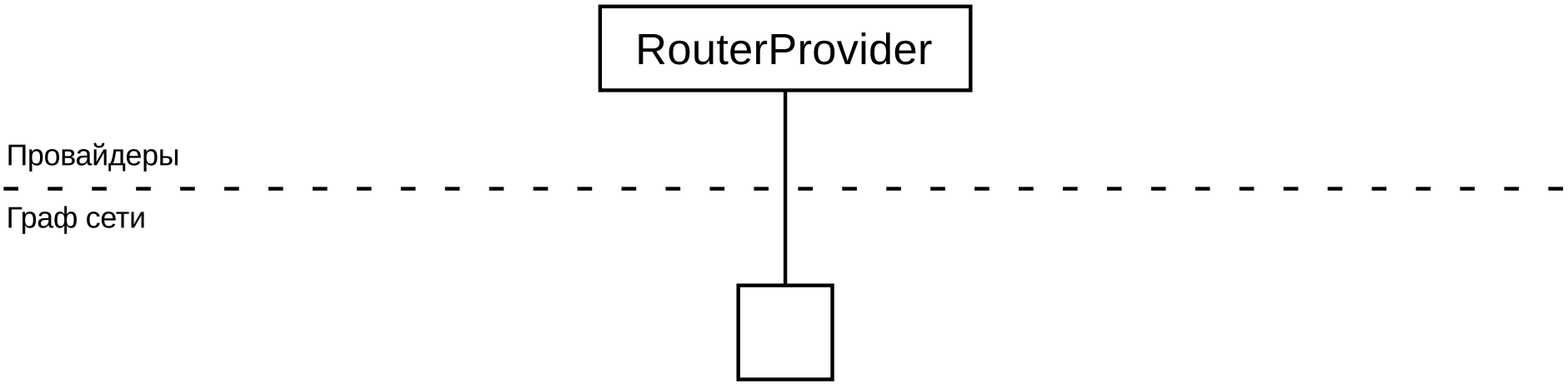


Иллюстрация работы подсистемы

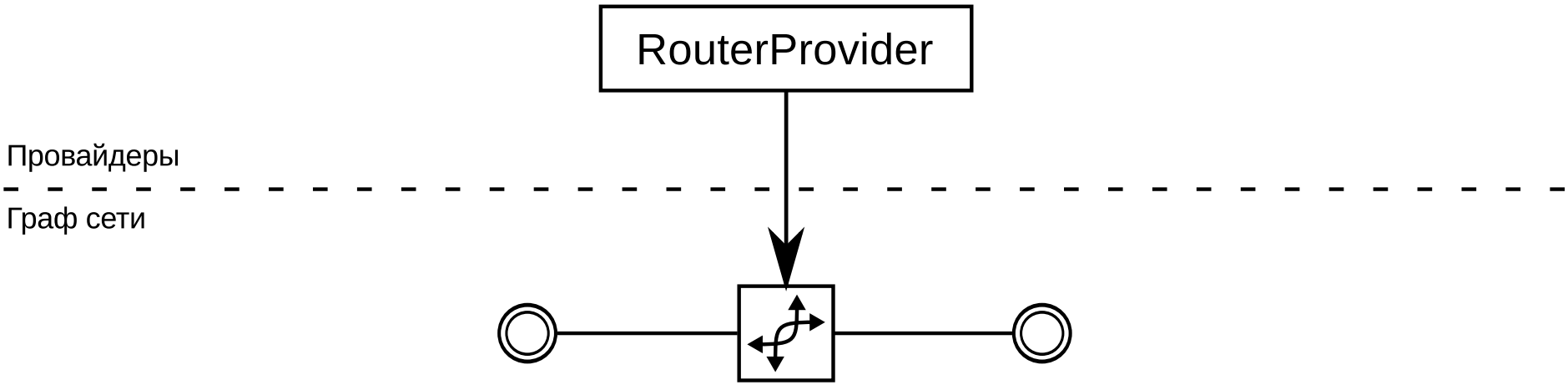


Иллюстрация работы подсистемы

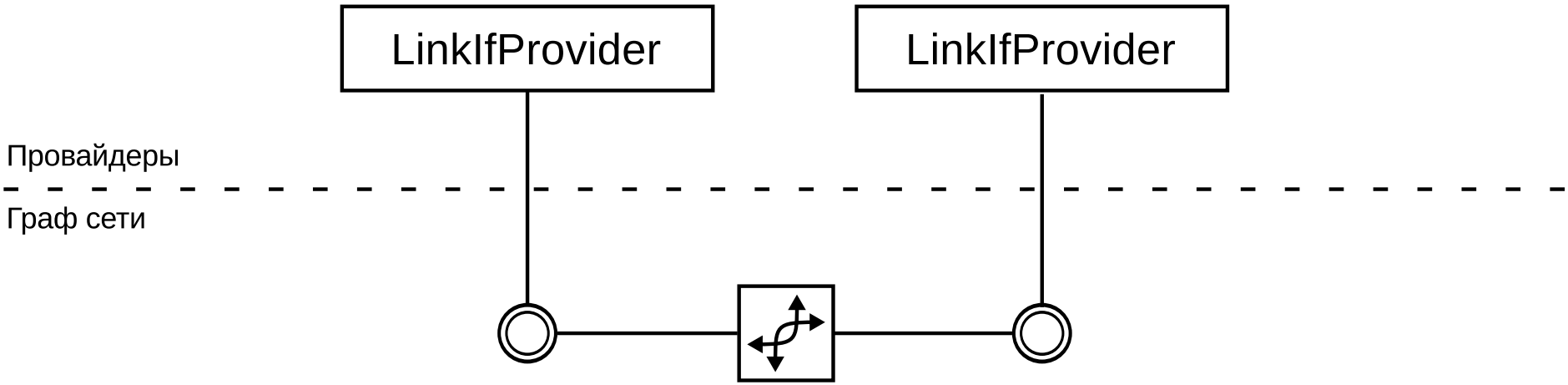


Иллюстрация работы подсистемы

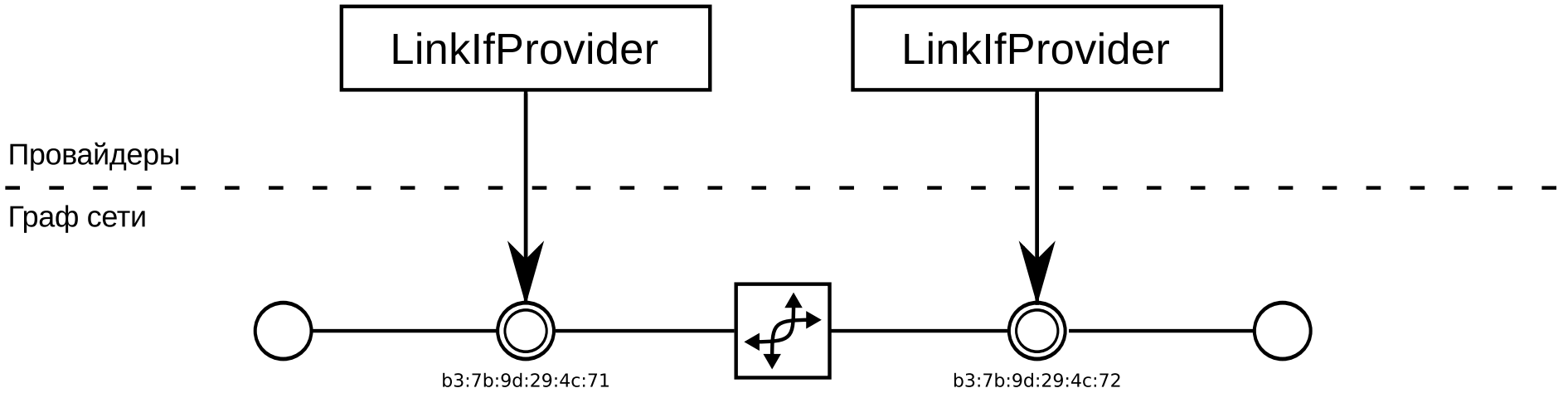


Иллюстрация работы подсистемы

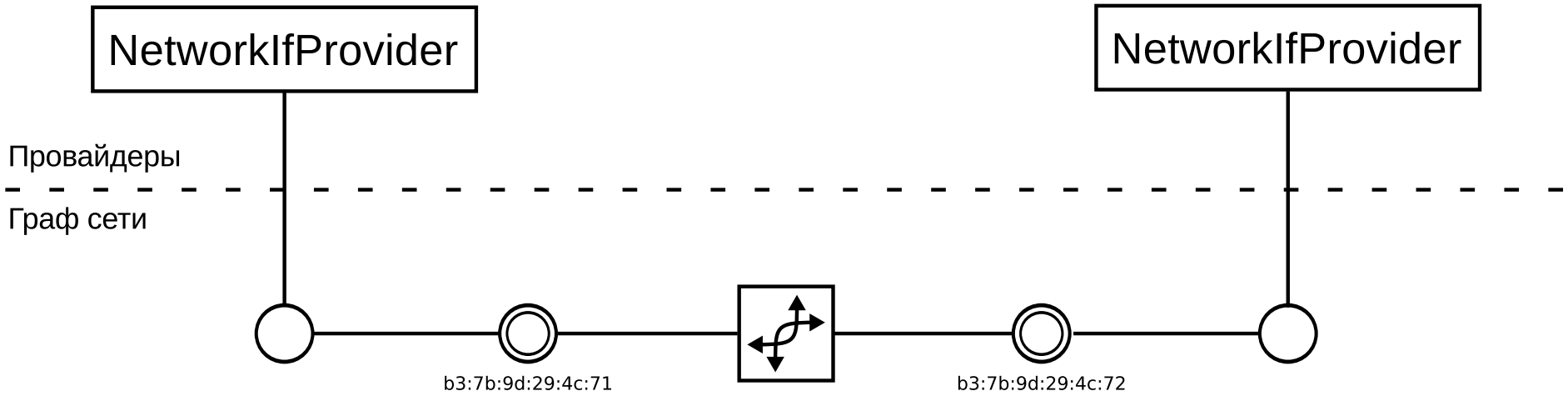


Иллюстрация работы подсистемы

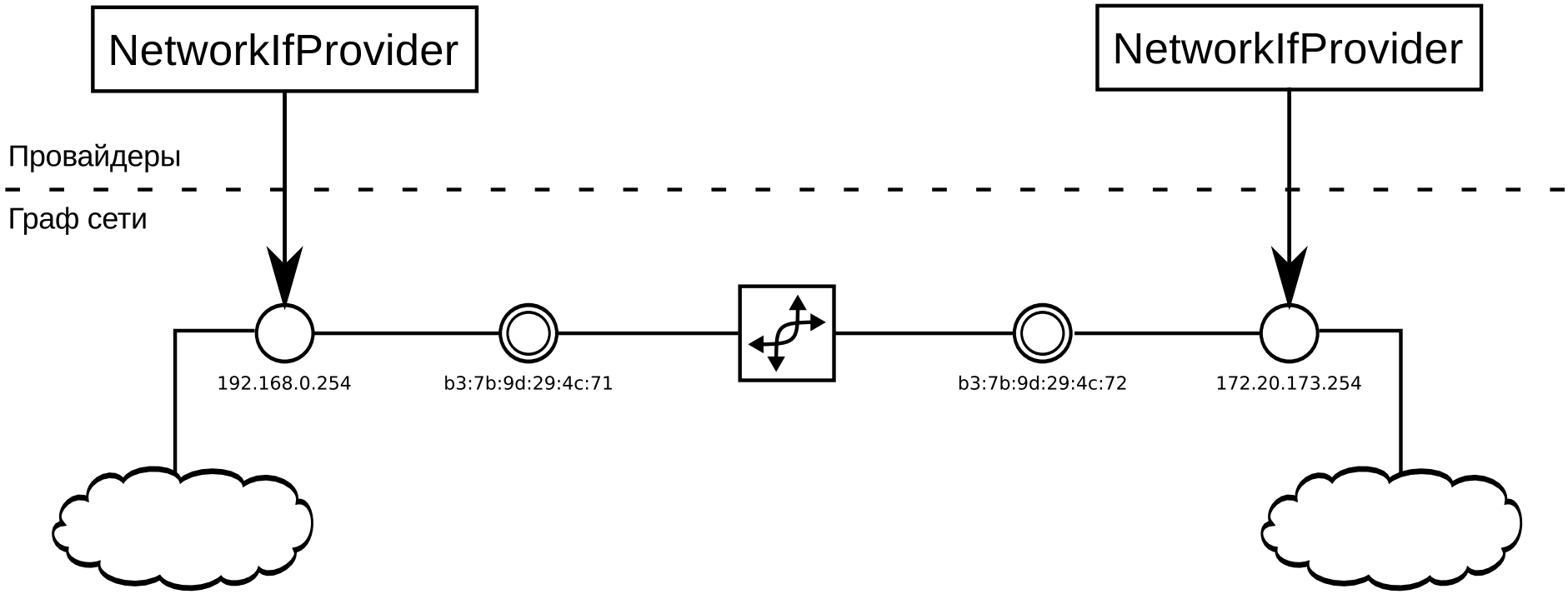
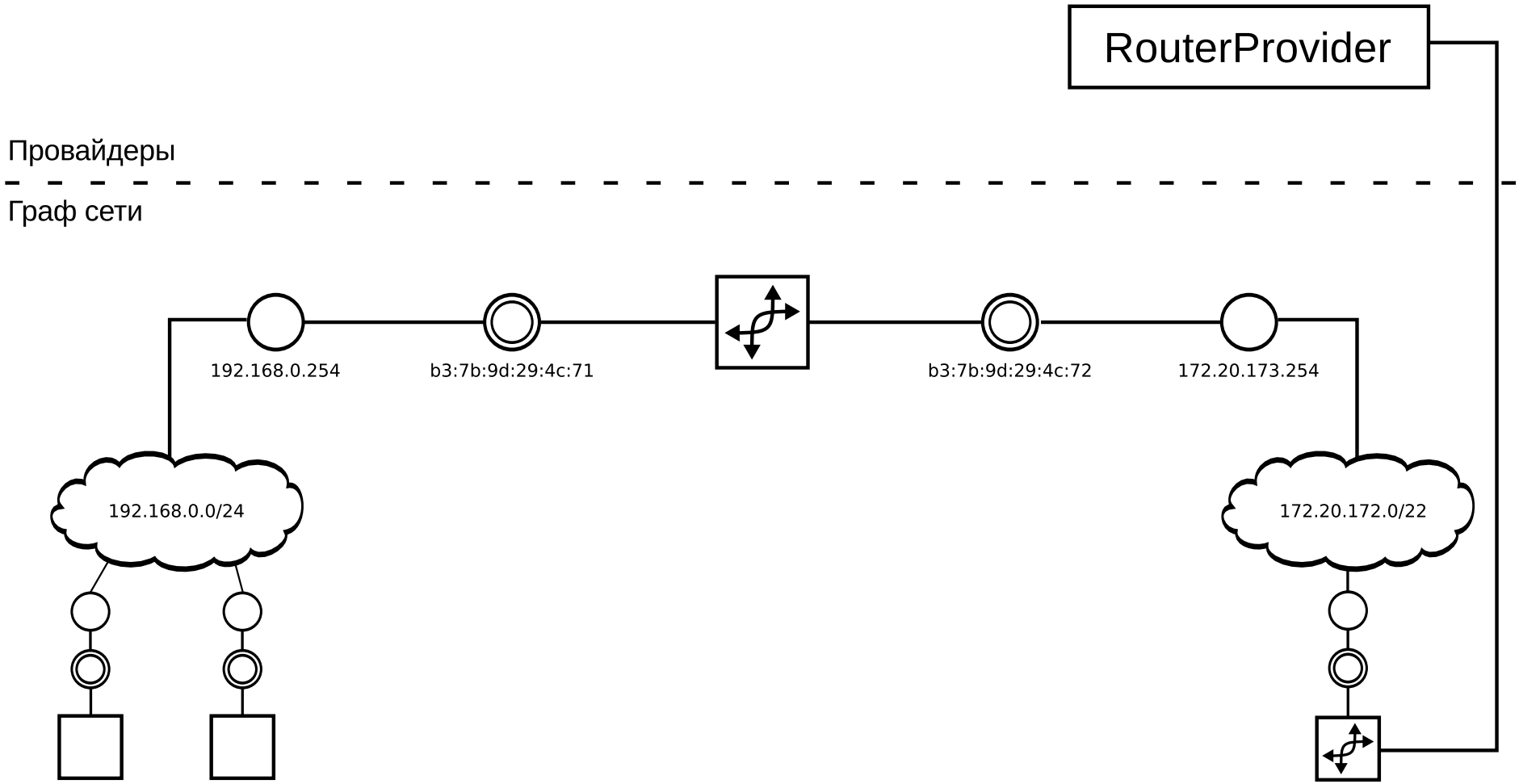


Иллюстрация работы подсистемы



Характеристики подсистемы получения данных и построения графа Сети

- Сбор данных об элементах и топологии лПСУ.
- Параллельная процедура построения графа.
- Сохранение графа топологии в объектной БД.
- Обновление существующего графа без повторного перестроения.
- Ведение истории изменений графа Сети.
- Определение multihoming-устройств.

Задача визуализации

- Визуализировать сетевую структуру лПСУ совместно с организационной и пространственной структурами модели SON.
- До 10 000 сетевых устройств.
- Сетевая структура не представима в виде дерева.
- Различные варианты визуального представления, определяемые пользователем.
- Интерактивная работа.
- Приемлемая производительность и эффективное использование вычислительных ресурсов, параллельные алгоритмы раскладки и отрисовки.

Метод визуализации

Граф объектов SON

Построенный граф сети, организационная и пространственная информация в ООБД.

Правила преобразования

Предметно-ориентированный язык на основе Clojure (Лисп). Правила: подграф SON $\rightarrow$ визуальный объект.

Предоставляется библиотека комбинируемых визуальных объектов.

Граф интерактивных визуальных объектов

Алгоритм раскладки графа

Визуализация графа методом физических аналогий.

Интерактивное визуальное представление графа

Изменение правил визуализации и обновление графа в ответ на действия пользователя.

Предметно-ориентированный язык визуализации

Набор правил — пары (p, f).
p : шаблон подграфа SON.
f : последовательности элементов SON $\rightarrow$ визуальные объекты.

Выбор по типу объекта SON:

`Device`, `Network`, `Building` и др.

Выбор по набору свойств:

`{:`*`свойство1 значение1`*`, :`*`свойство2 значение2`*`, ...}`

Соответствие любому из шаблонов:

`[`*`шаблон1 шаблон2 ...`*`]`

Выбор подграфа:

`(path `*`шаблон0 шаблон1 шаблон2 ...`*`)`

Шаблон0 задает начальную вершину. Выбирается множество путей графа от начальной вершины, таких что все вершины каждого пути удовлетворяют соответствующим шаблонам.

Основные графические объекты

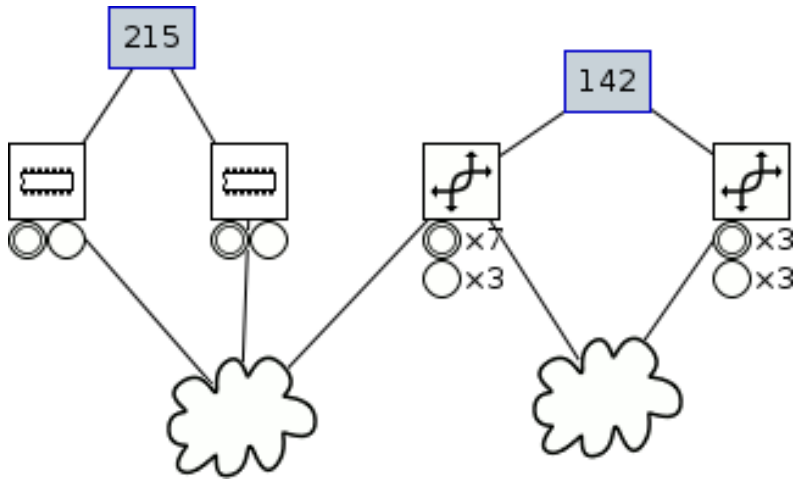
- Отображение элементов SON:
  `simple` — короткое описание или изображение,
  `verbose` — список свойств и значение,
  `combined` — `simple`, под курсором — `verbose`,
  `stack` — компактное представление набора элементов.
- Декораторы:
  `expandable` — добавляет реакцию на события мыши (изменение графа, контексное меню),
  `border` — добавляет рамку,
  `panel` — добавляет фон,
  `const-x`, `const-y`, `on-circle` и др. — задают ограничения на расположение объектов при визуализации графа.
- Группировка:
  `hbox`, `vbox` — горизонтальное и вертикальное размещение.
- Базовые:
  `label` — отображает текст, `image` — изображение.

Пример визуализации

```
;; Комната и все ее отделы
(path Room Occupancy)
  ;; Номер комнаты в прямоугольнике
  #(-> % first .getNumber label (panel 5) border)

;; Устройство и все его канальные и сетевые интерфейсы
(path Device LinkInterface NetworkInterface)
  ;; Набор элементов, ограничение координаты y=0
  #(-> % stack (const-y 0))
```

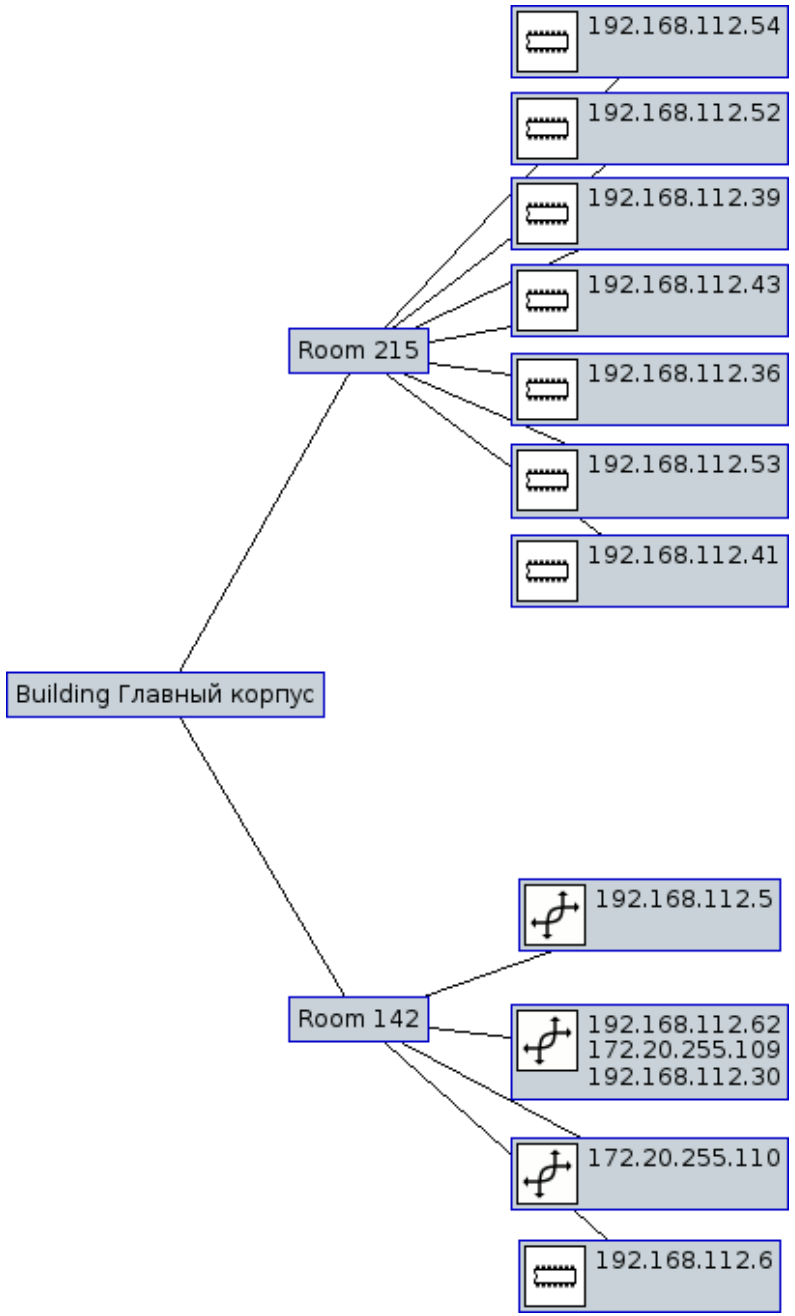
Network stack



```
(path Building Floor)
  #(-> % stack (const-x 0))

(path [{:number "142"} {:number "215"}]
      Occupancy)
  #(-> % stack (const-x 100))

(path Device LinkInterface NetworkInterface)
  (fn [[device & interfaces]] (->
   (hbox
    (popup device)
    (apply vbox
     (map
      (comp label display-string)
      (filter
       #(instance? NetworkInterface %)
       interfaces))))
   (panel 2)
   border
   (const-x 250)))
```



Визуализация графа

- Визуализация на основе физических аналогий (вершины — заряды, дуги — пружины).
  Хороший результат, гибкость, интерактивность.
- Учет размеров вершин.
- Группировка слабо взаимодействующих зарядов-вершин для улучшения производительности.
- Параллельно:
  - пересчет сил для каждой вершины (работа распределятся по доступным процессорным ядрам),
  - отрисовка видимой части графа,
  - отрисовка миниатюры всего графа (с меньшим приоритетом).
- Пример работы алгоритма.

Лаконичный язык запросов

Мотивация:

- известные языки запросов для объектной БД требуют от пользователя полных знаний о модели;
- модель с большим количеством классов делает написание запросов трудоемкой задачей;
- большой объем текста запроса способствует появлению ошибок;
- невозможно предугадать, что может потребоваться пользователю в конечном итоге.

Механизмы достижения лаконичности:

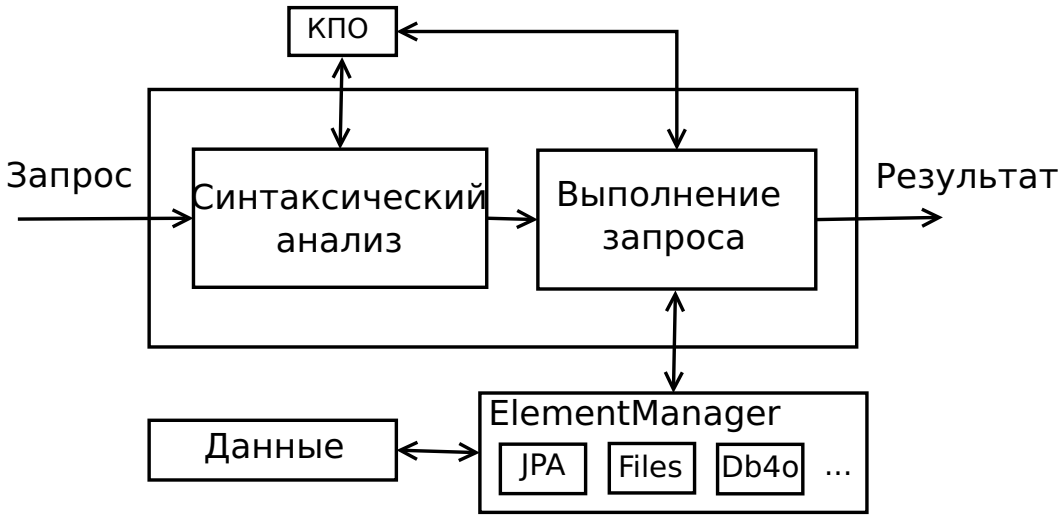
- элиминирование указания полной спецификации путей в объектном графе;
- минимизация ключевых слов;
- сокращенная запись имен классов;
- сокращенная запись сложных условий;
- введение свойства по умолчанию.

Выбор пути между двумя объектами

- Выбор пути осуществляется на основе метаинформации о модели (карта предметной области, КПО).
- КПО содержит информацию о путях между двумя любыми классами объектной модели.
- КПО формируется в полуавтоматическом режиме:
  1. создается на основе графа классов посредством алгоритма обхода в ширину;
  2. дополняется вручную по мере необходимости.
- Во время выполнения запроса не осуществляется поиска.
- Путь между двумя классами извлекается из КПО.
- Для изменения пути пользователь должен предоставить дополнительную информацию (промежуточные классы или свойства).

Лаконичный язык запросов

- Идея: лаконичность, достигнутая за счет использования объектной модели.
- Поддержка стандартных возможностей общепринятых языков запросов (OQL, HQL, JP-QL и др.).
- Независимость от хранилища.
- Древовидное и табличное представление результата в интерфейсе пользователя.



Сравнение текстов запросов

Найти здание, в котором находится маршрутизатор, обслуживающий заданное устройство:

- **HQL:**

```
select b from Building as b
        left join b.floors as f
        left join f.rooms as r
        left join r.occupancies as o
        left join o.devices as router
        left join router.linkInterfaces as li
        left join li.networkInterfaces as ni
        left join ni.network.networkInterfaces as ni2
        left join ni2.linkInterface.device as d
where router.forwarding = true and d.description = 'D'
```

```
building (device#(forwarding && network.device.description = "D"))
```

Сравнение текстов запросов

Найти устройства в здании главного корпуса:

- **HQL:**
  select d from Device d
  where d.occupancy.room.floor.building.name='ГК'

device#(building.name="ГК")

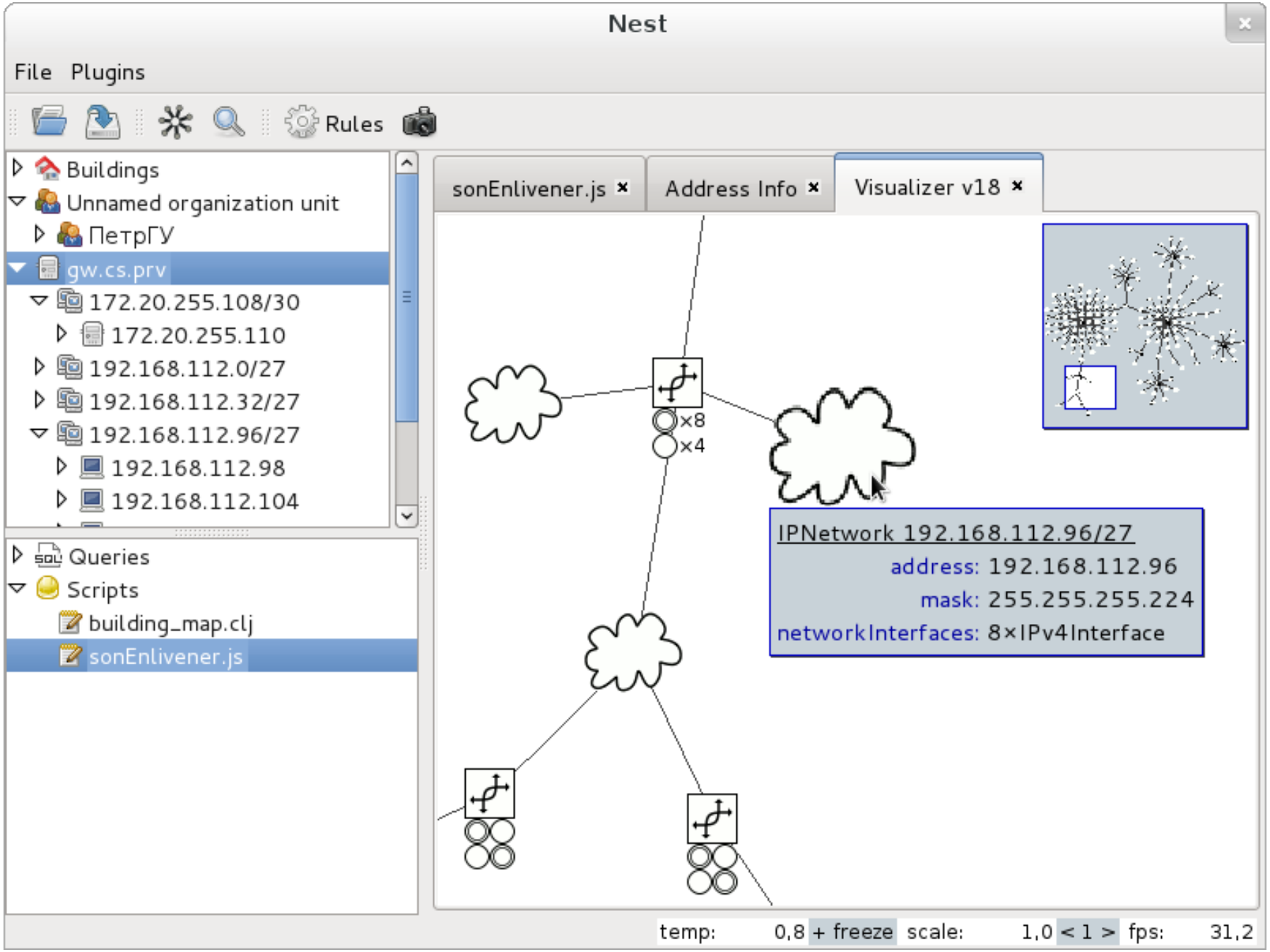
Найти здание, в котором находится устройство с именем "gw.cs.prv":

- **HQL:**
  select b from Building as b left join b.floors as f
  left join f.rooms as r left join r.occupancies as o
  left join o.devices as d where d.name='gw.cs.prv'

building#(device.name="gw.cs.prv")

Текущее состояние системы

- Разработан прототип системы Nest — экспериментальной платформы для исследования моделей и методов сетевого управления.
- Разработка для JVM на языках программирования Java и Clojure (19953 и 3158 строк кода соответственно, исключая комментарии).
- JavaScript и другие языки сценариев для расширения функциональности.
- Эксперименты в сети ПетрГУ. Подсистема построения графа сети обнаруживает более 1500 устройств, 168 IP-подсетей, более 3600 интерфейсов канального и сетевого уровня. Получаемый граф соответствует данным от администраторов Сети.

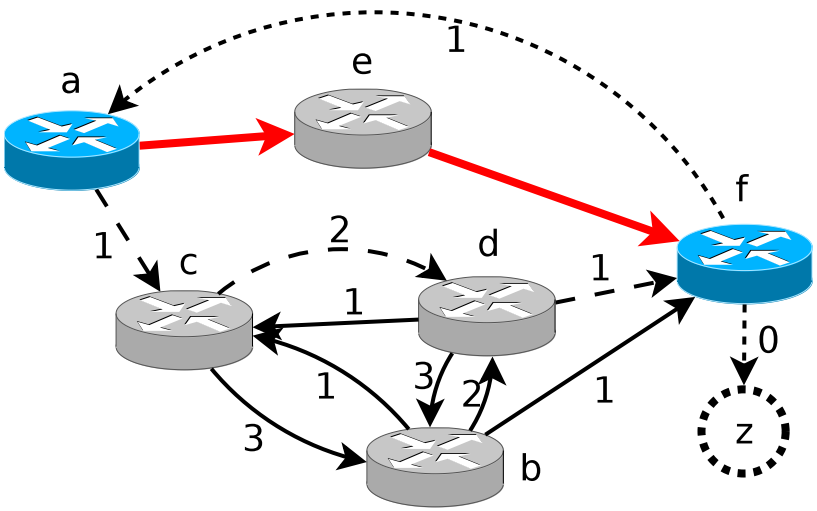

Nest
File Plugins
Rules
Buildings
Unnamed organization unit
ПетрГУ
gw.cs.prv
172.20.255.108/30
172.20.255.110
192.168.112.0/27
192.168.112.32/27
192.168.112.96/27
192.168.112.98
192.168.112.104
Queries
Scripts
building_map.clj
sonEnlivener.js
sonEnlivener.js
Address Info
Visualizer v18
×8
×4
IPNetwork 192.168.112.96/27
address: 192.168.112.96
mask: 255.255.255.224
networkInterfaces: 8×IPv4Interface
temp: 0,8 + freeze scale: 1,0 < 1 > fps: 31,2

Задача восстановления маршрутов MPLS

- Сеть MPLS (мультипротокольная коммутация по меткам):
  - Разделение трафика на классы эквивалентности FEC
  - Сопоставление метки пакету
  - Множество меток — таблица маршрутизации
- Потеря соединения:
  - нарушение линии связи или выход из строя узла
  - Задача построения обходного маршрута (поиск маршрута)
  - Задача переключения на новый маршрут (активация маршрута)
- Приложения:
  - Чувствительные к задержкам
  - Чувствительные к потере связности
- Требования:
  - Гарантированное время восстановления
  - Учет дополнительных критериев (число переходов, загруженность линий связи и узлов и др.)

Модель с характеристиками линий связи

- Характеристики линий связи
  (загруженность, приоритет, число промежуточных устройств)
- Добавление фиктивных вершины z и дуги (v, z)
- Вес дуг орграфа $A = (a_{wi})_{w \in N, i \in L}$, а дуг (v, u) и (v, z) равен 1 и 0
- Система одАНЛДУ
$$\begin{cases} x_{vu} + x_{vz} = \sum\limits_{i \notin I_v} a_{vi} x_i, \\ \sum\limits_{i \in I_w} x_i = \sum\limits_{i \notin I_w} a_{wi} x_i, \quad w \in N \setminus \{v\}. \end{cases}$$
- Решение — маршрут из u в v



$$\begin{cases} a: & x_{ac} = x_{fa} \\ b: & x_{bc} + x_{bd} + x_{bf} = 3x_{cb} + 3x_{db} \\ c: & x_{cb} + x_{cd} = x_{ac} + x_{bc} + x_{dc} \\ d: & x_{db} + x_{dc} + x_{df} = 2x_{bd} + 2x_{cd} \\ f: & x_{fa} + x_{fz} = x_{bf} + x_{df} \end{cases}$$

Базис Гильберта содержит 32 решения,
для 7 из которых $x_{fa} = 1$.

Благодарности

Н. С. Рузановой за внимание и поддержку данной работы,

М. А. Крышеню, А. С. Колосову, В. М. Димитрову и К. А. Кулакову за полезные идеи, обсуждения и упорную работу по реализации системы,

В. А. Пономареву, И. О. Суворову и И. А. Зиновику за системную поддержку и помощь в выполнении экспериментов.

Спасибо за внимание.

ybgv@cs.karelia.ru