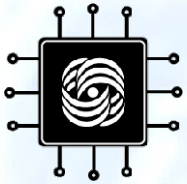
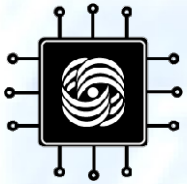


Имитационное моделирование в исследовании и разработке вычислительных систем и сетей

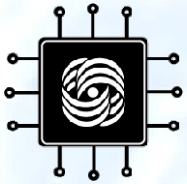
Лекция 2

Понятие модели. Основные виды моделей



Понятие модели

Модель – это объект, заменяющий исходный объект в ходе достижения заданных целей и при заданных предположениях.



Важные примеры целей моделирования (1)

- Исследование

  Установление закономерностей, связей между сущностями и явлениями

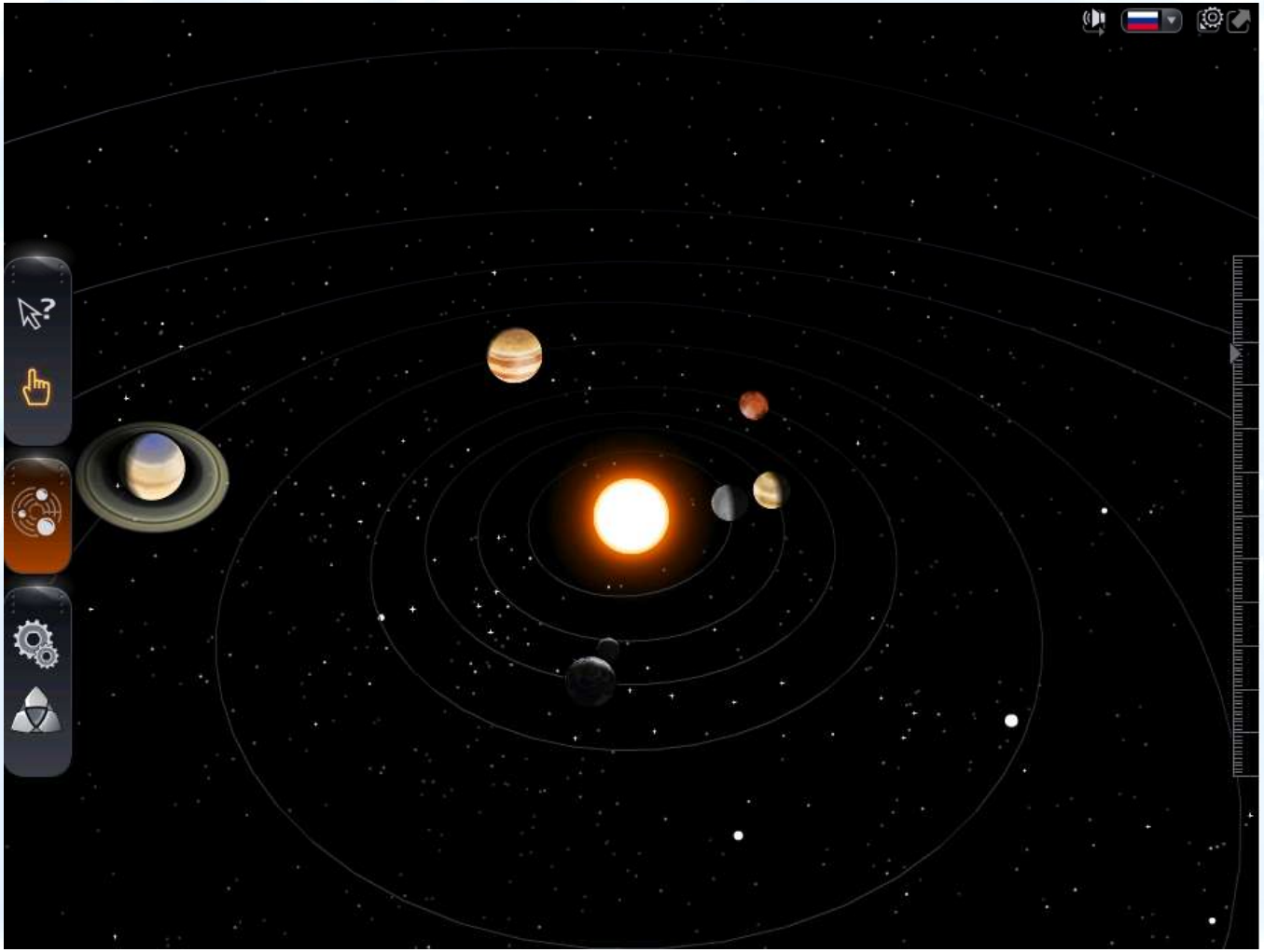
- Проектирование

  Предсказание характеристик объекта до его построения

- Обучение
  - (Действующее) наглядное пособие
  - Тренажёр

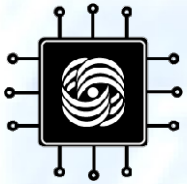
Наглядное пособие

https://space.utema.ru/sss/



Тренажёр

Тренажёр (2)

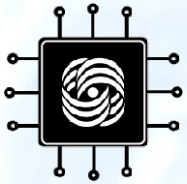


Важные примеры целей моделирования (2)

- Замещение моделируемого объекта в составе некоторой системы
  - эмуляторы системы команд процессора

…

- Взаимодействие между людьми, достижение взаимопонимания в отношении объекта
  - Карты, схемы, чертежи, …



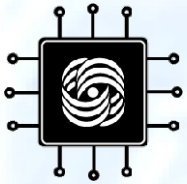
Понятие модели (2)

- Модель – это всегда **упрощение** исходного объекта (имеются упрощающие предположения) *(что-то отбрасываем и/или обобщаем);*
- Модель должна быть **адекватной** объекту (для заданной цели применения);
- Использование модели должно быть **проще** использования исходного объекта

См. также источник

http://simulation.su/uploads/files/default/2007-uch-posob-zamyatina-1.pdf

и другие учебные пособия на указанном сайте



Физические модели

- В модели используются материальные (физические) законы, процессы, устройства

Натурные модели: примеры

Компьютер в «коммерческом» исполнении вместо «промышленного» или «военного»

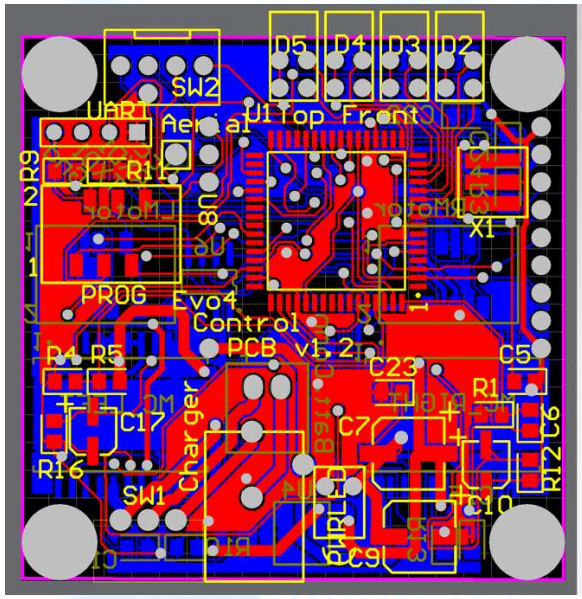
Объект

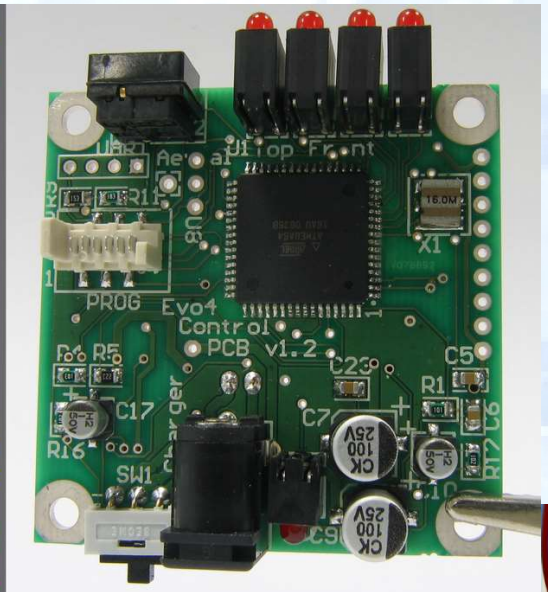
Модель

Натурные модели: примеры

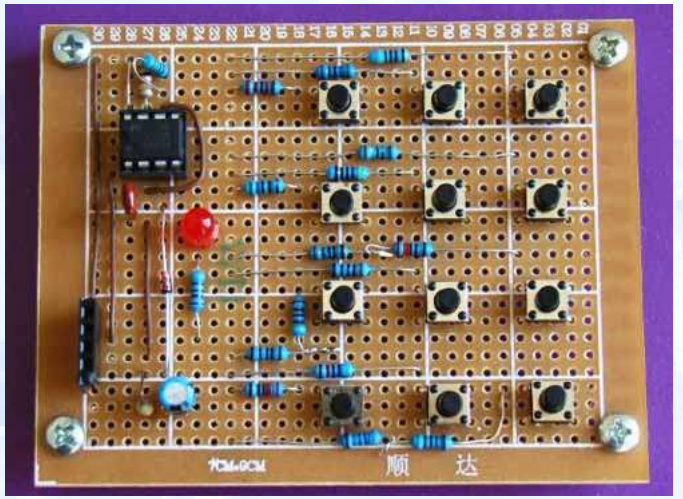
Электронная схема, собранная на макетной плате

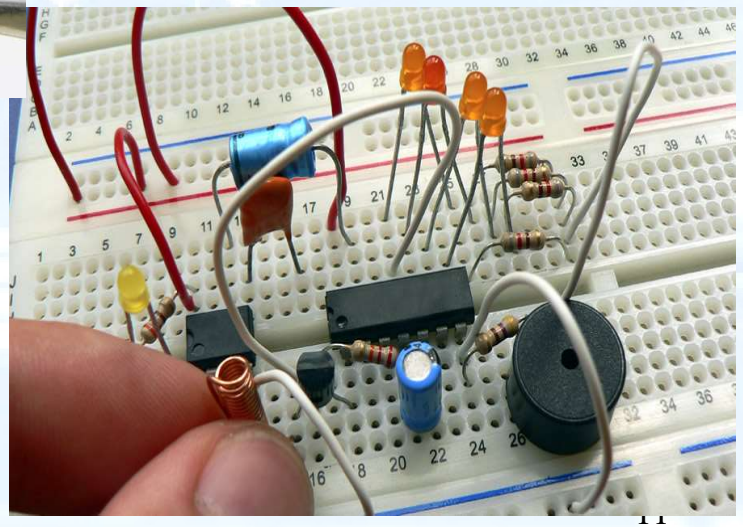
Объект





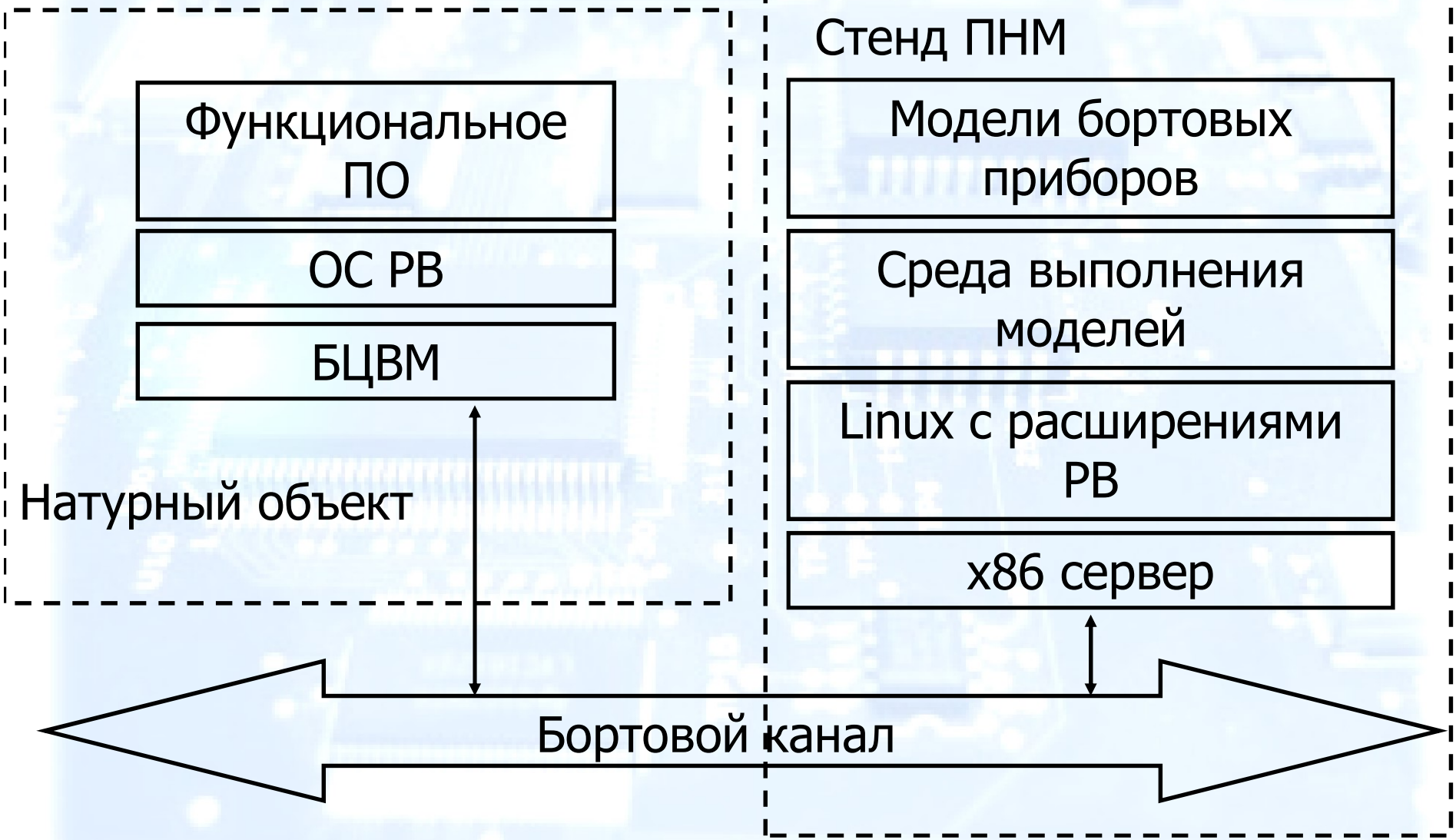
Модель



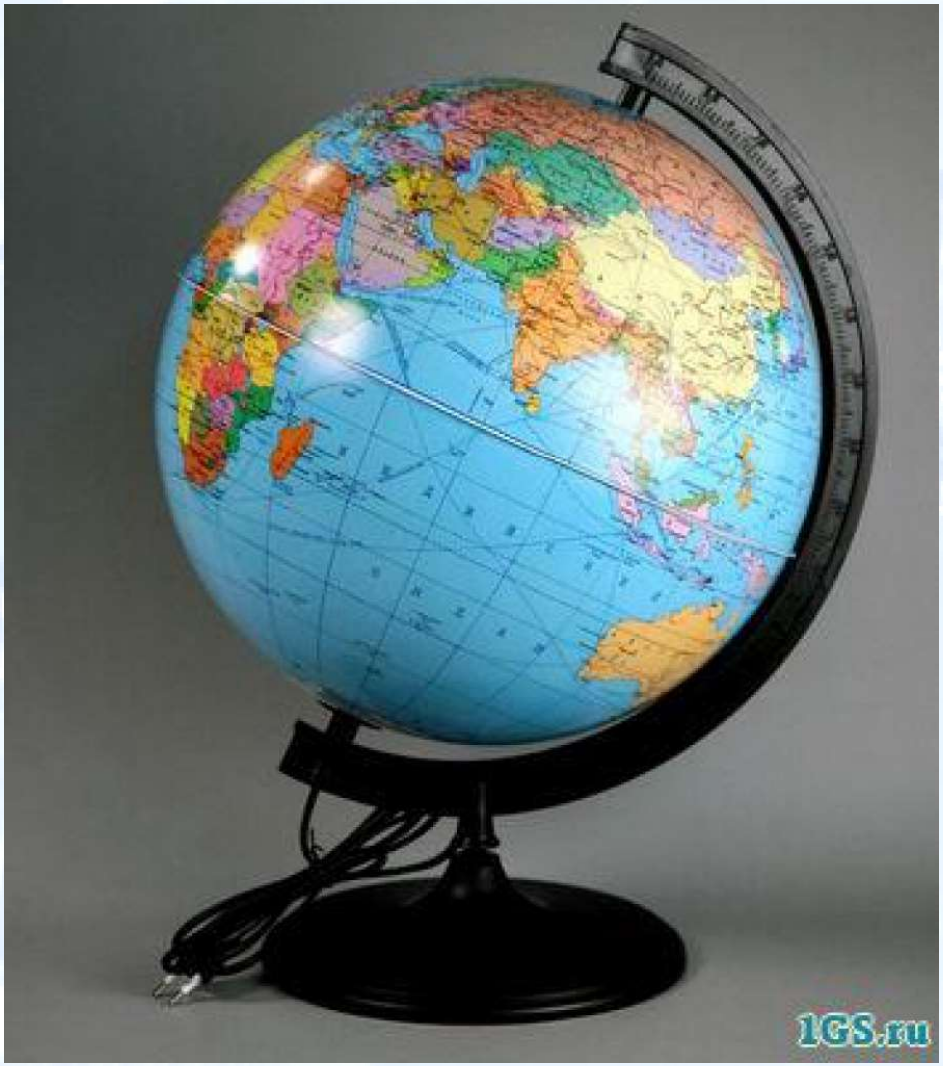


Автор: User Mike1024 - Photographed by User:Mike1024, Общественное достояние, https://commons.wikimedia.org/w/index.php?curid=1627980

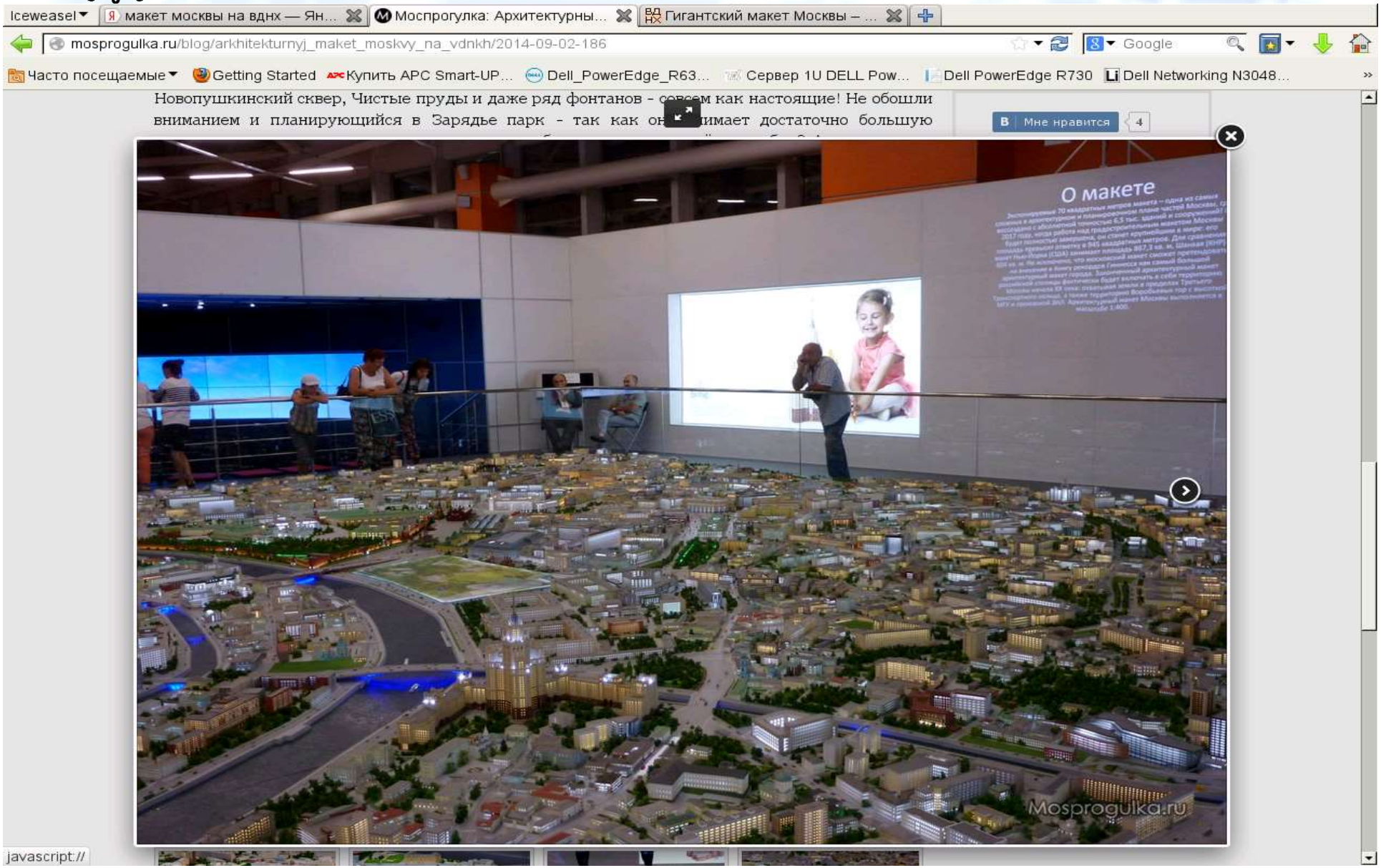
Полунатурные модели



Масштабные модели



Масштабные модели



Масштабные модели

Аналоговые модели

использование в модели иных физических явлений по отношению к объекту, но с теми же математическими формулировками законов работы

- Построение системы уравнений для объекта

– Построение электрической схемы по системе уравнений

– Включение схемы и измерение

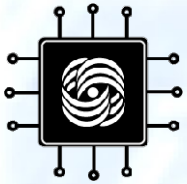
Аналоговые вычислительные машины:

http://computer-museum.ru/histussr/14.htm

Виды моделей (1)

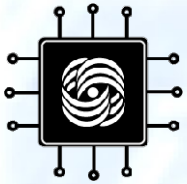
Физические модели

- Натурные: эксперименты с реальным объектом или его частью;
- полунатурные: часть объекта заменена моделью;
- масштабные: глобус, макеты зданий, макеты в аэро(гидро)динамической трубе
- аналоговые



Модель, макет, прототип

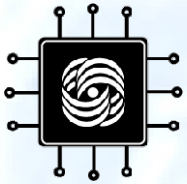
- Макет – модель, предварительный образец (© Ожегов)
- Как правило, уменьшенное в масштабе представление объекта
- Как правило, при создании макета от внутреннего содержания абстрагируются
- Цель создания макета – как правило, оценка взаимодействия макетируемого объекта с окружением, а не «проникания внутрь»
- Прототип программы для ЭВМ – «быстрая» реализация программы для проверки и уточнения функциональных требований к ней



Виды моделей (2)

- Знаковые
  - лингвистические (правила, кодексы …)
  - графические (схемы, чертежи)
  - математические
- Математические - по свойствам моделируемого объекта
  - структурные
  - функциональные

Доп. ссылки: [3], [4]

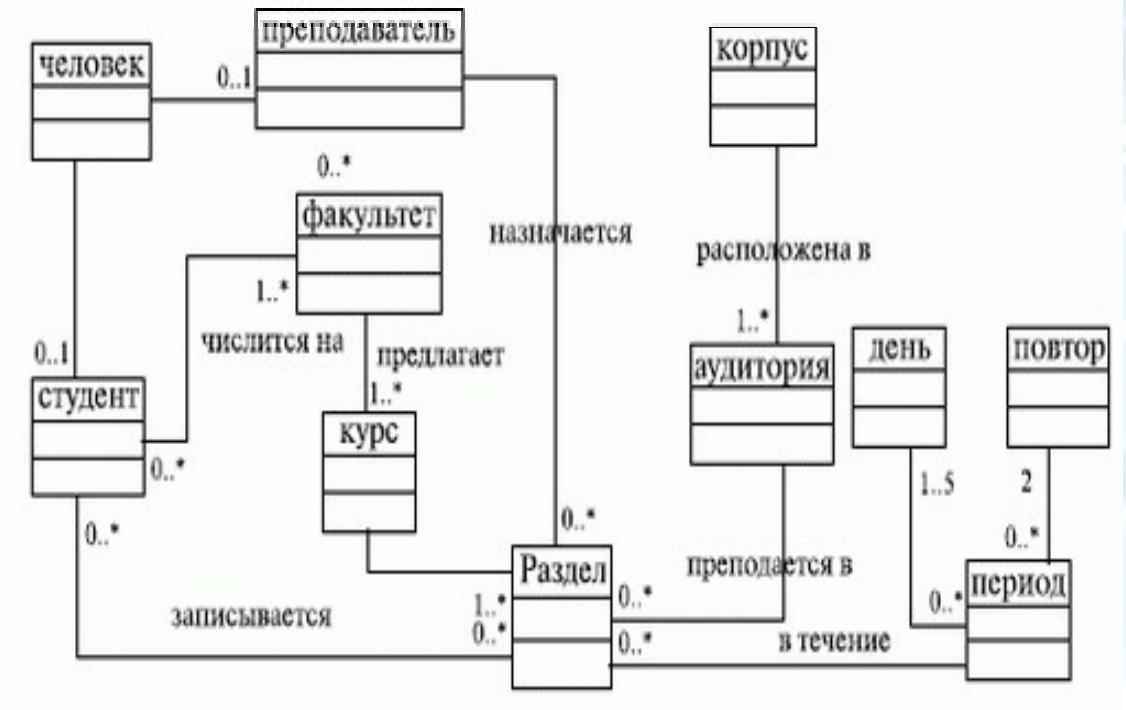


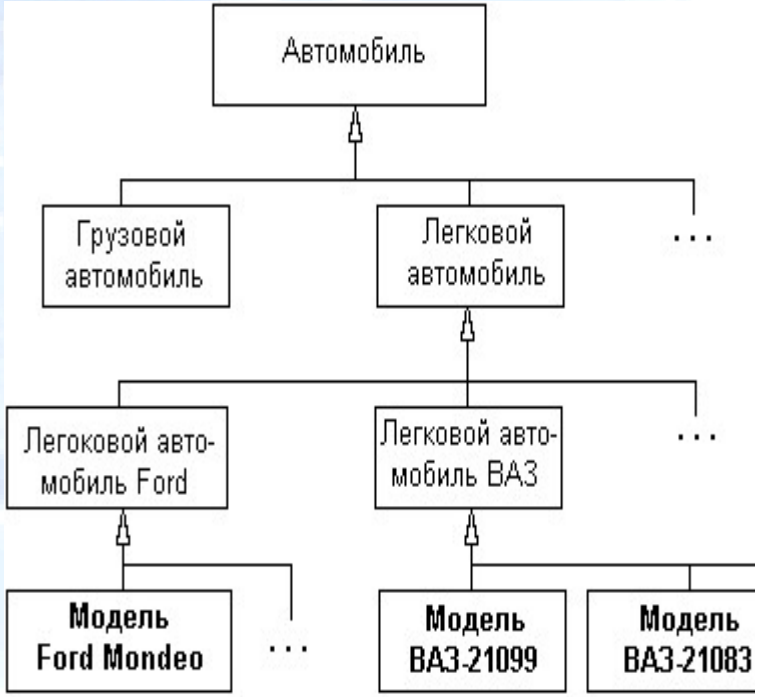
Структурные модели

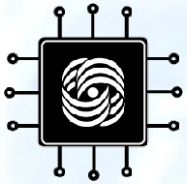
- Определяют:
  - Разбиение системы на компоненты
  - Связи между компонентами
- Используемые математические понятия
  - Иерархические древовидные модели
  - Графовые модели
  - Сетевые модели

Структурные модели (2)

- Важные примеры
  - Диаграммы классов UML
  - Диаграммы сущность-связь (ERD)

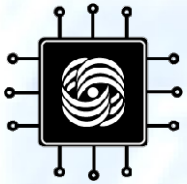






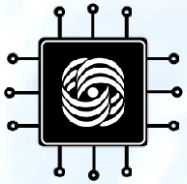
Важные примеры математических средств описания функциональных моделей

- Блок-схемы (диаграммы последовательности UML)
- Диаграммы состояний конечного автомата (state transition diagrams в языке UML)
- Сети Петри



Виды моделей (3)

- Математические функциональные:
- аналитические: построены математические зависимости результатов от входных данных;
- алгоритмические: построен алгоритм вычисления результатов по входным данным;
- имитационные: построен алгоритм, описывающий поведение объекта во времени с учётом входных воздействий на объект



Аналитические модели (2) производительности

V – количество операций в программе;

p – число процессоров (ядер);

t(p) – время выполнения на p ядрах;

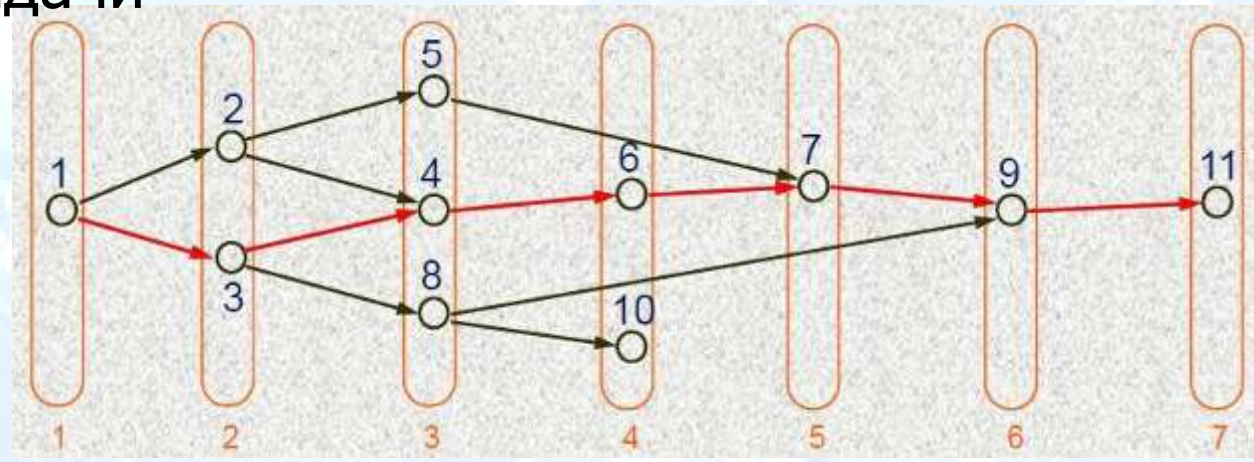
S(p) = t(1) / t(p) – ускорение;

- Полное распараллеливание: S(p) = p
- Пусть α – доля последовательных вычислений
- Закон Амдаля (Amdahl)

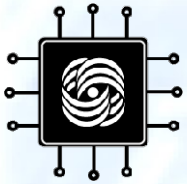
$S(p) \leq 1 / (\alpha + (1-\alpha)/p)$

Ярусно-параллельная модель параллельных вычислений

- Вершина графа– задача
- Вес вершины – время выполнения
- Ребро (стрелка) – зависимость по данным
- В одном ярусе – параллельно выполняемые задачи



Критический путь длиной Vкр



Аналитические модели (2)

Ярусно-параллельная форма

$t(p) \geq V / p + V_{кр} * (p-1)/p$

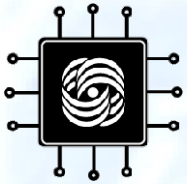
$s(p) \leq 1 / (1/p + V_{кр}/V * (p-1)/p)$

Конвейерные вычисления

p – длина конвейера; V – длина вектора;

$T(p) = p + V - 1$; $t(1) = pV$;

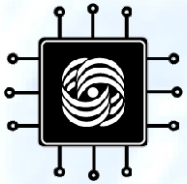
$a = p / (1 + (p-1)/V)$



Имитационное моделирование

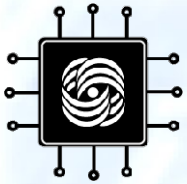
- Алгоритмические математические модели выражают связи выходных параметров с параметрами внутренними и внешними в форме алгоритма
- Имитационная математическая модель - это алгоритмическая модель, отражающая поведение исследуемого объекта во времени при задании внешних воздействий на объект.

Суть имитации – «пройти» заданный интервал времени работы системы, воспроизводя её поведение в промежуточные моменты времени



Имитационное моделирование (2)

Имитационная модель воспроизводит процесс функционирования системы во времени, причём имитируются элементарные явления, составляющие процесс, с сохранением их логической структуры и последовательности протекания во времени [Советов, Яковлев, с.34]



Модель ВС: немного терминологии

На входе

- параметры ВС
- модель рабочей нагрузки (возможно, со своими параметрами)

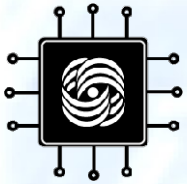
То, что можно варьировать в ходе эксперимента – ещё называется **факторами**

На выходе

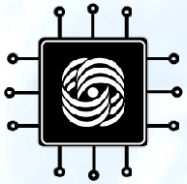
- временн**а**я диаграмма работы
- характеристики производительности

Внутри

- компоненты
- переменные состояний компонентов
- события



Пример модели с дискретными событиями (и с дискретным временем)



Описание моделируемой системы

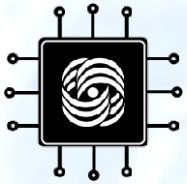
Сервер S обрабатывает два потока запросов от клиентов C1 и C2.

Для каждого из потоков заданы:

интервал времени между запросами dt1(n), dt2(n)

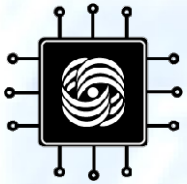
время обработки запроса

r1(n), r2(n)



Описание моделируемой системы (2)

- Сервер однопроцессорный
- Запрос, приходящий в момент обработки предыдущего запроса, ставится в очередь
- i-й клиент генерирует новый запрос через время $dt_i(n)$ не дожидаясь удовлетворения своего предыдущего запроса

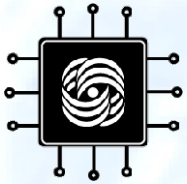


Цель моделирования

Определить на заданном интервале работы:

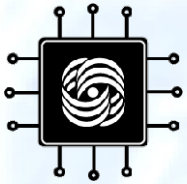
- максимальную длину очереди;
- загруженность сервера.

Для достижения цели нужно воспроизвести временну**ю** диаграмму работы сервера и клиентов



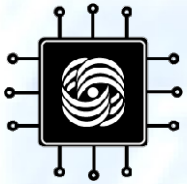
Предположения

- Реальные входные и выходные данные запроса (и их обработка) не моделируются
- Время обработки запроса записано в самом запросе
- Затраты времени на переключение процессов отсутствуют



Компоненты модели

- Клиент 1
- Клиент 2
- Сервер (с очередью)



Состояния

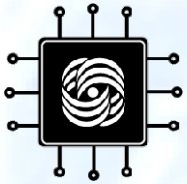
Клиенты: подготовка запроса

(так как выдача запроса мгновенна, состояние для неё не заводим)

Сервер:

IDLE – бездействие (ожидание);

RUN – выполнение запроса;



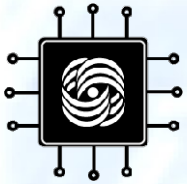
События

EV_REQ - создание запроса клиентом

EV_FIN - окончание обработки запроса

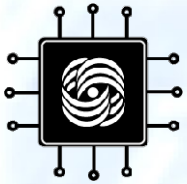
EV_INIT - начало моделирования

Событие начала обработки запроса можно для простоты не вводить (дальше будет понятно почему).



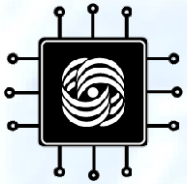
Моделирование посредством планирования событий

```
mod_time=0;
calendar.add( first_event, 0 ); //начальное событие(я)
while(!finish()) // пока не достигнуто условие окончания
{
  event=calendar.get_first_event(); // событие с мин. временем
  mod_time=event.time;
   switch(event.type)
  {
     case type1: /* обработка */ calendar.add(события,
     mod_time+интервал );
     case type2: ….
  }
}
```



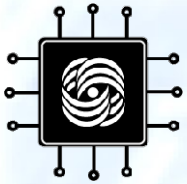
Моделирование посредством планирования событий

Полный текст модели см. в файле «simple_event.cpp» на странице курса



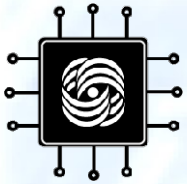
Моделирование посредством планирования событий

```
class Event  // событие в календаре
    {
    public:
      float time; // время свершения события
      int type;   // тип события
      int attr; // дополнительные сведения о событии в зависимости
    от типа
      Event(float t, int tt, int a) {time = t; type = tt; attr = a;}
    };
// События для нашей модели
#define EV_INIT 1
#define EV_REQ 2
#define EV_FIN 3
// состояния
#define RUN 1
#define IDLE 0
```



Календарь событий

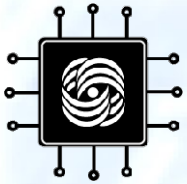
```
class Calendar: public list<Event*> //
   календарь событий
   {
     public:
     void put (Event* ev); // вставить
   событие в список с упорядочением по
   полю time
     Event* get (); // извлечь первое
   событие из календаря (с наименьшим
   модельным временем)
   }
```



Очередь запросов

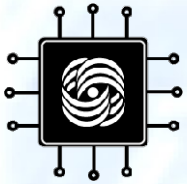
```
class Request // задание в очереди
    {
    public:
      float time;     // время выполнения задания без
    прерываний
      int source_num; // номер источника заданий (1
    или 2)
      Request(float t, int s) {time = t; source_num = s;}
    };

  typedef list<Request*> Queue; // очередь заданий к
    процессору
    float get_req_time(int source_num); // длительность
    задания
    float get_pause_time(int source_num); //
    длительность паузы между заданиями
```



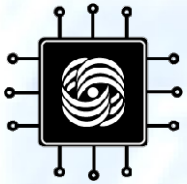
Подготовка моделирования

```
int main(int argc, char **argv )
    {
    Calendar calendar;
    Queue queue;
    float curr_time = 0;
    Event *curr_ev;
    float dt;
    int cpu_state = IDLE;
    float run_begin; //
    // начальное событие и инициализация
    календаря
    curr_ev = new Event(curr_time, EV_INIT, 0);
    calendar.put( curr_ev );
```



Основной цикл моделирования (1)

```
while((curr_ev = calendar.get()) != NULL )
   {
     cout << "time " << curr_ev->time << " type " <<
   curr_ev->type << endl;
     curr_time = curr_ev->time; // продвигаем время
     // обработка события
     if( curr_time >= LIMIT )break; // типичное
   дополнительное условие останова моделирования
     switch(curr_ev->type)
     {
     case EV_INIT:  // запускаем генераторы запросов
     calendar.put(new Event(curr_time, EV_REQ, 1));
     calendar.put(new Event(curr_time, EV_REQ, 2));
     break;
```

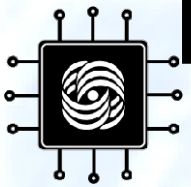


Основной цикл моделирования (2)

```
case EV_REQ:
    // планируем событие окончания обработки, если процессор свободен, иначе ставим в очередь
    dt = get_req_time(curr_ev->attr);
cout << "dt " << dt << " " << endl;
    if(cpu_state == IDLE)
{
  cpu_state = RUN;
  calendar.put(new Event(curr_time+dt, EV_FIN, curr_ev->attr));
  run_begin = curr_time;
}
    else
   queue.push_back(new Request(dt, curr_ev->attr));
  // планируем событие генерации следующего задания
    calendar.put(new Event(curr_time+get_pause_time(curr_ev->attr), EV_REQ, curr_ev->attr));
break;
```

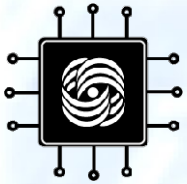
Основной цикл моделирования (3)

```
case EV_FIN:
    // объявляем процессор свободным и размещаем задание из очереди, если таковое есть
    cpu_state=IDLE;
    // выводим запись о рабочем интервале
    cout << "Работа с " << run_begin << " по " << curr_time << " длит. " << (curr_time-run_begin) << endl;
    if (!queue.empty())
    {
  Request *rq = queue.front();
  queue.pop_front();
  calendar.put(new Event(curr_time+rq->time, EV_FIN, rq->source_num));
  delete rq;
  run_begin = curr_time;
} break;
  } // switch
   delete curr_ev;
} // while
} // main
```



Недостатки «прямолинейного» событийного подхода

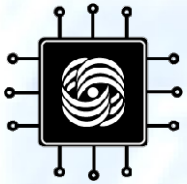
- Нет структуры
- Неудобства детализации модели
- Неудобства объединения моделей и построения иерархических моделей



Объектно-ориентированный подход

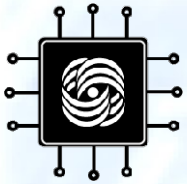
- Компонентам моделируемой системы ставим в соответствие объекты (будем называть их - процессы)
- Переменные состояния – располагаем внутри процессов
- События помещаем в календарь с указателем на процесс
- У каждого процесса – собственный обработчик событий
- События доставляются каждому процессу в порядке возрастания их модельного времени

Используем библиотеку SimpleSim:
http://www.inf.usi.ch/carzaniga/ssim/



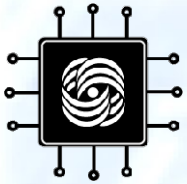
Применение объектного подхода (1)

```
class Req : public Event
{
public:
        Req(int num, float reqtime)
                : e_num(num), e_reqtime(reqtime) {}
                virtual ~Req() {}
        // получение атрибутов
        int getNum()
        {
                return e_num;
        }
        float getReqtime()
        {
                return e_reqtime;
        }
private:
        int e_num;
        float e_reqtime;
};
```



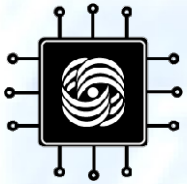
Применение объектного подхода (2)

```
class Fin : public Event
{
public:
        Fin() {}
        virtual ~Fin() {}
};
class Start : public Event
{
public:
        Start() {}
        virtual ~Start() {}
};
```



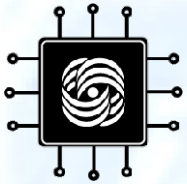
Применение объектного подхода (3)

```
class Client : public Process
{
public:
        Client(int num) : c_num(num) {}
        virtual ~Client() {}
        virtual void init() { Sim::self_signal_event(new Start, 0); }
        virtual void process_event(const Event* e)
        {
                const Event* ev;
                if ((ev = dynamic_cast<const Start*>(e)) != 0)
                {
                        float t = get_req_time(c_num);
                        Req* req = new Req(c_num, t);
                        Sim::signal_event(server_id, req);
                        // delete req;
```



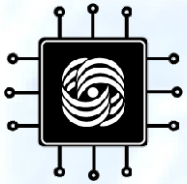
Применение объектного подхода (4)

```
                    Start* start = new Start;
                    Sim::self_signal_event(start, get_pause_time(c_num));
                                        }
                    else
                    {
                                cerr << "Client(" << c_num << ") received unknown
event" << endl;
                    }
                    } // process_event
          private:
                    int c_num;
          };
```



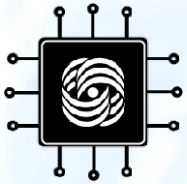
Применение объектного подхода (5)

```
class Server : public Process
        {
        private:        Queue queue;            int cpu_state;
                float run_begin;
        public:
                Server() {}
                ~Server() {}
                virtual void init() { cpu_state = IDLE; }
                virtual void process_event(const Event* e)
```



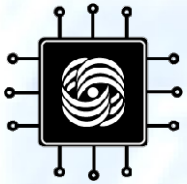
Применение объектного подхода (6)

```
{
        const Event* ev;
        if ((ev = dynamic_cast<const Req*>(e)) != 0)
        {
                if (cpu_state == IDLE)
                {
                        cpu_state = RUN;
                        run_begin = Sim::clock();
                        Fin* fin = new Fin();
                        Sim::self_signal_event(fin, ((const Req*)ev)-
>getReqtime());
                }
                else {
        queue.push_back(new Job(((const Req*)ev)->getReqtime(), ((const
Req*)ev)->getNum()));                                   }
        }                               else
```



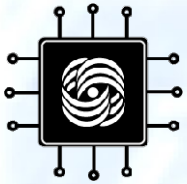
Применение объектного подхода (7)

```
if ((ev = dynamic_cast<const Fin*>(e)) != 0)                                    {
                                                        cpu_state = IDLE;
                                        // выводим запись о рабочем интервале
                                                cout << "Работа с " << run_begin << "
по " << Sim::clock() << " длит. " << (Sim::clock() - run_begin) << endl;
                                                if (!queue.empty())
                        {
                Job* rq = queue.front();
                        queue.pop_front();
                        Sim::self_signal_event(new Fin(), rq->time);
                        run_begin = Sim::clock();
                        cpu_state = RUN;
                        delete rq; }
                        }
                        else
                        {
                                                        cerr << "Server received
unknown event" << endl;
```



Применение объектного подхода (7)

```
int main(int argc, char **argv)
{
  Client c1(1), c2(2);
  Server srv;
  srand(2019);
  server_id = Sim::create_process(&srv);
  Sim::create_process(&c1);
  Sim::create_process(&c2);
  Sim::set_stop_time(LIMIT);
  Sim::run_simulation();
  return 0;
} // main
```

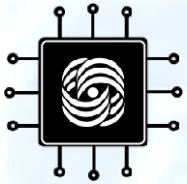


Задание 2 (4 варианта)

Отредактировать модель (в любом из вариантов simple_event, simple_event_ssim):

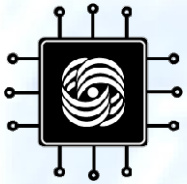
- а,б) добавить второй сервер, работающий параллельно с первым, и балансировщик нагрузки (в двух вариантах);
- в,г) добавить третьего клиента и планировщик (в двух вариантах)

Подробности в текстовом документе «Задание_2.doc»



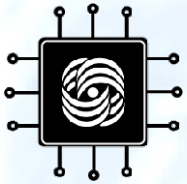
Особенности ИМ

- По сравнению с аналитическими моделями:
  - универсальность применения (+);
  - результат только для конкретного набора входных данных (-);
- По сравнению с «программами общего назначения»:
  - «ТЗ формируется по ходу дела…»
  - Необходимость поддержки понятий предметной области в средстве моделирования



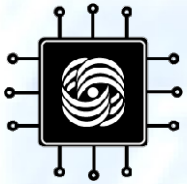
Этапы создания ИМ (1)

- Анализ требований и проектирование
  - Постановка цели моделирования
  - Построение концептуальной модели
  - Проверка достоверности концептуальной модели
- Реализация модели
  - Выбор языка и средств моделирования
  - Программирование модели
  - Отладка модели



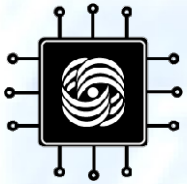
Этапы создания ИМ (2)

- Проведение экспериментов и анализ результатов
  - Планирование экспериментов
  - Прогон модели
  - Анализ результатов и формулирование выводов



Концептуальная модель

- Как правило, знаковая (лингв. или графич. модель) различной степени формализованности
- Построение – процесс неформальный, интуитивный
- Определяет структуру моделируемой системы, алгоритмы функционирования компонентов, их состояния, порядок взаимодействия, и т.д.
- Представляет собой решение по абстракции и упрощению иследуемой системы
- «техническое задание» на программирование имитационной модели

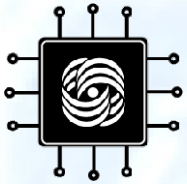


ИМ по способам продвижения времени

- С постоянным шагом
  - Непрерывные модели
  - Потактовые модели
- От события к событию
  - Дискретно-событийные модели
- Гибридные модели
  - Совместная работа компонентов разного рода
  - Переключение режимов «непрерывного» компонента

Литература

1. Р. Шеннон. Имитационное моделирование систем – искусство и наука. – М:Мир, 1978. Глава 1.
2. Гультяев А.К. Matlab 5.2. Имитационное моделирование в среде Windows. Практическое пособие. Главы 1-2.
3. О.М. Замятина. Моделирование систем – Томск, 2009.
4. Б.А. Советов, С.А. Яковлев. Моделирование систем – М:Высшая школа, 2001.



Спасибо за внимание!