

---

# TinyOs. Simulators. Programming for TinyOS.

Сайт курса:

<http://www.sumkino.com/wsn/course>

---

Садков Александр

Аспирант РФ

[axel@wl.unn.ru](mailto:axel@wl.unn.ru)

<http://www.wl.unn.ru>

---

# ΠΛΑΗ

- TinyOS
  - Simulators
    - TOSSIM
    - SNS
    - Aurora
  - Programming for TinyOs
-

---

# TinyOs

Почему нам нужна новая операционная система?

- Традиционные операционные системы:
    - Большие!
    - Многопоточковая архитектура – большие объемы памяти.
    - Пространство ядра системы и пользовательское пространство разделены.
    - Нет энергетических ограничений.
    - Большие вычислительные ресурсы.
-

# TinyOs

## Ограничения сенсорных сетей?

- Мощность.
- Ограниченный размер памяти.
- Слабый микропроцессор.
- Небольшие размеры.
- Ограниченный параллелизм.
- Передача информации с помощью маломощных трансиверов:
  - Малая скорость передачи
  - Короткие расстояния

---

# TinyOs

Какая операционная система нам нужна?

- Занимающая небольшой объем памяти.
  - Эффективная в плане энергии и вычислений.
  - Передача информации должно быть фундаментальным.
  - Real-time.
  - Достаточно гибкая.
-

---

# TinyOs

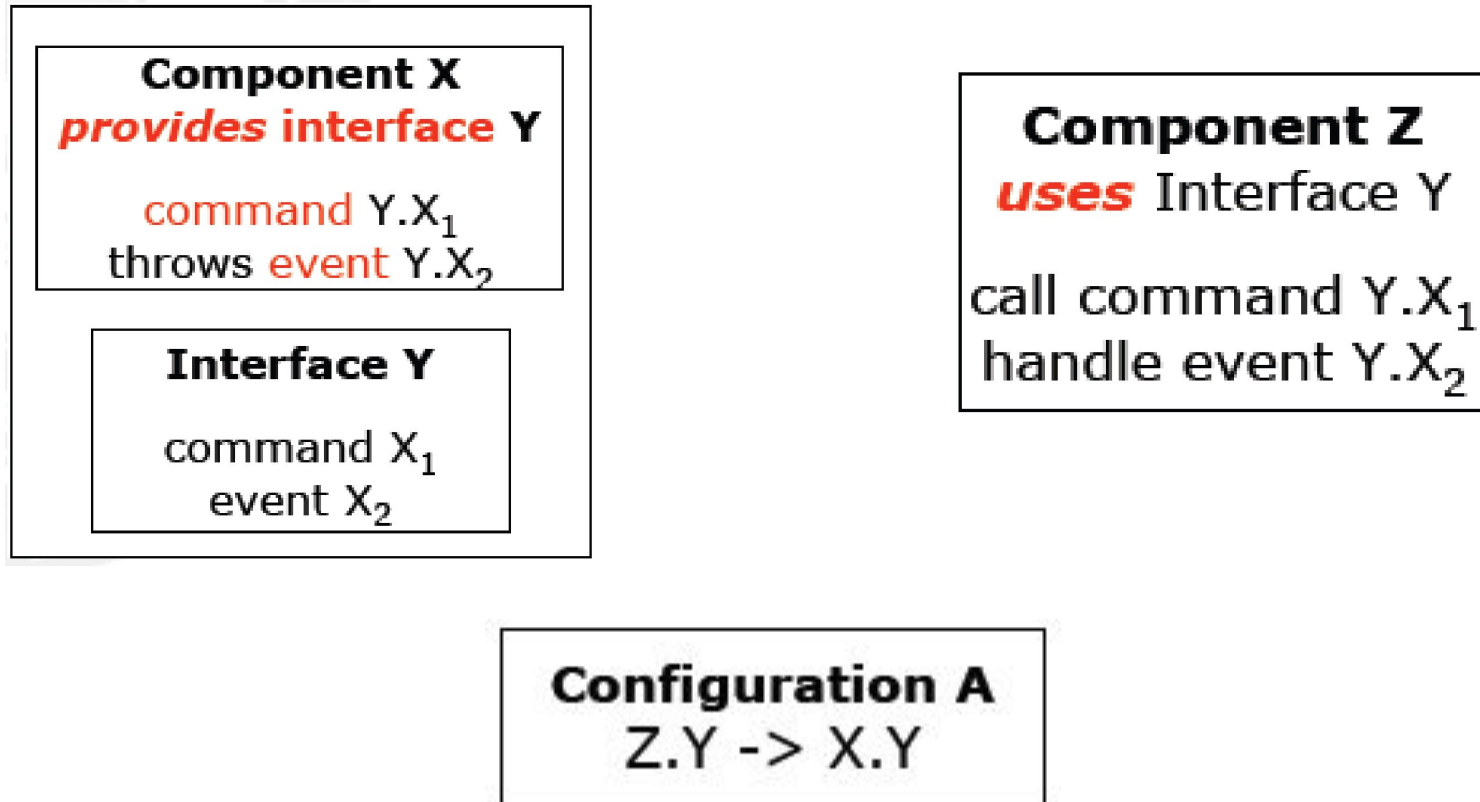
- Конкуренция : Использует event-driven архитектуру.
  - Модульность.
    - Приложения состояются из нескольких компонентов.
    - OS + Приложения компилируются в единый исполняемый модуль.
  - Использование event/command модели.
  - FIFO и системы планирования.
  - Нет границы между ядром системы и приложениями.
-

---

# TinyOs and NesC

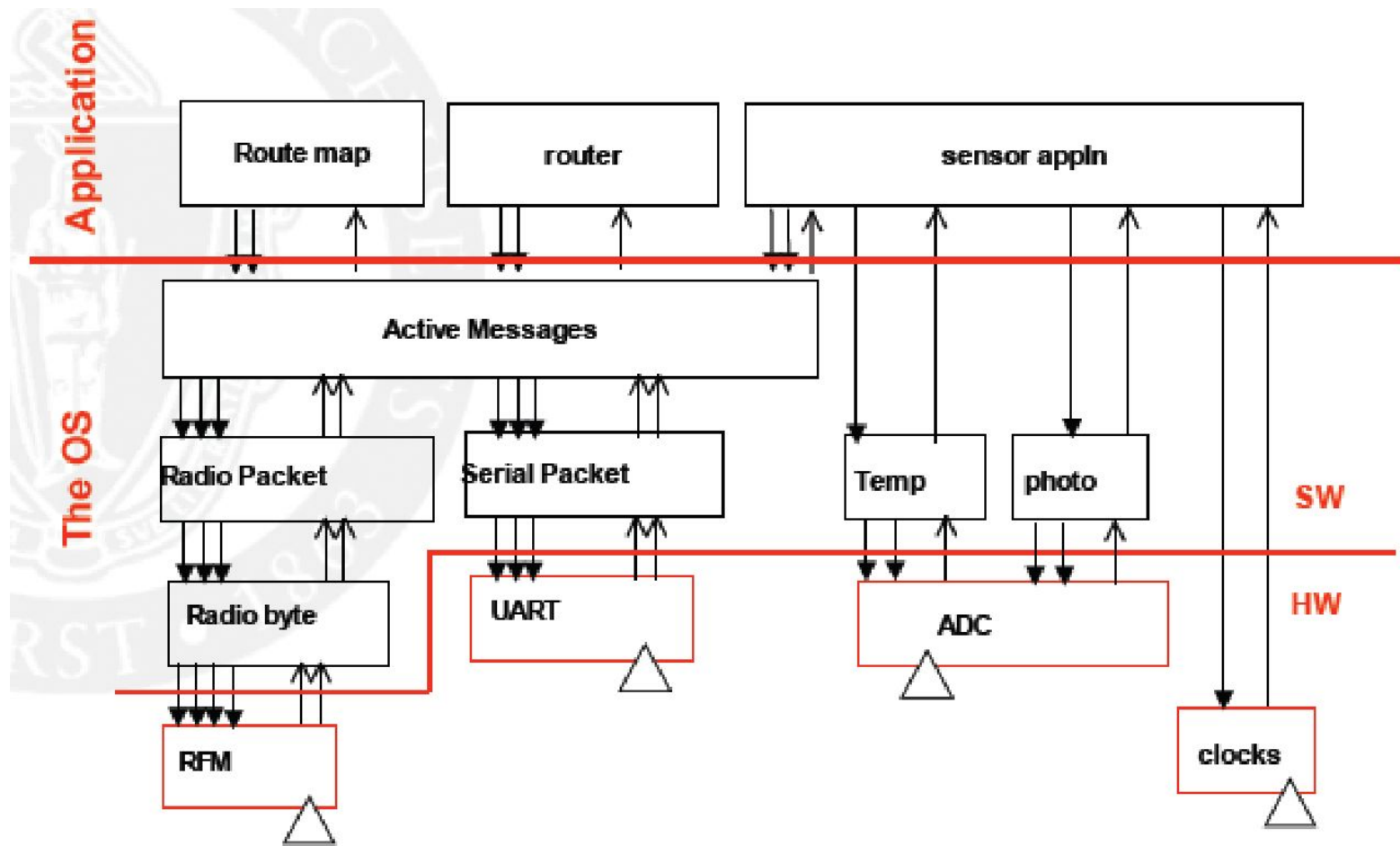
- Новый язык NesC, основная единица кода – компонент.
  - Компонент:
    - Обрабатывает команды (**commands**)
    - Посылает события (**events**)
    - Имеет Frame для хранения локального состояния.
    - Использует задачи (**tasks**) для конкуренции.
  - Компонент предоставляет интерфейсы (**interfaces**).
    - Используются другими компонентами для связи.
  - Компоненты связываются между собой с помощью конфигуратора (**configuration**).
-

# TinyOs and NesC





# Application = Graph of Components



---

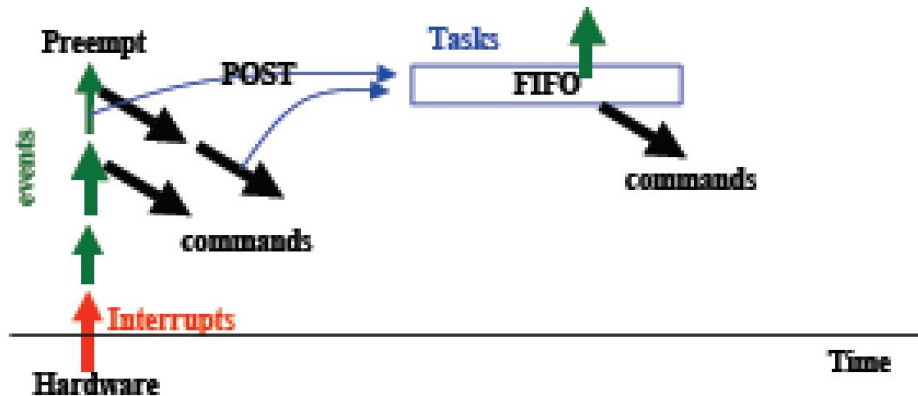
# Commands/Events/Tasks

- Команды (**commands**).
    - Должны быть не блокирующие.
    - То есть, берет параметры, начинает обработку и возвращается к приложению; откладывает выполнение трудоемких задач путем объявления задач (**tasks**).
  - События (**events**):
    - Могут вызывать команды, сигнализировать другие события, объявлять задачи, но не могут быть вызваны командами.
    - Заменяют задачи, но не наоборот.
  - Задачи (**tasks**):
    - Планирование FIFO.
    - Не могут быть заменены другими задачами, но заменяются событиями.
    - Используются для выполнения вычислительно интенсивных задач.
    - Могут быть объявлены командами или событиями.
-

# Scheduler

- В Планировщике два уровня: события (**events**) и задачи (**tasks**).
- Планировщик – простой FIFO
- Задача не может быть заменена другой задачей
- Событие может заменить задачу (имеет больший приоритет)

```
main {  
  ...  
  while(1) {  
    while(more_tasks)  
      schedule_task;  
    sleep;  
  }  
}
```



---

# Simulators

---

---

# Simulators

- TOSSIM
  - Avrora
  - EmStar
  - SNS
-

---

# TOSSIM

- Использует реалистичные модели канала.
- Высокая повторяемость экспериментов
- Масштабируемость до тысяч узлов.
- Компилирует напрямую TinyOs код.
- Помогает отловить наиболее общие баги.
- Не очень подходит для моделирования «временных» задач.

Philip Levis, Nelson Lee, Matt Welsh, and David Culler [TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications](#). In Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys 2003).

---

---

# Avrora

- Для AVR (Atmel) микроконтроллеров (узлы Mica2).
- Тестирование программного кода перед установкой на железо.
- Профилировщик – позволяет лучше понять поведение программы.
- Дебаггер.
- Моделирование энергозатрат.

<http://compilers.cs.ucla.edu/avrora/simulator.html>

---

---

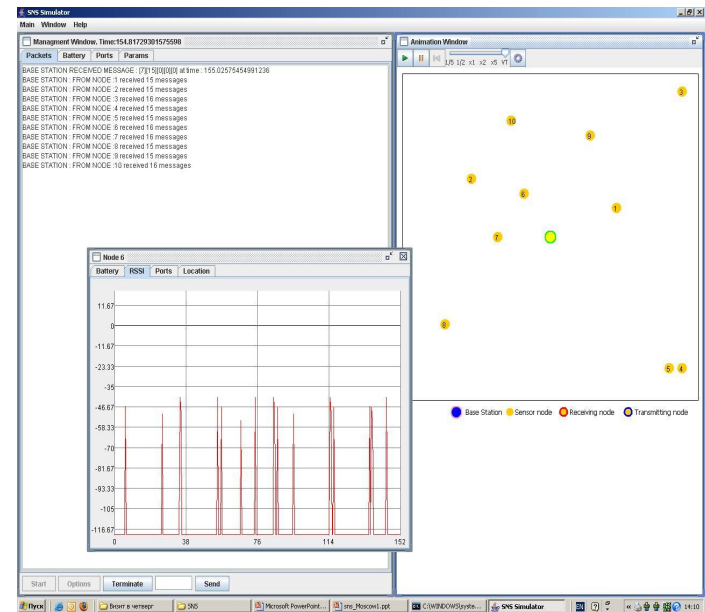
# EmStar

- Моделирует в коде.
  - От моделирования до реального эксперимента.
  - Имеет интерфейс для подключения реальных узлов.
  - Работает не со всеми узлами.
  - Невысокая скорость работы.
-



# SNS

- Детальное моделирование физического уровня.
- Модульность.
- Интуитивно понятный интерфейс.
- Не предназначен для моделирования кода. (не эмулятор)
- Ограниченная масштабируемость.
- Невысокая скорость работы.



---

# Programming for TinyOS

---

---

# Programming for TinyOS

- Нет динамической памяти.
  - Однозадачность; event-driven
  - Команды (commands) и события (events) должны делать небольшую работу.
  - Объявление задач(tasks) для выполнения долгих процессов.
  - Программный код должен представлять из себя машину состояний.
-

# Programming for TinyOS

- X вызывает компонент Y для чтения нескольких байт с флеш с помощью команды `Y.multiRead()`.
- Объявляем задачу A чтения первого байта из памяти вызывая `Flash.Read()`
- Возвращаем статус ОК.
- Когда возвращается `Flash.ReadDone()`, объявляем задачу A для чтения второго байта и т.д.
- Когда все байты считаны, сигнализируем событие (signal event) `Y.multiReadDone()`
- Если произошли какие-то ошибки сигнализируем событие `Y.multiReadDone()` с кодом ошибки.

# Programming for TinyOS

- Наименование файлов

- Расширение nesC файлов: .nc

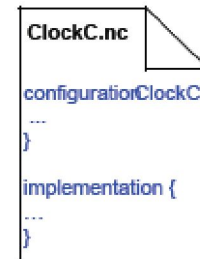
- Clock.nc: либо интерфейс, либо конфигуратор

- ClockC.nc: конфигуратор

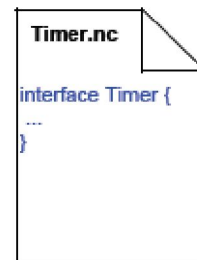
- ClockM.nc: модуль



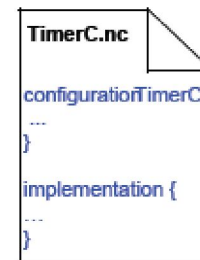
```
interface Clock {
  ...
}
```



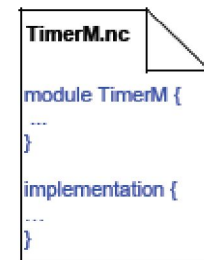
```
configuration ClockC {
  ...
}
implementation {
  ...
}
```



```
interface Timer {
  ...
}
```



```
configuration TimerC {
  ...
}
implementation {
  ...
}
```



```
module TimerM {
  ...
}
implementation {
  ...
}
```

# Programming for TinyOS

- Интерфейсы связывают между собой компоненты

```
interface StdControl
{
    command result_t init();
    command result_t start();
    command result_t stop();
}
```

```
interface X
{
    command result_t doSomething();
    event result_t doSomethingDone();
}
```

# Programming for TinyOS

- Модули могут реализовывать один или несколько интерфейсов.
- Модули могут использовать другие интерфейсы.

```
module Provider
{
  provides interface StdControl;
  provides interface X;
  uses interface Z;
}
implementation
{
  // C code
  ....
}
```

MyComp.nc

# Programming for TinyOS

- Интерфейсы могут передаваться с параметрами.
- Один и тот же интерфейс может несколько разных «вхождений».

```
module Provider
{ ...
  provides interface X[uint8_t id];
}
implementation {
  uint8_t idd;

  task void signaler()
  {
    signal X.doSomethingDone[idd]();
  }

  command result_t X.doSomething
  [uint8_t id] ()
  {
    idd = id;
    post signaler();
    return SUCCESS;
  }
}
```



# Programming for TinyOS

## Configuration

- Два компонента связываются(взаимодействуют) между собой с помощью интерфейсов.
- Интерфейсы в *user* компоненте связываются (**wired**) с такими же интерфейсами на *provider* компоненте.

- Три типа связи:

- $endpoint_1 = endpoint_2$
- $endpoint_1 \rightarrow endpoint_2$
- $endpoint_1 \leftarrow endpoint_2$   
(equivalent:  $endpoint_2 \rightarrow endpoint_1$ )

### Application.nc

```
configuration Application {  
}  
implementation {  
  components Main, Provider, User, SomeComp;  
  
  Main.StdControl -> Provider.StdControl;  
  User.X -> Provider.X;  
  Provider.Z -> SomeComp.Z;  
}
```

- Компонент, который использует интерфейс должен быть слева, компонент, который предоставляет интерфейс справа.

---

# Programming for TinyOS

---



---

# Литература

- Localization in Sensor Networks A. Savvides, L. Girod, M. Srivastava, and D. Estrin, Book Chapter, Wireless Sensor Networks, Edited by Znati, Radhavendra and Sivalingam.
  - Power-Efficient Sensor Placement and Transmission Structure for Data Gathering under Distortion Constraints D. Ganesan, R. Cristescu and B. Beresford-Lozano
  - Coverage in wireless ad hoc sensor networks X. Li, P. Wan, and O. Frieder
  - Impact of Heterogeneous Deployment on Lifetime Sensing Coverage in Sensor Networks, Jae-Joon Lee, Bhaskar Krishnamachari, C.C. Jay Kuo
  - Topology Control for Wireless Sensor Networks J. Pan, Y.T. Hou, L. Cai, Y. Shi, and S.X. Shen
-

---

The End

---