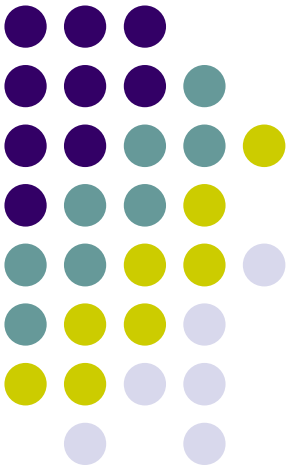
Lab course: Programming Sensor Networks

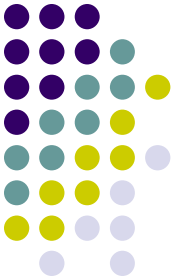
Lecture-2

Introduction to JVM and TakaTuka



Java Pros & Cons

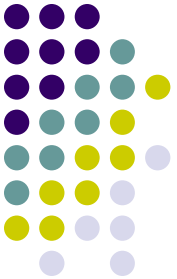
University of Freiburg
Institute of Computer Science
Computer Networks and Telematics
Prof. Christian Schindelhauer



- Pros
  - User friendly object oriented language
  - Many users and tools
- Cons
  - Slow
    - Bad for battery lifetime
  - Large binary file and virtual machine code
    - Cannot fit programs on mote flash
  - Large RAM requirements

TakaTuka JVM

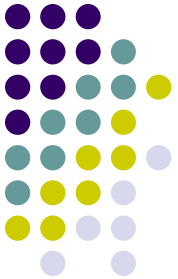
University of Freiburg
Institute of Computer Science
Computer Networks and Telematics
Prof. Christian Schindelhauer



- Objectives
  - To mitigate Java Cons
  - For tiny devices with very small resources
  - Support dozen of devices currently using TinyOS

TakaTuka Optimization

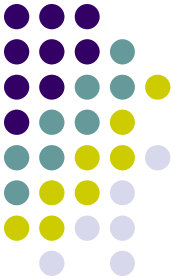
University of Freiburg
Institute of Computer Science
Computer Networks and Telematics
Prof. Christian Schindelhauer



- Flash optimization
  - To reduce the size:
    - Java binary
    - virtual machine
  - Current status:
    - Nearly finished
- RAM optimization
  - Should finished by June/July 2009
    - New garbage collection strategy
    - New operand stack management algorithm

TakaTuka Optimization

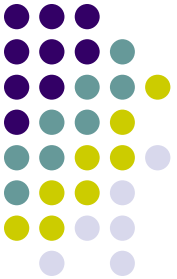
University of Freiburg
Institute of Computer Science
Computer Networks and Telematics
Prof. Christian Schindelhauer



- Performance optimization
  - Not yet started
  - Will start work after RAM optimization

Summary of Flash Optimization

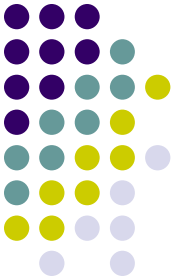
University of Freiburg
Institute of Computer Science
Computer Networks and Telematics
Prof. Christian Schindelhauer



- Class-file has three parts
  - Bytecode
    - Each method has a array of instructions that are executed by a JVM interpreter (or Just-in-Time compiler)
  - Constant pool
    - A set constant values used by the bytecode
  - Structural informations and flags

Constant Pool (CP) Optimization

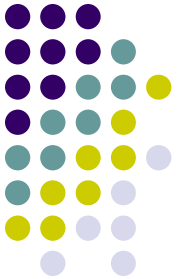
University of Freiburg
Institute of Computer Science
Computer Networks and Telematics
Prof. Christian Schindelhauer



- Typical CP has unique information per classfile
  - TakaTuka CP has unique information per project
- Typical CP has information required for debugging, extending a program, loading, running etc.
  - TakaTuka CP has only information required for running a program

Bytecode Optimization

University of Freiburg
Institute of Computer Science
Computer Networks and Telematics
Prof. Christian Schindelhauer

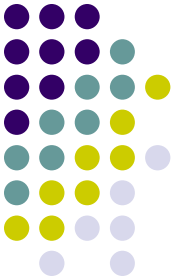


- Each bytecode instruction consist of:
  - **opcode** i.e. operation code
  - zero or more **operand.**
    - parameters for an operation

- Example:
  - ISTORE 5
    - ISTORE is *mnemonic* assigned to an opcode
    - 5 is operand

Bytecode Optimization

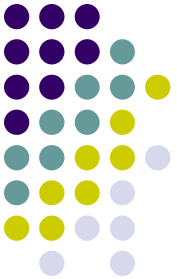
University of Freiburg
Institute of Computer Science
Computer Networks and Telematics
Prof. Christian Schindelhauer



- Three kind of optimization
  - Reduce the size of operand when possible
    - Example
      - IFNE 0x0005 → IFNE 0x05
  - Make operand explicit
    - IFNE 0x0005 → IFNE_0x05
  - Create a new instruction composed of many instructions
    - IFNE_GOTO 0x05 0x03 (two instruction combines)
- The optimized bytecode is still interpretable by a special interpreter

Structural Information Optimization

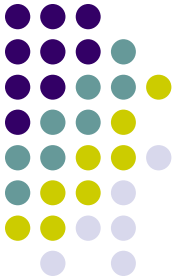
University of Freiburg
Institute of Computer Science
Computer Networks and Telematics
Prof. Christian Schindelhauer



- Instead of typical classfile format
  - TakaTuka Java binary is stored in a TUK file
  - TUK file has all the class files of a project

Lab-2

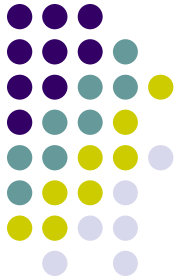
University of Freiburg
Institute of Computer Science
Computer Networks and Telematics
Prof. Christian Schindelhauer



- Blink LEDs using TinyOS code
- Application source */opt/tinyos-2.1.0/apps/Blink*
- Application code has three files
  - A module – BlinkC.ns
  - A configuration – BlinkAppC.ns
  - A makefile -- Makefile

Lab-2: Module

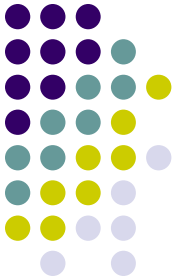
University of Freiburg
Institute of Computer Science
Computer Networks and Telematics
Prof. Christian Schindelhauer



```
#include "Timer.h"

module BlinkC
{
	uses interface Timer<TMilli> as Timer0; //see the interfaces at /opt/tinyos-2.1.0/tos/interfaces
	uses interface Timer<TMilli> as Timer1;
	uses interface Timer<TMilli> as Timer2;
	uses interface Leds;
	uses interface Boot;
}
implementation
{
	event void Boot.booted()
	{
		call Timer0.startPeriodic( 250 );
		call Timer1.startPeriodic( 500 );
		call Timer2.startPeriodic( 1000 );
	}

	event void Timer0.fired()
	{
		dbg("BlinkC", "Timer 0 fired @ %s.\n", sim_time_string());
		call Leds.led0Toggle();
	}

	event void Timer1.fired()
	{
		dbg("BlinkC", "Timer 1 fired @ %s \n", sim_time_string());
		call Leds.led1Toggle();
	}

	event void Timer2.fired()
	{
		dbg("BlinkC", "Timer 2 fired @ %s.\n", sim_time_string());
		call Leds.led2Toggle();
	}
}
```

Lab-2: Configuration

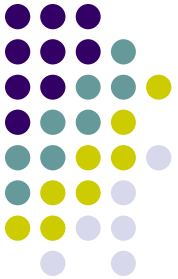
University of Freiburg
Institute of Computer Science
Computer Networks and Telematics
Prof. Christian Schindelhauer



```
configuration BlinkAppC
{
}
implementation
{
        components MainC, BlinkC, LedsC;
        components new TimerMilliC() as Timer0;
        components new TimerMilliC() as Timer1;
        components new TimerMilliC() as Timer2;


        BlinkC -> MainC.Boot;

        BlinkC.Timer0 -> Timer0;
        BlinkC.Timer1 -> Timer1;
        BlinkC.Timer2 -> Timer2;
        BlinkC.Leds -> LedsC;
}
```

Lab-2: Blink LEDs using TakaTuka

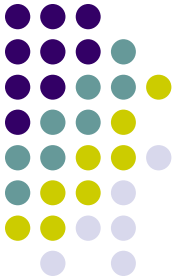
University of Freiburg
Institute of Computer Science
Computer Networks and Telematics
Prof. Christian Schindelhauer



- Use login "*guest*" and password "*ttguest9*"
- Get the TakaTuka from release
- Follow the tutorial given at page
  - *http://cone.informatik.uni-freiburg.de/people/aslam/takatuka/HelloWorldTutorial.html*

The End

University of Freiburg
Institute of Computer Science
Computer Networks and Telematics
Prof. Christian Schindelhauer



- Thank you for listening