

Component-Based Development of Networked Embedded Applications

Peter Volgyesi and Akos Ledeczi

Institute for Software Integrated Systems, Vanderbilt University

{peter.volgyesi, akos.ledeczi}@vanderbilt.edu

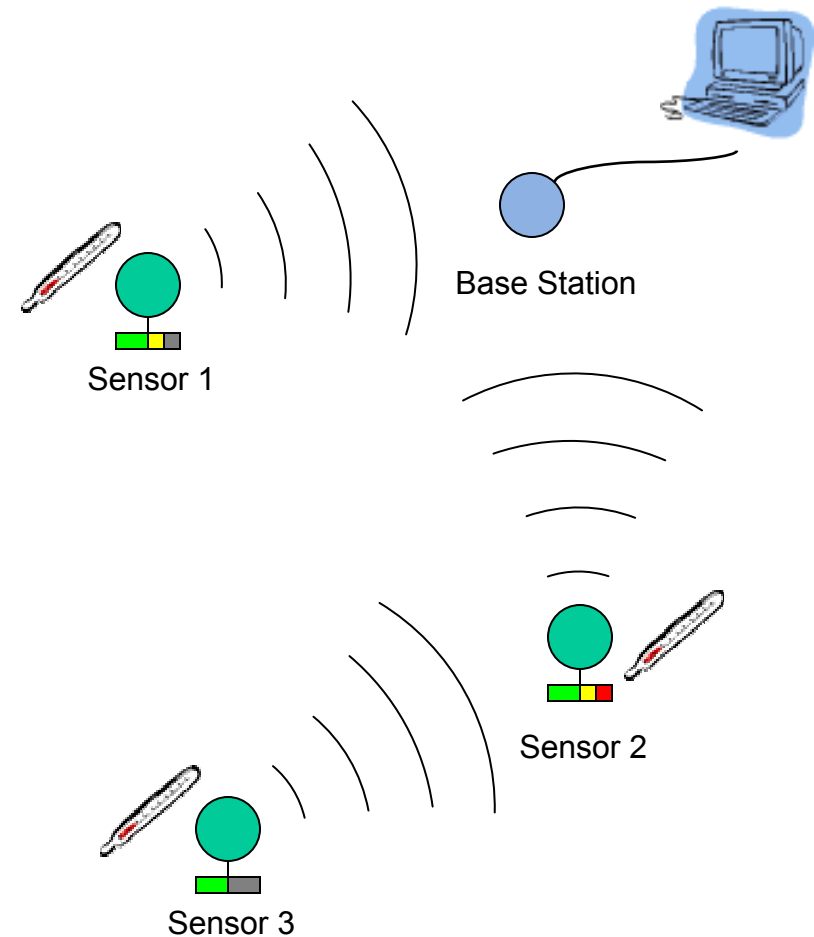
Euromicro 2002 Dortmund

Outline

- Background
- What we did
- How we did it
- Conclusions

What is a Networked Sensor ?

- Autonomous computer based system
 - CPU, memory, power
- Sensors
 - temp, light, etc.
- Actuators
- Comm. Channels
 - radio, UART
- Ad-hoc network

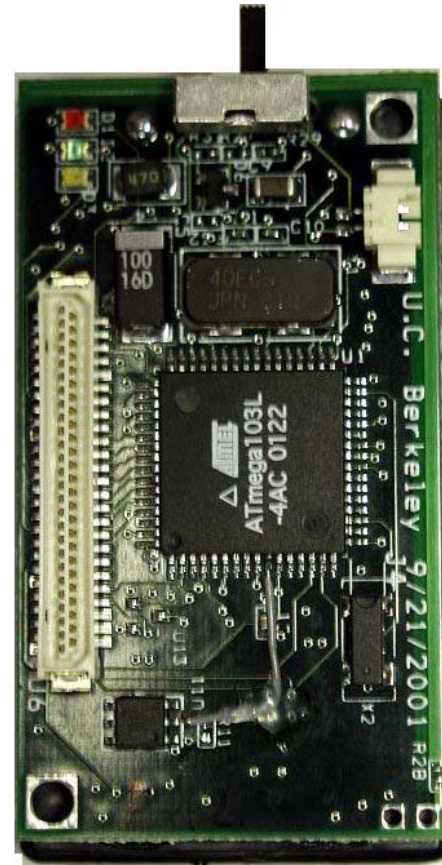


Networked Sensor Characteristics

- Resource constraints
 - power, size, memory, speed
- Concurrency
 - capturing sensor info, processing data, transmit to base station, maintain routing information
- Diversity in design and usage
 - OS must provide high level of modularity
- Shared and unreliable comm. channels

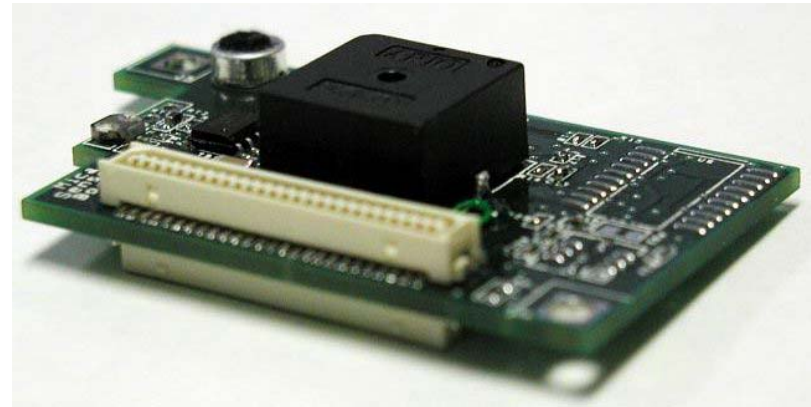
The MICA hardware platform

- UC Berkeley
- Name: Mote
- ATmega 103L MCU
 - 6Mhz RISC, 128kB Flash, 4kB EEPROM, 4kB SRAM
 - 8 A/D channels, UART, SPI
- Coprocessor
 - 4Mbit EEPROM
- RFM
 - 900Mhz, 50kBit/s, bit-level interface (on/off keyed), 4-5 meters
- LEDs
- 2 AA batteries



Sensors

- Microphone / Tone decoder
 - Buzzer (4kHz)
 - Temperature
 - Photo sensor
 - Accelerometer (2D)
 - Magnetometer (2D)
-
- A/D ports, PWR lines, control lines (I2C)

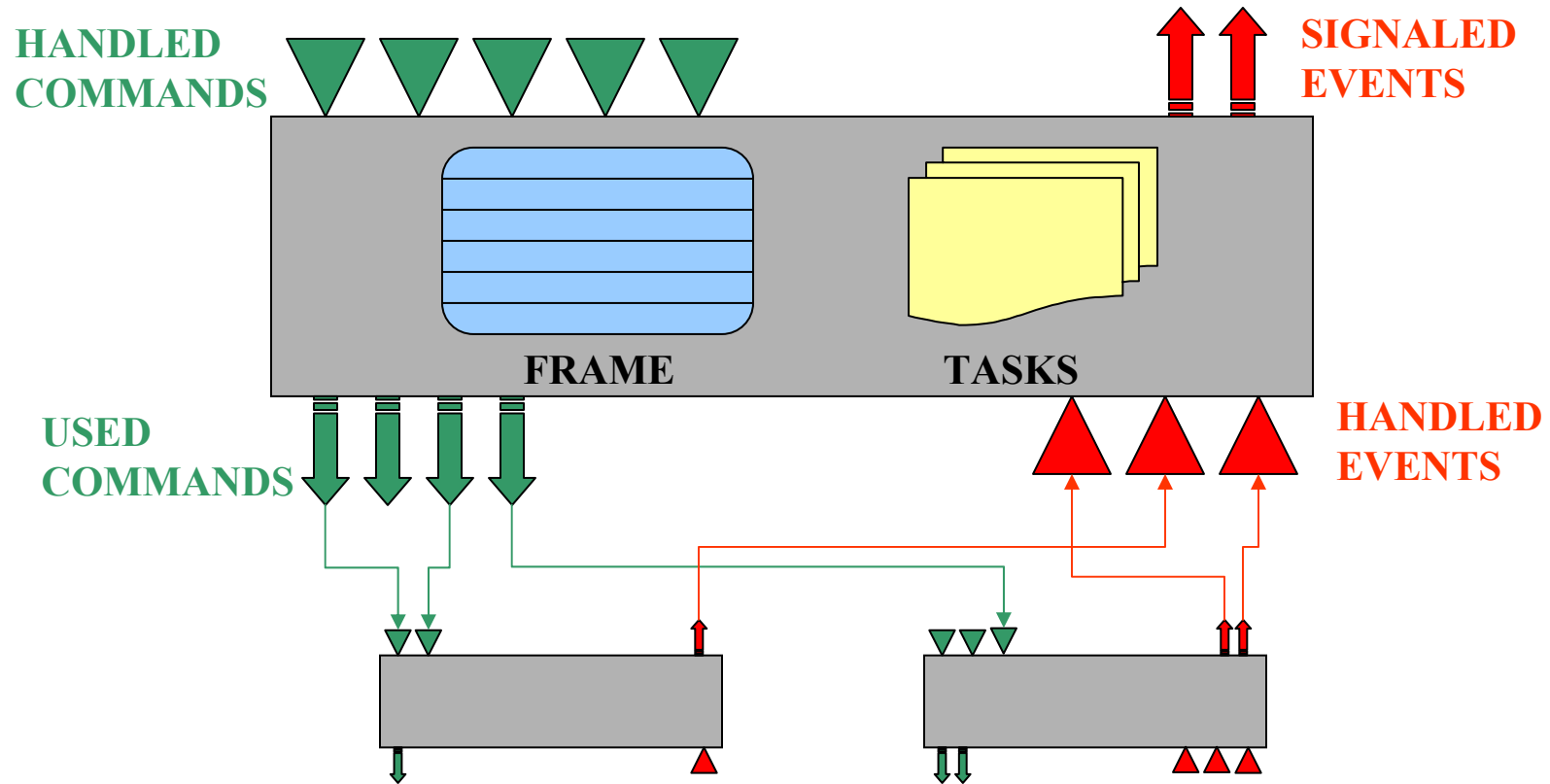


- Unused components can (should) be turned off

TinyOS System Architecture

- An application is a network of components
- The whole application is configured and linked at compile time (no dynamic reconfiguration)
- The OS components are also compiled and linked to the application (no permanent SW code on the MOTES)
- No dynamic memory allocations, no “real” multitasking

TinyOS components



TinyOS Components

- Frame is a static data structure for state info
- Event and command handlers are functions
- Events and commands are functions to be called (#define-based wiring)
- Tasks are functions to be called (these function pointers can be placed in the queue of the scheduler)
- Commands must not signal events
- Only events can interrupt tasks

Case study: BLINK

BLINK.desc

```
include modules {  
    MAIN;  
    BLINK;  
    CLOCK;  
    LEDES;  
};
```

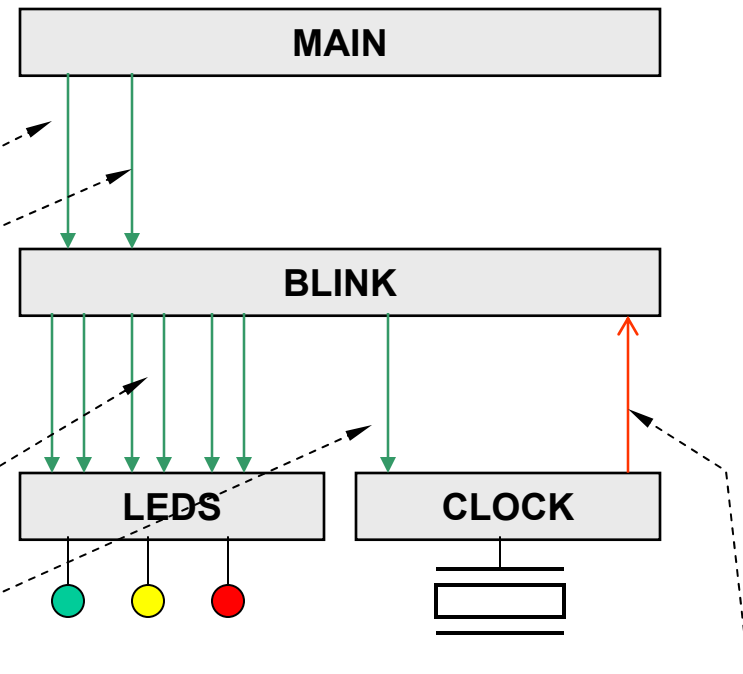
```
BLINK:BLINK_INIT MAIN:MAIN_SUB_INIT
```

```
BLINK:BLINK_START MAIN:MAIN_SUB_START
```

```
BLINK:BLINK_LEDy_on LEDES:YELLOW_LED_ON  
BLINK:BLINK_LEDy_off LEDES:YELLOW_LED_OFF  
BLINK:BLINK_LEDr_on LEDES:RED_LED_ON  
BLINK:BLINK_LEDr_off LEDES:RED_LED_OFF  
BLINK:BLINK_LEDg_on LEDES:GREEN_LED_ON  
BLINK:BLINK_LEDg_off LEDES:GREEN_LED_OFF
```

```
BLINK:BLINK_SUB_INIT CLOCK:CLOCK_INIT
```

```
BLINK:BLINK_CLOCK_EVENT CLOCK:CLOCK_FIRE_EVENT
```



Case study: BLINK

BLINK.comp

BLINK.c

```
TOS_MODULE BLINK;

ACCEPTS {
    char BLINK_INIT(void);
    char BLINK_START(void);
};

HANDLES {
    void BLINK_CLOCK_EVENT(void);
};

USES {
    char BLINK_SUB_INIT(char interval, char scale);
    char BLINK_LEDr_on();
    char BLINK_LEDr_off();
    char BLINK_LEDy_on();
    char BLINK_LEDy_off();
    char BLINK_LEDg_on();
    char BLINK_LEDg_off();
};

SIGNALS {
};
```

```
TOS_FRAME_BEGIN(BLINK_frame) {
    char state;
}

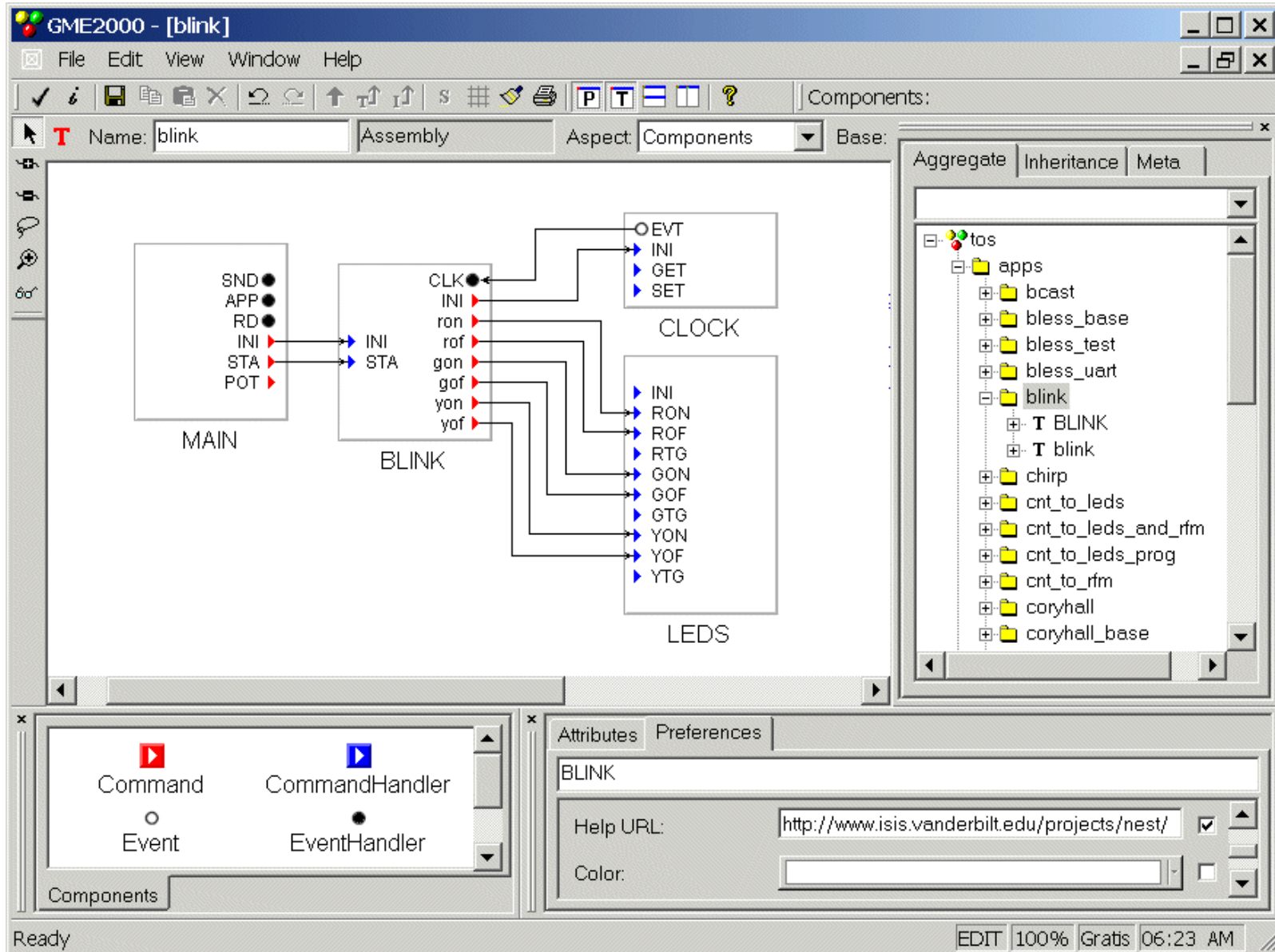
TOS_FRAME_END(BLINK_frame);

char TOS_COMMAND(BLINK_INIT)() {
    TOS_CALL_COMMAND(BLINK_LEDr_off)();
    . . .
    VAR(state) = 0
    TOS_CALL_COMMAND(BLINK_SUB_INIT)
        (tick1ps);
    return 1;
}

char TOS_COMMAND(BLINK_START)(){
    return 1;
}

void TOS_EVENT(BLINK_CLOCK_EVENT)() {
    VAR(state) = (VAR(state) + 1) % 2;
    if (VAR(state))
        TOS_CALL_COMMAND(BLINK_LEDr_on)
    else
        TOS_CALL_COMMAND(BLINK_LEDr_off)
}
```

Gratis Model of BLINK



Visual Programming Environment: Motivation/Requirements

- Problems with redundant information (eg.: interface points)
- Need for (good) visual representation of the wiring (hierarchical components)
- Visual Browser for the OS components
- Error checking, constraints
- Must support two way translation between graphical representation and the source files:
 - TinyOS parsing tool (written in Python)
 - Builds all possible applications/components/wiring
 - Extensive consistency checks

Further Motivation

- Domain-Specific Design Environments (Simulink, LabVIEW, Rose, etc.):
 - Modeling
 - Analysis
 - Simulation
 - Synthesis
- Problem: High cost of development
- Challenge: Rapid and cost-effective creation of domain-specific design environments like Gratis

Generic Modeling Environment

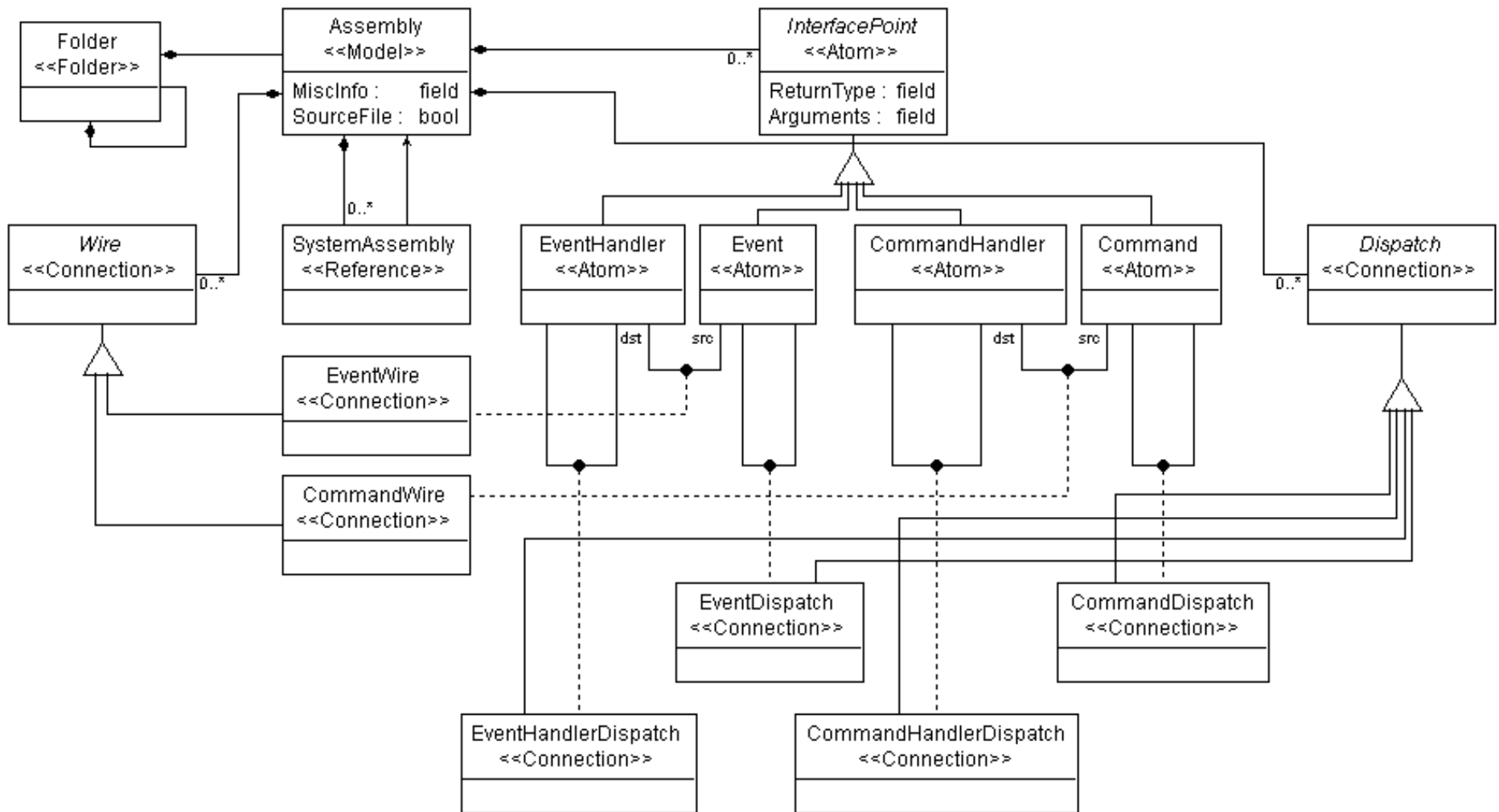
GME 2000

- Configurable (i.e. metaprogrammable)
- Highly modular architecture
- Extensibility: Standard interfaces (COM, XML, UML, OCL)
- Built-in constraint manager
- User-defined visualization
- Model database (object store)

→ Freely available at:

<http://www.isis.vanderbilt.edu/Projects/gme/default.html>

Gratis Metamodel



Conclusion

- Domain-specific modeling and code generation environments are a natural fit for component-based software development
- Applying metaprogrammable environments such as GME 2000 makes their creation feasible :
 - Gratis, a fully functional, sophisticated graphical development environment was about a *1 man-month* project!
- Thanks for **DARPA IXO** for their support