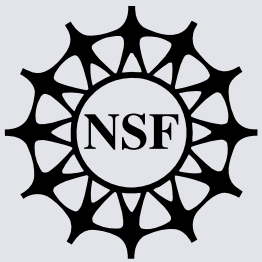
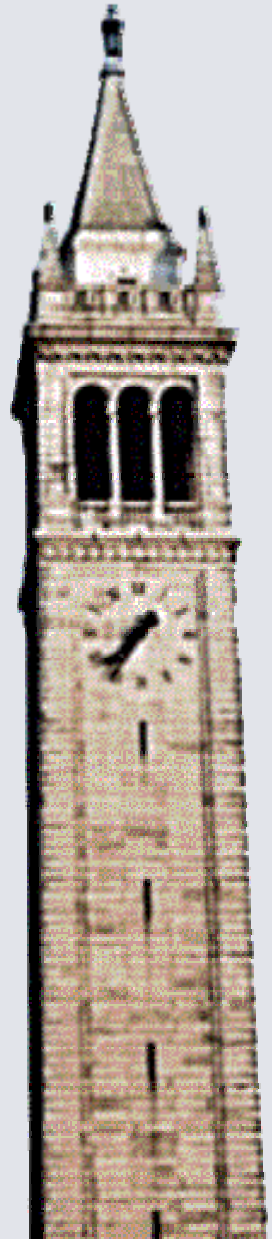
Sensor Network Design

Edited and presented by
Akos Ledeczi
ISIS, Vanderbilt University

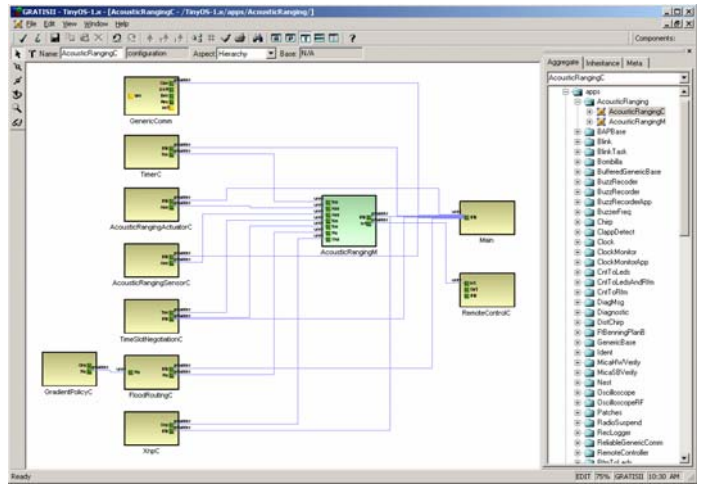


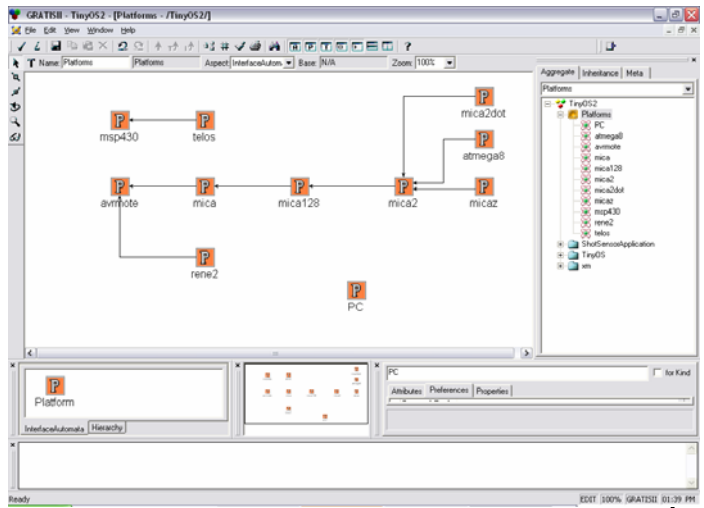
Chess Review
November 21, 2005
Berkeley, CA



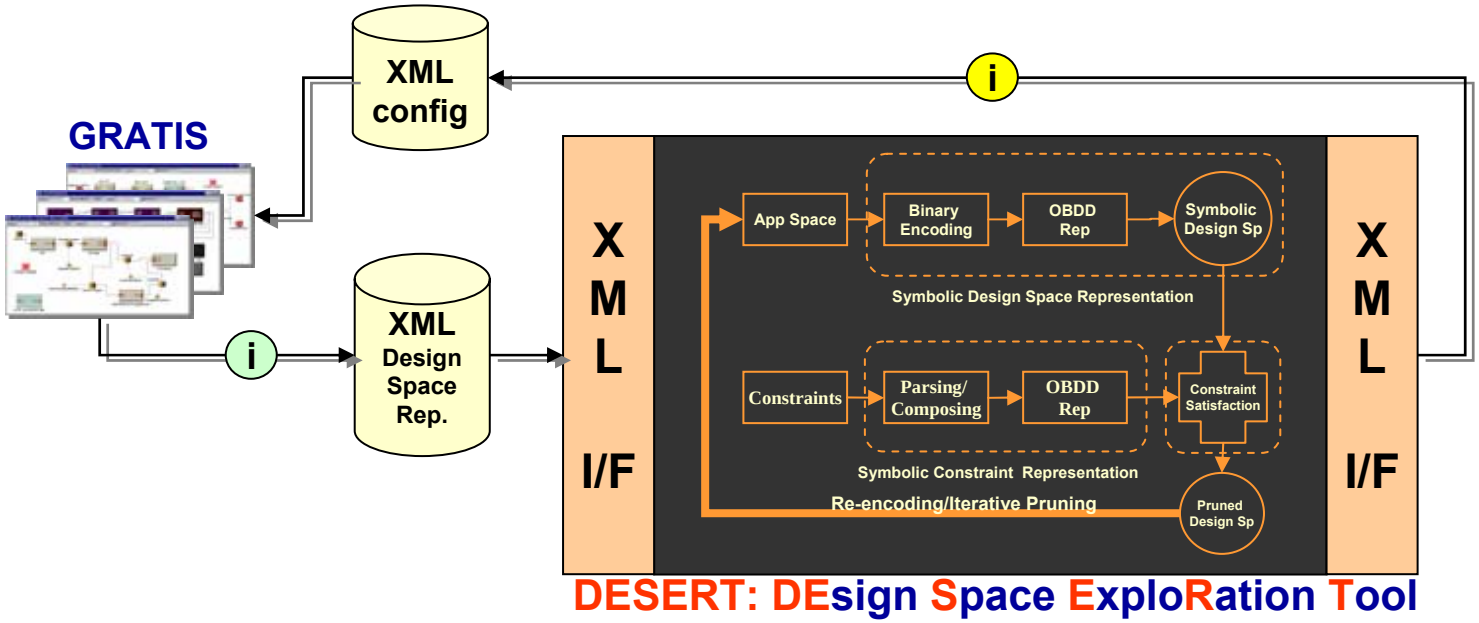
Composition and Synthesis

- Gratis: Visual Composition and Synthesis Environment for TinyOS
- Ported to nesC 1.2
- Built using the Generic Modeling Environment (GME)
  - Meta programmable toolkit using UML and OCL
- Hierarchical representation of TinyOS applications
  - Interfaces: set of events and commands
  - Modules: interface references and implementation code
  - Configurations: set of interface, module and configuration references
- Automatic generation of all configuration files from graphical models
- Can parse existing TinyOS applications and libraries and build equivalent graphical models
  - In TinyOS 1.2 over 10000 connections and objects are provided as a library to the user
  - Necessary to keep visual models and source base in sync
- Extensible through meta-model composition and add-ons



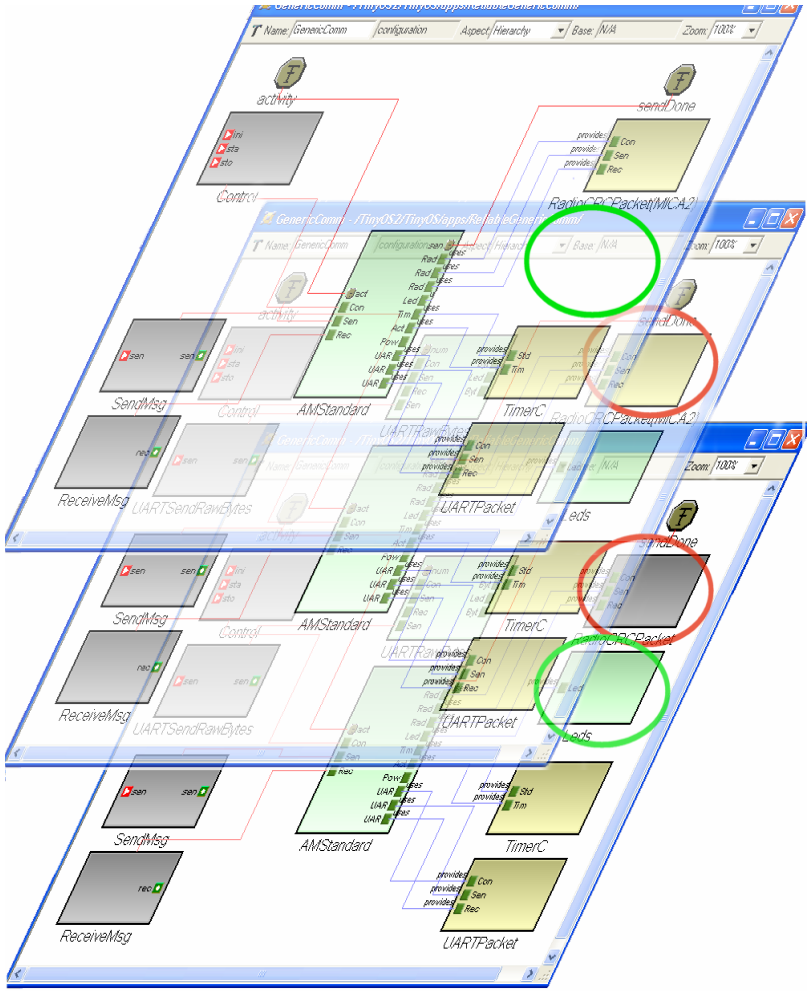


Design Space Exploration



- GRATIS modeling extensions:
  - Component and configuration alternatives
  - Resource usage attributes for components: number of tasks and timers and amount of RAM used etc.
  - OCL constraints to specify requirements and resource constraints
- GRATIS – DESERT integration:
  - A model interpreter configures DESERT w/ the design space and constraint captured in GRATIS
  - A second model interpreter takes the output of DESERT (i.e. a subset of the design space) and allows the user to select the desired configuration and then automatically configures the GRATIS models

Multiplatform support



Specific

General

GRATIS resolves:

- Alternatives capturing the design-space of the application. Selection is based on resource and other application-specific constraints and is carried out by the DESERT tool.
- Platform alternatives representing the target hardware platform. It is based on component platform attributes and is carried out by DESERT.
- Conditional compilation alternatives. It is based on preprocessor macro definitions and is carried out by the parser

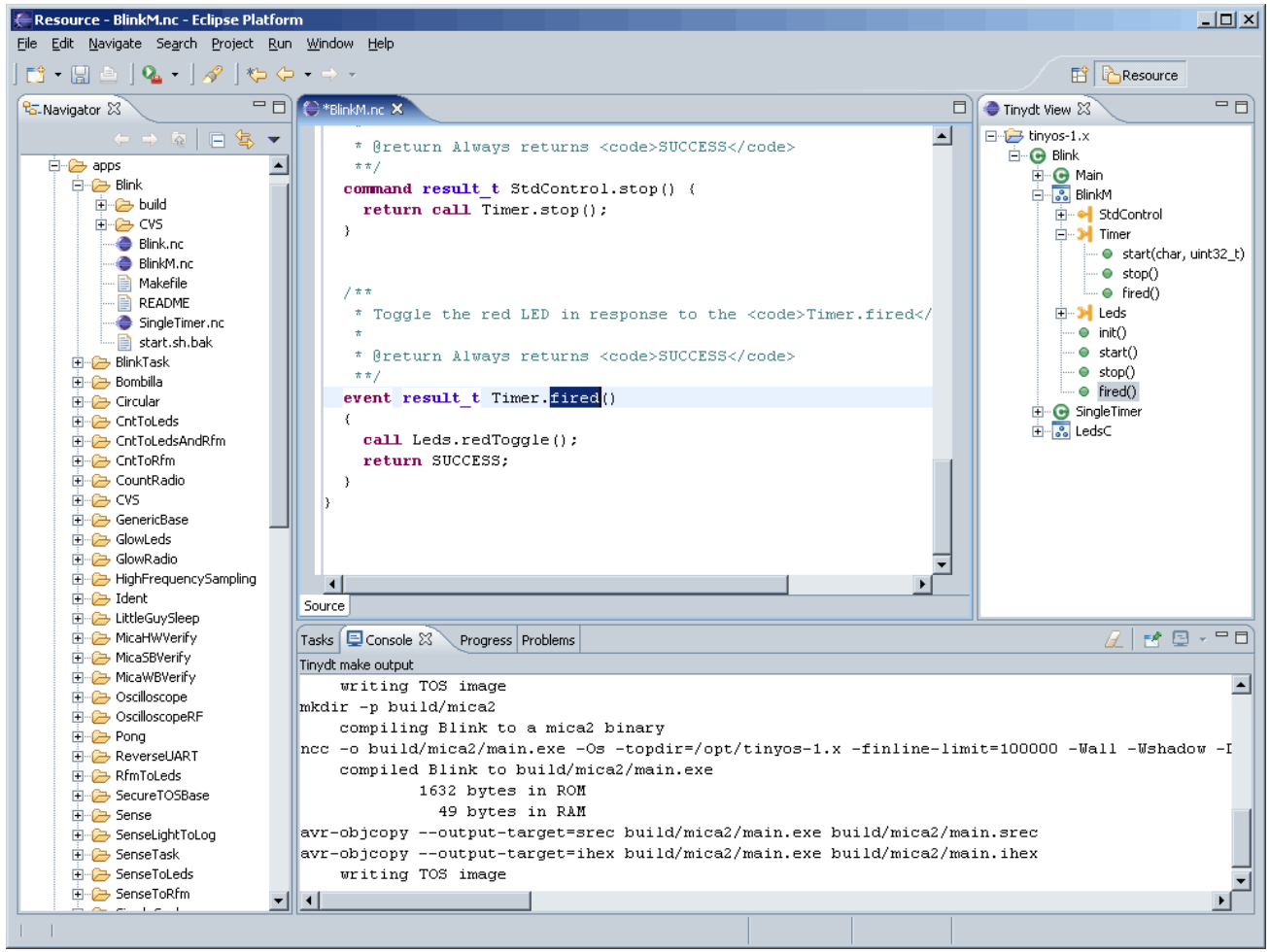
TinyDT: Eclipse IDE for TinyOS

Features:

- Syntax highlighting
- Code completion
- Component Browser
- Configuration View
- Integrated TinyOS Compiler Toolchain

Planned features:

- Model-based interface
- Integrated Loader
- Integrated Simulator
- Message Editor
- Sensor Network Management Console



http://www.tinydt.net

Virtual Thread Abstraction

Motivation

- modules which execute a series of actions that need to be broken up into commands and event handlers are hard to program
- C control constructs (for, while, do-while) cannot be used to iteratively execute split-phase operations
- variables shared between commands and event handlers must be declared in global scope (which is error-prone)
- component state has to be managed by the programmer (which is error-prone)
- components commonly implement state-machine like functionality, but nesC lacks explicit language support

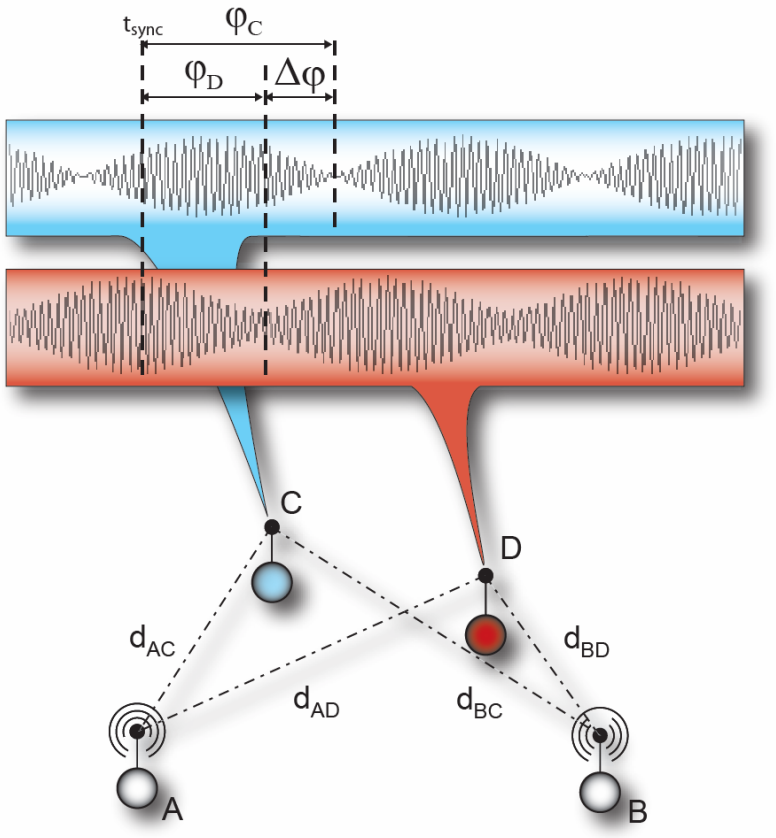
Approach

- Language extensions that provide support for blocking operations
- Implemented as a program translator
- Seamless integration with the TinyOS/nesC tool chain

Benefits

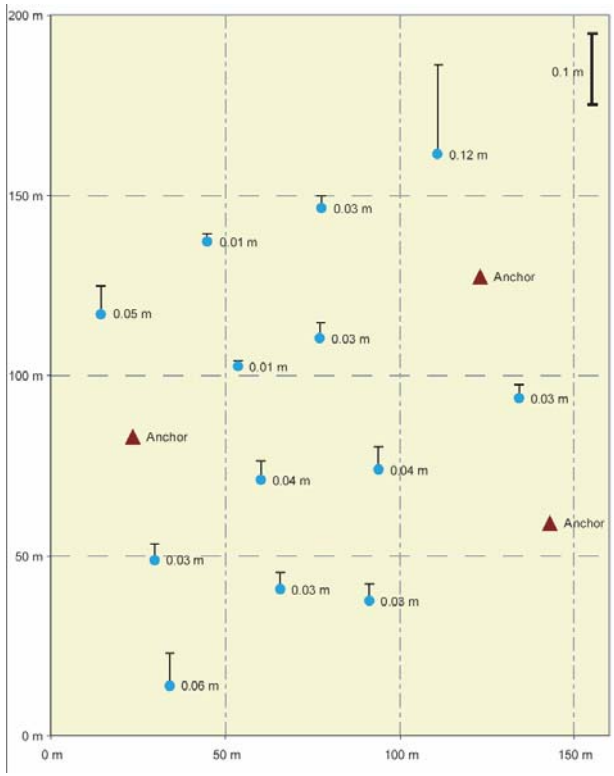
- more comprehensible source code
- familiar thread-like abstraction
- split-phase operations can be part of C control structures
- manages module state
- manages global variables (variables that are never used concurrently can be allocated at the same memory address)
- protects from unexpected events
- seamlessly integrates into the existing development tool chain

Radio Interferometric Localization



$$D_{ABCD} \pmod{\lambda} = \Delta\varphi * \frac{\lambda}{2\pi}$$

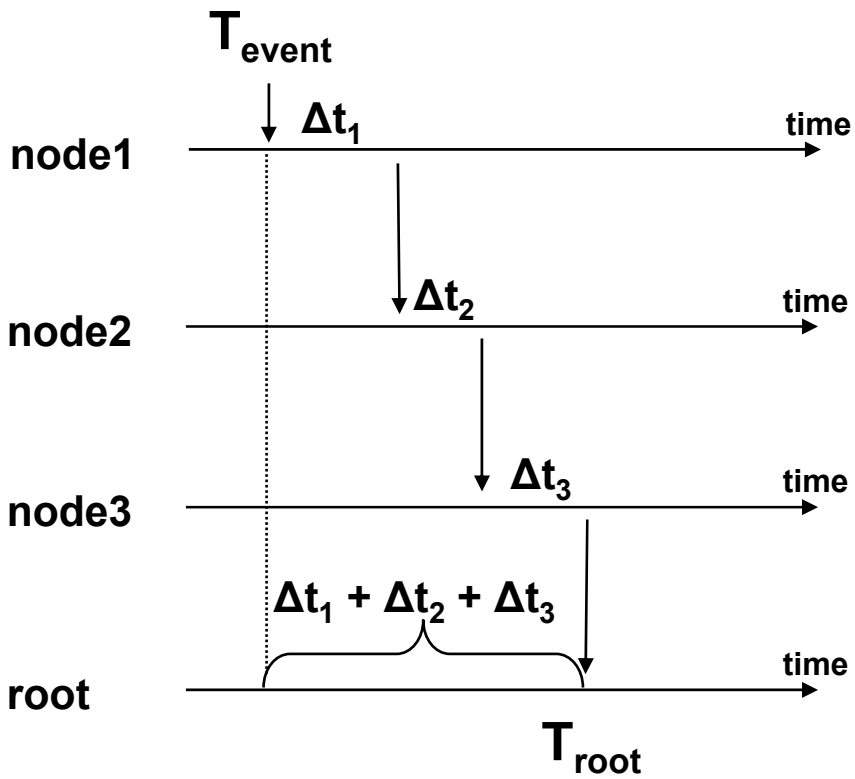
$$D_{ABCD} = d_{AD} - d_{BD} + d_{BC} - d_{AC}$$



- 12000m² area
- 16 XSM motes on the ground
- Minimum node distance 25m
- 3 anchor nodes
- Took 50 minutes
- Average loc error < 4cm
- Maximum loc error 12cm
- Maximum "range" 170m

RiTS: Routing Integrated Time Sync

- Combination of Time Synchronization and Message Routing
- No extra messages
- Stealth operation
- Uses the TimeStamping module that has 1.2 us precision per hop
- No clock skew estimation
- Precision depends on the hop count of the route and on the total routing time
- Integrated with the Directed Flood Routing Framework



$$T_{event} = T_{root} - \Delta t_1 - \Delta t_2 - \Delta t_3$$

RaTS: Rapid Time Synchronization

- 5x10 grid on desktop; neighbor to neighbor communication enforced in sw
- Infrequent resynchronization after initial startup phase: **1 hour interval**
- Rapid synchronization: 30 seconds startup time
- Initial error is 1 ms max
- After regression table is full with 8min interval data, error goes down to 360 us max, 100 us average.

- Idea: utilize RITS with broadcast
- Base station broadcast a message with current global time periodically. Every node receives it along with the measured delay.
- Global time estimate is put in a linear regression table.
- After three broadcasts, good clock skew estimate is already available.
- Short initial period to achieve rapid startup