Seminar HS 10
Wiselib: A Generic Algorithm Library for Heterogeneous Sensor Networks of WISEBED project

Zhongliang Zhao

University of Bern
Overview

> WISEBED

> Wiselib
  — Motivation
  — Objective
  — Challenge
  — Architecture

> Shawn simulator

> Conclusion

> Outlook
Objective

— Joint effort of academic and research institutes across Europe

— To provide a multi-level infrastructure of interconnected testbed of large-scale wireless sensor networks for research purposes

— Bring together heterogeneous small-scale devices and testbed to form well-organized large-scale integrated structures

— Pursue an interdisciplinary approach that integrates the aspects of hardware, software, algorithms, and data
Wiselib: A Generic Algorithm Library for Heterogeneous Sensor Networks of WISEBED project

Wiselib

> Motivation

— Few algorithms for WSNs have been tried in practice
— Many practice challenges are still awaiting efficient algorithmic solutions
— Programming sensor nodes are still at very technical level

> Objective

— Runs on all sensor nodes by the WISEBED partners
— Runs on different operating system & Shawn simulator
— Compiled for any supported system without changing any line of code
— Adaptive utilization of the capabilities of the devices
— Provides simple interface and ready-to-use data structures for development
— Doesn’t contain any platform-specific specializations
— Includes broad algorithms scope
— Single implementation to run natively over heterogeneous networks
Wiselib: A Generic Algorithm Library for Heterogeneous Sensor Networks of WISEBED project

Wiselib

> Challenge

— Heterogeneity: Compiler Variance, Data Access

<table>
<thead>
<tr>
<th>Sensor Node</th>
<th>CPU</th>
<th>OS</th>
<th>Language</th>
<th>Dyn. Mem</th>
<th>ROM</th>
<th>RAM</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>iSense</td>
<td>Jennic</td>
<td>iSense-FW</td>
<td>C++</td>
<td>Physical</td>
<td>128kB</td>
<td>92kB</td>
<td>32</td>
</tr>
<tr>
<td>ScatterWeb MSB</td>
<td>MSP430</td>
<td>SCW-FW</td>
<td>C</td>
<td>None</td>
<td>48kB</td>
<td>10kB</td>
<td>16</td>
</tr>
<tr>
<td>ScatterWeb ESB</td>
<td>MSP430</td>
<td>SCW-FW</td>
<td>C</td>
<td>None</td>
<td>60kB</td>
<td>2kB</td>
<td>16</td>
</tr>
<tr>
<td>Tmote Sky</td>
<td>MSP430</td>
<td>Contiki</td>
<td>C</td>
<td>Physical</td>
<td>48kB</td>
<td>10kB</td>
<td>16</td>
</tr>
<tr>
<td>MicaZ</td>
<td>ATMega128L</td>
<td>Contiki</td>
<td>C</td>
<td>Physical</td>
<td>128kB</td>
<td>4kB</td>
<td>8</td>
</tr>
<tr>
<td>TNOde</td>
<td>ATMega128L</td>
<td>TinyOS</td>
<td>nesC</td>
<td>Physical</td>
<td>128kB</td>
<td>4kB</td>
<td>8</td>
</tr>
<tr>
<td>iMote2</td>
<td>Intel XScale</td>
<td>TinyOS</td>
<td>nesC</td>
<td>Physical</td>
<td>32MB</td>
<td>32MB</td>
<td>32</td>
</tr>
<tr>
<td>GumStix</td>
<td>Intel XScale</td>
<td>Emb. Linux</td>
<td>C</td>
<td>Virtual</td>
<td>16MB</td>
<td>64MB</td>
<td>32</td>
</tr>
</tbody>
</table>

— Limited memory

— Limited Computation Power
> **Approach**

— C++ templates: motivated from STL & CGAL & Boost

— Inline functions

> **Benefits**

— Generic code resolved and bound at compile time, the price of generality paid at compile time

— No memory or computation overhead at run time

— Final binary contains highly efficient and specialized code therefore no overhead at runtime
Wiselib: C++ Template

— Allow the use of generic code that is fully resolved at compile time when specific types are given, no considerable overhead

— Only the code that is actually needed is generated, efficiently

— Concept & Model technique
  – Concept lists the required and provided types as well as the member function declaration
  – Model contains the implementation of the concepts, using the template specialization
**Wiselib: C++ Template**

> **C++ Template Advantage**
>  
> — Early Binding
>  
> — Inline Optimization
>  
> — Code Pruning
>  
> — Extensibility
>  
> — Layered structure

> **Core idea**
>  
> — Pass the important functionality as template parameters to an algorithm: implementation of OS specific code and data structure
>  
> — Make it possible to compile an algorithm exactly for the current needs
Wiselib: A Generic Algorithm Library for Heterogeneous Sensor Networks of WISEBED project

Wiselib

> **Architecture**

— *Concept*: a detailed description of the requirements for a class, that is, an informal prototype for it. It doesn’t contain any source code, but instead of part of Wiselib’s documentation. The idea is to provide a complete documentation from which one is able to produce a valid implementation.

— *Model*: a specific implementation of a concept.

> **Consists of three main pieces**

— Algorithms
— OS facets
— Data structure
Wiselib: Architecture I

> Architecture

External Interface

- Concept
  - OS Facets

- Model
  - OS Facet Implementation

Internal Interface

- Concept
  - Algorithm Category

- Model
  - Algorithm Implementation

- Model
  - Data Structure Implementation

Pass at Compile Time
Algorithm Category Concept

— Core of Wiselib

— One concept per category, whereby a category groups algorithms by their basic functionality.
  - Routing protocol
  - Time synchronization protocol
  - Energy management protocol
  - Clustering protocol
  - Localization protocol
  - MAC layer protocol

— Each algorithm category include a common concept, so that specific algorithms can be models of that concept

— Specify what data structures such an algorithm will typically employ, this is internal interface.

— Specify what the underlying system has to provide and define a concept for it, this is external interface.

— Any algorithm model implements one or multiple concepts, is basically a template expecting various parameters. These parameters can be both OS facets and data structures.
Wiselib: Architecture II

> **OS Facets Concept**
  
  — Represents the connection between Wiselib and underlying OS of target platform, provide basic types and access to OS functionality.
  
  — Each functional unit of an OS is reflected by an OS facet concept, such as the Radio, the Timer, local storage, debugging interfaces.

> **Data Structure Concept**
  
  — Provide adaptive data structure with which an algorithm could scale to the platform it is compiled for.
  
  — i.e. Static data structure can be passed on tiny platforms without dynamic memory management, whereas highly dynamic and efficient data structure are passed on powerful microcontrollers or desktop PC.
Wiselib: A Generic Algorithm Library for Heterogeneous Sensor Networks of WISEBED project

Wiselib: Architecture III

> **External Interface**
  — Provides access to the underlying OS (i.e. Contiki or simulator like Shawn).
  — Provides *basic concepts* that contain provided types and member function which in turn must be implemented for each supported OS.
  — Implementation of this facet are then passed to an algorithm as template arguments.

> **Internal Interface**
  — Provides special and useful kinds of data structures, which can be used by any algorithm implementation within Wiselib.
  — Diverse set of data structure implementations serve the goal of scalability: for each *general data structure*, e.g., routing table or position maps, a set of implementation matching the span of platforms should be provided.
  — So far, the only available concept is the Routing Table concept that is used by routing algorithm models.
Wiselib: Case Study of Routing algorithm I

> External Interface: How routing algorithm would be able to use different implementations of a radio concept on multiple hardware platforms

— Describe the general concept of a radio.
— Radio includes only method for message sending as well as data types.
— Implementation of a radio must provide the data types, including type of the node identifier, the type of block data, and the general size type.
— Method message sending will use all the data types.
  - type of node ID: receiver
  - type of block data: pointer to the data that is going to be sent
  - type of general size: number of block data units
Wiselib: A Generic Algorithm Library for Heterogeneous Sensor Networks of WISEBED project

Wiselib: Case Study of Routing algorithm II

> The Prototype for Radio concept

```cpp
template <typename OsModel>
struct RadioModel
{
    typedef ... node_id_t;
    typedef ... block_data_t;
    typedef ... size_t;

    static inline void send_message( node_id_t,
                                       block_data_t,
                                       size_t );

};
```

— This general concept now could be specialized for different system.
Wiselib: A Generic Algorithm Library for Heterogeneous Sensor Networks of WISEBED project

Wiselib: Case Study of Routing algorithm III

iSense OS

template <>
structure RadioModel <iSense OS>
{
  typedef isense :: uint16 node_id_t;
  typedef isense :: uint8 block_data_t;
  typedef isense :: uint8 size_t;

  static inline void send_message ( node_id_t dst, block_data_t* buf, size_t len )
  {
    os -> radio.send( dst, buf, len, 0, NULL );
  }
};

ScatterWeb OS

template <>
structure RadioModel <ScatterWeb OS>
{
  typedef uint16_t node_id_t;
  typedef uint8_t block_data_t;
  typedef uint8_t size_t;

  static inline void send_message ( node_id_t dst, block_data_t* buf, size_t len )
  {
    send() adapted to call the needed function of the ScatterWeb API
  }
};

5. Mai 2010
template < typename OsModel, 
typename RadioModel = RadioModel<OsModel> > 

class Routing
{
    Routing (OsModel& os) : os_( os )
    { ... }

    void foo ( void )
    {
        RadioModel :: node_id_t dst = ... ;
        RadioModel :: block_data_t buf [64] = ... ;
        ... 
        ... 

        RadioModel :: send_message ( dst, buf, 64, os_ ) ;
    }

    OsModel& os_ ;
};
Internal Interface: Which data structures should be provided for the routing algorithm

— Target platforms variety: Heavily limited platform vs Rich platform

Limited platform

```cpp
template < typename OsModel, unsigned int table_size>
class StaticArrayRoutingTable {
    typedef pair<OsModel::NodeId, OsModel::NodeID> TableEntry;
    staticArrayRoutingTable ( ) {
        for ( TableEntry* elem = table_ ; elem != table_ + table_size ;  ++ elem )
            elem->first = OsModel::NULL_NODE_ID; ++ elem ;
    }

    bool add_new_entry (OsModel ::NodeId dest, OsModel ::NodeID next_hop) {
        for ( TableEntry* elem = table_ ; elem != table_ + table_size ;  ++ elem )
            if ( elem->first == OsModel::NULL_NODE_ID )
                elem->first = dest, elem->second = next_hop; return true
        return false;
    }

    TableEntry table_ [ table_size ] ;
}
```

Rich platform

```cpp
template < typename OsModel>
class STLRoutingTable {
    typedef std :: map < OsModel ::NodeId, OsModel ::NodeID> Table;

    bool add_new_entry ( OsModel ::NodeId dest, OsModel ::NodeID next_hop) {
        table_ [ dest ] =  next_hop; return true;
    }

    Table table_;
}
```
> A routing algorithm can now be implemented with routing table as a black box.

```
template<typename OsModel, typename RoutingTable>
class Routing
{
    ...
    RoutingTable routing_table_;
};
```

— Such an algorithm will be instantiated with a specific type for the `RoutingTable` template parameter by the application code.

— The compiler will resolve all the templates and fully embed the data structure into `Routing` class.
Shawn Simulator

> **Motivation**
>  — Large scale sensor networks simulation: thousands of sensor nodes
>  — Detailed lower-level networking simulation is always necessary?

> **Objective**
>  — High level abstraction point of view
>  — Intrinsically designed for the evaluation of the high level protocol
>  — Substitute the physical environment with well-chosen random distribution on message delay and loss.
>  — Focus more on the design of the algorithm rather than on the simulation process

> **Core idea**
>  — Simulate the effects caused by the phenomenon, not the phenomenon
>  — Replace the low-level effects with abstract well-chosen and exchangeable models, the simulation can be used for huge networks in reasonable time.
Wiselib: A Generic Algorithm Library for Heterogeneous Sensor Networks of WISEBED project

Sequencer
— Central coordinating unit as it configures the simulation, executes tasks sequentially and controls the simulation

Simulation Environment
— home for the virtual world in which the simulated sensors nodes reside & behave.
Conclusion

> **Wiselib**
>  — Platform independent
>  — OS independent
>  — Broad algorithm scope
>  — Cross-layer algorithm
>  — Scalability and efficiency

> **Shawn**
>  — Simulate the effects caused by the phenomenon, not the phenomenon itself
>  — Replace the low-level effects with abstract and exchangeable models, the simulation can be used for huge networks in reasonable time
Outlook

> Accomplishment of the OLSR routing protocol for Wiselib

> Wiselib in Shawn of OLSR

> DYMO implementation for Wiselib
Discussion