Wiselib School 2010

Session 4
Insight into Wiselib Internals

Tobias Baumgartner

Institute of Operating Systems and Computer Networks
Handout on http://www.ibr.cs.tu-bs.de/alg/winterschool/

October 2010

Outline

Wiselib Algorithm Integration

- Two options for Wiselib Algorithm Integration
  - **Generic Wiselib Application**
    $\Rightarrow$ Compile the same application for several platforms
  - **Direct Integration**
    $\Rightarrow$ Use Wiselib algorithms directly on your platform

Generic Wiselib Application

- Already known from hands-on task
- Write code **once**, run on **different** platforms
  → Change make target (`shawn`, `isense`, `contiki_sky`, `scw_msb`, ...)
- **Advantages**
  - Totally flexible, especially for **debugging**
    ⇒ **Test** code on different platforms
  - **Easy** to start with
  - Very nice for **new platform** integration
- **Disadvantages**
  - **Bound to** generic application
  - **No direct access** to operating system
  - Limited when trying to pass **different facet** implementations
  - Restrictions like in several **middleware** approaches

The Wiselib is an **algorithm library**, **no** middleware

Direct Integration

- Directly use a Wiselib algorithm in your OS/firmware
- **Disadvantages**
  - Algorithm integration needed to be done for **each platform**
  - **Tedious** work for **testing** on different platforms
  - **Perhaps hard** for new platform support
    $\Rightarrow$ Makefile adaptation, C++ compiler for own firmware, ...
- **Advantages**
  - Full **flexibility** in hardware/OS access
  - Do **not bind** the application developer to anything
  - Existing systems **not affected**, but **extendable**
  - **Still possible** to develop **heterogeneous** systems
    $\Rightarrow$ When **TinyOs** can communicate with **Contiki**, they can also run a **common Wiselib algorithm**

Main design goal of the Wiselib

Outline

Outline

Generic Wiselib Application Revisited

- Separate application with init method

```
1 class MyGenericApplication {
2    public:
3       void init( Os::AppMainParameter& value )
4       {
5          radio_ = &wiselib::FacetProvider<Os, Os::Radio>::get_facet( value );
6       };
7
8       Os::Radio::self_pointer_t radio_;
9 }
```

- Create Wiselib Application

```
10 wiselib::WiselibApplication<Os, MyGenericApplication> my_app;
```

- Implement `application_main()`, call init

```
11 void application_main( Os::AppMainParameter& value )
12 {
13    my_app.init( value );
14 }
```

Wiselib Application Template

- Definition

```
1 wiselib::WiselibApplication<Os, MyGenericApplication> my_app;
```

- Global Variable
  - Only **one** instance
  - Constructor is called before system initialization (before `main()`)
- Given facts for supported systems
  - No dynamic memory support on Contiki, SCW, TinyOs by default
  - Shawn has usually more processors
    $\Rightarrow$ Multiple instances of `MyGenericApplication` needed

Default Wiselib Application

```
1   template<typename OsModel_P,
2            typename Application_P>
3   class WiselibApplication
4   {
5   public:
6      typedef OsModel_P OsModel;
7      typedef Application_P Application;
8      typedef typename OsModel::AppMainParameter AppMainParameter;
9
10     void init( AppMainParameter& os )
11     {
12        application_.init( os );
13     };
14
15  private:
16     Application application_;
17  };
```

- No constructor
- Private member of passed `Application_P` (template parameter)
- Only **one** instance of application
- Call to `init(...)` is *forwarded*

Shawn Wiselib Application

```
1   template<typename Application_P>
2   class WiselibApplication<ShawnOsModel, Application_P>
3   {
4   public:
5       typedef ShawnOsModel OsModel;
6       typedef Application_P Application;
7
8       void init( ShawnOs& os )
9       {
10          Application *app = new Application();
11          app->init( os );
12      };
13  };
```

- Template specialization of `WiselibApplication` for Shawn
- No constructor
- No private member
- Create a **new** application **instance** per call
- Initialize newly created application

Outline

The AppMainParameter

- Passed in `application_main()`

```
1 void application_main( Os::AppMainParameter& value )
2 {
3     my_app.init( value );
4 }
```

- System specific information
- Defined by `OsModel`
  - Processor in Shawn
  - `isense::Os` in iSense
  - Not needed in Contiki: Dummy parameter

Calling application_main() in iSense

```
1   class iSenseWiselibApplication: public isense::Application
2   {
3     public:
4     iSenseWiselibApplication(isense::Os &os)
5       : isense::Application( os ),
6         os_( os )
7     {}
8
9     virtual void boot ( void )
10    { application_main( os_ ); }
11
12    isense::Os &os_;
13  }
```

- Internally, an ordinary iSense application
- Call `application_main` (the one in Generic Wiselib Application)
- Inside `application_main`:
  - Initialize Generic Wiselib Application

Calling application_main() in Shawn

```
1   void
2   WiselibShawnStandaloneProcessor::
3   boot( void )
4     throw()
5   {
6     os_.proc = this;
7     application_main( os_ );
8   }
```

- Internally, an ordinary Shawn processor
- Set pointer to processor, then pass to application_main
- Inside application_main:
  - Initialize Generic Wiselib Application

Outline

Generic Wiselib Application

- Facet instantiation

```
1 radio_ = &wiselib::FacetProvider<Os, Os::Radio>::get_facet( value );
2 ...
3 Os::Radio::self_pointer_t radio_;
```

- Different requirements for different systems
  - Shawn: Create one radio per processor
  - iSense: Create one radio that has access to iSense Os pointer
  - Contiki: Create one radio that only calls C funtions
- Solution (again): Template Specialization
  - Generic facet provider
  - Specialization for sytems with specific needs

Generic Facet Provider

```
1   template<typename OsModel_P,
2            typename Facet_P>
3   class FacetProvider
4   {
5   public:
6      typedef OsModel_P OsModel;
7      typedef Facet_P Facet;
8      typedef typename OsModel::AppMainParameter AppMainParameter;
9
10     static Facet& get_facet( AppMainParameter& os )
11     { return facet_; };
12
13  private:
14     static Facet facet_;
15  };
```

- Static data member
- Only **one** facet instance
- Only default constructor used

Example: Contiki Timer Model

```
1   template<typename OsModel_P>
2   class ContikiTimerModel
3   {
4   public:
5       typedef OsModel_P OsModel;
6
7       typedef ContikiTimerModel<OsModel> self_type;
8       typedef self_type* self_pointer_t;
9
10      typedef uint32_t millis_t;
11
12      template<typename T, void (T::*TMethod)(void*)>
13      int set_timer( millis_t millis, T *obj_pnt, void *userdata )
14      {
15        // only C function calls!
16      }
17  };
```

- No constructor
- Only function calls in timer used

iSense Facet Provider

```
1   template<typename Facet_P>
2   class FacetProvider<iSenseOsModel, Facet_P>
3   {
4   public:
5      typedef iSenseOsModel OsModel;
6      typedef Facet_P Facet;
7
8      static Facet& get_facet( isense::Os& os )
9      {
10        if ( facet_ == 0 )
11           facet_ = new Facet(os);
12
13        return *facet_;
14     }
15
16  private:
17     static Facet *facet_;
18  };
```

- Template Specialization: **Only used for iSense**
- Static data member: Pointer!
- Only **one** instance per facet
- Create new instance when calling `get_facet()`
- Pass `isense::Os` in constructor

Example: iSense Radio Model

```
1   template<typename OsModel_P>
2   class iSenseRadioModel
3       : public isense::Receiver
4   {
5     iSenseRadioModel( isense::Os& os )
6       : os_(os)
7     {
8       os_.dispatcher().add_receiver( this );
9     }
10
11    int send( node_id_t id, size_t len, block_data_t *data )
12    {
13      os_.radio().send( id, len, data, 0, 0 );
14      ...
15    }
16
17    isense::Os& os_;
```

- `isense::Os` passed in constructor
- `isense::Os` can be used within the whole model
- Possible, because we **know** how the facet provider looks like

Outline

Integration in Native Applications

- Developer knows the Os Facets for the platform
- **Not needed**
  - Facet provider
  - Wiselib Application
  - `application_main(...)`
  - `Os::AppMainParameter`
- **Instead**
  - Use the knowledge of your platform
  - Create facets directly (as required for the corresponding platform)
  - `Os::AppMainParameter` is known

Example: iSense Integration (1 of 2)

- Directly include Os Facets

```
#include "external_interface/isense/isense_os.h"
#include "external_interface/isense/isense_radio.h"
...
#include "algorithms/routing/dsdv/tree_routing.h"
```

- When declare typedefs, use knowledge of OsModel

```
typedef wiselib::iSenseOsModel Os;
typedef wiselib::TreeRouting<Os, Os::Timer, Os::Radio> tree_routing_t;
```

- Define needed data members and methods

```
class iSenseDemoApplication
{
  void receive_routing_message (uint16 src_addr, uint8 len, uint8 *buf);
  ...
  Os::Radio radio_;
  Os::Timer timer_;
  Os::Debug debug_;
  tree_routing_t routing_;
```

Example: iSense Integration (2 of 2)

- Pass in `isense::Os` in constructor to facets

```
1 iSenseDemoApplication::
2 iSenseDemoApplication(isense::Os& os)
3     : isense::Application(os),
4       radio_( os ),
5       timer_( os ),
6       debug_( os )
7 {}
```

- In `boot()`, init routing and register callback

```
1 void iSenseDemoApplication::boot(void) {
2   routing_.init( radio_, timer_, debug_ );
3   routing_.enable_radio();
4   routing_.reg_recv_callback<
5     iSenseDemoApplication,
6     &iSenseDemoApplication::receive_routing_message>(this);
7 }
```

- Receive messages in iSense, which were sent over Wiselib algorithm

```
1 void
2 iSenseDemoApplication::
3 receive_routing_message (uint16 src_addr, uint8 len, uint8 *buf)
4 {
5    os_.debug("received routing message from %i", src_addr);
6 }
```

Outline

Design Issues for Callback Structure

- Ability to register callback (method pointer) in another class
- Requirements
  - Flexible
  - Fast
  - Small (in code space)
- **Our Solution:** Delegates
- Known from C#

```
1 public delegate int DelegateToMethod(int x, int y);
2
3 public static int Add(int first, int second)
4 { return first + second; }
5
6 DelegateToMethod aDelegate = new DelegateToMethod(Math.Add);
7 Console.WriteLine(aDelegate(5,5));
```

- With C++ and Templates
  - We use the approach of Sergey Ryazanov
  - www.codeproject.com/KB/cpp/ImpossiblyFastCppDelegate.aspx

Delegates: Internals (1 of 2)

- Basic class `delegate`
  - Object pointer of type void
  - Stub pointer that points to a static template function (here: return type void, expect parameter of type int)

```
1 class delegate
2 {
3     ...
4     typedef void (*stub_type)(void* object_ptr, int);
5
6     void* object_ptr;
7     stub_type stub_ptr;
8     ...
9 };
```

- Meaning of pointers
  - Object pointer: Pointer to *callbacked* class with member function
  - Stub pointer: Call the member function of the object pointer:

```
1 template <class T, void (T::*TMethod)(int)>
2 static void method_stub(void* object_ptr, int a1)
3 {
4   T* p = static_cast<T*>(object_ptr);
5   return (p->*TMethod)(a1);
6 }
```

Delegates: Internals (2 of 2)

- Stub pointer is created via static member function `from_method` of the delegate class

```
1 template <class T, void (T::*TMethod)(int)>
2 static delegate from_method(T* object_ptr)
3 {
4   delegate d;
5   d.object_ptr = object_ptr;
6   d.stub_ptr = &method_stub<T, TMethod>;
7
8   return d;
9  }
```

- Finally: Invocation of a delegate

```
1 void operator()(int a1) const
2 {
3   return (*stub_ptr)(object_ptr, a1);
4 }
```

Delegates: Users' view

- Internals hidden by Wiselib
- Example: Callback in radio

```
1 #include "util/delegates/delegate.hpp"
2
3 typedef delegate3<void, node_id_t, size_t, block_data_t*> radio_delegate_t;
4 radio_delegate_t cb_;
5
6 template<class T, void (T::*TMethod)(node_id_t, size_t, block_data_t*)>
7 int
8 reg_recv_callback( T *obj_pnt )
9 {
10   cb_ = radio_delegate_t::from_method<T, TMethod>( obj_pnt );
11 }
12
13 void on_receive(...)
14 {
15   if (cb_)
16     cb_( from, len, buf );
17 }
```

- Callback registration

```
1 radio_->reg_recv_callback<MyApp, &MyApp::rcv_radio_message>(this);
```

Outline

Outline

Os Facets: Overview

	OS	Radio (TX Power Ext.) (RSSI/LQI Ext.)	Timer	Logging	Clock (Set Clock Ext.)	Serial Comm.	Position	
WP2 OSA	⊕	○	○	⊕				**4/7**
Contiki	⊕	○	⊕	⊕		○		**5/7**
TinyOS	⊕			⊕				**2/7**
iSense	⊕	⊕ ● ●	⊕	⊕	⊕ ●	⊕	○	**7/7**
ScatterWeb	⊕	○	⊕	⊕				**4/7**
Shawn	⊕	⊕ ● ●	⊕	⊕	⊕		⊕	**6/7**

(⊕ = fully supported, ○ = works / proof of concept, ● = with extension)

Example: Contiki Timer

Example: Contiki Timer (1 of 2)

- Timer item that is used for Contiki etimer

```
1 typedef delegate1<void, void*> contiki_timer_delegate_t;
2
3 struct timer_item
4 {
5   struct timer_item *next;
6   struct etimer etimer;
7   struct process *p;
8   contiki_timer_delegate_t cb;
9   void *ptr;
10 };
```

- `set_timer` implementation

```
1 template<typename T, void (T::*TMethod)(void*)>
2 set_timer( millis_t millis, T *obj_pnt, void *userdata )
3 {
4   uint16_t secs = millis >> 10;
5
6   timer_item *item = get_timer_item_ref();
7   item->p = PROCESS_CURRENT();
8   item->cb = contiki_timer_delegate_t::from_method<T, TMethod>( obj_pnt );
9   item->ptr = userdata;
10
11  PROCESS_CONTEXT_BEGIN( &timer_process );
12  etimer_set( &item->etimer, CLOCK_SECOND * secs );
13  PROCESS_CONTEXT_END( &timer_process );
14  return SUCCESS;
15 }
```

Example: Contiki Timer (2 of 2)

- Contiki internal timer

```
1   PROCESS( timer_process, "timer process" );
2   PROCESS_THREAD( timer_process, ev, data )
3   {
4       PROCESS_BEGIN();
5
6       while (1)
7       {
8           PROCESS_YIELD_UNTIL( ev == PROCESS_EVENT_TIMER );
9           for ( int i = 0; i < MAX_REGISTERED_TIMER; i++ )
10          {
11              struct timer_item *c = &timer_items[i];
12              if ( &c->etimer == data )
13              {
14                  PROCESS_CONTEXT_BEGIN( c->p );
15                  if ( c->cb )
16                      c->cb( c->ptr );
17                  PROCESS_CONTEXT_END( c->p );
18                  c->cb = contiki_timer_delegate_t();
19                  break;
20              }
21          }
22      }
23
24      PROCESS_END();
25  }
```

Example: iSense Radio

Example: iSense radio

- iSense Radio: Direct calls to `isense::Os`

```
1 template<typename OsModel_P>
2 class iSenseRadioModel
3   : public isense::Receiver
4 {
5 public:
6   typedef OsModel_P OsModel;
7   ...
8   iSenseRadioModel( isense::Os& os )
9     : os_(os)
10  { os_.dispatcher().add_receiver( this ); }
11
12  int send( node_id_t id, size_t len, block_data_t *data )
13  { os_.radio().send( id, len, data, 0, 0 ); return SUCCESS; }
14
15  int enable_radio()
16  { os_.radio().enable(); return SUCCESS; }
17
18  int disable_radio()
19  { os_.radio().disable(); return SUCCESS; }
20
21  node_id_t id()
22  { return os_.id(); }
```

Outline

Porting to a new Platform

- Facet selection
  - OsModel: Provide basic data types (0 methods)
  - Radio: Send messages (6 methods)
  - Timer: Register timed callbacks (1 method)
  - Debug: Write out text messages (1 method)

  $\Rightarrow$ Basic algorithms can be started

  $\Rightarrow$ **Only 8** methods to implement
- Makefile support
  - Generic Wiselib Application

    $\rightarrow$ Provide call to `application_main()`

    $\rightarrow$ Link application to firmware
  - Direct integration

    $\rightarrow$ Compile application with C++ compiler

    $\rightarrow$ In case of need, use `extern "C"`

Example: Contiki Makefile for Generic Wiselib Application

Generic Wiselib Application: Contiki Example

```
1  export CONTIKI=$(CONTIKI_PATH)
2  include $(CONTIKI_PATH)/Makefile.include
3
4  CXXFLAGS += ...
5  LDFLAGS  += -L. -L$(WISELIB_BASE)/applications/lib \
6        $(OBJECTDIR)/contiki-$(TARGET)-main.o \
7        contiki-$(TARGET).a
8
9  contiki_msp: $(PROJECT_OBJECTFILES) $(PROJECT_LIBRARIES) contiki-$(TARGET).a
10 @echo "compiling..."
11 msp430-g++ $(CXXFLAGS) -o out/contiki_os.o \
12   -c $(WISELIB_PATH)/external_interface/contiki/contiki_os.cpp
13 msp430-g++ $(CXXFLAGS) -o out/contiki_timer.o \
14   -c $(WISELIB_PATH)/external_interface/contiki/contiki_timer.cpp
15 ...
16 @echo "linking..."
17 msp430-g++ $(LDFLAGS) -o out/$(BIN_OUT).elf out/$(BIN_OUT).o \
18   out/contiki_os.o out/contiki_timer.o out/contiki_radio.o \
19   out/contiki_uart.o
20 @echo "make hex..."
21 msp430-objcopy -O ihex out/$(BIN_OUT).elf out/$(BIN_OUT).hex
22 msp430-objcopy -O binary out/$(BIN_OUT).elf out/$(BIN_OUT).bin
```

- Generate Contiki library
- Compile own code with C++-Compiler, then link Contiki

Example: Direct Integration on iSense

Direct Integration: iSense Example

- Add Wiselib include directories to iSense Makefile

Thoughts: Direct Integration in Contiki

Direct Integration: Contiki

- This is a little bit more tricky...
- Not done (**yet!**)
- How it would work in principle
  - Influence make process
  - Compile firmware with C-Compiler
  - In main application, embed all Contiki includes in `extern "C"`
  - Compile main application with C++ compiler

Thanks