Pervasive Systems

Ioannis Chatzigiannakis

Sapienza University of Rome
Department of Computer, Control, and Management Engineering (DIAG)

Lecture 21:
Wiselib: Algorithmic Library for WSN

A Library Of Algorithms

Typical Problems In WSN Programming

- Theoreticians are not interested in programming
  - Ideally they just have to write their algorithms
  - And do not need to care about boilerplate code
- Practioners are not interested in theory
  - Just need a good algorithm for their task
  - Without having to study the field for years

⇒ There is need for an algorithm library

- With lots of algorithms for all kinds of tasks
- That are easy to integrate into existing systems
- And are combinable
- And easily enhanceable

A Library Of Algorithms

Typical Problems In WSN Programming

- Theoreticians are not interested in programming
  - Ideally they just have to write their algorithms
  - And do not need to care about boilerplate code
- Practioners are not interested in theory
  - Just need a good algorithm for their task
  - Without having to study the field for years

⇒ There is need for an algorithm library

- With lots of algorithms for all kinds of tasks
- That are easy to integrate into existing systems
- And are combinable
- And easily enhanceable

A Library Of Algorithms

Typical Problems In WSN Programming

- Theoreticians are not interested in programming
  - Ideally they just have to write their algorithms
  - And do not need to care about boilerplate code
- Practioners are not interested in theory
  - Just need a good algorithm for their task
  - Without having to study the field for years

⇒ There is need for an algorithm library

- With lots of algorithms for all kinds of tasks
- That are easy to integrate into existing systems
- And are combinable
- And easily enhanceable

Typical Problems In WSN Programming

- Theoreticians are not interested in programming
  - Ideally they just have to write their algorithms
  - And do not need to care about boilerplate code
- Practioners are not interested in theory
  - Just need a good algorithm for their task
  - Without having to study the field for years

⇒ There is need for an algorithm library

- With lots of algorithms for all kinds of tasks
- That are easy to integrate into existing systems
- And are combinable
- And easily enhanceable

Motivation Design Of The Wiselib Hardware Abstraction with OS Facets
A Library Of Algorithms
Typical Problems In WSN Programming
- Theoreticians are not interested in programming
  - Ideally they just have to write their algorithms
  - And do not need to care about boilerplate code
- Practioners are not interested in theory
  - Just need a good algorithm for their task
  - Without having to study the field for years

⇒ There is need for an algorithm library

- With lots of algorithms for all kinds of tasks
- That are easy to integrate into existing systems
- And are combinable
- And easily enhanceable

Ioannis Chatzigiannakis Pervasive Systems Lecture 21 2 / 49

Solution

The Wiselib

A library of about 50 algorithms, lots more to come! These are

- Extensible
- Combineable
- Exchangeable

Currently includes the following algorithm categories

- Clustering
- Graph Coloring
- Crypto
- Energy Preservation
- Localization
- Metrics
- Routing
- Synchronization
- Topology Control
- Tracking

The Wiselib is. . .

- A C++ project
- Free (as in freedom), licensed under LGPL
- **NOT** a middleware (we will see later why)

`github.com/ibr-alg/wiselib`

There you'll find:

- The Documentation Wiki
- The Wiselib Sourcecode
- The Bugtracker
- Instructions on how to download & install the Wiselib

Wiselib Distributions

Testing

- Under development
- Not necessarily tested on all platforms
- New things that may still change their interface
- "Release early, release often"

Stable

- Tested on all supported platforms
- Interfaces will not change anymore

Platform Independence

- When scientists all over the world work together, they likely use different experimentation environments
- The Wiselib aims to be **versatile**
  - So it can be used for **different tasks**
  - Which also require **different hardware**
- In lots of applications we need **heterogeneous nodes**
  - But do not want to write the same code again and again for each node type

→ We want the Wiselib to be platform independent!

Platform Independence

	iSense	**iMote2**	**ScatterWeb MSB**	**Tmote Sky**
				
Hardware	Jennic	Intel XScale	MSP430	MSP 430
Operating System	iSense	TinyOS	Scatterweb / Contiki	Contiki / TinyOS
ROM / RAM	128kB / 92kB	32MB / 32MB	48kB / 10kB	48kB / 10kB
Memory Management	Dynamic	Dynamic	Static	Dynamic
Programming Language	C++	nesC	C	C, nesC

Platform Independence

- Some platforms do not provide dynamic memory
- And/or have limited RAM
- Some do not provide a C++ environment
  - No libstdc++
  - So no exception handling, RTTI, virtual inheritance, etc...

The "extremely portable" subset of C++

- C (except `malloc` / `free`)
- Static memory management
- "Simple" (non-virtual) inheritance
- Templates
- Use C-Headers (`<math.h>` instead of `<cmath>`)

The Wiselib adheres to those conditions!

Memory Management

Platform independence demands:

- No `malloc/free` or `new/delete`

→ Data can be allocated in 3 ways:

- Global
- Static
- On the stack (function-local)

- Constructors of global/static variables will be called before `main()`
- . . . in undefined order!
- That can be very undesirable:

```
1 Radio radio;
2 SomeAlgorithm algo(radio); // Might receive uninitialized radio!
```

→ Provide `init()/destruct()` methods, call them manually

→ Hide initialization method of system objects ("Facets")

Motivation | Design Of The Wiselib | Hardware Abstraction with OS Facets
Memory Management
Memory Management
Platform independence demands:
- No `malloc/free` or `new/delete`
→ Data can be allocated in 3 ways:
- Global
- Static
- On the stack (function-local)
- Constructors of global/static variables will be called before `main()`
- . . . in undefined order!
- That can be very undesirable:
1 Radio radio;
2 SomeAlgorithm algo(radio); // Might receive uninitialized radio!
→ Provide `init()/destruct()` methods, call them manually
→ Hide initialization method of system objects ("Facets")

Ioannis Chatzigiannakis | Pervasive Systems | Lecture 21 9 / 49
Motivation | Design Of The Wiselib | Hardware Abstraction with OS Facets
Memory Management
Memory Management
Platform independence demands:
- No `malloc/free` or `new/delete`
→ Data can be allocated in 3 ways:
- Global
- Static
- On the stack (function-local)
- Constructors of global/static variables will be called before `main()`
- . . . in undefined order!
- That can be very undesirable:
1 Radio radio;
2 SomeAlgorithm algo(radio); // Might receive uninitialized radio!
→ Provide `init()/destruct()` methods, call them manually
→ Hide initialization method of system objects ("Facets")

Ioannis Chatzigiannakis | Pervasive Systems | Lecture 21 9 / 49
Motivation | Design Of The Wiselib | Hardware Abstraction with OS Facets
Memory Management
Memory Management
Platform independence demands:
- No `malloc/free` or `new/delete`
→ Data can be allocated in 3 ways:
- Global
- Static
- On the stack (function-local)
- Constructors of global/static variables will be called before `main()`
- . . . in undefined order!
- That can be very undesirable:
1 Radio radio;
2 SomeAlgorithm algo(radio); // Might receive uninitialized radio!
→ Provide `init()/destruct()` methods, call them manually
→ Hide initialization method of system objects ("Facets")

Ioannis Chatzigiannakis | Pervasive Systems | Lecture 21 9 / 49

Memory Management

Platform independence demands:

- No `malloc/free` or `new/delete`

→ Data can be allocated in 3 ways:

- Global
- Static
- On the stack (function-local)

- Constructors of global/static variables will be called before `main()`
- . . . in undefined order!
- That can be very undesirable:

```
1 Radio radio;
2 SomeAlgorithm algo(radio); // Might receive uninitialized radio!
```

→ Provide `init()/destruct()` methods, call them manually

→ Hide initialization method of system objects ("Facets")

Inheritance

Problem: Virtual inheritance is not portable.

What would we use virtual inheritance for?

→ **Code reuse**
Base class provides functionality which can be used by derived class
- Still possible with non-virtual inheritance

→ **Abstraction**
Define an *interface* which *classes* can use to interact with each other
- An algorithm only has to know the interface of the things its using, the concrete implementation is exchangeable

We want both!

Inheritance

Problem: Virtual inheritance is not portable.

What would we use virtual inheritance for?

→ **Code reuse**
Base class provides functionality which can be used by derived class
- Still possible with non-virtual inheritance

→ **Abstraction**
Define an *interface* which *classes* can use to interact with each other
- An algorithm only has to know the interface of the things its using, the concrete implementation is exchangeable

We want both!

Inheritance

Problem: Virtual inheritance is not portable.

What would we use virtual inheritance for?

→ **Code reuse**
Base class provides functionality which can be used by derived class
- Still possible with non-virtual inheritance

→ **Abstraction**
Define an *interface* which *classes* can use to interact with each other
- An algorithm only has to know the interface of the things its using, the concrete implementation is exchangeable

We want both!

Inheritance

Problem: Virtual inheritance is not portable.

What would we use virtual inheritance for?

→ **Code reuse**
Base class provides functionality which can be used by derived class
- Still possible with non-virtual inheritance

→ **Abstraction**
Define an *interface* which *classes* can use to interact with each other
- An algorithm only has to know the interface of the things its using, the concrete implementation is exchangeable

We want both!

Motivation Design Of The Wiselib Hardware Abstraction with OS Facets
Abstraction With Templates

Inheritance

Problem: Virtual inheritance is not portable.

What would we use virtual inheritance for?

→ **Code reuse**
Base class provides functionality which can be used by derived class
- Still possible with non-virtual inheritance

→ **Abstraction**
Define an *interface* which *classes* can use to interact with each other
- An algorithm only has to know the interface of the things its using, the concrete implementation is exchangeable

We want both!

Inheritance

Do it with templates!

- The "interface" is given by a piece of documentation, called **Concept**
- An algorithm expects a template parameter for the type of the concrete class, which is called **Model**

Template Based Design

```
1 class iSenseRadioModel {
2   int enable_radio() {}
3 }
```

```
1 class ShawnRadioModel {
2   int enable_radio() {}
3 }
```

```
1 template<typename Radio_P>
2 class Algorithm
3 {
4   typedef Radio_P Radio;
5
6   int init( Radio& radio ) {
7     radio_ = &radio;
8     radio_->enable_radio();
9   }
10
11  Radio::self_pointer_t radio_;
12 }
```

```
1 Algorithm<iSenseRadioModel> algrithm_isense;
2 Algorithm<ShawnRadioModel> algrithm_shawn;
```

Template Based Design

```
1 class iSenseRadioModel {
2   int enable_radio() {}
3 }
```

```
1 class ShawnRadioModel {
2   int enable_radio() {}
3 }
```

```
1 template<typename Radio_P>
2 class Algorithm
3 {
4   typedef Radio_P Radio;
5
6   int init( Radio& radio ) {
7     radio_ = &radio;
8     radio_->enable_radio();
9   }
10
11  Radio::self_pointer_t radio_;
12 }
```

```
1 Algorithm<iSenseRadioModel> algrithm_isense;
2 Algorithm<ShawnRadioModel> algrithm_shawn;
```

Motivation | Design Of The Wiselib | Hardware Abstraction with OS Facets
Abstraction With Templates

Template Based Design

```
1 class iSenseRadioModel {
2   int enable_radio() {}
3 }
```

```
1 class ShawnRadioModel {
2   int enable_radio() {}
3 }
```

```
1 template<typename Radio_P>
2 class Algorithm
3 {
4   typedef Radio_P Radio;
5
6   int init( Radio& radio ) {
7     radio_ = &radio;
8     radio_->enable_radio();
9   }
10
11  Radio::self_pointer_t radio_;
12 }
```

```
1 Algorithm<iSenseRadioModel> algrithm_isense;
2 Algorithm<ShawnRadioModel> algrithm_shawn;
```

Ioannis Chatzigiannakis | Pervasive Systems | Lecture 21 12 / 49

Motivation | Design Of The Wiselib | Hardware Abstraction with OS Facets
Abstraction With Templates

Template Based Design

```
1 class iSenseRadioModel {
2   int enable_radio() {}
3 }
```

```
1 class ShawnRadioModel {
2   int enable_radio() {}
3 }
```

```
1 template<typename Radio_P>
2 class Algorithm
3 {
4   typedef Radio_P Radio;
5
6   int init( Radio& radio ) {
7     radio_ = &radio;
8     radio_->enable_radio();
9   }
10
11  Radio::self_pointer_t radio_;
12 }
```

```
1 Algorithm<iSenseRadioModel> algrithm_isense;
2 Algorithm<ShawnRadioModel> algrithm_shawn;
```

Ioannis Chatzigiannakis | Pervasive Systems | Lecture 21 12 / 49

Motivation | Design Of The Wiselib | Hardware Abstraction with OS Facets
Abstraction With Templates

Template Based Design

```
1 class iSenseRadioModel {
2   int enable_radio() {}
3 }
```

```
1 class ShawnRadioModel {
2   int enable_radio() {}
3 }
```

```
1 template<typename Radio_P>
2 class Algorithm
3 {
4   typedef Radio_P Radio;
5
6   int init( Radio& radio ) {
7     radio_ = &radio;
8     radio_->enable_radio();
9   }
10
11  Radio::self_pointer_t radio_;
12 }
```

```
1 Algorithm<iSenseRadioModel> algrithm_isense;
2 Algorithm<ShawnRadioModel> algrithm_shawn;
```

Abstraction

Concept

- Describes behaviour of components
- E.g. "A Radio has a void send(char*) method"
- Only documentation

Model

- Actual class
- Implements any number of concepts
- E.g. A routing protocol may implement the radio concept
- ...so it can be used like one

Motivation | Design Of The Wiselib | Hardware Abstraction with OS Facets
Abstraction With Templates
Abstraction
Concept
Describes behaviour of components
E.g. "A Radio has a void send(char*) method"
Only documentation
Model
Actual class
Implements any number of concepts
E.g. A routing protocol may implement the radio concept
...so it can be used like one
Ioannis Chatzigiannakis | Pervasive Systems | Lecture 21 14 / 49
Motivation | Design Of The Wiselib | Hardware Abstraction with OS Facets
Abstraction With Templates
Abstraction
Concept
Describes behaviour of components
E.g. "A Radio has a void send(char*) method"
Only documentation
Model
Actual class
Implements any number of concepts
E.g. A routing protocol may implement the radio concept
...so it can be used like one
Ioannis Chatzigiannakis | Pervasive Systems | Lecture 21 14 / 49
Motivation | Design Of The Wiselib | Hardware Abstraction with OS Facets
Abstraction With Templates
Abstraction
Concept
Describes behaviour of components
E.g. "A Radio has a void send(char*) method"
Only documentation
Model
Actual class
Implements any number of concepts
E.g. A routing protocol may implement the radio concept
...so it can be used like one
Ioannis Chatzigiannakis | Pervasive Systems | Lecture 21 14 / 49

Abstraction

Concept

- Describes behaviour of components
- E.g. "A Radio has a void send(char*) method"
- Only documentation

Model

- Actual class
- Implements any number of concepts
- E.g. A routing protocol may implement the radio concept
- ...so it can be used like one

Motivation | Design Of The Wiselib | Hardware Abstraction with OS Facets
Abstraction With Templates

Abstraction

Concept

- Describes behaviour of components
- E.g. "A Radio has a void send(char*) method"
- Only documentation

Model

- Actual class
- Implements any number of concepts
- E.g. A routing protocol may implement the radio concept
- ...so it can be used like one

Motivation | Design Of The Wiselib | Hardware Abstraction with OS Facets
Abstraction With Templates

Abstraction

Concept

- Describes behaviour of components
- E.g. "A Radio has a void send(char*) method"
- Only documentation

Model

- Actual class
- Implements any number of concepts
- E.g. A routing protocol may implement the radio concept
- ...so it can be used like one

How Usable Is The Template Approach?

- There are other ways to provide abstraction
  - In C, one would usually abstract with **function pointers**
  - In C++ one would use **virtual inheritance**

How do they compare to the template approach?

Abstracting with C function pointers

```
1 // C
2 typedef struct {
3   int (*value)(void);
4 } Concept;
5
6 int model_value() { return 5; }
7 Concept model = { .value = &model_value };
8
9 void algorithm(Concept *c) {
10   // pointer->pointer->function
11   int v = c->value();
12 }
13
14 int main(int argc, char** argv) {
15   algorithm(&model);
16 }
```

Abstracting with virtual inheritance

```
1 // C++
2 class Concept {
3   public:
4     virtual int value();
5 };
6
7 class Model : public Concept {
8   public:
9     int value() { return 5; }
10 };
11
12 class Algorithm {
13   public:
14     // reference->vtable->function
15     void init(Concept& c) { v = c.value(); }
16     int v;
17 };
18
19 int main(int argc, char** argv) {
20   Model m;
21   Algorithm a;
22   a.init(m);
23 }
```

Abstracting with templates

```
1 // C++
2
3 // concept "Concept" {
4 //   has an 'int value()' method
5 // }
6
7 class Model {
8   public:
9     int value() { return 5; }
10 };
11
12 template<typename Concept_P>
13 class Algorithm {
14   public:
15     // reference->function
16     void init(Concept_P& c) { v = c.value(); }
17     int v;
18 };
19
20 int main(int argc, char** argv) {
21   Model m;
22   Algorithm<Model> a;
23   a.init(m);
24 }
```

Comparing the results

After compiling (for jennic, using ba-elf-gcc/ba-elf-g++) with -Os:

```
1 text    data     bss     dec     hex filename
2   56       4       0      60      3c c.o
3   16       0       0      16      10 template.o
4  143       0       0     143      8f virtual.o
```

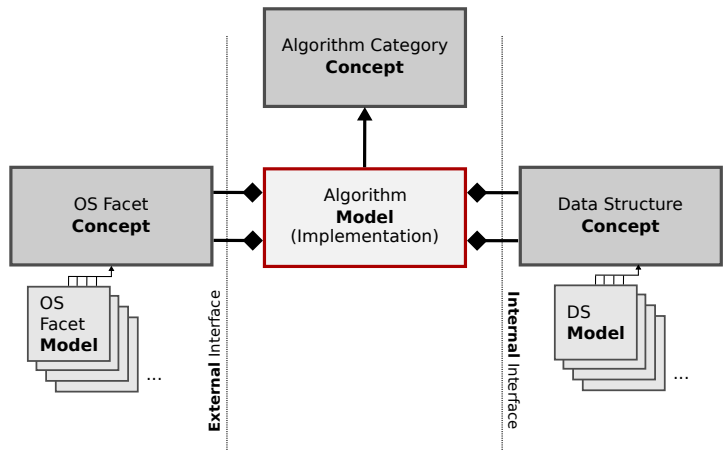
→ Template-based design is space efficient!
→ Template-based design produces fast code!
→ Template-based design is portable!

Comparing the results

After compiling (for jennic, using ba-elf-gcc/ba-elf-g++) with -Os:

```
1 text    data     bss     dec     hex filename
2   56       4       0      60      3c c.o
3   16       0       0      16      10 template.o
4  143       0       0     143      8f virtual.o
```

→ Template-based design is space efficient!
→ Template-based design produces fast code!
→ Template-based design is portable!



Types Of Concepts

Concept Organization

- Lots of models
- Lots of concepts
- Models that behave similar should share concepts
- E.g. A routing algorithm should be usable like a radio
  For the user, both are just things that
  - Can receive messages
  - Can send messages to nodes
  - Only the neighborhood is different!
- But a routing algorithm might have additional methods!

→ We want a (loose) hierarchy of concepts
→ We want to express concept inheritance
→ We want to have "base concepts" for general things

Motivation | Design Of The Wiselib | Hardware Abstraction with OS Facets
Concept Architecture

Concept Organization

- Lots of models
- Lots of concepts
- Models that behave similar should share concepts
- E.g. A routing algorithm should be usable like a radio
  For the user, both are just things that
  - Can receive messages
  - Can send messages to nodes
  - Only the neighborhood is different!
- But a routing algorithm might have additional methods!

→ We want a (loose) hierarchy of concepts
→ We want to express concept inheritance
→ We want to have "base concepts" for general things

Ioannis Chatzigiannakis | Pervasive Systems | Lecture 21 21 / 49

Concept Organization

- Lots of models
- Lots of concepts
- Models that behave similar should share concepts
- E.g. A routing algorithm should be usable like a radio
  For the user, both are just things that
  - Can receive messages
  - Can send messages to nodes
  - Only the neighborhood is different!
- But a routing algorithm might have additional methods!

→ We want a (loose) hierarchy of concepts
→ We want to express concept inheritance
→ We want to have "base concepts" for general things

Motivation | Design Of The Wiselib | Hardware Abstraction with OS Facets
Concept Architecture

Concept Organization

- Lots of models
- Lots of concepts
- Models that behave similar should share concepts
- E.g. A routing algorithm should be usable like a radio
  For the user, both are just things that
  - Can receive messages
  - Can send messages to nodes
  - Only the neighborhood is different!
- But a routing algorithm might have additional methods!

→ We want a (loose) hierarchy of concepts
→ We want to express concept inheritance
→ We want to have "base concepts" for general things

Motivation | Design Of The Wiselib | Hardware Abstraction with OS Facets
Concept Architecture

Concept Organization

- Lots of models
- Lots of concepts
- Models that behave similar should share concepts
- E.g. A routing algorithm should be usable like a radio
  For the user, both are just things that
  - Can receive messages
  - Can send messages to nodes
  - Only the neighborhood is different!
- But a routing algorithm might have additional methods!

→ We want a (loose) hierarchy of concepts
→ We want to express concept inheritance
→ We want to have "base concepts" for general things

Motivation | Design Of The Wiselib | Hardware Abstraction with OS Facets
Concept Architecture

The OsModel Facet

```
1  concept OsModel {
2    typedef ... size_t;
3    typedef ... block_data_t; // "byte"-like type for buffers
4    enum ReturnValues { SUCCESS = ..., ERR_UNSPEC = ..., ... };
5
6    typedef ... Radio; // Wireless communication facet
7    typedef ... Timer;
8    typedef ... Debug; // Send debug messages
9
10   static const Endianess endianess; // WISELIB_LITTLE_ENDIAN or
         WISELIB_BIG_ENDIAN
11 }
```

- Holds platform properties (like endianess, size type, etc...)
- Constants for return values
  - Include at least `SUCCESS` and `ERR_UNSPEC` (unspecified error)
  - May/will include more, similar to `errno`
- Holds types of other OS Facets

Concept Inheritance

```
1  concept RadioFacet {
2    typedef ... OsModel;
3    typedef ... node_id_t;
4    typedef ... block_data_t;
5    typedef ... size_t;
6
7    typedef ... message_id_t;
8
9    enum SpecialNodeIds {
10     BROADCAST_ADDRESS = ...,
11     NULL_NODE_ID      = ...
12   };
13   enum Restrictions {
14     MAX_MESSAGE_LENGTH = ...
15   };
16
17   int enable_radio();
18   int disable_radio();
19
20   int send(node_id_t receiver,
         size_t len, block_data_t *
         data );
21
22   node_id_t id();
23
24   // ....
25 };
```

← We "derive" another concept from this one:

```
1  concept VariablePowerRadioFacet
2    : public RadioFacet
3  {
4    // Everything in RadioFacet plus:
5
6    typedef ... TxPower;
7
8    int set_power(TxPower p);
9    TxPower power();
10 };
```

Motivation | Design Of The Wiselib | Hardware Abstraction with OS Facets
Concept Architecture

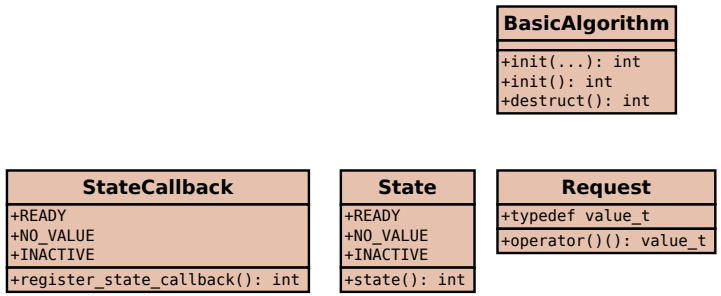
Concept Inheritance


```
1  concept RadioFacet {
2    typedef ... OsModel;
3    typedef ... node_id_t;
4    typedef ... block_data_t;
5    typedef ... size_t;
6
7    typedef ... message_id_t;
8
9    enum SpecialNodeIds {
10     BROADCAST_ADDRESS = ...,
11     NULL_NODE_ID      = ...
12   };
13   enum Restrictions {
14     MAX_MESSAGE_LENGTH = ...
15   };
16
17   int enable_radio();
18   int disable_radio();
19
20   int send(node_id_t receiver,
         size_t len, block_data_t *
         data );
21
22   node_id_t id();
23
24   // ....
25 };
```

← We "derive" another concept from this one:

```
1  concept VariablePowerRadioFacet
2    : public RadioFacet
3  {
4    // Everything in RadioFacet plus:
5
6    typedef ... TxPower;
7
8    int set_power(TxPower p);
9    TxPower power();
10 };
```


Base Concepts



Basic Algorithm Manual initialization & destruction (so the order is defineable)

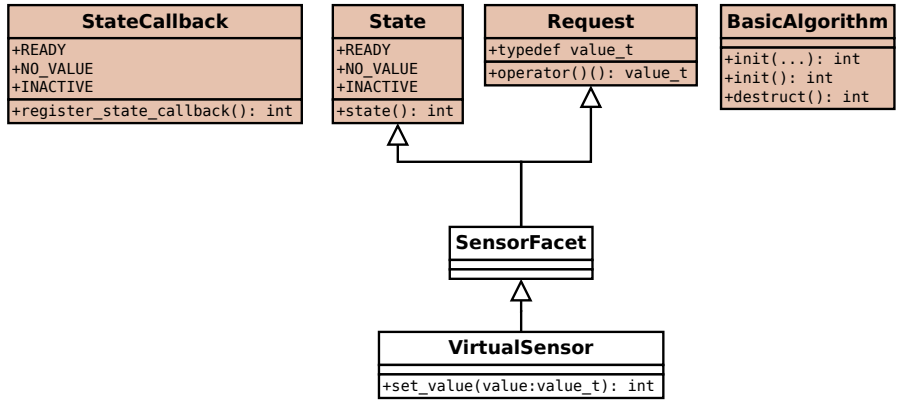
Request Produces values (can be polled with call-operator)

State Object is not guaranteed to be able to operate all the time

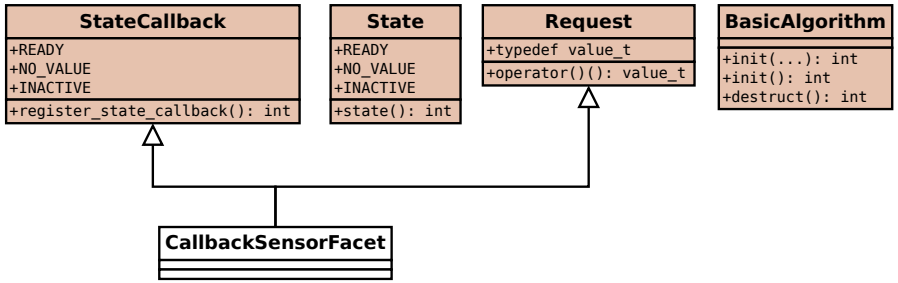
StateCallback Object can inform its user about state changes

Motivation | Design Of The Wiselib | Hardware Abstraction with OS Facets
Concept Architecture

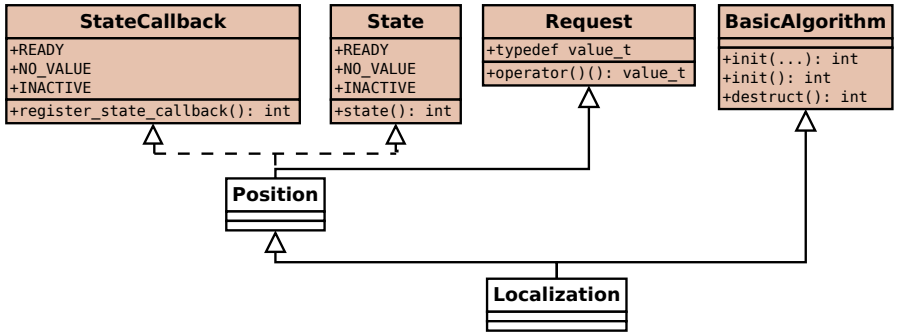
Base Concepts



Base Concepts



Base Concepts



Stackability

Idea: Things with similar behaviour should share a concept!

Routing algorithms behave like radios
- They send and receive data to other nodes

→ Routing algorithms implement the *Radio Concept*

Localization algorithms produce a stream of values
- So do sensors!

→ Localization algorithms implement the *Sensor Concept* or the *CallbackSensor Concept*

Etc. . .

Benefit
- Say some algorithm uses a radio (i.e. transmits data)
- We can pass a routing algorithm instead
- And extend the algorithms functionality that way!

Stackability

Idea: Things with similar behaviour should share a concept!

Routing algorithms behave like radios
- They send and receive data to other nodes

→ Routing algorithms implement the *Radio Concept*

Localization algorithms produce a stream of values
- So do sensors!

→ Localization algorithms implement the *Sensor Concept* or the *CallbackSensor Concept*

Etc. . .

Benefit
- Say some algorithm uses a radio (i.e. transmits data)
- We can pass a routing algorithm instead
- And extend the algorithms functionality that way!

Stackability

Idea: Things with similar behaviour should share a concept!

Routing algorithms behave like radios
- They send and receive data to other nodes

→ Routing algorithms implement the *Radio Concept*

Localization algorithms produce a stream of values
- So do sensors!

→ Localization algorithms implement the *Sensor Concept* or the *CallbackSensor Concept*

Etc...

Benefit
- Say some algorithm uses a radio (i.e. transmits data)
- We can pass a routing algorithm instead
- And extend the algorithms functionality that way!

Motivation | Design Of The Wiselib | Hardware Abstraction with OS Facets
Concept Architecture

Stackability

Idea: Things with similar behaviour should share a concept!

Routing algorithms behave like radios
- They send and receive data to other nodes

→ Routing algorithms implement the *Radio Concept*

Localization algorithms produce a stream of values
- So do sensors!

→ Localization algorithms implement the *Sensor Concept* or the *CallbackSensor Concept*

Etc...

Benefit
- Say some algorithm uses a radio (i.e. transmits data)
- We can pass a routing algorithm instead
- And extend the algorithms functionality that way!

Ioannis Chatzigiannakis | Pervasive Systems | Lecture 21 28 / 49

Stackability

- Create arbitrary complex applications
- Just by plugging together algorithms

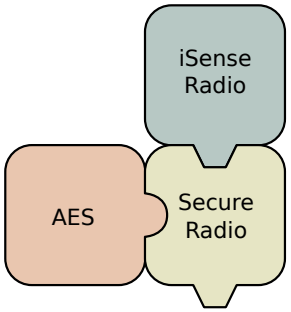
Here:

1. "Physical" radio by iSense
2. AES-Encrypted node-to-node radio
3. Routing, all packets AES-encrypted node-to-node
4. All packets AES-encrypted node-to-node, payload ECC encrypted end-to-end

...can be used like a single simple radio!

Motivation | Design Of The Wiselib | Hardware Abstraction with OS Facets
Concept Architecture

Stackability



- Create arbitrary complex applications
- Just by plugging together algorithms

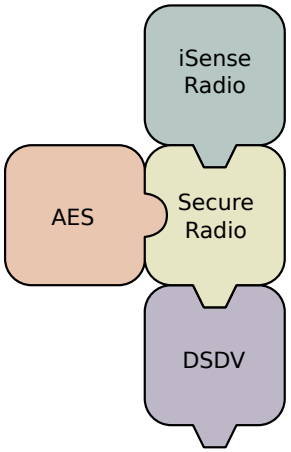
Here:

1. "Physical" radio by iSense
2. AES-Encrypted node-to-node radio
3. Routing, all packets AES-encrypted node-to-node
4. All packets AES-encrypted node-to-node, payload ECC encrypted end-to-end

...can be used like a single simple radio!

Ioannis Chatzigiannakis | Pervasive Systems | Lecture 21 29 / 49

Stackability



- Create arbitrary complex applications
- Just by plugging together algorithms

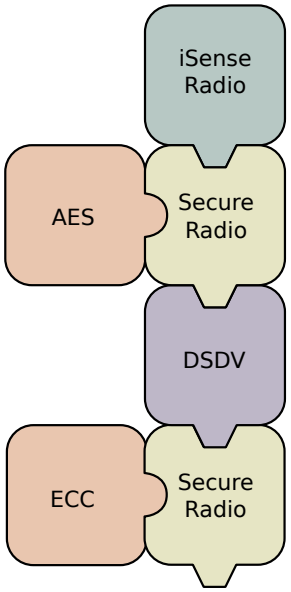
Here:

1. "Physical" radio by iSense
2. AES-Encrypted node-to-node radio
3. Routing, all packets AES-encrypted node-to-node
4. All packets AES-encrypted node-to-node, payload ECC encrypted end-to-end

...can be used like a single simple radio!

Motivation | Design Of The Wiselib | Hardware Abstraction with OS Facets
Concept Architecture

Stackability



- Create arbitrary complex applications
- Just by plugging together algorithms

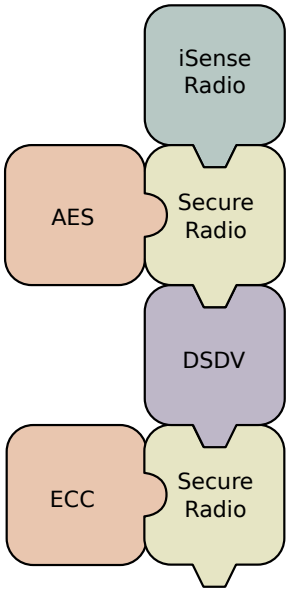
Here:

1. "Physical" radio by iSense
2. AES-Encrypted node-to-node radio
3. Routing, all packets AES-encrypted node-to-node
4. All packets AES-encrypted node-to-node, payload ECC encrypted end-to-end

...can be used like a single simple radio!

Motivation | Design Of The Wiselib | Hardware Abstraction with OS Facets
Concept Architecture

Stackability



- Create arbitrary complex applications
- Just by plugging together algorithms

Here:

1. "Physical" radio by iSense
2. AES-Encrypted node-to-node radio
3. Routing, all packets AES-encrypted node-to-node
4. All packets AES-encrypted node-to-node, payload ECC encrypted end-to-end

...can be used like a single simple radio!

Integration Demands

- Wiselib components should be easily integrable into existing code
- We want and/or need the full power of the platform. Examples:
  - Dynamically discover attached sensors
  - Fine-tuned device configuration

BUT

- Sometimes you want to run the same application on different platforms
- Advanced hardware settings are relatively unimportant

Two different integration mechanisms needed!

Integration Demands

- Wiselib components should be easily integrable into existing code
- We want and/or need the full power of the platform.
  Examples:
  - Dynamically discover attached sensors
  - Fine-tuned device configuration

BUT

- Sometimes you want to run the same application on different platforms
- Advanced hardware settings are relatively unimportant

Two different integration mechanisms needed!

Motivation Design Of The Wiselib Hardware Abstraction with OS Facets
Integrating The Wiselib

Integration Demands

- Wiselib components should be easily integrable into existing code
- We want and/or need the full power of the platform.
  Examples:
  - Dynamically discover attached sensors
  - Fine-tuned device configuration

BUT

- Sometimes you want to run the same application on different platforms
- Advanced hardware settings are relatively unimportant

Two different integration mechanisms needed!

Ioannis Chatzigiannakis Pervasive Systems Lecture 21 30 / 49

Integration Mechanisms

Direct Integration

→ Just use whatever parts of the Wiselib you like

⊕ Retain full power of your platform

⊕ Good if you have existing code

⊖ Not portable

Motivation Design Of The Wiselib Hardware Abstraction with OS Facets
Integrating The Wiselib

Integration Mechanisms

Generic Application

→ Write a Wiselib application class

⊕ Can be compiled for all Wiselib backends

⊖ You can only access the operating system through facets

⊖ But functionality will be limited to a common subset

E.g. you have to write "extremely portable" C++ (no `new`/`delete`, RTTI, exceptions, ...) in order to retain portability

Ioannis Chatzigiannakis Pervasive Systems Lecture 21 31 / 49

Direct Integration

```
// ...

void iSenseDemoApplication::boot(void) {
    os_.debug("WiselibExample::boot");
    routing_.enable();
    routing_.reg_recv_callback<
        iSenseDemoApplication,
        &iSenseDemoApplication::receive_routing_message>(this);

    os_.allow_sleep(false);
    os_.add_task_in(isense::Time(MILLISECONDS), this, 0);
}

// ...
```

- iSense specific code
- Wiselib specific code

Motivation | Design Of The Wiselib | Hardware Abstraction with OS Facets
Integrating The Wiselib
Direct Integration
// ...
void iSenseDemoApplication::boot(void) {
os_.debug("WiselibExample::boot");
routing_.enable();
routing_.reg_recv_callback<
iSenseDemoApplication,
&iSenseDemoApplication::receive_routing_message>(this);
os_.allow_sleep(false);
os_.add_task_in(isense::Time(MILLISECONDS), this, 0);
}
// ...
iSense specific code
Wiselib specific code
Ioannis Chatzigiannakis | Pervasive Systems | Lecture 21 32 / 49

Motivation | Design Of The Wiselib | Hardware Abstraction with OS Facets
Integrating The Wiselib
Direct Integration
// ...
void iSenseDemoApplication::boot(void) {
os_.debug("WiselibExample::boot");
routing_.enable();
routing_.reg_recv_callback<
iSenseDemoApplication,
&iSenseDemoApplication::receive_routing_message>(this);
os_.allow_sleep(false);
os_.add_task_in(isense::Time(MILLISECONDS), this, 0);
}
// ...
iSense specific code
Wiselib specific code
Ioannis Chatzigiannakis | Pervasive Systems | Lecture 21 32 / 49

Motivation | Design Of The Wiselib | Hardware Abstraction with OS Facets
Integrating The Wiselib
Direct Integration
// ...
void iSenseDemoApplication::boot(void) {
os_.debug("WiselibExample::boot");
routing_.enable();
routing_.reg_recv_callback<
iSenseDemoApplication,
&iSenseDemoApplication::receive_routing_message>(this);
os_.allow_sleep(false);
os_.add_task_in(isense::Time(MILLISECONDS), this, 0);
}
// ...
iSense specific code
Wiselib specific code
Ioannis Chatzigiannakis | Pervasive Systems | Lecture 21 32 / 49

Generic Application

```
1 #include "external_interface/external_interface.h"
2 #include "external_interface/external_interface_testing.h"
3 // ...
4
5 typedef wiselib::PCOsModel Os;
6 class DemoApplication {
7   public:
8     void init(Os::AppMainParameter& amp) {
9       radio_ = &wiselib::FacetProvider<Os, Os::Radio>::get_facet(amp);
10      debug_ = &wiselib::FacetProvider<Os, Os::Debug>::get_facet(amp);
11
12      algorithm_.init();
13
14      radio_->enable_radio();
15      debug_->debug("Initialized.\n");
16    }
17
18  private:
19    Os::Debug::self_pointer_t debug_;
20    Os::Radio::self_pointer_t radio_;
21    SomeAlgorithm algorithm_;
22 };
23
24 wiselib::WiselibApplication<Os, DemoApplication> demo_app;
25 void application_main(Os::AppMainParameter& amp) {
26   demo_app.init(amp);
27 }
```

Generic Application

```
1 #include "external_interface/external_interface.h"
2 #include "external_interface/external_interface_testing.h"
3 // ...
4
5 typedef wiselib::PCOsModel Os;
6 class DemoApplication {
7   public:
8     void init(Os::AppMainParameter& amp) {
9       radio_ = &wiselib::FacetProvider<Os, Os::Radio>::get_facet(amp);
10      debug_ = &wiselib::FacetProvider<Os, Os::Debug>::get_facet(amp);
11
12      algorithm_.init();
13
14      radio_->enable_radio();
15      debug_->debug("Initialized.\n");
16    }
17
18  private:
19    Os::Debug::self_pointer_t debug_;
20    Os::Radio::self_pointer_t radio_;
21    SomeAlgorithm algorithm_;
22 };
23
24 wiselib::WiselibApplication<Os, DemoApplication> demo_app;
25 void application_main(Os::AppMainParameter& amp) {
26   demo_app.init(amp);
27 }
```

Platform selection

Generic Application

```
1 #include "external_interface/external_interface.h"
2 #include "external_interface/external_interface_testing.h"
3 // ...
4
5 typedef wiselib::PCOsModel Os;
6 class DemoApplication {
7   public:
8     void init(Os::AppMainParameter& amp) {
9       radio_ = &wiselib::FacetProvider<Os, Os::Radio>::get_facet(amp);
10      debug_ = &wiselib::FacetProvider<Os, Os::Debug>::get_facet(amp);
11
12      algorithm_.init();
13
14      radio_->enable_radio();
15      debug_->debug("Initialized.\n");
16    }
17
18  private:
19    Os::Debug::self_pointer_t debug_;
20    Os::Radio::self_pointer_t radio_;
21    SomeAlgorithm algorithm_;
22 };
23
24 wiselib::WiselibApplication<Os, DemoApplication> demo_app;
25 void application_main(Os::AppMainParameter& amp) {
26   demo_app.init(amp);
27 }
```

Initialization: FacetProvider for OS facets / Manual for algorithms

Generic Application

```
1 #include "external_interface/external_interface.h"
2 #include "external_interface/external_interface_testing.h"
3 // ...
4
5 typedef wiselib::PCOsModel Os;
6 class DemoApplication {
7   public:
8     void init(Os::AppMainParameter& amp) {
9       radio_ = &wiselib::FacetProvider<Os, Os::Radio>::get_facet(amp);
10      debug_ = &wiselib::FacetProvider<Os, Os::Debug>::get_facet(amp);
11
12      algorithm_.init();
13
14      radio_->enable_radio();
15      debug_->debug("Initialized.\n");
16    }
17
18  private:
19    Os::Debug::self_pointer_t debug_;
20    Os::Radio::self_pointer_t radio_;
21    SomeAlgorithm algorithm_;
22 };
23
24 wiselib::WiselibApplication<Os, DemoApplication> demo_app;
25 void application_main(Os::AppMainParameter& amp) {
26   demo_app.init(amp);
27 }
```

`application_main` getting called by Wiselib ↔ OS adaptor

What is a Facet?

- **Connection** between algorithms and OS
- **OS Facets** (Concepts)
  - OS Facet
  - Radio Facet
  - Timer Facet
  - ...
- For **each** supported OS at least **one model** per facet
  - iSenseOsModel
  - ContikiRadioModel
  - ShawnTimerModel
  - ...
- **Possible** to provide **muliple models** per facet
  - ContikiRimeRadioModel
  - Contiki6LowPanRadioModel
  - ...

Motivation | Design Of The Wiselib | Hardware Abstraction with OS Facets
Introduction
What is a Facet?
Connection between algorithms and OS
OS Facets (Concepts)
OS Facet
Radio Facet
Timer Facet
...
For each supported OS at least one model per facet
iSenseOsModel
ContikiRadioModel
ShawnTimerModel
...
Possible to provide muliple models per facet
ContikiRimeRadioModel
Contiki6LowPanRadioModel
...
Ioannis Chatzigiannakis | Pervasive Systems | Lecture 21 34 / 49

Motivation | Design Of The Wiselib | Hardware Abstraction with OS Facets
Introduction
What is a Facet?
Connection between algorithms and OS
OS Facets (Concepts)
OS Facet
Radio Facet
Timer Facet
...
For each supported OS at least one model per facet
iSenseOsModel
ContikiRadioModel
ShawnTimerModel
...
Possible to provide muliple models per facet
ContikiRimeRadioModel
Contiki6LowPanRadioModel
...
Ioannis Chatzigiannakis | Pervasive Systems | Lecture 21 34 / 49

Motivation | Design Of The Wiselib | Hardware Abstraction with OS Facets
Introduction
What is a Facet?
Connection between algorithms and OS
OS Facets (Concepts)
OS Facet
Radio Facet
Timer Facet
...
For each supported OS at least one model per facet
iSenseOsModel
ContikiRadioModel
ShawnTimerModel
...
Possible to provide muliple models per facet
ContikiRimeRadioModel
Contiki6LowPanRadioModel
...
Ioannis Chatzigiannakis | Pervasive Systems | Lecture 21 34 / 49

OS Facet 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)

Exchangeability with Algorithms

- Basic design issue: **Flexibility**
- Pass an **algorithm** where a **facet** is expected
- Examples
  - Pass routing algorithm where radio is expected
    ⇒ Enable flexible multihop neighborhoods
  - Pass time-synchronization algorithm where clock is expected
    ⇒ Enable system-wide time basis
  - Pass localization algorithm where position is expected
    ⇒ Only some nodes in the network need to know their position
  - Pass routing-based debug model where debug facet is expected
    ⇒ Debug nodes that are not connected to a gateway position
- Advantage: Totally **transparent** for algorithm

Motivation | Design Of The Wiselib | Hardware Abstraction with OS Facets
Introduction

Exchangeability with Algorithms

- Basic design issue: **Flexibility**
- Pass an **algorithm** where a **facet** is expected
- Examples
  - Pass routing algorithm where radio is expected
    ⇒ Enable flexible multihop neighborhoods
  - Pass time-synchronization algorithm where clock is expected
    ⇒ Enable system-wide time basis
  - Pass localization algorithm where position is expected
    ⇒ Only some nodes in the network need to know their position
  - Pass routing-based debug model where debug facet is expected
    ⇒ Debug nodes that are not connected to a gateway position
- Advantage: Totally **transparent** for algorithm

Motivation | Design Of The Wiselib | Hardware Abstraction with OS Facets
Introduction

Exchangeability with Algorithms

- Basic design issue: **Flexibility**
- Pass an **algorithm** where a **facet** is expected
- Examples
  - Pass routing algorithm where radio is expected
    ⇒ Enable flexible multihop neighborhoods
  - Pass time-synchronization algorithm where clock is expected
    ⇒ Enable system-wide time basis
  - Pass localization algorithm where position is expected
    ⇒ Only some nodes in the network need to know their position
  - Pass routing-based debug model where debug facet is expected
    ⇒ Debug nodes that are not connected to a gateway position
- Advantage: Totally **transparent** for algorithm

Exchangeability with Algorithms

- Basic design issue: **Flexibility**
- Pass an **algorithm** where a **facet** is expected
- Examples
  - Pass routing algorithm where radio is expected
    ⇒ Enable flexible multihop neighborhoods
  - Pass time-synchronization algorithm where clock is expected
    ⇒ Enable system-wide time basis
  - Pass localization algorithm where position is expected
    ⇒ Only some nodes in the network need to know their position
  - Pass routing-based debug model where debug facet is expected
    ⇒ Debug nodes that are not connected to a gateway position
- Advantage: Totally **transparent** for algorithm

Motivation | Design Of The Wiselib | Hardware Abstraction with OS Facets
Introduction
Exchangeability with Algorithms
- Basic design issue: Flexibility
- Pass an algorithm where a facet is expected
- Examples
  - Pass routing algorithm where radio is expected
    ⇒ Enable flexible multihop neighborhoods
  - Pass time-synchronization algorithm where clock is expected
    ⇒ Enable system-wide time basis
  - Pass localization algorithm where position is expected
    ⇒ Only some nodes in the network need to know their position
  - Pass routing-based debug model where debug facet is expected
    ⇒ Debug nodes that are not connected to a gateway position
- Advantage: Totally transparent for algorithm
Ioannis Chatzigiannakis | Pervasive Systems | Lecture 21 36 / 49

Motivation | Design Of The Wiselib | Hardware Abstraction with OS Facets
Introduction
Exchangeability with Algorithms
- Basic design issue: Flexibility
- Pass an algorithm where a facet is expected
- Examples
  - Pass routing algorithm where radio is expected
    ⇒ Enable flexible multihop neighborhoods
  - Pass time-synchronization algorithm where clock is expected
    ⇒ Enable system-wide time basis
  - Pass localization algorithm where position is expected
    ⇒ Only some nodes in the network need to know their position
  - Pass routing-based debug model where debug facet is expected
    ⇒ Debug nodes that are not connected to a gateway position
- Advantage: Totally transparent for algorithm
Ioannis Chatzigiannakis | Pervasive Systems | Lecture 21 36 / 49

Motivation | Design Of The Wiselib | Hardware Abstraction with OS Facets
Introduction
Exchangeability with Algorithms
- Basic design issue: Flexibility
- Pass an algorithm where a facet is expected
- Examples
  - Pass routing algorithm where radio is expected
    ⇒ Enable flexible multihop neighborhoods
  - Pass time-synchronization algorithm where clock is expected
    ⇒ Enable system-wide time basis
  - Pass localization algorithm where position is expected
    ⇒ Only some nodes in the network need to know their position
  - Pass routing-based debug model where debug facet is expected
    ⇒ Debug nodes that are not connected to a gateway position
- Advantage: Totally transparent for algorithm
Ioannis Chatzigiannakis | Pervasive Systems | Lecture 21 36 / 49

The OS Facet

```
concept OsFacet {
  typedef ... size_t;
  typedef ... block_data_t; // "byte"-like type for buffers
  enum ReturnValues { SUCCESS, EUNSPEC, ... }; // Define constants for return
       values

  typedef ... Radio; // Wireless communication facet
  typedef ... Timer;
  typedef ... Debug; // Send debug messages

  static const Endianess endianess; // WISELIB_LITTLE_ENDIAN or
       WISELIB_BIG_ENDIAN
}
```

- **Only** facet which does **not need** to be **instantiated**
- Provide **type definitions** and **constants**
- Platform properties (endianess, size type, ...)
- Constants for return values
  - Include at least SUCCESS and ERR_UNSPEC (unspecified error)
  - May/will include more, similar to errno
- Default types for basic OS Facets

Motivation | Design Of The Wiselib | Hardware Abstraction with OS Facets
Important Facets
The OS Facet
concept OsFacet {
  typedef ... size_t;
  typedef ... block_data_t; // "byte"-like type for buffers
  enum ReturnValues { SUCCESS, EUNSPEC, ... }; // Define constants for return values
  typedef ... Radio; // Wireless communication facet
  typedef ... Timer;
  typedef ... Debug; // Send debug messages
  static const Endianess endianess; // WISELIB_LITTLE_ENDIAN or WISELIB_BIG_ENDIAN
}
Only facet which does not need to be instantiated
Provide type definitions and constants
Platform properties (endianess, size type, ...)
Constants for return values
Include at least SUCCESS and ERR_UNSPEC (unspecified error)
May/will include more, similar to errno
Default types for basic OS Facets
Ioannis Chatzigiannakis | Pervasive Systems | Lecture 21 37 / 49

Motivation | Design Of The Wiselib | Hardware Abstraction with OS Facets
Important Facets
The OS Facet
concept OsFacet {
  typedef ... size_t;
  typedef ... block_data_t; // "byte"-like type for buffers
  enum ReturnValues { SUCCESS, EUNSPEC, ... }; // Define constants for return values
  typedef ... Radio; // Wireless communication facet
  typedef ... Timer;
  typedef ... Debug; // Send debug messages
  static const Endianess endianess; // WISELIB_LITTLE_ENDIAN or WISELIB_BIG_ENDIAN
}
Only facet which does not need to be instantiated
Provide type definitions and constants
Platform properties (endianess, size type, ...)
Constants for return values
Include at least SUCCESS and ERR_UNSPEC (unspecified error)
May/will include more, similar to errno
Default types for basic OS Facets
Ioannis Chatzigiannakis | Pervasive Systems | Lecture 21 37 / 49

Motivation | Design Of The Wiselib | Hardware Abstraction with OS Facets
Important Facets
The OS Facet
concept OsFacet {
  typedef ... size_t;
  typedef ... block_data_t; // "byte"-like type for buffers
  enum ReturnValues { SUCCESS, EUNSPEC, ... }; // Define constants for return values
  typedef ... Radio; // Wireless communication facet
  typedef ... Timer;
  typedef ... Debug; // Send debug messages
  static const Endianess endianess; // WISELIB_LITTLE_ENDIAN or WISELIB_BIG_ENDIAN
}
Only facet which does not need to be instantiated
Provide type definitions and constants
Platform properties (endianess, size type, ...)
Constants for return values
Include at least SUCCESS and ERR_UNSPEC (unspecified error)
May/will include more, similar to errno
Default types for basic OS Facets
Ioannis Chatzigiannakis | Pervasive Systems | Lecture 21 37 / 49

The OS Facet

```
concept OsFacet {
  typedef ... size_t;
  typedef ... block_data_t; // "byte"-like type for buffers
  enum ReturnValues { SUCCESS, EUNSPEC, ... }; // Define constants for return values

  typedef ... Radio; // Wireless communication facet
  typedef ... Timer;
  typedef ... Debug; // Send debug messages

  static const Endianess endianess; // WISELIB_LITTLE_ENDIAN or WISELIB_BIG_ENDIAN
}
```

- **Only** facet which does **not need** to be **instantiated**
- Provide **type definitions** and **constants**
- Platform properties (endianess, size type, ...)
- Constants for return values
  - Include at least `SUCCESS` and `ERR_UNSPEC` (unspecified error)
  - May/will include more, similar to `errno`
- Default types for basic OS Facets

Radio Facet (1 of 4)

- Design issues
  - Abstraction to underlying hardware radio
  - Complex routing algorithms
  - *Virtual* radio providing *virtual* ids
- Send messages to other nodes
- Callback registration for received messsages
- Provide node id (and its **type**!)
  - Node id type is defined **per radio**
  - E.g., provide IP addresses, but run on 16-bit addresses
  - Only restriction: Be passed to `sizeof()`

Radio Facet (1 of 4)

- Design issues
  - Abstraction to underlying hardware radio
  - Complex routing algorithms
  - *Virtual* radio providing *virtual* ids
- Send messages to other nodes
- Callback registration for received messsages
- Provide node id (and its **type**!)
  - Node id type is defined **per radio**
  - E.g., provide IP addresses, but run on 16-bit addresses
  - Only restriction: Be passed to `sizeof()`

Radio Facet (1 of 4)

- Design issues
  - Abstraction to underlying hardware radio
  - Complex routing algorithms
  - *Virtual* radio providing *virtual* ids
- Send messages to other nodes
- Callback registration for received messsages
- Provide node id (and its **type**!)
  - Node id type is defined **per radio**
  - E.g., provide IP addresses, but run on 16-bit addresses
  - Only restriction: Be passed to `sizeof()`

Radio Facet (1 of 4)

- Design issues
  - Abstraction to underlying hardware radio
  - Complex routing algorithms
  - *Virtual* radio providing *virtual* ids
- Send messages to other nodes
- Callback registration for received messsages
- Provide node id (and its **type**!)
  - Node id type is defined **per radio**
  - E.g., provide IP addresses, but run on 16-bit addresses
  - Only restriction: Be passed to `sizeof()`

Radio Facet (2 of 4)

```
1  concept RadioFacet {
2    typedef ... node_id_t;
3    typedef ... block_data_t;
4    typedef ... size_t;
5    typedef ... message_id_t;
```

- Ability to provide **arbitrary** node ID types
- Message ID type to identify received messages

```
9    enum SpecialNodeIds { BROADCAST_ADDRESS = ..., NULL_NODE_ID = ... };
10   enum Restrictions { MAX_MESSAGE_LENGTH = ... };
```

- Basic constants for broadcasting and unknown nodes
- Maximal message length defined **per radio**

```
11   int enable_radio();
12   int disable_radio();
```

- Turn on/off radio
- Return `SUCCESS` or error code if failed

Motivation | Design Of The Wiselib | Hardware Abstraction with OS Facets
Important Facets
Radio Facet (2 of 4)
concept RadioFacet {
typedef ... node_id_t;
typedef ... block_data_t;
typedef ... size_t;
typedef ... message_id_t;
Ability to provide arbitrary node ID types
Message ID type to identify received messages
enum SpecialNodeIds { BROADCAST_ADDRESS = ..., NULL_NODE_ID = ... };
enum Restrictions { MAX_MESSAGE_LENGTH = ... };
Basic constants for broadcasting and unknown nodes
Maximal message length defined per radio
int enable_radio();
int disable_radio();
Turn on/off radio
Return SUCCESS or error code if failed
Ioannis Chatzigiannakis | Pervasive Systems | Lecture 21 | 39 / 49

Radio Facet (3 of 4)

```
13      int send(node_id_t receiver, size_t len, block_data_t *data );
```

- Send message to `receiver` (either unicast or broadcast)
- Return `SUCCESS` or error code if failed

```
14      node_id_t id();
```

- Return node id: Can be of **arbitrary** type

```
15    template<class T, void (T::*TMethod)(node_id_t, size_t, block_data_t*)>
16    int reg_recv_callback(T *obj_pnt);
17
18    int unreg_recv_callback(int cid);
19  }
```

- Callback registration: Return *callback id* (or -1 if failed)
- Pass *callback id* to unregister callback

Motivation | Design Of The Wiselib | Hardware Abstraction with OS Facets
Important Facets

Radio Facet (3 of 4)

13 int send(node_id_t receiver, size_t len, block_data_t *data);

- Send message to receiver (either unicast or broadcast)
- Return SUCCESS or error code if failed

14 node_id_t id();

- Return node id: Can be of arbitrary type

15 template<class T, void (T::*TMethod)(node_id_t, size_t, block_data_t*)>
16 int reg_recv_callback(T *obj_pnt);
17
18 int unreg_recv_callback(int cid);
19 }

- Callback registration: Return callback id (or -1 if failed)
- Pass callback id to unregister callback

Ioannis Chatzigiannakis | Pervasive Systems | Lecture 21 | 40 / 49

Motivation | Design Of The Wiselib | Hardware Abstraction with OS Facets
Important Facets

Radio Facet (3 of 4)

13 int send(node_id_t receiver, size_t len, block_data_t *data);

- Send message to receiver (either unicast or broadcast)
- Return SUCCESS or error code if failed

14 node_id_t id();

- Return node id: Can be of arbitrary type

15 template<class T, void (T::*TMethod)(node_id_t, size_t, block_data_t*)>
16 int reg_recv_callback(T *obj_pnt);
17
18 int unreg_recv_callback(int cid);
19 }

- Callback registration: Return callback id (or -1 if failed)
- Pass callback id to unregister callback

Ioannis Chatzigiannakis | Pervasive Systems | Lecture 21 | 40 / 49

Radio Facet (4 of 4) - Derived Concepts

- VariablePowerRadio

```
1       typedef ... TxPower;
2
3       int set_power(TxPower p);
4       TxPower power();
```

  - Set transmission power
  - Read out TX power to work on value (increment, decrement, ...)

- ExtendedDataRadio

```
1       typedef ... ExtendedData;
2
3       template<class T, void (T::*TMethod)(node_id_t, size_t,
            block_data_t*,
4                                            ExtendedData&)>
5       int reg_recv_callback( T *obj_pnt );
```

  - Register **receive** method with additional parameter
  - Extended data can be LQI, RSSI, ...
    → Again a concept with different `ExtendedData`-models

Radio Facet (4 of 4) - Derived Concepts

- VariablePowerRadio

```
1   typedef ... TxPower;
2
3   int set_power(TxPower p);
4   TxPower power();
```

  - Set transmission power
  - Read out TX power to work on value (increment, decrement, ...)

- ExtendedDataRadio

```
1   typedef ... ExtendedData;
2
3   template<class T, void (T::*TMethod)(node_id_t, size_t,
        block_data_t*,
4                                        ExtendedData&)>
5   int reg_recv_callback( T *obj_pnt );
```

  - Register **receive** method with additional parameter
  - Extended data can be LQI, RSSI, ...
    → Again a concept with different `ExtendedData`-models

Timer Facet

- Event mechanism
- *Wait for given time, then call me back...*

```
1   concept TimerFacet {
2     typedef ... millis_t;
3
4     template<typename T, void (T::*TMethod)(void*)>
5     int set_timer(millis_t millis, T *obj_pnt, void *userdata);
6   }
```

  - Time given in milliseconds
  - Callback registration: Call passed method in given time
  - `userdata` is passed on callback

Timer Facet

- Event mechanism
- *Wait for given time, then call me back...*

```
1   concept TimerFacet {
2     typedef ... millis_t;
3
4     template<typename T, void (T::*TMethod)(void*)>
5     int set_timer(millis_t millis, T *obj_pnt, void *userdata);
6   }
```

  - Time given in milliseconds
  - Callback registration: Call passed method in given time
  - `userdata` is passed on callback

Debug/Logging Facet

- Write out debug or logging data
- Equivalent to `printf()`

```
1   concept DebugFacet
2   {
3     void debug( const char *msg, ... );
4   }
```

  - Only one method: `debug(...)`
  - Usage as `printf()`
    ⇒ debug_->debug("print an int: %d", my_int);

Debug/Logging Facet

- Write out debug or logging data
- Equivalent to `printf()`

```
concept DebugFacet
{
  void debug( const char *msg, ... );
}
```

- Only one method: `debug(...)`
- Usage as `printf()`
  ⇒ debug_->debug("print an int: %d", my_int);

Clock Facet

- Access to system time
- Type defined by model (platform dependent)

```
concept ClockFacet {
  typedef ... time_t;

  enum ClockSpecificData { CLOCKS_PER_SEC = ..., };

  time_t time();
}
```

- Only one method: `time()`
- Number of clock tics per second (`CLOCKS_PER_SEC`):
  → Deal with platform independent time calculations

- Derived Concept: Settable Clock facet

```
int set_time( time_t time );
```

  - Set time (e.g., for time-synchronization)
  - Currently only implemented for iSense

Clock Facet

- Access to system time
- Type defined by model (platform dependent)

```
concept ClockFacet {
  typedef ... time_t;

  enum ClockSpecificData { CLOCKS_PER_SEC = ..., };

  time_t time();
}
```

- Only one method: `time()`
- Number of clock tics per second (`CLOCKS_PER_SEC`):
  → Deal with platform independent time calculations

- Derived Concept: Settable Clock facet

```
int set_time( time_t time );
```

  - Set time (e.g., for time-synchronization)
  - Currently only implemented for iSense

Clock Facet

- Access to system time
- Type defined by model (platform dependent)

```
concept ClockFacet {
  typedef ... time_t;

  enum ClockSpecificData { CLOCKS_PER_SEC = ..., };

  time_t time();
}
```

- Only one method: `time()`
- Number of clock tics per second (`CLOCKS_PER_SEC`):
  → Deal with platform independent time calculations

- Derived Concept: Settable Clock facet

```
int set_time( time_t time );
```

  - Set time (e.g., for time-synchronization)
  - Currently only implemented for iSense

Clock Facet

- Access to system time
- Type defined by model (platform dependent)

```
1  concept ClockFacet {
2    typedef ... time_t;
3
4    enum ClockSpecificData { CLOCKS_PER_SEC = ..., };
5
6    time_t time();
7  }
```

- Only one method: `time()`
- Number of clock tics per second (`CLOCKS_PER_SEC`):
  → Deal with platform independent time calculations

- Derived Concept: Settable Clock facet

```
1      int set_time( time_t time );
```

- Set time (e.g., for time-synchronization)
- Currently only implemented for iSense

Facet Structure

- **Construction** of facets **system dependent**
  - Shawn: A facet needs to know to which processor it belongs
  - iSense: Require access to `isense::Os`
  - Contiki: Only calls to C functions
- Each system with **own constructors**
- Generic Wiselib Application
  - **Construction** must be **hidden** for user
  - Solution: Template based **facet provider**
- Direct Integration
  - Facets are **known** to user
  - **Directly** initialize facets

Facet Structure

- **Construction** of facets **system dependent**
  - Shawn: A facet needs to know to which processor it belongs
  - iSense: Require access to `isense::Os`
  - Contiki: Only calls to C functions
- Each system with **own constructors**
- Generic Wiselib Application
  - **Construction** must be **hidden** for user
  - Solution: Template based **facet provider**
- Direct Integration
  - Facets are **known** to user
  - **Directly** initialize facets

Facet Structure

- **Construction** of facets **system dependent**
  - Shawn: A facet needs to know to which processor it belongs
  - iSense: Require access to `isense::Os`
  - Contiki: Only calls to C functions
- Each system with **own constructors**
- Generic Wiselib Application
  - **Construction** must be **hidden** for user
  - Solution: Template based **facet provider**
- Direct Integration
  - Facets are **known** to user
  - **Directly** initialize facets

Facet Structure

- **Construction** of facets **system dependent**
  - Shawn: A facet needs to know to which processor it belongs
  - iSense: Require access to `isense::Os`
  - Contiki: Only calls to C functions
- Each system with **own constructors**
- Generic Wiselib Application
  - **Construction** must be **hidden** for user
  - Solution: Template based **facet provider**
- Direct Integration
  - Facets are **known** to user
  - **Directly** initialize facets

Generic Wiselib Application (1 of 2)

- Template `FacetProvider`
  → Internals in Session 4

```
1   template<typename OsModel_P,
2            typename Facet_P>
3   class FacetProvider {
4     static Facet& get_facet( AppMainParameter& os );
5   }
```

- Template **specialization** for different platforms
- Method `get_facet()` returns **reference to facet**

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

Motivation | Design Of The Wiselib | Hardware Abstraction with OS Facets
Facet Instantiation

Generic Wiselib Application (1 of 2)

- Template `FacetProvider`
  → Internals in Session 4

```
1   template<typename OsModel_P,
2            typename Facet_P>
3   class FacetProvider {
4     static Facet& get_facet( AppMainParameter& os );
5   }
```

- Template **specialization** for different platforms
- Method `get_facet()` returns **reference to facet**

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

Motivation | Design Of The Wiselib | Hardware Abstraction with OS Facets
Facet Instantiation

Generic Wiselib Application (1 of 2)

- Template `FacetProvider`
  → Internals in Session 4

```
1   template<typename OsModel_P,
2            typename Facet_P>
3   class FacetProvider {
4     static Facet& get_facet( AppMainParameter& os );
5   }
```

- Template **specialization** for different platforms
- Method `get_facet()` returns **reference to facet**

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

Generic Wiselib Application (2 of 2)

- Special Issue: The self_pointer_t

```
void init( Os::AppMainParameter& value )
{
   radio_ = &wiselib::FacetProvider<Os, Os::Radio>::get_facet(
        value );
 }
 ...
Os::Radio::self_pointer_t radio_;
```

- Each facet/algorithm provides self_pointer_t
- Access via radio_->enable_radio()
- Usually, this is just a pointer
  → typedef self_t* self_pointer_t
- For C systems, this can be used to **optimize** code space

Motivation | Design Of The Wiselib | Hardware Abstraction with OS Facets
Facet Instantiation
Generic Wiselib Application (2 of 2)
Special Issue: The self_pointer_t
void init(Os::AppMainParameter& value)
{
   radio_ = &wiselib::FacetProvider<Os, Os::Radio>::get_facet(
        value);
 }
 ...
Os::Radio::self_pointer_t radio_;
Each facet/algorithm provides self_pointer_t
Access via radio_->enable_radio()
Usually, this is just a pointer
→ typedef self_t* self_pointer_t
For C systems, this can be used to optimize code space
Ioannis Chatzigiannakis | Pervasive Systems | Lecture 21 47 / 49

Motivation | Design Of The Wiselib | Hardware Abstraction with OS Facets
Facet Instantiation
Generic Wiselib Application (2 of 2)
Special Issue: The self_pointer_t
void init(Os::AppMainParameter& value)
{
   radio_ = &wiselib::FacetProvider<Os, Os::Radio>::get_facet(
        value);
 }
 ...
Os::Radio::self_pointer_t radio_;
Each facet/algorithm provides self_pointer_t
Access via radio_->enable_radio()
Usually, this is just a pointer
→ typedef self_t* self_pointer_t
For C systems, this can be used to optimize code space
Ioannis Chatzigiannakis | Pervasive Systems | Lecture 21 47 / 49

Motivation | Design Of The Wiselib | Hardware Abstraction with OS Facets
Facet Instantiation
Generic Wiselib Application (2 of 2)
Special Issue: The self_pointer_t
void init(Os::AppMainParameter& value)
{
   radio_ = &wiselib::FacetProvider<Os, Os::Radio>::get_facet(
        value);
 }
 ...
Os::Radio::self_pointer_t radio_;
Each facet/algorithm provides self_pointer_t
Access via radio_->enable_radio()
Usually, this is just a pointer
→ typedef self_t* self_pointer_t
For C systems, this can be used to optimize code space
Ioannis Chatzigiannakis | Pervasive Systems | Lecture 21 47 / 49

iSense Application

- iSense facets usually expect `isense::Os` in constructor

```
template<typename OsModel_P>
class iSenseRadioModel
  : public isense::Receiver
{
  iSenseRadioModel( isense::Os& os )
    : os_(os)
  {
    os_.dispatcher().add_receiver( this );
  }
  ...
}
```

- Directly used as members

```
#include "external_interface/isense/isense_radio.h"
typedef wiselib::iSenseOsModel Os;

class iSenseDemoApplication {

  iSenseDemoApplication( isense::Os& os )
    : isense::Application(os),
      radio_( os )
  {}

  Os::Radio radio_;
}
```

Motivation | Design Of The Wiselib | Hardware Abstraction with OS Facets
Facet Instantiation

iSense Application

- iSense facets usually expect `isense::Os` in constructor

```
template<typename OsModel_P>
class iSenseRadioModel
  : public isense::Receiver
{
  iSenseRadioModel( isense::Os& os )
    : os_(os)
  {
    os_.dispatcher().add_receiver( this );
  }
  ...
}
```

- Directly used as members

```
#include "external_interface/isense/isense_radio.h"
typedef wiselib::iSenseOsModel Os;

class iSenseDemoApplication {

  iSenseDemoApplication( isense::Os& os )
    : isense::Application(os),
      radio_( os )
  {}

  Os::Radio radio_;
}
```

Ioannis Chatzigiannakis | Pervasive Systems | Lecture 21 48 / 49

Shawn Application

- Shawn facets usually expect `ShawnOs` in constructor
  → Defined in `external_interface/shawn/shawn_types.h`

```
template<typename OsModel_P>
class ShawnRadioModel {
  ShawnRadioModel( ShawnOs& os )
    : os_(os)
  {}
  ...
  ShawnOs& os_;
```

- Directly used as members

```
#include "external_interface/shawn/shawn_radio.h"
typedef wiselib::ShawnOsModel Os;

class WiselibExampleProcessor
  : public virtual ExtIfaceProcessor {

  WiselibExampleProcessor()
    : wiselib_radio_ ( os_ )
  {}

  void boot() { os_.proc = this; }

  ShawnOs os_;
  Os::Radio wiselib_radio_;
```

Motivation | Design Of The Wiselib | Hardware Abstraction with OS Facets
Facet Instantiation

Shawn Application

- Shawn facets usually expect `ShawnOs` in constructor
  → Defined in `external_interface/shawn/shawn_types.h`

```
template<typename OsModel_P>
class ShawnRadioModel {
  ShawnRadioModel( ShawnOs& os )
    : os_(os)
  {}
  ...
  ShawnOs& os_;
```

- Directly used as members

```
#include "external_interface/shawn/shawn_radio.h"
typedef wiselib::ShawnOsModel Os;

class WiselibExampleProcessor
  : public virtual ExtIfaceProcessor {

  WiselibExampleProcessor()
    : wiselib_radio_ ( os_ )
  {}

  void boot() { os_.proc = this; }

  ShawnOs os_;
  Os::Radio wiselib_radio_;
```

Ioannis Chatzigiannakis | Pervasive Systems | Lecture 21 49 / 49