Pervasive Systems

Ioannis Chatzigiannakis

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

Lecture 17:
Wiselib: Algorithmic Library for WSN

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 17 2 / 49

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 17 2 / 49

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

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)

`http://wiselib.org`

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

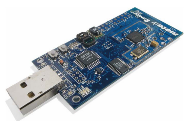
- Tested on all supported platforms
- Interfaces will not change anymore

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 17 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 17 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 17 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:

```
Radio radio;
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!

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!

Ioannis Chatzigiannakis · Pervasive Systems · Lecture 17 · 10 / 49

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!

Ioannis Chatzigiannakis · Pervasive Systems · Lecture 17 · 10 / 49

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

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

```
class iSenseRadioModel {
  int enable_radio() {}
}
```

```
class ShawnRadioModel {
  int enable_radio() {}
}
```

```
template<typename Radio_P>
class Algorithm
{
  typedef Radio_P Radio;

  int init( Radio& radio ) {
    radio_ = &radio;
    radio_->enable_radio();
  }

  Radio::self_pointer_t radio_;
}
```

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

Template Based Design

```
class iSenseRadioModel {
  int enable_radio() {}
}
```

```
class ShawnRadioModel {
  int enable_radio() {}
}
```

```
template<typename Radio_P>
class Algorithm
{
  typedef Radio_P Radio;

  int init( Radio& radio ) {
    radio_ = &radio;
    radio_->enable_radio();
  }

  Radio::self_pointer_t radio_;
}
```

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


Template Based Design

```
class iSenseRadioModel {
  int enable_radio() {}
}
```

```
class ShawnRadioModel {
  int enable_radio() {}
}
```

```
template<typename Radio_P>
class Algorithm
{
  typedef Radio_P Radio;

  int init( Radio& radio ) {
    radio_ = &radio;
    radio_->enable_radio();
  }

  Radio::self_pointer_t radio_;
}
```

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


Template Based Design

```
class iSenseRadioModel {
  int enable_radio() {}
}
```

```
class ShawnRadioModel {
  int enable_radio() {}
}
```

```
template<typename Radio_P>
class Algorithm
{
  typedef Radio_P Radio;

  int init( Radio& radio ) {
    radio_ = &radio;
    radio_->enable_radio();
  }

  Radio::self_pointer_t radio_;
}
```

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


Template Based Design

```
class iSenseRadioModel {
  int enable_radio() {}
}
```

```
class ShawnRadioModel {
  int enable_radio() {}
}
```

```
template<typename Radio_P>
class Algorithm
{
  typedef Radio_P Radio;

  int init( Radio& radio ) {
    radio_ = &radio;
    radio_->enable_radio();
  }

  Radio::self_pointer_t radio_;
}
```

```
Algorithm<iSenseRadioModel> algrithm_isense;
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 17 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 17 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 17 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

Ioannis Chatzigiannakis | Pervasive Systems | Lecture 17 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 17 14 / 49

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

```
// C
typedef struct {
  int (*value)(void);
} Concept;

int model_value() { return 5; }
Concept model = { .value = &model_value };

void algorithm(Concept *c) {
  // pointer->pointer->function
  int v = c->value();
}

int main(int argc, char** argv) {
  algorithm(&model);
}
```


Abstracting with virtual inheritance

```
// C++
class Concept {
  public:
    virtual int value();
};

class Model : public Concept {
  public:
    int value() { return 5; }
};

class Algorithm {
  public:
    // reference->vtable->function
    void init(Concept& c) { v = c.value(); }
    int v;
};

int main(int argc, char** argv) {
  Model m;
  Algorithm a;
  a.init(m);
}
```


Abstracting with templates

```
// C++

// concept "Concept" {
//   has an 'int value()' method
// }

class Model {
  public:
    int value() { return 5; }
};

template<typename Concept_P>
class Algorithm {
  public:
    // reference->function
    void init(Concept_P& c) { v = c.value(); }
    int v;
};

int main(int argc, char** argv) {
  Model m;
  Algorithm<Model> a;
  a.init(m);
}
```


Comparing the results

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

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

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

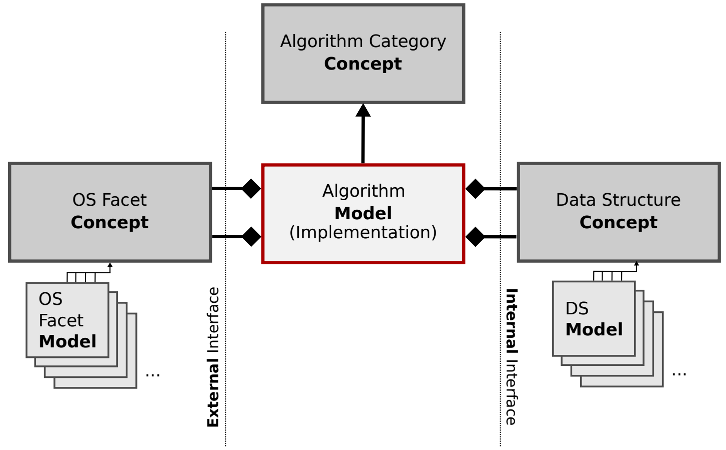
Comparing the results

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

text	data	bss	dec	hex	filename
56	4	0	60	3c	c.o
16	0	0	16	10	template.o
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

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

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

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

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

The OsModel Facet

```
concept OsModel {
  typedef ... size_t;
  typedef ... block_data_t; // "byte"-like type for buffers
  enum ReturnValues { SUCCESS = ..., ERR_UNSPEC = ..., ... };

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

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

- 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

```
concept RadioFacet {
  typedef ... OsModel;
  typedef ... node_id_t;
  typedef ... block_data_t;
  typedef ... size_t;

  typedef ... message_id_t;

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

  int enable_radio();
  int disable_radio();

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

  node_id_t id();

  // ....
};
```

← We “derive” another concept from this one:

```
concept VariablePowerRadioFacet
  : public RadioFacet
{
  // Everything in RadioFacet plus:

  typedef ... TxPower;

  int set_power(TxPower p);
  TxPower power();
};
```


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

Concept Inheritance

```
concept RadioFacet {
  typedef ... OsModel;
  typedef ... node_id_t;
  typedef ... block_data_t;
  typedef ... size_t;

  typedef ... message_id_t;

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

  int enable_radio();
  int disable_radio();

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

  node_id_t id();

  // ....
};
```

← We “derive” another concept from this one:

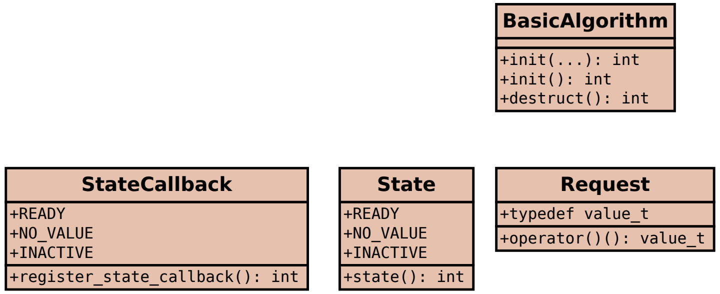
```
concept VariablePowerRadioFacet
  : public RadioFacet
{
  // Everything in RadioFacet plus:

  typedef ... TxPower;

  int set_power(TxPower p);
  TxPower power();
};
```

Ioannis Chatzigiannakis | Pervasive Systems | Lecture 17 | 23 / 49

Base Concepts



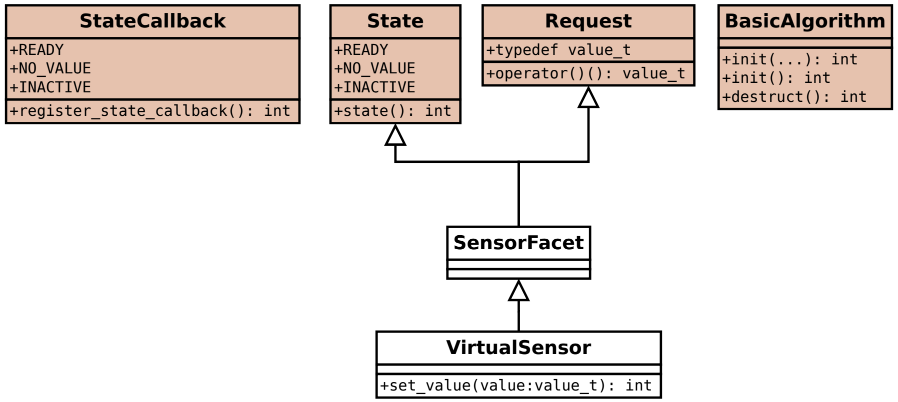
Basic Algorithm Manual initialization & destruction (so the order is defineable)

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

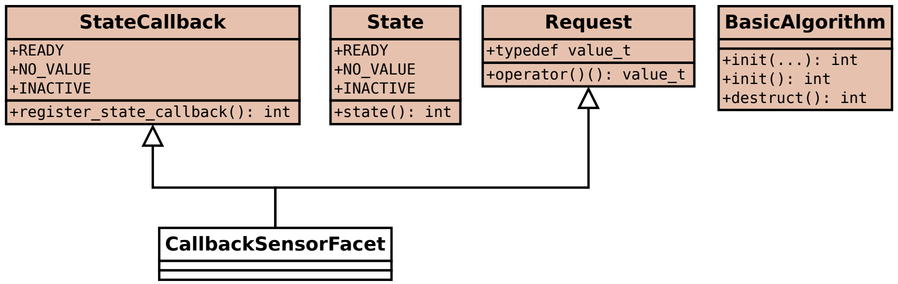
State Object is not guaranteed to be able to operate all the time

StateCallback Object can inform its user about state changes

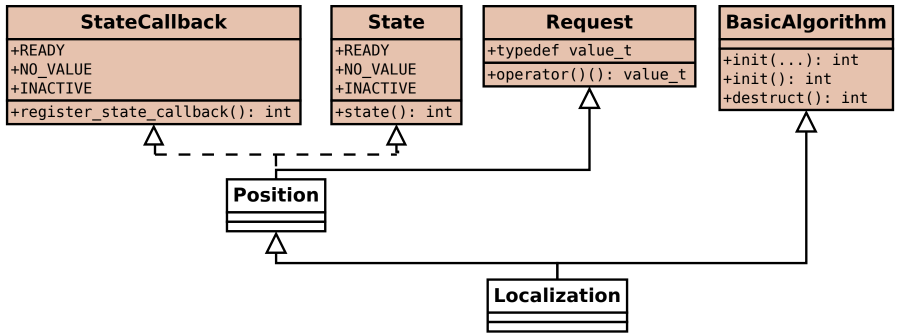
Base Concepts



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*

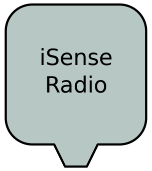
Etc. . .

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 17 28 / 49

Stackability



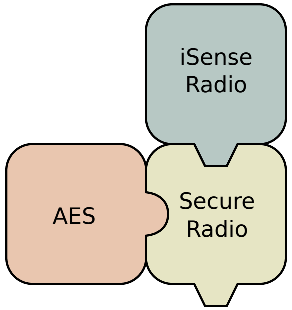
- Create arbitrary complex applications
- Just by plugging together algorithms

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

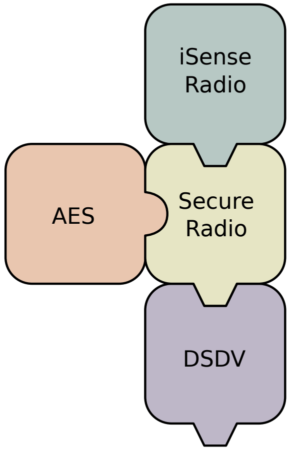
. . . can be used like a single simple radio!

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!

Stackability



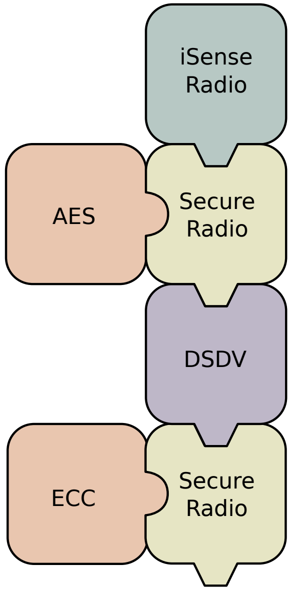
- Create arbitrary complex applications
- Just by plugging together algorithms

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!

Stackability



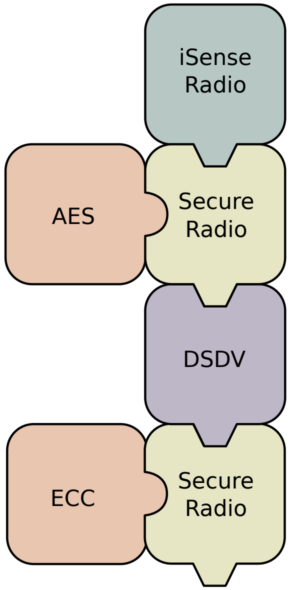
- Create arbitrary complex applications
- Just by plugging together algorithms

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!

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 17 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

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

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 17 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 17 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 17 32 / 49

Generic Application

```
#include "external_interface/external_interface.h"
#include "external_interface/external_interface_testing.h"
// ...

typedef wiselib::PCOsModel Os;
class DemoApplication {
  public:
    void init(Os::AppMainParameter& amp) {
      radio_ = &wiselib::FacetProvider<Os, Os::Radio>::get_facet(amp);
      debug_ = &wiselib::FacetProvider<Os, Os::Debug>::get_facet(amp);

      algorithm_.init();

      radio_->enable_radio();
      debug_->debug("Initialized.\n");
    }

  private:
    Os::Debug::self_pointer_t debug_;
    Os::Radio::self_pointer_t radio_;
    SomeAlgorithm algorithm_;
};

wiselib::WiselibApplication<Os, DemoApplication> demo_app;
void application_main(Os::AppMainParameter& amp) {
  demo_app.init(amp);
}
```


Generic Application


```
#include "external_interface/external_interface.h"
#include "external_interface/external_interface_testing.h"
// ...

typedef wiselib::PCOsModel Os;
class DemoApplication {
  public:
    void init(Os::AppMainParameter& amp) {
      radio_ = &wiselib::FacetProvider<Os, Os::Radio>::get_facet(amp);
      debug_ = &wiselib::FacetProvider<Os, Os::Debug>::get_facet(amp);

      algorithm_.init();

      radio_->enable_radio();
      debug_->debug("Initialized.\n");
    }

  private:
    Os::Debug::self_pointer_t debug_;
    Os::Radio::self_pointer_t radio_;
    SomeAlgorithm algorithm_;
};

wiselib::WiselibApplication<Os, DemoApplication> demo_app;
void application_main(Os::AppMainParameter& amp) {
  demo_app.init(amp);
}
```


Platform selection

Generic Application


```
#include "external_interface/external_interface.h"
#include "external_interface/external_interface_testing.h"
// ...

typedef wiselib::PCOsModel Os;
class DemoApplication {
  public:
    void init(Os::AppMainParameter& amp) {
      radio_ = &wiselib::FacetProvider<Os, Os::Radio>::get_facet(amp);
      debug_ = &wiselib::FacetProvider<Os, Os::Debug>::get_facet(amp);

      algorithm_.init();

      radio_->enable_radio();
      debug_->debug("Initialized.\n");
    }

  private:
    Os::Debug::self_pointer_t debug_;
    Os::Radio::self_pointer_t radio_;
    SomeAlgorithm algorithm_;
};

wiselib::WiselibApplication<Os, DemoApplication> demo_app;
void application_main(Os::AppMainParameter& amp) {
  demo_app.init(amp);
}
```


Initialization: FacetProvider for OS facets / Manual for algorithms

Generic Application


```
#include "external_interface/external_interface.h"
#include "external_interface/external_interface_testing.h"
// ...

typedef wiselib::PCOsModel Os;
class DemoApplication {
  public:
    void init(Os::AppMainParameter& amp) {
      radio_ = &wiselib::FacetProvider<Os, Os::Radio>::get_facet(amp);
      debug_ = &wiselib::FacetProvider<Os, Os::Debug>::get_facet(amp);

      algorithm_.init();

      radio_->enable_radio();
      debug_->debug("Initialized.\n");
    }

  private:
    Os::Debug::self_pointer_t debug_;
    Os::Radio::self_pointer_t radio_;
    SomeAlgorithm algorithm_;
};

wiselib::WiselibApplication<Os, DemoApplication> demo_app;
void application_main(Os::AppMainParameter& amp) {
  demo_app.init(amp);
}
```


`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 17 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 17 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 17 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

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

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 17 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 17 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 17 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 17 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 17 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 17 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)

```
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

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 17 39 / 49

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 17 39 / 49

Radio Facet (3 of 4)

```
    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

```
    node_id_t id();
```

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

```
  template<class T, void (T::*TMethod)(node_id_t, size_t, block_data_t*)>
  int reg_recv_callback(T *obj_pnt);

  int unreg_recv_callback(int cid);
}
```

- 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)
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
node_id_t id();
Return node id: Can be of arbitrary type
template<class T, void (T::*TMethod)(node_id_t, size_t, block_data_t*)>
int reg_recv_callback(T *obj_pnt);
int unreg_recv_callback(int cid);
}
Callback registration: Return callback id (or -1 if failed)
Pass callback id to unregister callback
Ioannis Chatzigiannakis Pervasive Systems Lecture 17 40 / 49

Motivation Design Of The Wiselib Hardware Abstraction with OS Facets
Important Facets
Radio Facet (3 of 4)
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
node_id_t id();
Return node id: Can be of arbitrary type
template<class T, void (T::*TMethod)(node_id_t, size_t, block_data_t*)>
int reg_recv_callback(T *obj_pnt);
int unreg_recv_callback(int cid);
}
Callback registration: Return callback id (or -1 if failed)
Pass callback id to unregister callback
Ioannis Chatzigiannakis Pervasive Systems Lecture 17 40 / 49

Radio Facet (4 of 4) - Derived Concepts

- VariablePowerRadio

```
    typedef ... TxPower;

    int set_power(TxPower p);
    TxPower power();
```

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

- ExtendedDataRadio

```
    typedef ... ExtendedData;

    template<class T, void (T::*TMethod)(node_id_t, size_t,
        block_data_t*,
                                          ExtendedData&)>
    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

```
typedef ... TxPower;

int set_power(TxPower p);
TxPower power();
```

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

- ExtendedDataRadio

```
typedef ... ExtendedData;

template<class T, void (T::*TMethod)(node_id_t, size_t,
      block_data_t*,
                                      ExtendedData&)>
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...*

```
concept TimerFacet {
  typedef ... millis_t;

  template<typename T, void (T::*TMethod)(void*)>
  int set_timer(millis_t millis, T *obj_pnt, void *userdata);
}
```

> - 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...*

```
concept TimerFacet {
  typedef ... millis_t;

  template<typename T, void (T::*TMethod)(void*)>
  int set_timer(millis_t millis, T *obj_pnt, void *userdata);
}
```

> - 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()`

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

> - 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)

```
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

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

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

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

Ioannis Chatzigiannakis | Pervasive Systems | Lecture 17 45 / 49

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

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

Ioannis Chatzigiannakis | Pervasive Systems | Lecture 17 45 / 49

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

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

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

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


Generic Wiselib Application (1 of 2)

- Template `FacetProvider`
  → Internals in Session 4

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

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

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


Generic Wiselib Application (1 of 2)

- Template `FacetProvider`
  → Internals in Session 4

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

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

```
void init( Os::AppMainParameter& value )
{
   radio_ = &wiselib::FacetProvider<Os, Os::Radio>::get_facet(
          value );
 }
 ...
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 17 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 17 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 17 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 17 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 17 49 / 49