Object-Oriented Modelling and Heterogeneous Networks

Joakim Bjørk Einar Broch Johnsen Marcel Kyas Olaf Owe

Department of Informatics, University of Oslo, Norway

Formal Methods for Components and Objects
October 23, 2008, Sophia Antipolis, France

Creol at a glance

- an executable OO modelling language
- targets open distributed systems
- allows to abstracts from the particular properties of the (object) scheduling and of the (network) environment
- operational semantics formally defined in rewriting logic
- the language design supports formal verification

Talk Overview

- Biomedical sensor networks
- Interfaces, classes, and types
- Concurrency, interaction
- Network awareness
- Modelling biomedical sensor networks

No pictures

But the pictures are not the subject matter of geometry and we are not permitted to reason from them. It is true that most people including mathematicians, lean upon these pictures as a crutch and find themselves unable to walk when the crutch is removed.

Morris Kline in the chapter “A Discourse on Method” from “Mathematics in Western Culture”, Oxford University Press, 1953

No pictures

> *But the pictures are not the subject matter of geometry and we are not permitted to reason from them. It is true that most people including mathematicians, lean upon these pictures as a crutch and find themselves unable to walk when the crutch is removed.*

Morris Kline in the chapter “A Discourse on Method” from
“Mathematics in Western Culture”,
Oxford University Press, 1953

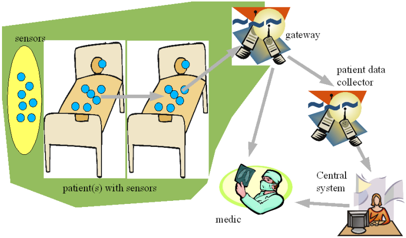
- Pictorial representation of software and structure is only adequate for *tiny* systems
- Pictures tend to abstract too many aspects of a model $\leadsto$ many pictures representing different views
- People usually cannot combine all views and pictures into a consistent model

Important features

- Encourages non-determinism and concurrency
- At all times at most one activity
- Intra-object communication is by shared attributes and cooperative scheduling
- Inter-object communication is by asynchronous method calls only
- Expressions have never side effects
- Statements are the only means to change the state

Biomedical sensor networks

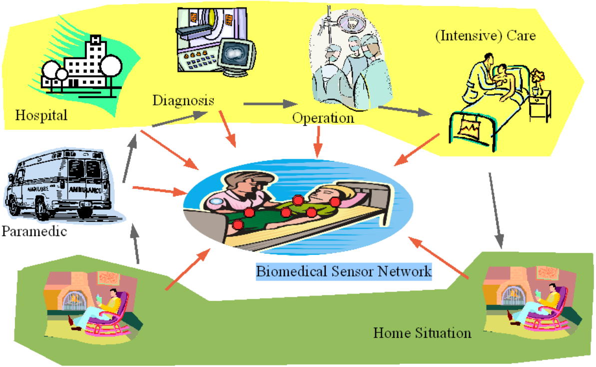
- Biomedical sensors measure body values, e.g. body temperature, heart frequency (EKG), oxygen saturation (SPO_2)
- Systems aggregate and visualise these
- Values out of bounds should be reported
- Data transmission and energy supply by cables



- Cables are obstructive and shall be replaced by wireless transmission
- Multi-hop communication to conserve energy

Context of biomedical sensor networks

- At home
- Emergency/ Paramedics
- Hospital
  - Diagnosis
  - Operation
  - post-operative case



Test platform

- 8 *MHz* TI MSP430 microcontroller (10 *k* RAM, 48 *k* Flash)
- IEEE 802.15.4 Chipcon Wireless Transceiver
- Four modes
  - Sendinf (Tx)
  - Receiving (Rx)
  - Idle
  - Shutdown

Figure: Tmote Sky

Anatomy of a sensors

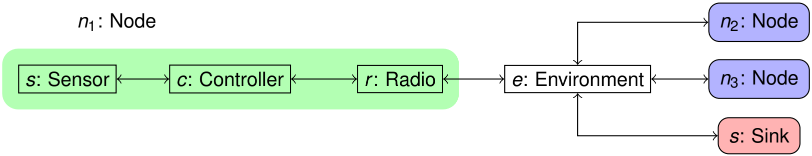


Figure: Structure of the object-oriented model

The “Controller”

- collects data
- processes data
- sends it on the radio
- receives messages on the radio
- decides whether to route
- manages energy

- “Sensor” reads values from patient when commanded by controller
- “Radio” sends and receives data

Properties of communication

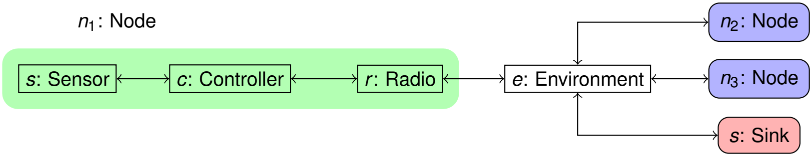


Figure: Structure of the object-oriented model

Internal communication realised with a bus:

- Reliable
- Queued

External communication realised by wireless channels:

- Higher probability of collisions
- Requires rendezvous of sender and receiver

Properties of communication

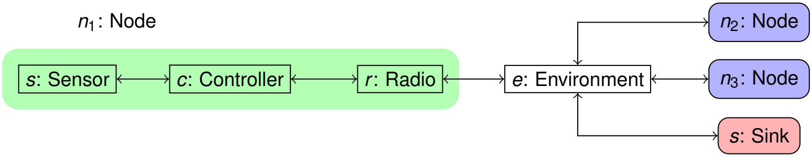


Figure: Structure of the object-oriented model

Internal communication realised with a bus:

- Reliable
- Queued

External communication realised by wireless channels:

- Higher probability of collisions
- Requires rendezvous of sender and receiver

Properties like “throughput” depend on the exact channel used, therefore we need a model of heterogeneous networks.

Properties of communication

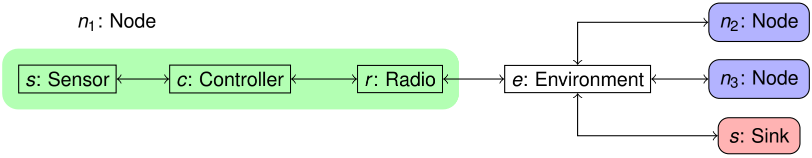


Figure: Structure of the object-oriented model

Internal communication realised with a bus:

- Reliable
- Queued

External communication realised by wireless channels:

- Higher probability of collisions
- Requires rendezvous of sender and receiver

Nodes will turn off their Radios to save energy: Communication is disrupted during these times.

Properties of communication

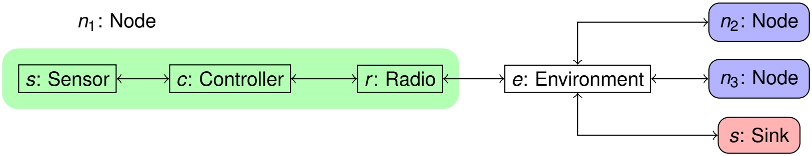


Figure: Structure of the object-oriented model

Internal communication realised with a bus:

- Reliable
- Queued

External communication realised by wireless channels:

- Higher probability of collisions
- Requires rendezvous of sender and receiver

Successful communication between nodes requires that all radio components of participating nodes are on.

Creol

An object-oriented modelling language with:

- Classes, interfaces
- Multiple inheritance, typing and inheritance are disjoint
- Asynchronous method calls with future variables
- Functional expression language
- Library of common data types
- Simple semantics and simple proof system

Creol differs from a programming language in:

- Unspecified scheduling,
- Non-deterministic choice,
- Allowing logical expressions (similar to Hilbert's ε)

Creol language constructs

Syntactic categories	*Definitions*
$C, I, m \in$ Names	$IF ::= \textbf{interface } I \; [\textbf{inherits } \{I\}]$ $\textbf{begin } \{\textbf{with } I \; \{Sg\}\} \textbf{ end}$
$n \in$ Network	$CL ::= \textbf{class } C \, [\{x : I\}] \, [\textbf{inherits } \{C\}] \, [\textbf{implements } \{I\}]$ $\textbf{begin } [\textbf{var } \{\{x\} : \mathrm{I} \, [:= e]\}] \; \{[\textbf{with } I] \; \{M\}\} \textbf{ end}$
$t \in$ Label	$M ::= Sg == [\textbf{var } \{\{x\} : \mathrm{I} \, [:= e]\};] \; s$
$g \in$ Guard	$Sg ::= \textbf{op } m \; ([\textbf{in } \{x : I\}][\textbf{out } \{x : I\}])$
$p \in$ MtdCall	$g ::= b \mid t? \mid g \wedge g \mid g \vee g$
$s \in$ Stmt	$s ::= \textbf{begin } s \textbf{ end} \mid s; s \mid s \,\Box\, s \mid x := e \mid \textbf{release}$
$x \in$ Var	$\mid x := \textbf{new } [\textbf{component}] \; \mathrm{C}[(\{e\})] \mid \textbf{skip}$
$e \in$ Expr	$\mid \textbf{if } b \textbf{ then } s \; [\textbf{else } s] \textbf{ end} \mid [t]![o.]m(\{e\}) \mid t?(x)$
$o \in$ ObjExpr	$\mid \textbf{await } g \mid [\textbf{await}][o.]m(\{e\}; \{x\})$
$b \in$ BoolExpr	

Creol language constructs

Syntactic categories	*Definitions*
$C, I, m \in$ Names	$IF ::=$ **interface** I [**inherits** $\{I\}$] **begin** $\{$**with** I $\{Sg\}\}$ **end**
$n \in$ Network	$CL ::=$ **class** $C\,[\{x : I\}]$ [**inherits** $\{C\}$] [**implements** $\{I\}$] **begin** [**var** $\{\{x\} : \mathrm{I}\,[:= e]\}$] $\{$[**with** I] $\{M\}\}$ **end**
$t \in$ Label	$M ::= Sg ==$ [**var** $\{\{x\} : \mathrm{I}\,[:= e]\};$] s
$g \in$ Guard	$Sg ::=$ **op** m ([**in** $\{x : I\}$][**out** $\{x : I\}$])
$p \in$ MtdCall	$g ::= b \mid t? \mid g \wedge g \mid g \vee g$
$s \in$ Stmt	$s ::=$ **begin** s **end** $\mid s; s \mid s \,\Box\, s \mid x := e \mid$ **release**
$x \in$ Var	$\mid x :=$ **new** [**component**] $\mathrm{C}[(\{e\})] \mid$ **skip**
$e \in$ Expr	$\mid$ **if** b **then** s [**else** s] **end** $\mid [t]![o.]m(\{e\}) \mid t?(x)$
$o \in$ ObjExpr	$\mid$ **await** $g \mid$ [**await**]$[o.]m(\{e\}; \{x\})$
$b \in$ BoolExpr	

Creol language constructs

Syntactic categories	*Definitions*
$C, I, m \in$ Names	$IF ::= \textbf{interface } I \; [\textbf{inherits } \{I\}]$ $\quad \textbf{begin } \{\textbf{with } I \; \{Sg\}\} \textbf{ end}$
$n \in$ Network	$CL ::= \textbf{class } C\,[\{x : I\}]\,[\textbf{inherits } \{C\}]\,[\textbf{implements } \{I\}]$
$t \in$ Label	$\quad \textbf{begin } [\textbf{var } \{\{x\} : \mathrm{I}\,[:= e]\}] \; \{[\textbf{with } I] \; \{M\}\} \textbf{ end}$
$g \in$ Guard	$M ::= Sg == [\textbf{var } \{\{x\} : \mathrm{I}\,[:= e]\};]\; s$
$p \in$ MtdCall	$Sg ::= \textbf{op } m\,([\textbf{in } \{x : I\}][\textbf{out } \{x : I\}])$
$s \in$ Stmt	$g ::= b \mid t? \mid g \wedge g \mid g \vee g$
$x \in$ Var	$s ::= \textbf{begin } s \textbf{ end} \mid s; s \mid s \,\Box\, s \mid x := e \mid$ **release**
$e \in$ Expr	$\quad \mid x := \textbf{new } [\textbf{component}]\; \mathrm{C}[(\{e\})] \mid \textbf{skip}$
$o \in$ ObjExpr	$\quad \mid \textbf{if } b \textbf{ then } s\; [\textbf{else } s] \textbf{ end} \mid [t]![o.]m(\{e\}) \mid t?(x)$
$b \in$ BoolExpr	$\quad \mid \textbf{await } g \mid [\textbf{await}][o.]m(\{e\}; \{x\})$

CREDO

Creol language constructs

Syntactic categories | *Definitions*

$C, I, m \in$ Names
$n \in$ Network
$t \in$ Label
$g \in$ Guard
$p \in$ MtdCall
$s \in$ Stmt
$x \in$ Var
$e \in$ Expr
$o \in$ ObjExpr
$b \in$ BoolExpr

$$IF ::= \textbf{interface } I \ [\textbf{inherits } \{I\}] \ \textbf{begin } \{\textbf{with } I \ \{Sg\}\} \textbf{ end}$$

$$CL ::= \textbf{class } C [\{x : I\}] [\textbf{inherits } \{C\}] [\textbf{implements } \{I\}] \ \textbf{begin } [\textbf{var } \{\{x\} : \mathrm{I} [:= e]\}] \ \{[\textbf{with } I] \ \{M\}\} \textbf{ end}$$

$$M ::= Sg == [\textbf{var } \{\{x\} : \mathrm{I} [:= e]\};] \ s$$

$$Sg ::= \textbf{op } m \ ([\textbf{in } \{x : I\}][\textbf{out } \{x : I\}])$$

$$g ::= b \mid t? \mid g \wedge g \mid g \vee g$$

$$s ::= \textbf{begin } s \textbf{ end} \mid s; s \mid s \,\Box\, s \mid x := e \mid \textbf{release} \mid x := \textbf{new } [\textbf{component}] \ \mathrm{C}[(\{e\})] \mid \textbf{skip} \mid \textbf{if } b \textbf{ then } s \ [\textbf{else } s] \textbf{ end} \mid [t]![o.]m(\{e\}) \mid t?(x) \mid \textbf{await } g \mid [\textbf{await}][o.]m(\{e\}; \{x\})$$

CREDO

Creol language constructs

Syntactic categories	*Definitions*
$C, I, m \in$ Names	$IF ::=$ **interface** I [**inherits** $\{I\}$] **begin** $\{$**with** I $\{Sg\}\}$ **end**
$n \in$ Network	$CL ::=$ **class** $C\,[\{x : I\}]$ [**inherits** $\{C\}$] [**implements** $\{I\}$] **begin** [**var** $\{\{x\} : \mathrm{I}\,[:= e]\}$] $\{$[**with** I] $\{M\}\}$ **end**
$t \in$ Label	$M ::= Sg ==$ [**var** $\{\{x\} : \mathrm{I}\,[:= e]\};$] s
$g \in$ Guard	$Sg ::=$ **op** m ([**in** $\{x : I\}$][**out** $\{x : I\}$])
$p \in$ MtdCall	$g ::= b \mid t? \mid g \wedge g \mid g \vee g$
$s \in$ Stmt	$s ::=$ **begin** s **end** $\mid s; s \mid s \Box s \mid x := e \mid$ **release**
$x \in$ Var	$\mid x :=$ **new** [**component**] $\mathrm{C}[(\{e\})] \mid$ **skip**
$e \in$ Expr	$\mid$ **if** b **then** s [**else** s] **end** $\mid [t]![o.]m(\{e\}) \mid t?(x)$
$o \in$ ObjExpr	$\mid$ **await** $g \mid$ [**await**]$[o.]m(\{e\}; \{x\})$
$b \in$ BoolExpr	

CREDO

Creol language constructs

Syntactic categories	*Definitions*
$C, I, m \in$ Names	$IF ::=$ **interface** I [**inherits** $\{I\}$] **begin** $\{$**with** I $\{Sg\}\}$ **end**
$n \in$ Network	$CL ::=$ **class** $C\,[\{x : I\}]$ [**inherits** $\{C\}$] [**implements** $\{I\}$] **begin** [**var** $\{\{x\} : \mathrm{I}\,[:= e]\}$] $\{$[**with** I] $\{M\}\}$ **end**
$t \in$ Label	$M ::= Sg ==$ [**var** $\{\{x\} : \mathrm{I}\,[:= e]\};$] s
$g \in$ Guard	$Sg ::=$ **op** m ([**in** $\{x : I\}$][**out** $\{x : I\}$])
$p \in$ MtdCall	$g ::= b \mid t? \mid g \wedge g \mid g \vee g$
$s \in$ Stmt	$s ::=$ **begin** s **end** $\mid s; s \mid s \Box s \mid x := e \mid$ **release**
$x \in$ Var	$\mid x :=$ **new** [**component**] $\mathrm{C}[(\{e\})] \mid$ **skip**
$e \in$ Expr	$\mid$ **if** b **then** s [**else** s] **end** $\mid [t]![o.]m(\{e\}) \mid t?(x)$
$o \in$ ObjExpr	$\mid$ **await** $g \mid$ [**await**]$[o.]m(\{e\}; \{x\})$
$b \in$ BoolExpr	

Creol language constructs

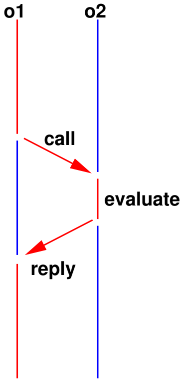
Syntactic categories	*Definitions*
$C, I, m \in$ Names	$IF ::=$ **interface** I [**inherits** $\{I\}$]
	begin $\{$**with** I $\{Sg\}\}$ **end**
$n \in$ Network	$CL ::=$ **class** $C\,[\{x : I\}]$ [**inherits** $\{C\}$] [**implements** $\{I\}$]
$t \in$ Label	**begin** [**var** $\{\{x\} : \mathrm{I}\,[:= e]\}$] $\{$[**with** I] $\{M\}\}$ **end**
$g \in$ Guard	$M ::= Sg == $ [**var** $\{\{x\} : \mathrm{I}\,[:= e]\};$] s
$p \in$ MtdCall	$Sg ::=$ **op** m ([**in** $\{x : I\}$][**out** $\{x : I\}$])
$s \in$ Stmt	$g ::= b \mid t? \mid g \wedge g \mid g \vee g$
$x \in$ Var	$s ::=$ **begin** s **end** $\mid s; s \mid s \,\Box\, s \mid x := e \mid$ **release**
$e \in$ Expr	$\mid x :=$ **new** [**component**] $\mathrm{C}[(\{e\})] \mid$ **skip**
$o \in$ ObjExpr	$\mid$ **if** b **then** s [**else** s] **end** $\mid [t]![o.]m(\{e\}) \mid t?(x)$
$b \in$ BoolExpr	$\mid$ **await** $g \mid$ [**await**]$[o.]m(\{e\}; \{x\})$

CREDO

Creol language constructs

Syntactic categories	*Definitions*
$C, I, m \in \text{Names}$	$IF ::= \textbf{interface}\ I\ [\textbf{inherits}\ \{I\}]$
	$\quad \textbf{begin}\ \{\textbf{with}\ I\ \{Sg\}\}\ \textbf{end}$
$n \in \text{Network}$	$CL ::= \textbf{class}\ C\,[\{x : I\}]\ [\textbf{inherits}\ \{C\}]\ [\textbf{implements}\ \{I\}]$
$t \in \text{Label}$	$\quad \textbf{begin}\ [\textbf{var}\ \{\{x\} : \mathrm{I}\,[:= e]\}]\ \{[\textbf{with}\ I]\ \{M\}\}\ \textbf{end}$
$g \in \text{Guard}$	$M ::= Sg == [\textbf{var}\ \{\{x\} : \mathrm{I}\,[:= e]\};]\ s$
$p \in \text{MtdCall}$	$Sg ::= \textbf{op}\ m\ ([\textbf{in}\ \{x : I\}][\textbf{out}\ \{x : I\}])$
$s \in \text{Stmt}$	$g ::= b \mid t? \mid g \wedge g \mid g \vee g$
$x \in \text{Var}$	$s ::= \textbf{begin}\ s\ \textbf{end} \mid s; s \mid s \,\Box\, s \mid x := e \mid \textbf{release}$
$e \in \text{Expr}$	$\quad \mid x := \textbf{new}\ [\textbf{component}]\ \mathrm{C}[(\{e\})] \mid \textbf{skip}$
$o \in \text{ObjExpr}$	$\quad \mid \textbf{if}\ b\ \textbf{then}\ s\ [\textbf{else}\ s]\ \textbf{end} \mid [t]![o.]m(\{e\}) \mid t?(x)$
$b \in \text{BoolExpr}$	$\quad \mid \textbf{await}\ g \mid [\textbf{await}][o.]m(\{e\}; \{x\})$

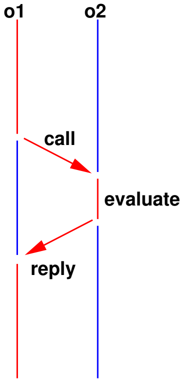
Object orientation: Remote Method Calls



RMI/RPC method call model

- Control threads follow call stack
- Derived from sequential setting
- Hides / ignores distribution!
- Tightly synchronized!

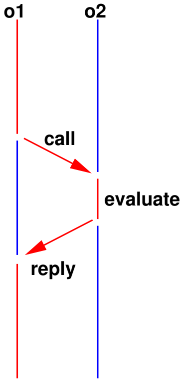
Object orientation: Remote Method Calls



ODS setting:

- Distributed, unstable
- Delays waste processor time
- Message overtaking / loss
- Callee not available?
- Lack of reply: block / deadlock!
- Highly non-deterministic!

Object orientation: Remote Method Calls

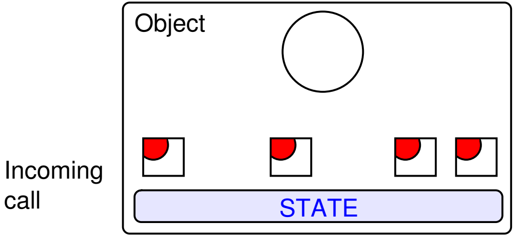


Creol:

- Show / exploit distribution!
- Asynchronous method calls
  - more efficient in distributed environments
  - *triggers* of concurrent activity
- Special cases:
  - *Synchronized communication:* the caller decides to wait for the reply
  - *Sequential computation:* only synchronized computation

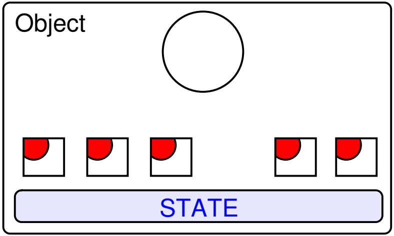
Execution model

- *Concurrent objects* encapsulate a (virtual) processor
- *No assumptions* about the (network) environment
- Execution in objects should *adapt* to the environment
- *Cooperative scheduling* between internal processes inside an object



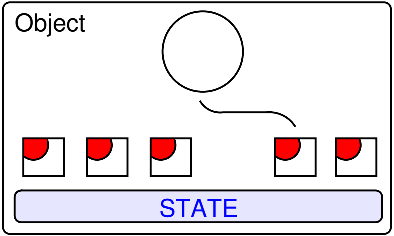
Execution model

- *Concurrent objects* encapsulate a (virtual) processor
- *No assumptions* about the (network) environment
- Execution in objects should *adapt* to the environment
- *Cooperative scheduling* between internal processes inside an object



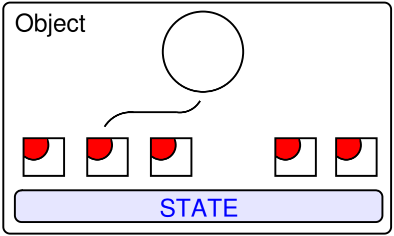
Execution model

- *Concurrent objects* encapsulate a (virtual) processor
- *No assumptions* about the (network) environment
- Execution in objects should *adapt* to the environment
- *Cooperative scheduling* between internal processes inside an object



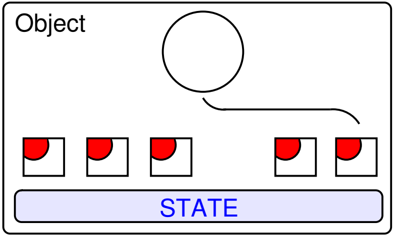
Execution model

- *Concurrent objects* encapsulate a (virtual) processor
- *No assumptions* about the (network) environment
- Execution in objects should *adapt* to the environment
- *Cooperative scheduling* between internal processes inside an object



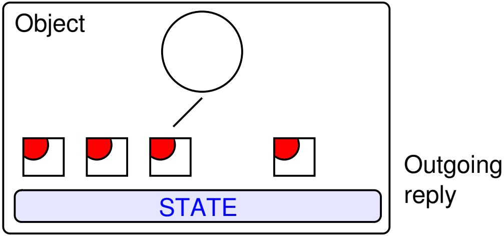
Execution model

- *Concurrent objects* encapsulate a (virtual) processor
- *No assumptions* about the (network) environment
- Execution in objects should *adapt* to the environment
- *Cooperative scheduling* between internal processes inside an object



Execution model

- *Concurrent objects* encapsulate a (virtual) processor
- *No assumptions* about the (network) environment
- Execution in objects should *adapt* to the environment
- *Cooperative scheduling* between internal processes inside an object



Active objects

```
class Radio(sendtime:Int, sleeptime:Int , cycle: Int , sync:Int)
  implements Controllable
begin var on:Bool := true, timer: Int := 0
  op run == while on do
    await (clock − sync) % cycle = 0; timer:= clock;
    while clock <timer + sleeptime do
      if clock = timer + sendtime then send else receive end end end
with Any
  op turnoff == on := false
  op turnon == on := true
  op reset (in time: Int) == sync := time
  op setSend (in time: Int) ==
            if time <sleeptime then sendtime :=time end
  op setSleep (in time: Int) ==
            if sendtime <time ∧ time <cycle then sleeptime := time end
end
```

Communication model

- Originally, *all* communication is asynchronous (sending and receiving need not be simultaneous)
- This fails to capture properties of biomedical sensor networks
- Requirements could neither be expressed nor validated

Communication model

- Originally, *all* communication is asynchronous (sending and receiving need not be simultaneous)
- This fails to capture properties of biomedical sensor networks
- Requirements could neither be expressed nor validated
- From a programmer's point of view, the details of the communication link are irrelevant
- The properties become relevant when analysing the model

Communication model

- Originally, *all* communication is asynchronous (sending and receiving need not be simultaneous)
- This fails to capture properties of biomedical sensor networks
- Requirements could neither be expressed nor validated
- From a programmer's point of view, the details of the communication link are irrelevant
- The properties become relevant when analysing the model

Modelling networks of objects:

- Objects in a distributed system may communicate by links with different properties
- Communication may be asynchronous or synchronous
- Communication may be reliable or lossy

Objects, links, and networks

Refinement of the model:

- Objects have *references* to other objects, i.e., names on which they can invoke methods
- Objects also have *links* to other objects, i.e., channels on which those calls and their replies are transported
- A reference to one object does *not* imply a link to that object
- Objects may need to *route* calls to other objects
- Sometimes modellers need to control the routing mechanism.

Objects, links, and networks

Refinement of the model:

- Objects have *references* to other objects, i.e., names on which they can invoke methods
- Objects also have *links* to other objects, i.e., channels on which those calls and their replies are transported
- A reference to one object does *not* imply a link to that object
- Objects may need to *route* calls to other objects
- Sometimes modellers need to control the routing mechanism. Provide ways to program routing (cross-layer design)

CREDO

Link types

Asynchronous link The link provides for buffering. Sending always succeeds. Messages are received when the receiver wants to. Trying to read a message from such a link will usually block the receiver, if no message is available.

Link types

Asynchronous link The link provides for buffering. Sending always succeeds. Messages are received when the receiver wants to. Trying to read a message from such a link will usually block the receiver, if no message is available.

Rendezvous link The link does not provide buffering. Sending succeeds only when the receiver is receiving (rendezvous). Trying to read a message from such a link will usually block the receiver, if no message is available.

Link types

Asynchronous link The link provides for buffering. Sending always succeeds. Messages are received when the receiver wants to. Trying to read a message from such a link will usually block the receiver, if no message is available.

Rendezvous link The link does not provide buffering. Sending succeeds only when the receiver is receiving (rendezvous). Trying to read a message from such a link will usually block the receiver, if no message is available.

Wireless link The link does not provide buffering. Sending and receiving always “succeed”. Data is transmitted when sending and receiving is *simultaneous*. Sending a message while no object is receiving will lose the message. Receiving without sending will result in a default message.

Link types

Asynchronous link: The link provides for buffering. Sending always succeeds. Messages are received when the receiver wants to. Trying to read a message from such a link will usually block the receiver, if no message is available.

Rendezvous link: The link does not provide buffering. Sending succeeds only when the receiver is receiving (rendezvous). Trying to read a message from such a link will usually block the receiver, if no message is available.

Wireless link: The link does not provide buffering. Sending and receiving always "succeed". Data is transmitted when sending and receiving is *simultaneous*. Sending a message while no object is receiving will lose the message. Receiving without sending will result in a default message.

For the operational model, we need a formalisation of "simultaneous".

Refining the network

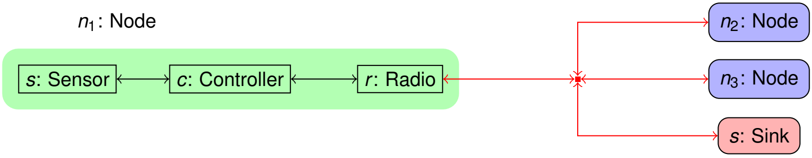


Figure: Structure of the object-oriented model

Red links represent *wireless links*
Black links represent *asynchronous links*

Refining the network

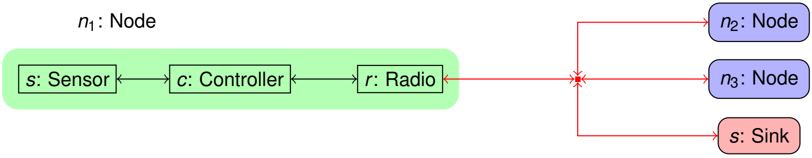


Figure: Structure of the object-oriented model

Red links represent *wireless links*
Black links represent *asynchronous links*

The *environment* object, which controlled the possibility of *collisions* in the network, has been removed. This function is now performed by the run-time environment.

What is in a step?

How do we model *simultaneous* actions in Creol?

- Today, we use a fictitious time model
- Time is abstracted to natural numbers
- Events with the same time stamp occur simultaneously

What is in a step?

How do we model *simultaneous* actions in Creol?

- Today, we use a fictitious time model
- Time is abstracted to natural numbers
- Events with the same time stamp occur simultaneously

We assume that all locally executed statements are instantaneous. Only communication with *external* entities takes time.

What is in a step?

How do we model *simultaneous* actions in Creol?

- Today, we use a fictitious time model
- Time is abstracted to natural numbers
- Events with the same time stamp occur simultaneously

We assume that all locally executed statements are instantaneous. Only communication with *external* entities takes time.

This is sufficient to model wireless links:

- Two sends at the same time: collision
- One send and many reads at the same time: communication
- No send: reads return “no message”

What is in a step?

How do we model *simultaneous* actions in Creol?

- Today, we use a fictitious time model
- Time is abstracted to natural numbers
- Events with the same time stamp occur simultaneously

We assume that all locally executed statements are instantaneous. Only communication with *external* entities takes time.
This is sufficient to model wireless links:

- Two sends at the same time: collision
- One send and many reads at the same time: communication
- No send: reads return “no message”

We do not consider *topology* and *signal strength* here, which can be added to the model.

Network components

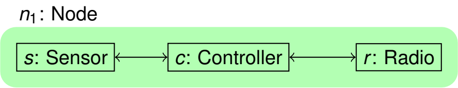


Figure: A network component

- The statement **new component** *C* creates an instance of *C* as a new *component*
- Components are *groups of active objects*
- They share one input and one output queue
- The *group leader* (here the Radio), which is the first object of a component, controls the queue
  - Special statements allow to send a message on a wireless link or to receive from a wireless link
  - This control is needed for controlling possible collisions.
- This way, Creol enables *cross-layer design* for network components
- Objects within one component usually use *asynchronous links*
- Inter-component communication may use user-defined links

What is a component?

- Syntax: **component** N **provides** $\{I\}$ **requires** $\{I\}$ **begin** N $\{C\}$ **end**
- A component aggregates classes
- All used *interfaces* are either internally satisfied or *required* from the environment.
- A subset of interface provided by component classes are *provided* to the environment.
- There is (usually) one instance of N, which is the group leader
- Only the group leader may use statements **send** and **receive** to send or receive messages on wireless links

Conclusion and future work

- The desire to reason about "throughput" in networks with heterogeneous communication links forces us to reveal certain implementation details
- We avoid ad hoc modelling of links by defining the nature of our links precisely in terms of their characteristics
- A light-weight component model aids in describing the system's behaviour
- Cross-layer design and removing abstractions seem to be the only means to meet deployment criteria
- Minimising "middle-ware" (possibly removing it) is a necessity, too.
- We want to go from the abstract model to real implementations
- The case study is a *hard real-time* system, which changes the rules: We need a refined model of time, resource awareness, scheduling, . . .