ReaGeniX Solutions to Some Concurrent System Design Problems

Ari Okkonen

OBP Research Oy

Solution Principles

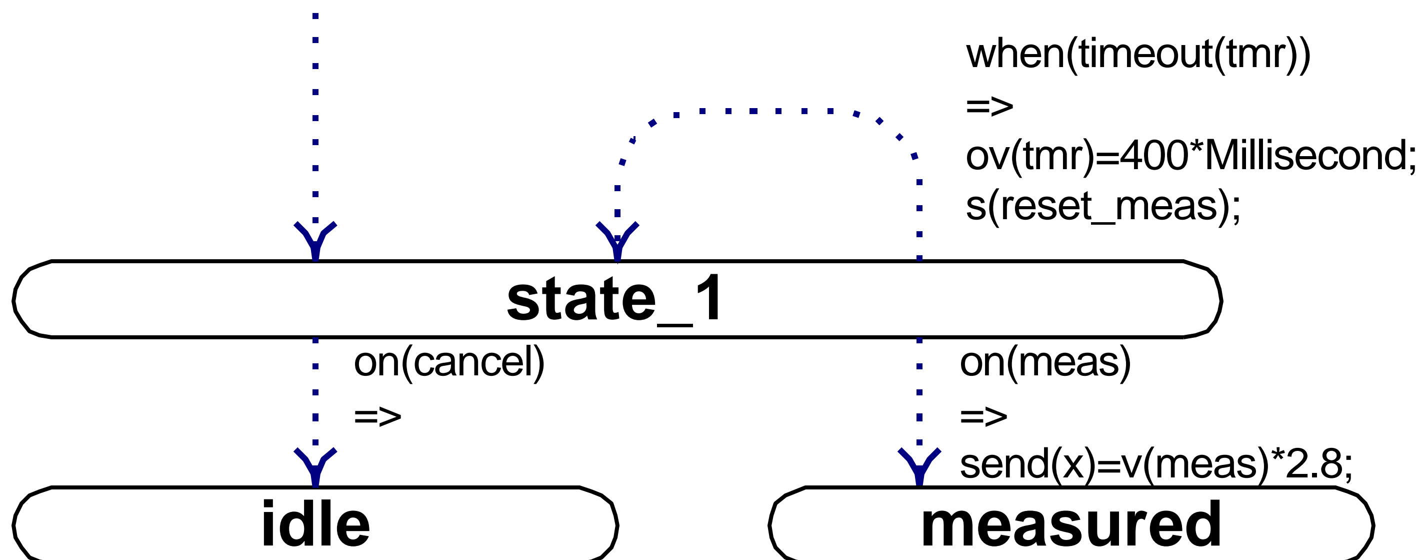
- For more or less independent control or monitoring sequences (processes) use separate **state machines**.
- For same kind of sequences use separate **instances** of the same state machine type.
- State machines can be packed hierarchically to more complex **active components**. A state machine is a simple one.
- Active components use **asynchronous communication**.
- Call C-functions or C++ methods from state machines for nontrivial data processing.
- Use ReaGOS or other RT-OS to use priorities in processing responses to events and to interface to non-ReaGeniX parts of the system.

State Machines for Independent Sequences

- Handles its responsibilities **independently** of the other parts of the system
- Communicates via its **interface**. It does not know, where the inputs are coming from, or where the outputs are going to.
- Connections to other parts of the system are defined outside of the state machine.
- Waits passively for a choice of input events and timed events.
- Responds to events by computation, outputs, setting timers, and changing its state including internal data.

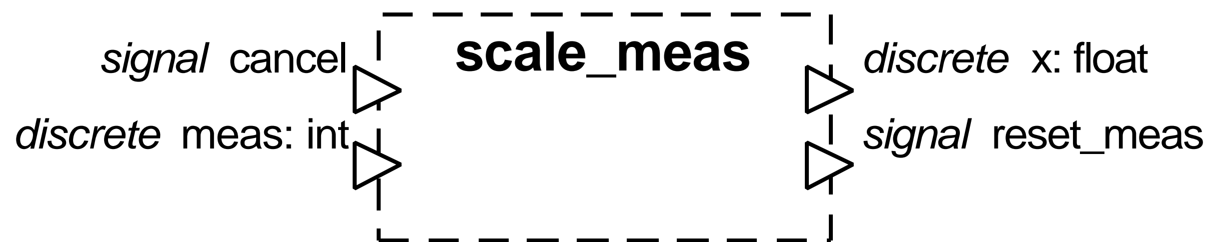
Awaiting a Choice of Events

- When in the *state_1* the state machine is ready to react to the *cancel* signal, to the *meas* message, and to a timeout of the *tmr* timer.



Strict Interfaces for Reusability

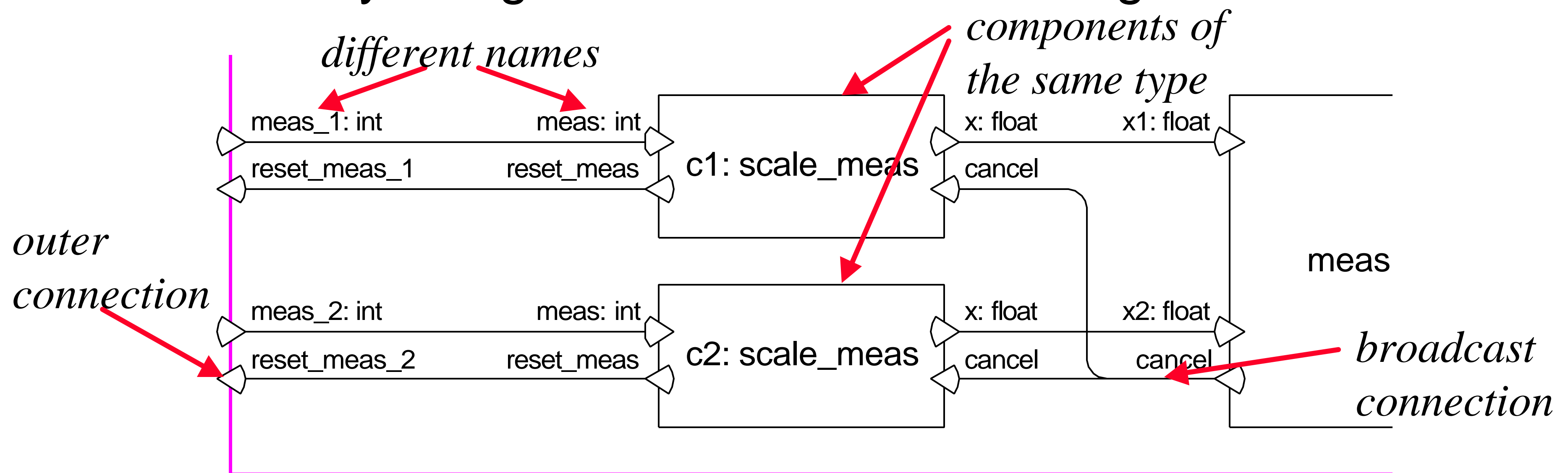
- All communication operations within a state machine are directed to **ports** of the interface.
- The **names** of the ports **are local** making the components portable and reusable.



Interface declaration

Connecting Active Components

- An **architecture diagram** defines the connection of the ports to other components or to the outer interface.
- Several functionally independent instances of the same active component type may be used in the system.
- A single state machine and a hierarchical active component are used similarly in higher level architecture diagrams.



Passing Data Structures and Objects Between State Machines

- **Discrete** connection is used to transmit packets of data.
- Built-in or self defined C types and C++ classes can be used.
- Transmission: `send(`*portname*`) =` *value*`;`
- Transmission of an object:

```
v(portname).method(params); /* setup value */
emit(portname);
```

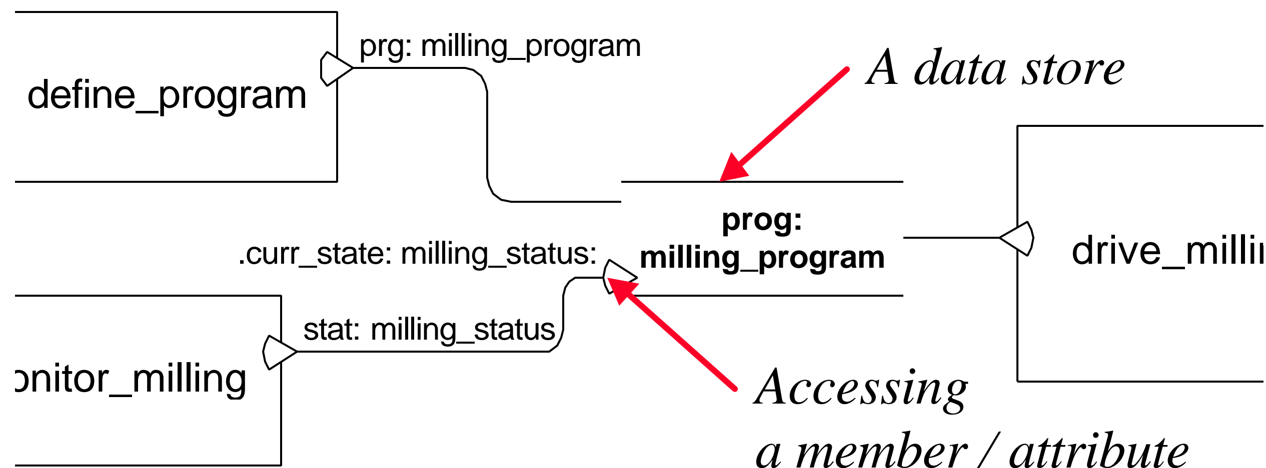
- Condition for receiving: `on(`*portname*`)`
- Expression for received value: `v(`*portname*`)`
- Message to received object: `v(`*portname*`).`*method*`(`*params*`);`
- Special communication type **signal** is a pure event without data.

Continuously Available Values

- Continuous connection has a defined value for all times. It has an initial value, and it retains the latest value until overwritten.
- Continuous connection is used when a change of value must sometimes be recognised immediately, sometimes it does not matter, and sometimes the current value is needed.
- The receiver can react to the event associated to new value.
- The receiver can read the current value when needed.
- The transmitting and receiving operations are same as for discrete communication.
- A continuous connection with Boolean data is called **flag**. It is useful for locking, reporting safe situations, etc.

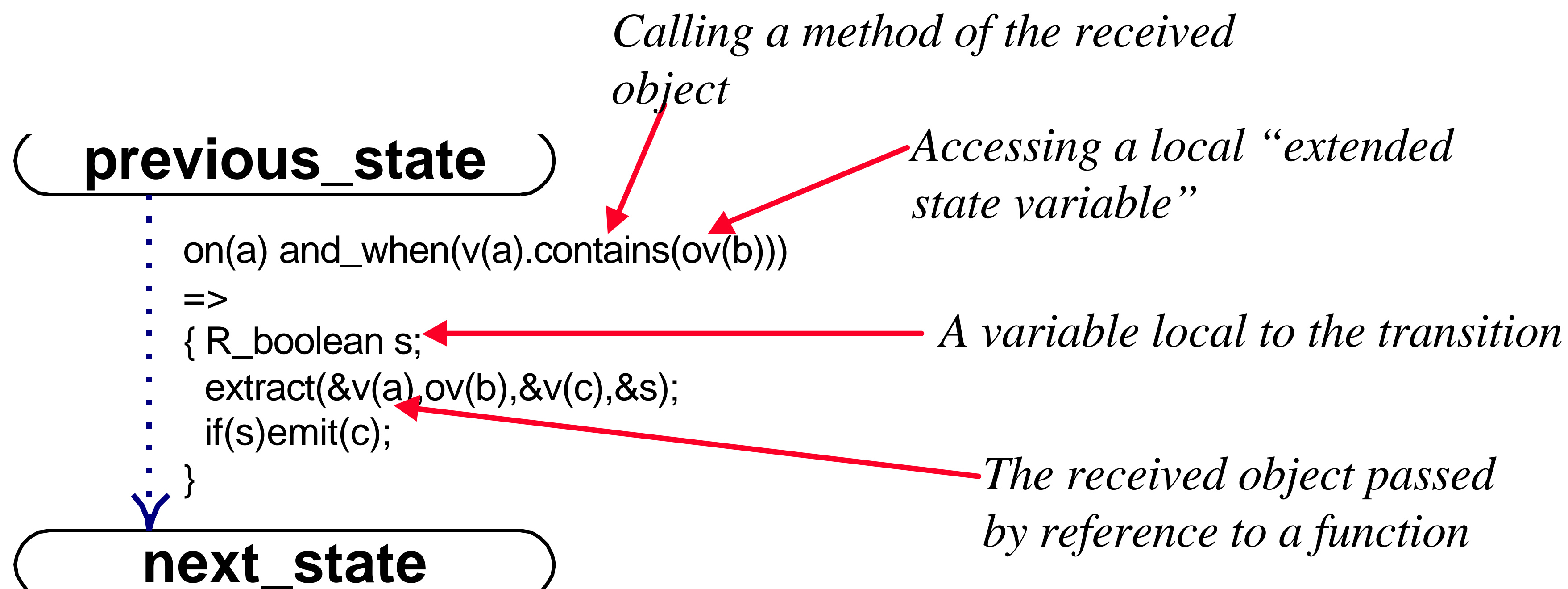
Shared Data Stores

- A data store is a passive component, which can be used among active components to build a higher level active component.
- A data store may contain an object or a traditional data item.
- The data in a store retains its value until explicitly changed.
- The state machines access a store via their ports.



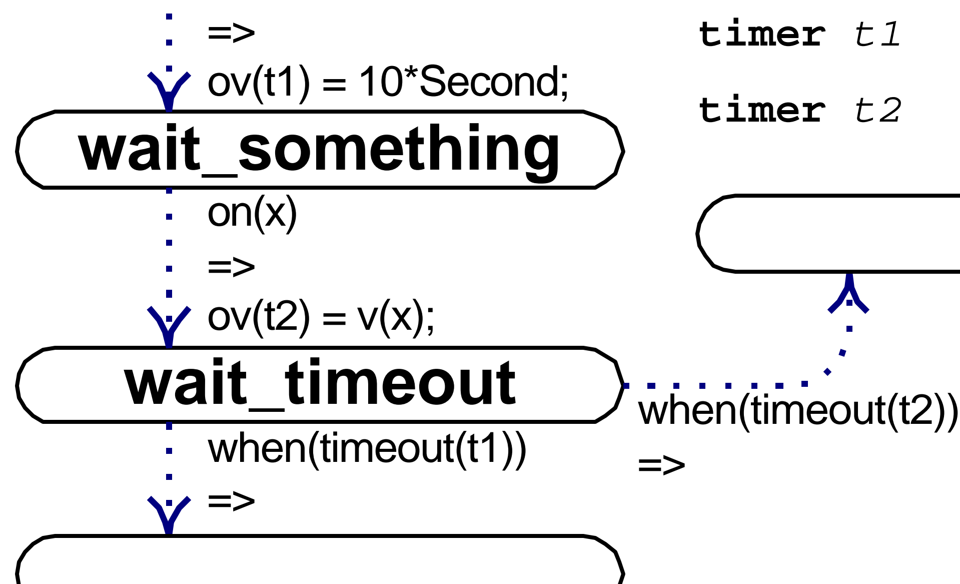
Data Processing Operations

- Trivial data processing can be written directly to actions of transitions using C or C++.
- Nontrivial data processing happens in functions or methods called from transitions. (Maybe generated using an OO tool.)



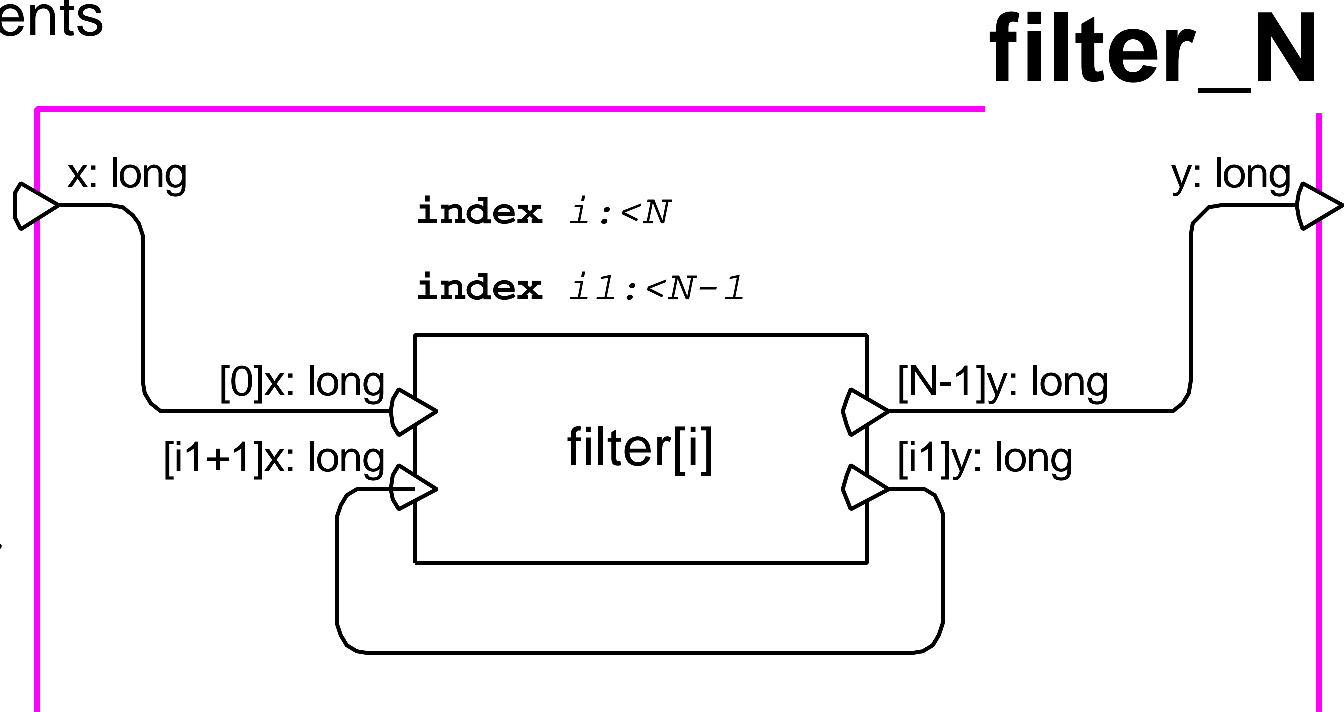
Timing

- You may define as many timers as needed to keep track of time and to give an event after predetermined time.
- A timer counts backwards until zero.
- Timeout stays true until the timer is set again.



If Many Similar Things are Needed

- ReaGeniX language includes powerful indexing mechanisms
- Using indexing, you may set up arrays of
  - active components
  - ports
  - timers



An N-stage filter component

Interfacing to External World

- For an active component the ReaGeniX generates:
  - an **event handler** function
  - a **data structure** definition.
- The event handler accesses the variables (objects) containing the input and output values via port pointers.
  - no unnecessary copying of data
- Input and output events are passed in Boolean variables.
- The system may contain as many event handlers in tasks as needed (without excess code).

Interfacing to a Traditional RT/OS

- The task main is a simple message loop.

```
while(running){
  wait_os_message
  switch(message_tag){
    case XYZ:{
      arrived(xyz)=R_true;
    } break;
    …
  }
  calculate_elapsed_time
  call_event_handler;
  if(arrived(abc)){
    send_os_message(&v(abc));
  }
  …
  request_timer_message_if_needed
}
```

Using Without any OS or RT-Kernel

- Main program is a simple loop:
  - check for inputs (from H/W ports or buffers loaded by interrupts)
  - calculate elapsed time
  - call event handler
  - pass outputs to H/W drivers
- If the system is interrupt driven and power must be conserved, at the end of the loop:
  - if the event handler requests a timer event, setup for timer interrupt
  - wait interrupts in halt state

Setting Priorities Using ReaGOS

- ReaGOS is an application specific RT kernel generated from a task architecture diagram.
- The event handler generated by the ReaGeniX interface directly to ReaGOS without manual coding.
- ReaGOS is quite portable. Macros to enable and disable interrupts may have to be redefined for a new hardware.
- ReaGOS assists to interface the interrupt drivers.