Metropolis Framework

Design of Function Processes
Design of Communication Media
Design of Architecture Components

Metropolis Infrastructure
- Model of computation
- Design methodology
  - Abstraction levels
  - Refinement
- Base tools
  - Design imports
  - User interface
  - Simulation

Metropolis Point Tools:
Synthesis/Refinement

Metropolis Point Tools:
Analysis/Verification
Metropolis: Model of Computation

- System: a network of concurrent processes
  - process: sequential function + ports

- Do not commit to particular communication semantics
  - processes communicate through communication media
  - communication media: define communication semantics
e.g. queues, shared memory, …, generic, …

- Do not commit to particular firing rules of processes
  - a special construct to define firing rules and atomicity
Communication medium:
- **state**: snapshot of the medium
- **interfaces implementation**: the means to access the medium
- **properties**: # of writers, transaction, arbitration, ...

- The state changes only by the execution of interface functions.
- An interface may be implemented by more than one media.
- Interface functions at different abstraction levels to support refinement.

- Library of pre-defined media
Computation

- **Ports:**
  - Interaction with communication is always through ports.
  - Each port is specified with an interface it has access to.
    
    All and only the functions of the interface can be used through the port.

- **Sequential program:**
  - Firing rules and atomic execution
    ```
    await(cond){ st1; st2; ... stk;}
    “if cond is TRUE, then atomically execute {st1; ... stk;}.”
    ```
  - Non determinism
  - Bounded loops
  - Parameters
process Filter {
    filter_interface port1, port2, port3;

    coeff = 1;
    while(TRUE){
        dataready = FALSE;
        await(port1.dataready() || port3.cfready()) {
            if(port3.cfready()) coeff = port3.cfread();
            if(port1.dataready()) dataready = TRUE;
        }

        if(dataready)
            bounded_loop(i, 0, LINES_IN_FRAME, 1) {
                port1.dataread(line, PIX_IN_LINE);
                filtering(line, coeff);
                port2.datawrite(line, PIX_IN_LINE);
            }
    }
}
Interface with Communication

```c
medium bfifo breader bwriter{
    state int size, space, n, storage[];

    void write(data, N){
        await(N <= space)
        do_write(storage, data, N);
    }

    void read(data, N){
        ...
    }

    void n(){ // the number of elements
        return n;
    }
}

medium Fwrapper filter_interface{
    breader port1;
    bwriter port2;
    lsreader port3;

    bool dataready(){
        return (port1.n() >= PIX_IN_LINE);
    }

    bool cfready(){
        return (port3.n() > 0);
    }

    void dataread(data, N){
        port1.read(data, N);
    }
    ...
}
```
medium df_fifo df_reader df_writer{
  state int size, space, n, storage[];
  state bool req_wr, req_rd, wake_wr, wake_rd;

  void write(data, N){
    req_wr = TRUE;  num = N;
    while(num > 0)
      await(num <= space || wake_wr ||
        req_rd && space > 0){
        do_write(data, min(num,space));
        num = num - min(num,space);
        if(req_rd)  wake_rd = TRUE;
        wake_wr = FALSE;
      }
    req_wr = FALSE;
  }

  bool req_wr(){ return req_wr;}
}

medium Fwrapper filter_interface{
  df_reader port1;
  bwriter port2;
  lsreader port3;

  bool dataready(){
    return (port1.n() > 0 || port1.req_wr());
  }

  bool cfready(){
    return (port3.n() > 0);
  }

  void dataread(data, N){
    port1.read(data, N);
  }
  ....
Mapping to Architecture

- All software mapping
- The states of the media to the system memory.

• RTOS_R&W
• Memory

Producer

RTOS_W

Filter

RTOS_R

Consumer

RTOS_W Scheduler

State_Intf.
Mapping to Architecture

- RTOS_W: implements df_writer using RTOS API, e.g. memcpy().
- RTOS_R&W: implements df_reader, bwriter, lsreader using RTOS API.
- RTOS_R: implements breader using RTOS API.
- Memory: implements the memory interface (used by RTOS API functions).
- RTOS_Scheduler and State_Interfaces:

```c
process Filter {
    coeff = 1;
    while(TRUE)
        dataready = FALSE;
        await(port1.dataready() || port3.cfready()){
            if(port3.cfready()) ...
        }
        if(dataready) ...
}
```

State_Interface:
- keep state info of Filter
- report the info to Scheduler
- tell Scheduler’s command to Filter

RTOS_Scheduler:
- access state info of processes
- make a schedule decision
- tell Scheduler’s command to State_Interfaces