A Model-driven development and verification framework for embedded software (2)

Yunja Choi

Software Safety Engineering Laboratory
School of Electrical Engineering and Computer Science
Kyungpook National University

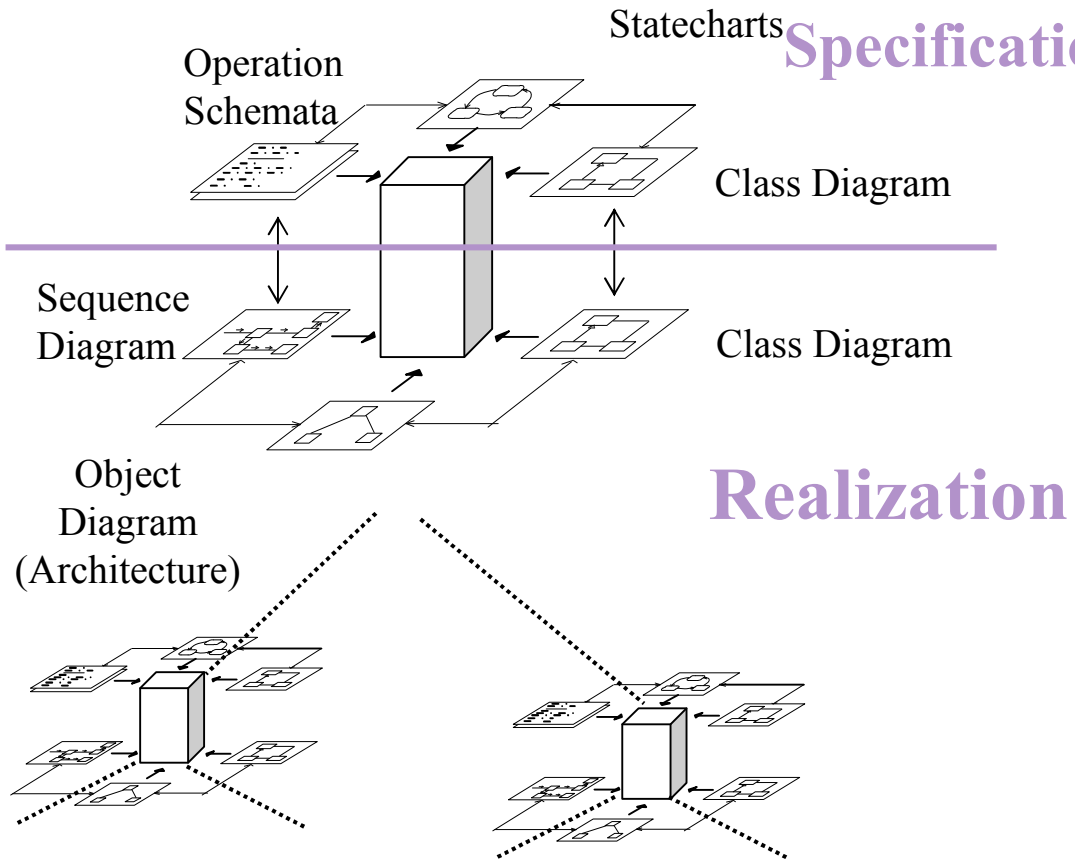
Our approach

- Integration of verification techniques into existing development process
  - We chose one of the component-based, model-driven development (MDD) methodologies named MARMOT (CBSE08)
- Provide a framework for the V&V-integrated development methodology including
  - Modeling language
  - Design simulation
  - Design verification
  - Code generation
- Provide automation to support the V&V-integrated development framework
  - UML subset + action language for the modeling language
  - Extension of existing UML support tools
  - Integration of model checking techniques

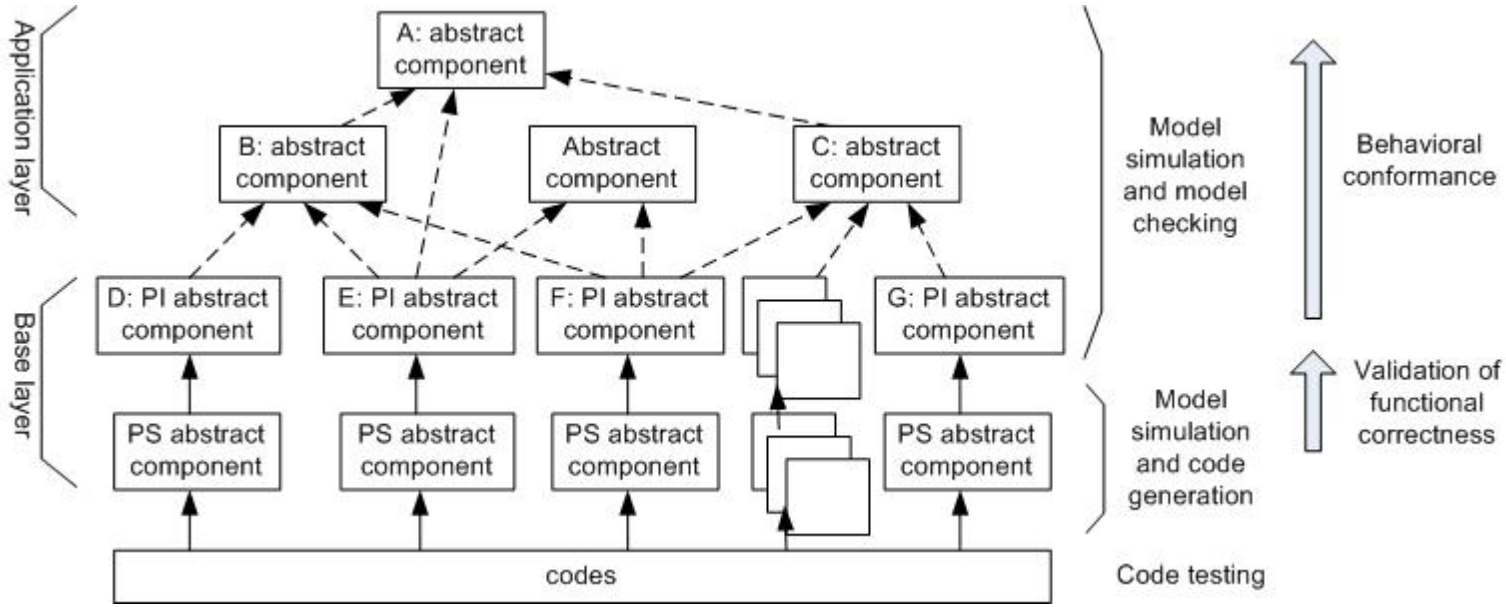
Work in progress

- Constructing behavior model of abstract component
- Verification of abstract component using communication patterns

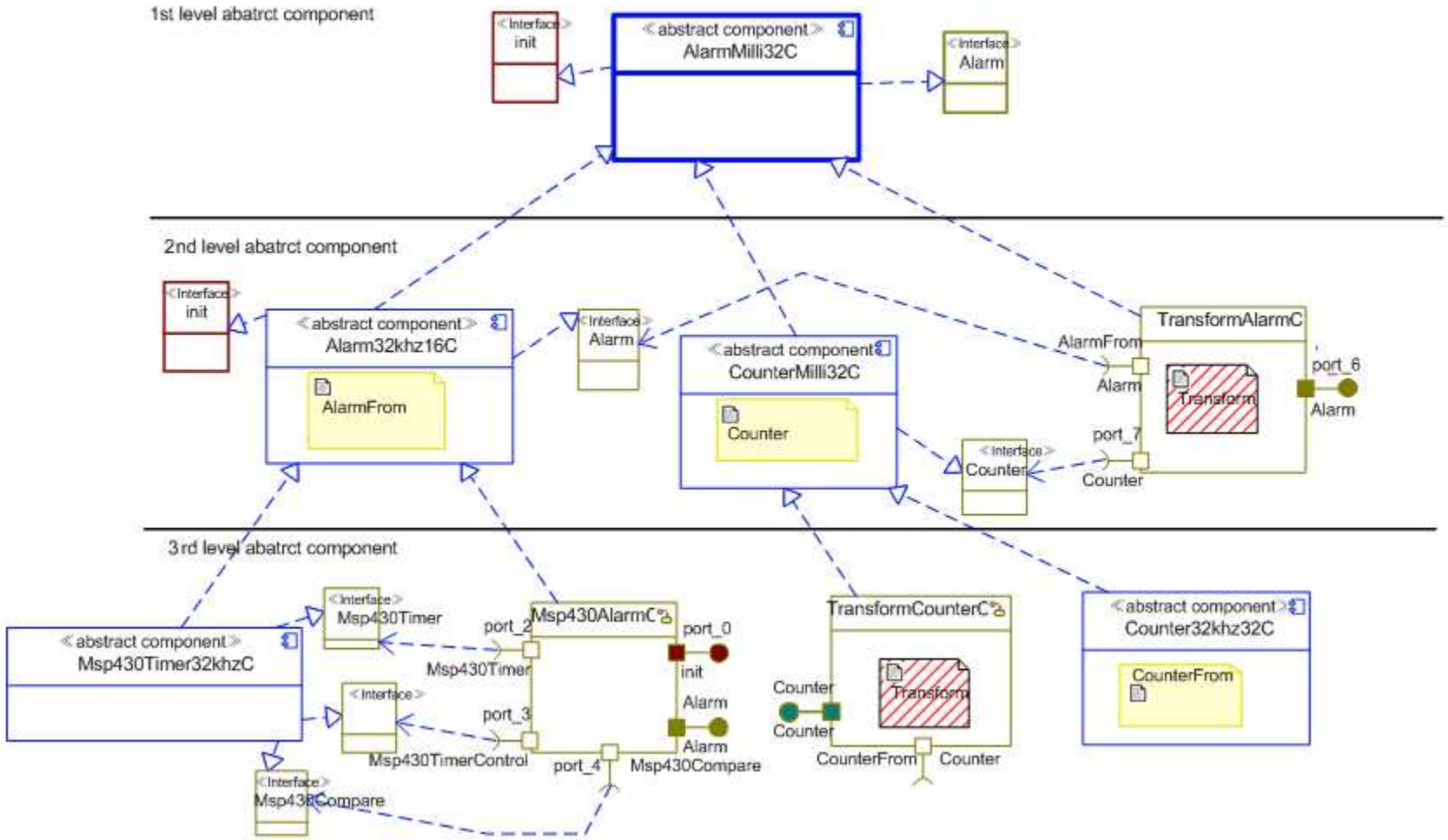
Abstract component



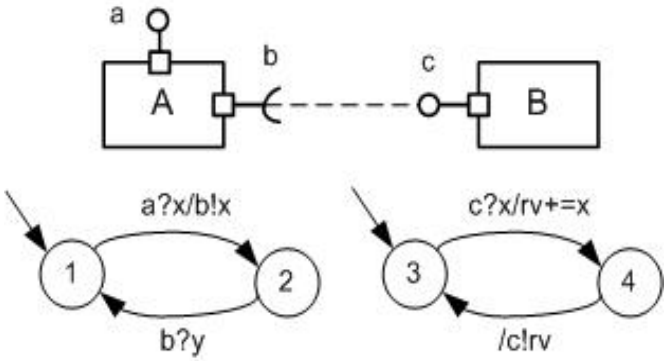
V&V of abstract component



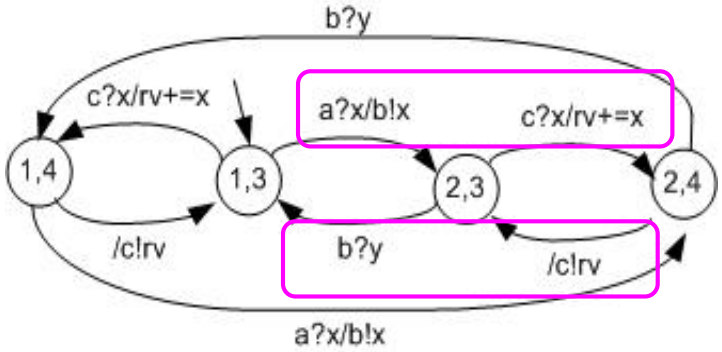
Example


1st level abatrct component
Interface
init
abstract component
AlarmMilli32C
Interface
Alarm
2nd level abatrct component
Interface
init
abstract component
Alarm32khz16C
AlarmFrom
Interface
Alarm
abstract component
CounterMilli32C
Counter
Interface
Counter
TransformAlarmC
AlarmFrom
Alarm
Transform
port_6
Alarm
port_7
Counter
3rd level abatrct component
abstract component
Msp430Timer32khzC
Interface
Msp430Timer
Msp430AlarmC
port_2
Msp430Timer
port_0
init
Alarm
Interface
port_3
Msp430TimerControl
port_4
Alarm
Msp430Compare
Interface
Msp430Compare
TransformCounterC
Counter
Counter
Transform
CounterFrom
Counter
abstract component
Counter32khz32C
CounterFrom

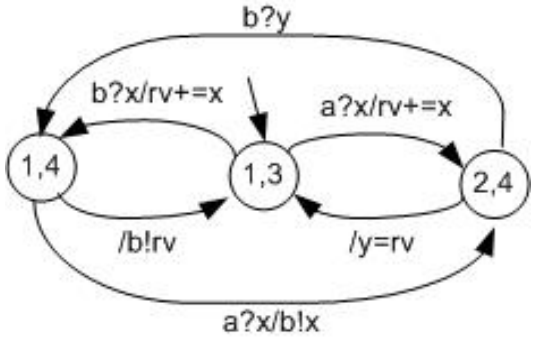
Construction of abstract behavior (1)



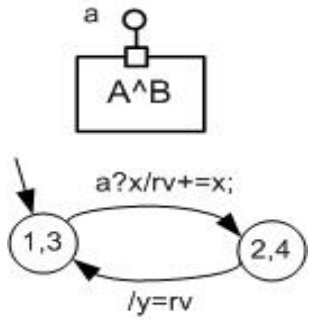
(a) two dependent abstract components



(b) free composition

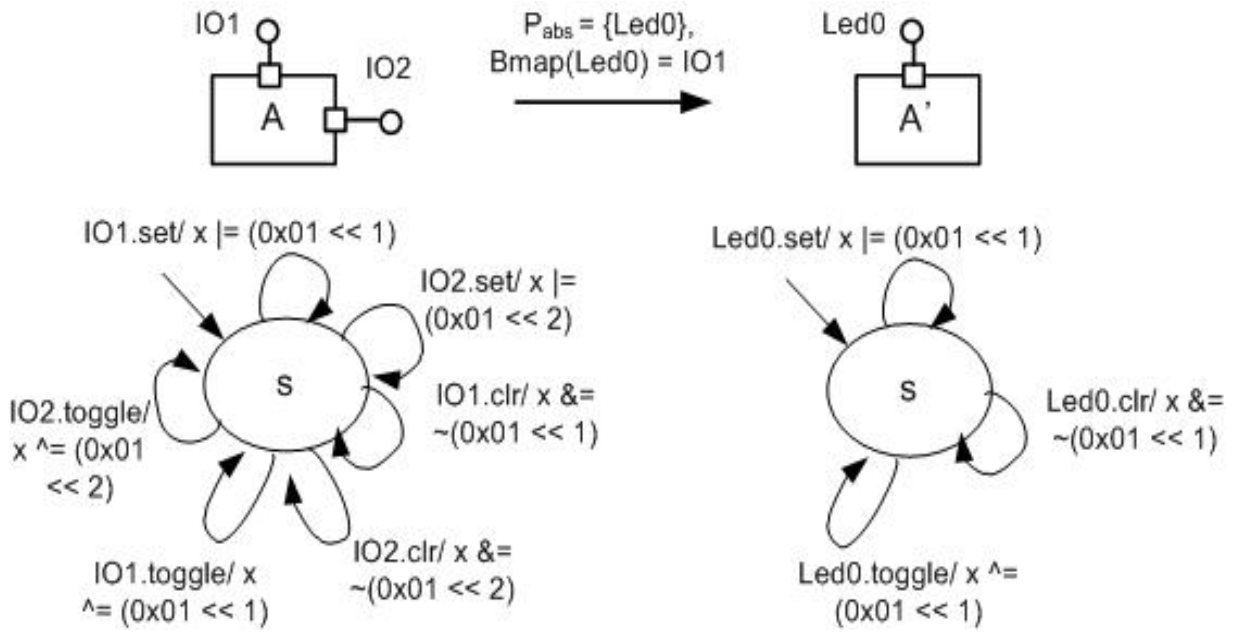


(c) synchronized reduction

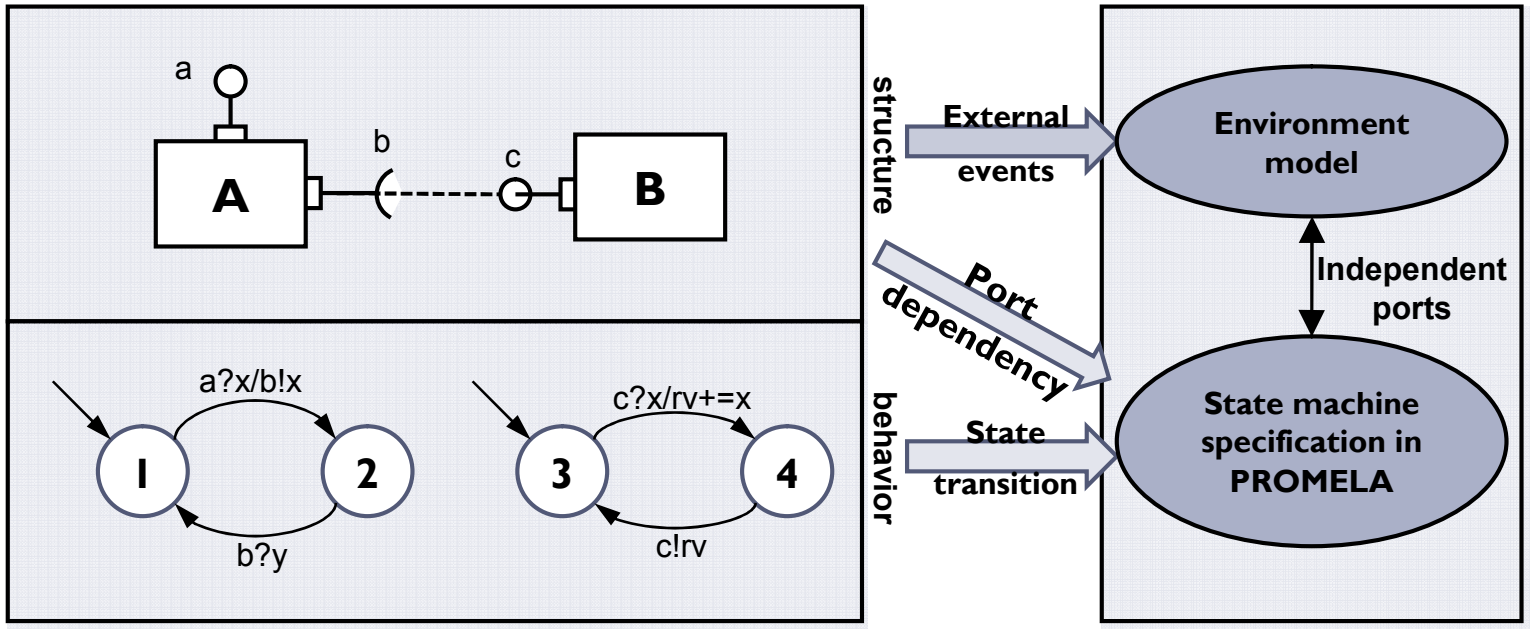


(d) abstraction

Construction of abstract behavior (2)



Checking communication consistency



Experiments

name	type	depth	states	transitions	Memory (M bytes)	Time (Seconds)
Msp430Timer32khzC	realization	1,494,716	2.4e+07	1.22e+08	5,034.0 (997.5)	1.69e+03 (3.46e+03)
	specification	392,826	8e+06	2.92e+07	843.5 (300.4)	224 (649)
Alarm32khz16C	realization	9,047	4.3e+07	2.02e+08	14,492.5 (1,488.9)	3.97e+03 (7.56e+03)
	specification	761	283,461	438,704	15.5 (6.7)	2.44 (7.8)

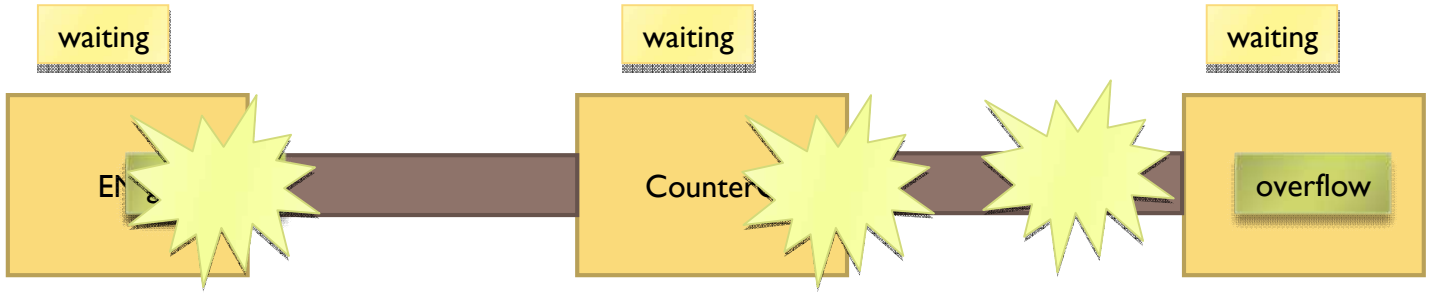
* On Sun Workstation T5240 1.2GHz 64G

No Priority and No Fired Channel

1. Skip Statement
2. One Channel

Timer Interface

Interface를 Use하는 Component에서
Interface 를 Provide하는 Component로
overflow 메시지를 전달함



Issues

- One thread model vs. multiple thread model
- How to identify and specify a thread model from a set of abstract components to be composed

Communication patterns

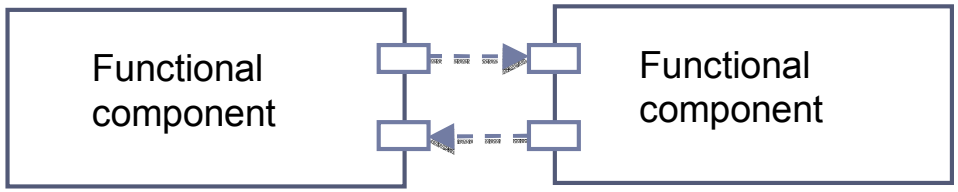
- Motivation
  - At the early stage, modeling is focused on functional behavior
  - Details on communication mechanism are determined much later
  - However, we need to somehow specify communication behavior for verification purposes since the verification result may depend on specific communication mechanism
- We define six representative communication patterns that can be exercised at the early modeling stage.

Communication structure

- Three types of communication structures are considered
  - Default
  - Refined
  - With separate handlers

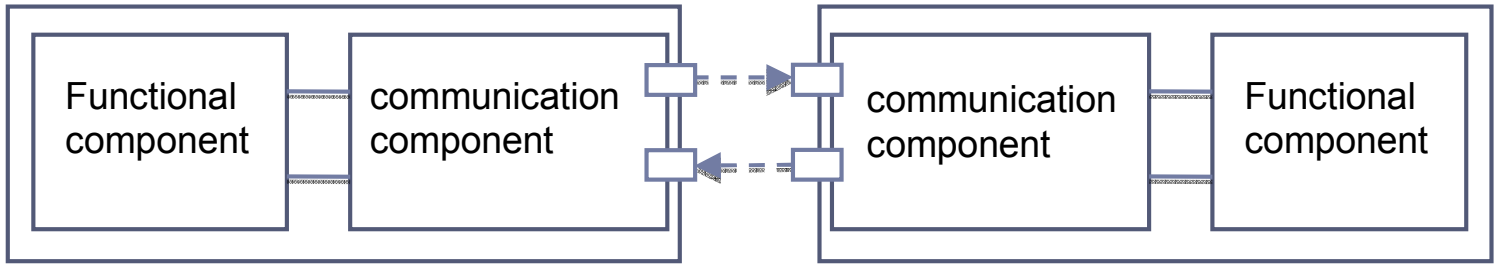
Communication structure -default

$$(P, P_{in}, P_{out}, P_{msg})$$



$$Connect(P,Q) \Leftrightarrow \exists p_{out} \in P_{out}, \exists q_{in} \in Q_{in} \bullet$$
$$type(p_{out}) = type(q_{in}) \,\&\, \forall m \in P_{msg}, (p_{out}!m \Rightarrow q_{in}\,?x \wedge x = m)$$

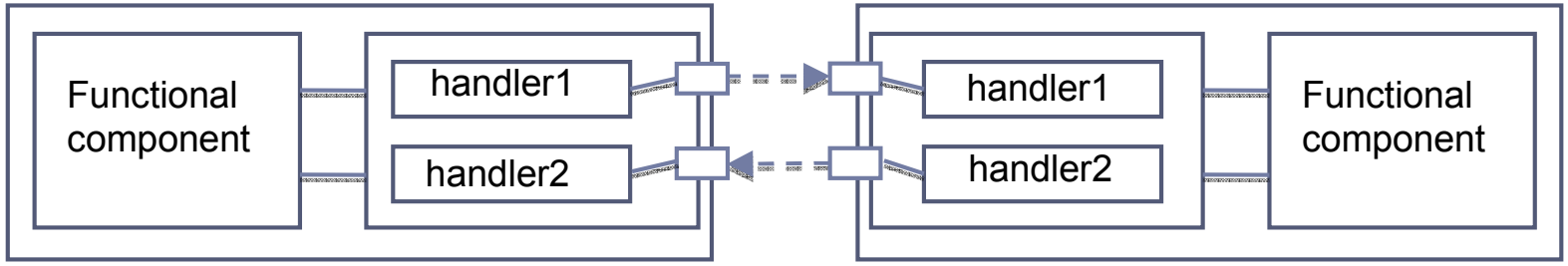
Communication structure - refined



$$Comp\_spec\,(i,o,op\_set,action\_set) = \\ new\ \ u,v(I(i,o,u,v) \quad | \quad Spec(u,v,op\_set,action\_set))$$

$$I\,(i,o,u,v) = i?x.u!x.I(i,o,u,v) + v?y.o!y.I(i,o,u,v)$$

Communication structure – separate handlers



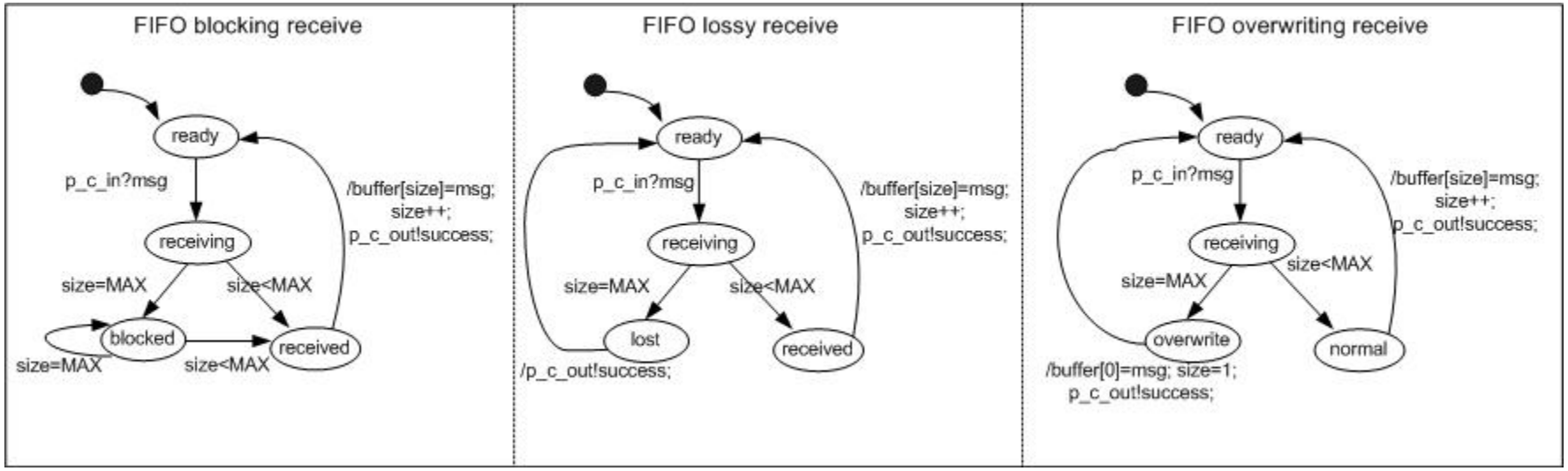
$$Comp\_spec(i,o,op\_set,action\_set) =$$
$$new\ u,v(I(i,o,u,v) \mid Spec(u,v,op\_set,action\_set))$$

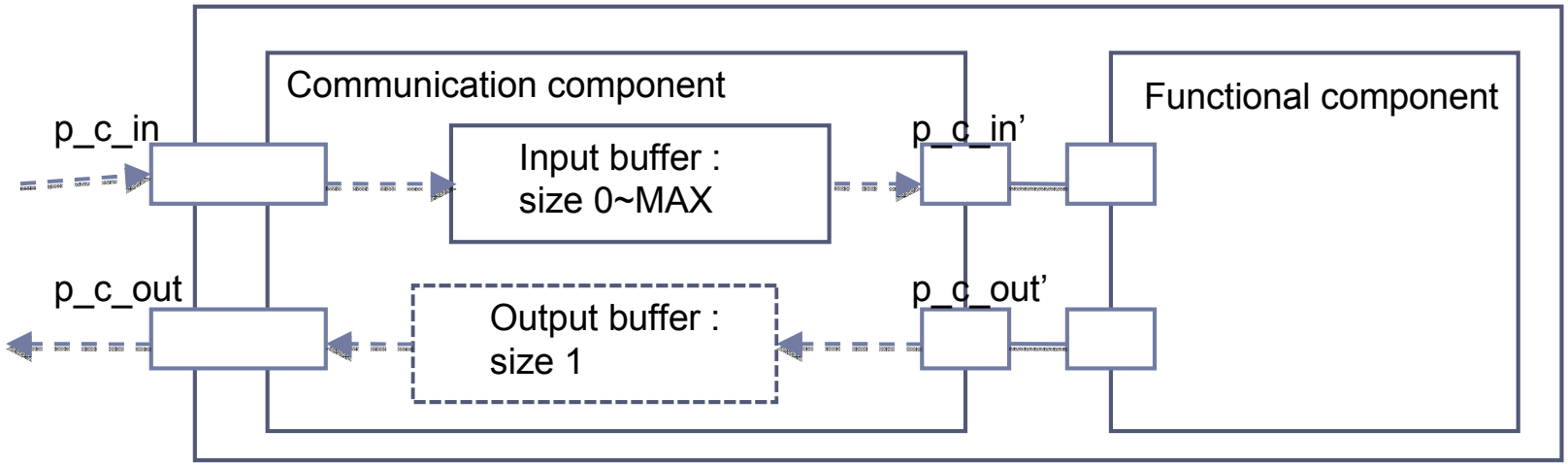
$$I(i,o,u,v) = I_{in}(i,u) \mid I_{out}(o,v)$$
$$I_{in}(i,u) = i?x.u!x.I_{in}(i,u)$$
$$I_{out}(v,o) = v?x.o!x.I_{out}(v,o)$$

Behavioral Patterns

- Three considerations
  - Message passing mechanism
    - FIFO, Value-based
  - Buffering methods
    - Blocking, Lossy, Overwriting
  - Synchrony
    - Synchronous vs. asynchronous message passing
- This work supports FIFO message passing mechanism
  - Outgoing messages : {FIFO} × {Blocking} ×{Synchronous, Asynchronous}
  - Incoming messages : {FIFO} × {Blocking, Overwriting, Lossy} ×{Synchronous, Asynchronous}

Example: synchronous FIFO patterns





Verification using communication patterns

$$run\ \ Spec(in, out);$$
$$run\ \ env(out, in);$$

- With default communication
  - A process deadlock is highly-likely due to the under-specified communication behavior
- Three steps for using communication patterns
  - Introduce the communication process
  - Apply the communication patterns and perform verification
  - Re-apply the communication patterns after separating the communication process with two independent input/output handlers

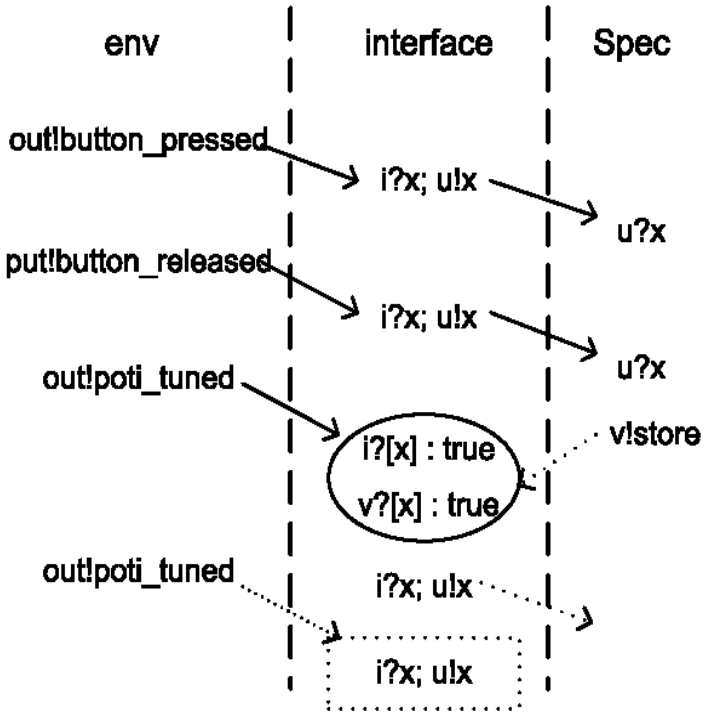
Verification using refined structure – in PROMELA

Communication process

```
proctype Interface(chan i,o,u,v){
    mtype x;
    do
    :: i?[x] -> i?x; u!x;
    :: v?[x] -> v?x; printf("%d", x);
    od;
}
```

Communication process

```
run env(in, out);
run Interface(out, in, u, v);
run Spec(v,u);
```



Verification using separate handlers

Communication handlers

```
proctype Interface(chan i,o,u,v){
      run input_handler(i,u);
      run output_handler(o,v);
}

proctype  input_handler(chan i, u){
    mtype x;
    do
    ::i?[x] -> i?x; u!x;
    od;
}

proctype output_handler(chan o,v){
    mtype x;
    do
    :: v?[x] -> v?x; printf("%d", x);
    od;
}
```

$$I\ (i,o,u,v) = I_{in}(i,u) \mid I_{out}(o,v)$$

$$I_{in}(i,u) = i?x.u!x.I_{in}(i,u)$$

$$I_{out}(v,o) = v?x.o!x.I_{out}(v,o)$$

Communication process

```
run env(in, out);
run Interface(out, in, u, v);
run Spec(v,u);
```

Discussion

- Communication patterns help designers model system without regard for the specifics of communication mechanism
- Patterns can be adopted as desired
- The default communication pattern is not an abstraction of the refined communication
  - Refine communication patterns??

Participants

- Yunja Choi
- Hoon Jang
- Seungmin Choo