Towards Seamless Integration of N-Version Programming in Model-Based Design

Tingting Hu [*], *Ivan Cibrario Bertolotti* [**], *Nicolas Navet* [*]

[*] National Research Council of Italy – IEIIT, Torino, Italy
[**] University of Luxembourg – FSTC, Esch-sur-Alzette, Luxembourg

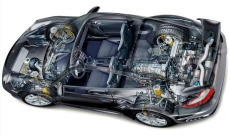
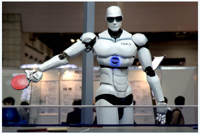
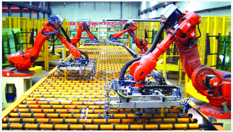
Sept. 12 – 15, 2017, Limassol, Cyprus

Table of Content

Cyber Physical Systems

"... A cyber-physical system (CPS) integrates computing and communication capabilities with monitoring and/or control of entities in the physical world dependably, safely, securely, efficiently and in real-time ..."

– *S. Shankar Sastry*

Cyber Physical Systems

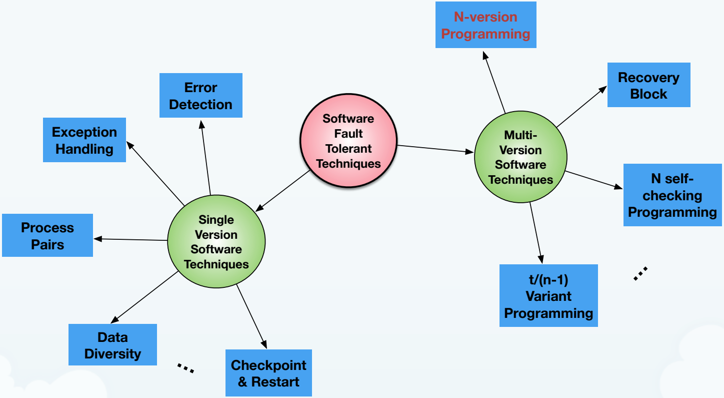
"... A cyber-physical system (CPS) integrates computing and communication capabilities with monitoring and/or control of entities in the physical world dependably, safely, securely, efficiently and in real-time ..."

– *S. Shankar Sastry*

- System dependability is impaired by faults, including hw faults, sw faults
- As CPS becomes more and more software centric, software faults become the dominant factor
- Suitable fault prevention, fault removal, or fault tolerance techniques should be employed.

Software Fault Tolerance

Techniques that enable a system to tolerant software faults remaining in the system after its development.



State of the art

- Theoretical foundation of software fault tolerance is well established in the 80s
- Generally, fault tolerance is introduced at a later phase of system development, e.g. implementation phase
- The selection of fault tolerance techniques is mainly driven by experience
- Few work is done on including fault tolerant analysis at the system design phase

Model based design is an enabling technique to this direction

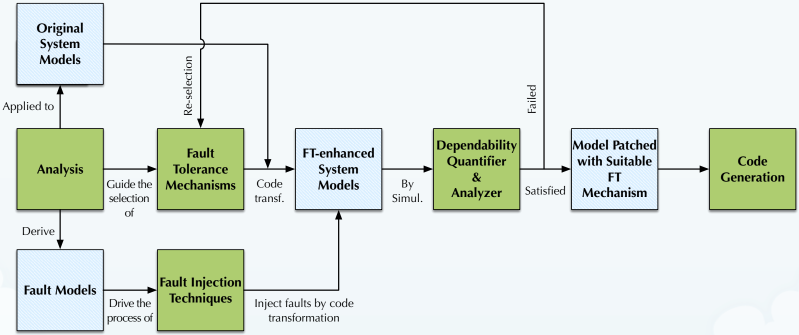
Model Based Design

Models play an important role in various engineering disciplines. They are used to guide the development process.

- Advantages
  - Support rapid prototyping
  - Early verification of system correctness
  - Explore different design and implementation choice
  - Early detection of design errors
- Widely adopted in automotive, aerospace, etc
- Popular tools include UML, Matlab/Simulink, SCADE, AADL

Automated FT/FI framework

Enhance the dependability of CPS by introducing fault tolerance features, without changing its functional behavior and still honoring the non-funtional requirements, e.g. timings.



Targeting safety-critical real-time systems

N Version Programming

NVP

N-fold replication of the same computation, carried out by means of N software modules, called member versions.

N Version Programming

NVP

N-fold replication of the same computation, carried out by means of N software modules, called member versions.

- Member versions run in parallel, operating on the same inputs;
- Result reached by consensus, e.g majority voting
- Requires member versions to generate comparison vectors at predefined cross-check points
- Feedback to the member versions depending on the result: termination/continuation, recovery actions

N Version Programming

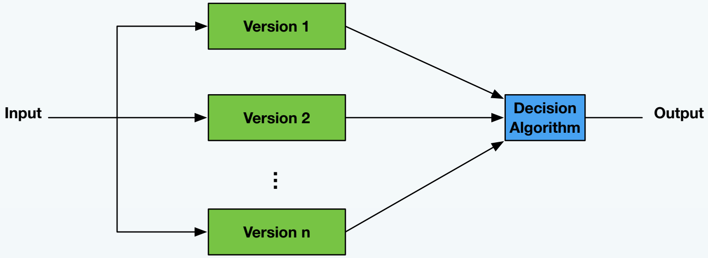
NVP

N-fold replication of the same computation, carried out by means of N software modules, called member versions.

- Member versions run in parallel, operating on the same inputs;
- Result reached by consensus, e.g majority voting
- Requires member versions to generate comparison vectors at predefined cross-check points
- Feedback to the member versions depending on the result: termination/continuation, recovery actions

- Error protected: software design faults
- Basic principle: increase software diversity

N Version Programming



CPAL

The Cyber Physical Action Language (CPAL)

A language offers high-level abstractions that are suitable for the modeling, simulation, verification and progamming of CPSs.

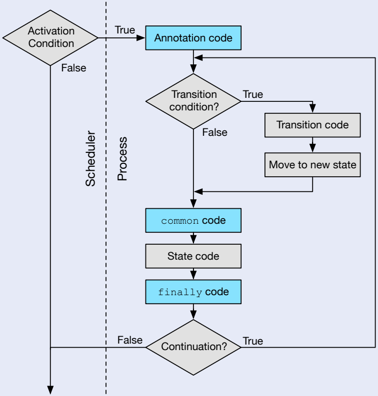
- Finite State Machine (FSM) based
- Expressiveness: functional and non-functional behavior, e.g. timings
- Modeling language & development language
- Timing equivalence between simulation time and run-time
- Supported platforms:
  - Windows 32/64bit, Linux 64bit, Mac OS X
  - Raspberry Pi, Freescale FRDM-K64F, Embedded Linux 64bit, Embedded Windows 32/64bit

CPAL

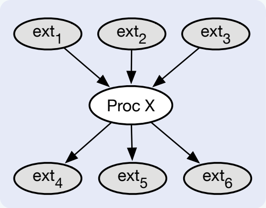
CPAL sample

```
processdef P(params) {
  common {
    code
  }

  state Warning {
    code
  }
  on (cond) {code} to Alarm_Mode;
  after (time) if (cond) to Normal_Mode;

  finally {
    code
  }
}

process P: inst[period,offset][cond](args);

@cpal:time:inst{
    annotation code
}
```

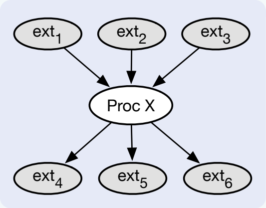
Elementary execution step



The NVP modeling framework

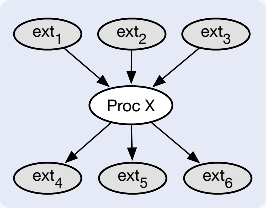


The NVP modeling framework

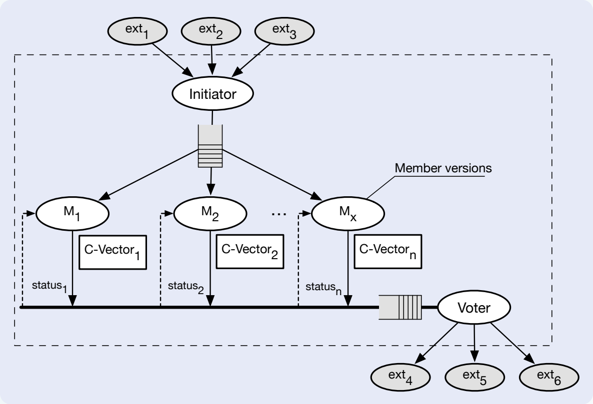


$\Rightarrow$

The NVP modeling framework



⇒



Design goal

Software patterns

"... Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem in such a way that you can use this solution a million times over, without ever doing it the same way twice ..."

– Christopher Alexander

Design goal

Software patterns

Patterns capture important practice in a form that makes the practice accessible

Design goal

Software patterns

Patterns capture important practice in a form that makes the practice accessible

- Re-usability
- Seamless integration with existing system model
  - Maintain the same interface with surroundings
  - Preserve the functional behavior and non-functional properties
- Code-generation friendly

CPAL model: the initiator

Driving force of the NVP process

- Collect and populate data to member versions for processing by means of the communication channel
- Preserve the interface with other modules and the timings of the original process

```
processdef Initiator(in uint32: a, in uint32: b,
                     out queue<Replica_In>: inputs
                     in queue<uint32>: active_members)
{
  var Replica_In: tmp;

  state Main{
    encapsulate_inputs(a, b, tmp);

    loop over active_members with it {
      inputs.push(tmp);
    }
  }
}
```

CPAL model: member versions

Operate on the input data, done within the user-provided FSM

```
processdef Member_Version(in queue<Replica_In>: x,
                          out queue<Comp_Vector>: z,
                          out queue<uint32>: status,
                          in uint32: id)
{
  common{
    tmp_in = x.pop();
    unfold_inputs(tmp_in, a, b);

    /* Other common code goes here. */
  }

  /* User-written FSM goes here without modification. */
  state Main{
    sum = a + b;
  }

  finally{
    encapsulate_outputs(sum, tmp_out);
    my_result.mem_res = tmp_out;
    my_result.mem_id = id;

    z.push(my_result);
    status.push(id);
  }
}
```

CPAL model: member versions

Export a comparison vector and indicate its execution status to the voter, by means of communication channel

```
processdef Member_Version(in queue<Replica_In>: x,
                          out queue<Comp_Vector>: z,
                          out queue<uint32>: status,
                          in uint32: id)
{
  common{
    tmp_in = x.pop();
    unfold_inputs(tmp_in, a, b);

    /* Other common code goes here. */
  }

  /* User-written FSM goes here without modification. */
  state Main{
    sum = a + b;
  }

  finally{
    encapsulate_outputs(sum, tmp_out);
    my_result.mem_res = tmp_out;
    my_result.mem_id = id;

    z.push(my_result);
    status.push(id);
  }
}
```

CPAL model: member versions

Cross-check point is implicitly set to the end of each execution step

```
processdef Member_Version(in queue<Replica_In>: x,
                          out queue<Comp_Vector>: z,
                          out queue<uint32>: status,
                          in uint32: id)
{
  common{
    tmp_in = x.pop();
    unfold_inputs(tmp_in, a, b);

    /* Other common code goes here. */
  }

  /* User-written FSM goes here without modification. */
  state Main{
    sum = a + b;
  }

  finally{
    encapsulate_outputs(sum, tmp_out);
    my_result.mem_res = tmp_out;
    my_result.mem_id = id;

    z.push(my_result);
    status.push(id);
  }
}
```

CPAL model: the voter

Perform majority voting based on the comparison vectors from the member versions

```
processdef Voter(in queue<Comp_Vector>: v,
                 in queue<uint32>: status_queue,
                 out uint32: sum,
                 out queue<uint32>: alive_members)
{
  state Main{
    Majority_Voting(v, majority, summary);

    unfold_outputs(majority, sum);

    alive_members.clear();
    loop over v with it{
      if(comp_ballot(it.current.mem_res, majority.value)){
        alive_members.push(it.current.mem_id);
      }
    }

    v.clear();
    status_queue.clear();
  }
}
```

CPAL model: the voter

Export output data to other modules of the modeled system

```
processdef Voter(in queue<Comp_Vector>: v,
                 in queue<uint32>: status_queue,
                 out uint32: sum,
                 out queue<uint32>: alive_members)
{
  state Main{
    Majority_Voting(v, majority, summary);

    unfold_outputs(majority, sum);

    alive_members.clear();
    loop over v with it{
      if(comp_ballot(it.current.mem_res, majority.value)){
        alive_members.push(it.current.mem_id);
      }
    }

    v.clear();
    status_queue.clear();
  }
}
```

CPAL model: the voter

Determine whether a member version should be terminated

```
processdef Voter(in queue<Comp_Vector>: v,
                 in queue<uint32>: status_queue,
                 out uint32: sum,
                 out queue<uint32>: alive_members)
{
  state Main{
    Majority_Voting(v, majority, summary);

    unfold_outputs(majority, sum);

    alive_members.clear();
    loop over v with it{
      if(comp_ballot(it.current.mem_res, majority.value)){
        alive_members.push(it.current.mem_id);
      }
    }

    v.clear();
    status_queue.clear();
  }
}
```

Process coordination

The initiator preserves the interface to the surrounding systems and timings

```
/* process Original_Process: origin_proc[100ms]
                            (input1, input2, output1); */

process Initiator: initiator[100ms]
              (input1, input2, input_queue, active_members);

process Member_Version: m1[]
              [member_alive(active_members, id)
               and input_queue.not_empty()
               and (not exec_complete(status_queue, id))]
              (input_queue, id, comp_vectors, status_queue);

process Voter: voter1[]
              [comp_vectors.not_empty() and
               comp_vectors.count() == status_queue.count()]
              (comp_vectors, status_queue,
               output1, active_members);
```

Process coordination

The initiator preserves the interface to the surrounding systems and timings

```
/* process Original_Process: origin_proc[100ms]
                             (input1, input2, output1); */

process Initiator: initiator[100ms]
              (input1, input2, input_queue, active_members);

process Member_Version: m1[]
              [member_alive(active_members, id)
               and input_queue.not_empty()
               and (not exec_complete(status_queue, id))]
              (input_queue, id, comp_vectors, status_queue);

process Voter: voter1[]
              [comp_vectors.not_empty() and
               comp_vectors.count() == status_queue.count()]
              (comp_vectors, status_queue,
               output1, active_members);
```

Process coordination

The initiator populates inputs to member versions

```
/* process Original_Process: origin_proc[100ms]
                                 (input1, input2, output1); */

process Initiator: initiator[100ms]
                   (input1, input2, input_queue, active_members);

process Member_Version: m1[]
                   [member_alive(active_members, id)
                    and input_queue.not_empty()
                    and (not exec_complete(status_queue, id))]
                   (input_queue, id, comp_vectors, status_queue);

process Voter: voter1[]
                   [comp_vectors.not_empty() and
                    comp_vectors.count() == status_queue.count()]
                   (comp_vectors, status_queue,
                    output1, active_members);
```

Process coordination

The member versions carry out its own computation and report their comparison vectors

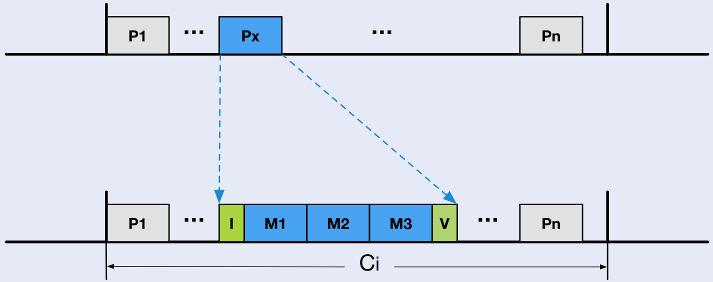
```
/* process Original_Process: origin_proc[100ms]
                            (input1, input2, output1); */

process Initiator: initiator[100ms]
              (input1, input2, input_queue, active_members);

process Member_Version: m1[]
              [member_alive(active_members, id)
               and input_queue.not_empty()
               and (not exec_complete(status_queue, id))]
              (input_queue, id, comp_vectors, status_queue);

process Voter: voter1[]
              [comp_vectors.not_empty() and
               comp_vectors.count() == status_queue.count()]
              (comp_vectors, status_queue,
               output1, active_members);
```

Process coordination

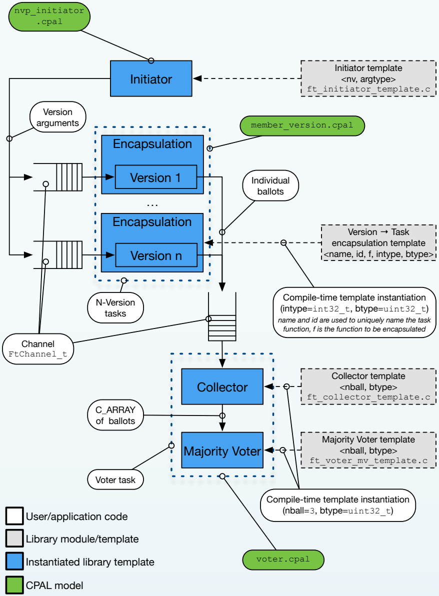
The voter determines the outputs of the NVP module and health of member versions

```
/* process Original_Process: origin_proc[100ms]
                                (input1, input2, output1); */

process Initiator: initiator[100ms]
                   (input1, input2, input_queue, active_members);

process Member_Version: m1[]
                   [member_alive(active_members, id)
                    and input_queue.not_empty()
                    and (not exec_complete(status_queue, id))]
                   (input_queue, id, comp_vectors, status_queue);

process Voter: voter1[]
                   [comp_vectors.not_empty() and
                    comp_vectors.count() == status_queue.count()]
                   (comp_vectors, status_queue,
                    output1, active_members);
```

Concurrency of member versions



NVP implementation in C



Conclusion

- Propose software patterns corresponding to NVP
- Can be seamlessly integrated with the existing system model, without changing its interface and timings.
- Derive a C language implementation of NVP from the CPAL model
- The same methodology can be profitably applied to the modeling of other fault tolerant mechanisms

- Future work:
  - Automatic code generation by means of code transformation
  - Automated fault injection for the validatation of the fault tolerant mechanism

Thank you for your attention