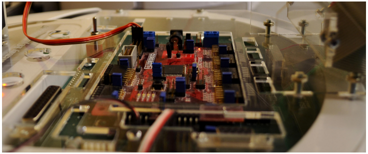
Transparent Fault Tolerance Support in Model-Based Design

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

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



2nd Italian Workshop on Embedded Systems (IWES)
September 7 – 8, 2017, Rome, Italy

Outline

- Overview and Motivation
- Fault Tolerance Framework
- Fault Injection Capabilities
- Conclusion

UNIVERSITÉ DU LUXEMBOURG

The CPAL Language

Cyber Physical Action Language

A high-level DSL to model, simulate, verify, and implement CPSs

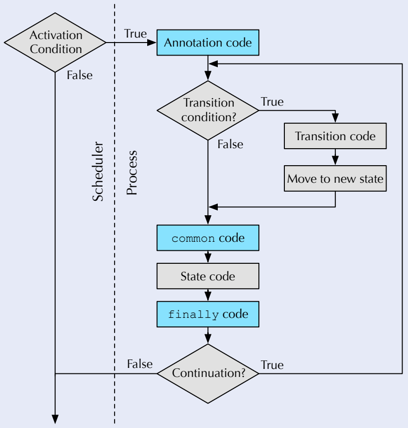
- It can express both functional and non-functional behaviors
- It can be executed in real time on an embedded platform, by means of an interpreter
- Simulation and execution are timing equivalent
- The language natively supports multiple periodic and/or event-driven processes, each modeled by means of a Mealy Finite State Machine (FSM)

The CPAL Language

Sample Process

```
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



Using CPAL

A "Real-World" Modeling Language

- C-like syntax
- Suitable as an implementation language
- Schedulability analysis and timing-accurate simulation
- Run-time introspection (e.g. for overload detection)

Goals

- Use CPAL to model a communication protocol for fault tolerance (interactive consistency)
- Compare it with another prominent language (Promela)

UNIVERSITÉ DU LUXEMBOURG

Using CPAL

A "Real-World" Modeling Language

- C-like syntax
- Suitable as an implementation language
- Schedulability analysis and timing-accurate simulation
- Run-time introspection (e.g. for overload detection)

Goals

- Use CPAL to model a communication protocol for fault tolerance (interactive consistency)
- Compare it with another prominent language (Promela)

UNIVERSITÉ DU LUXEMBOURG

CPAL vs. Promela — Main Remarks

- Promela is meant for verification, rather than execution
  - Non-determinism is at the core of the language
  - No I/O support
  - No floating point data types
  - It can be translated to C and Java (with varying success)

- CPAL supports timing-accurate simulation and interpreted execution
  - Non-determinism must be avoided in most real systems
  - No formal proofs (except for schedulability analysis)
  - Executable model
  - The execution platform is decoupled from the application

UNIVERSITÉ DU LUXEMBOURG

Fault-Tolerant CPSs

As CPSs become more and more software intensive, software defects tend to become the major source of faults

- Fault tolerance enables a system to tolerate software faults after its development
- Few work is done on automatic fault tolerance analysis and implementation at the system design phase

Goals

- Improve system dependability . . .
- . . . without affecting its functional behavior and timings
- Full integration with MBD workflow

Fault-Tolerant CPSs

As CPSs become more and more software intensive, software defects tend to become the major source of faults

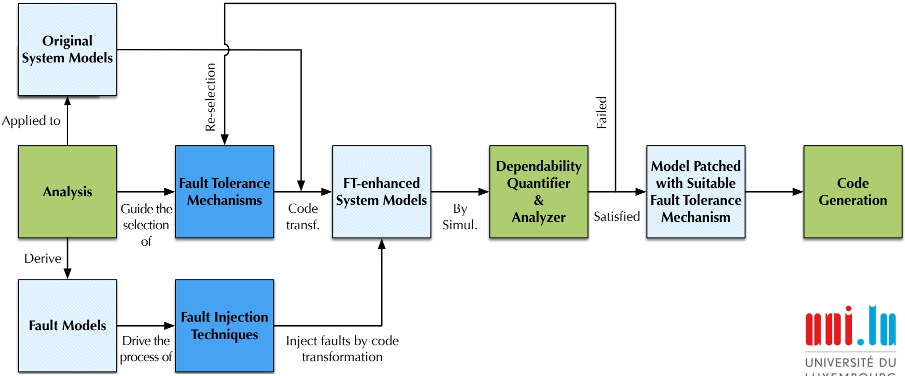
- Fault tolerance enables a system to tolerate software faults after its development
- Few work is done on automatic fault tolerance analysis and implementation at the system design phase

Goals

- Improve system dependability ...
- ... without affecting its functional behavior and timings
- Full integration with MBD workflow

UNIVERSITÉ DU LUXEMBOURG

Model-Based FT System Design/Development



N-Version Programming (NVP)

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

- 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 (terminate/continue, recovery)

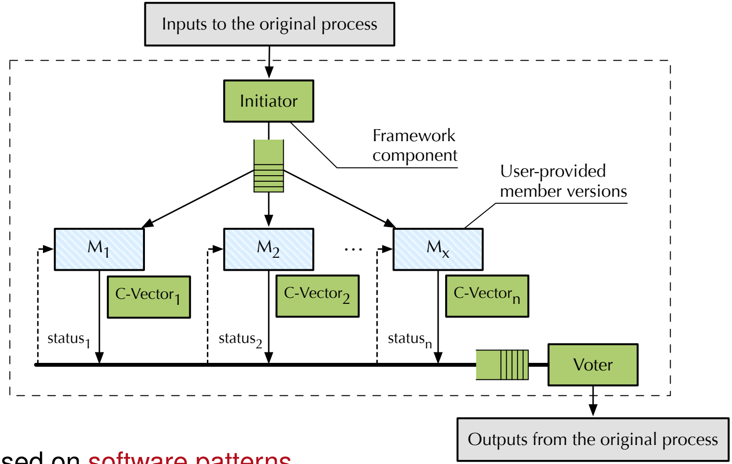
N-Version Programming (NVP)

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

- 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 (terminate/continue, recovery)

UNIVERSITÉ DU LUXEMBOURG

NVP Framework



Based on software patterns,
automatic code-generation friendly,
cross-check points set at execution step boundaries

Main Achievements

- Fault-tolerant mechanisms are kept independent from the logic of the application
- System designers may explore their use early in the design phase, focusing only on the application-dependent functional logic
- Minimal or no user involvement in low-level implementation details
- A C-language implementation derived from the model is also available (when direct model execution is impractical)

The same methodology can be applied to other fault tolerant mechanisms

UNIVERSITÉ DU LUXEMBOURG

Software Fault Injection

Motivation

No fault tolerance framework can be considered complete without the ability of injecting faults into the model

- Very powerful, well-understood assessment technique
- Time consuming, requires extensive know-how

Goals

- Automate software fault injection
- Integrate it with the design flow ...
- ... by means of software patterns

Software Fault Injection

Motivation

No fault tolerance framework can be considered complete without the ability of injecting faults into the model

- Very powerful, well-understood assessment technique
- Time consuming, requires extensive know-how

Goals

- Automate software fault injection
- Integrate it with the design flow . . .
- . . . by means of software patterns

What Can We Model? — Fault Categories

- Global State: State information resides in a pool of RAM statically allocated at link time $\rightarrow$ Its corruption can model various kinds of memory corruption
- Activation Arguments: Processes access state information through arguments, passed by value or by reference $\rightarrow$ Better granularity (down to the process activation level)
- Instance Variables: Local (stack-based) process storage is often implemented differently than global storage $\rightarrow$ Support the distinction between how different kinds of memory fail
- Control Flow Disruption: Most details of control flow are hidden in the model $\rightarrow$ Tampering with state transition conditions provides a useful surrogate

UNIVERSITÉ DU LUXEMBOURG

How? — Injection Mechanisms and Patterns

- External Injector: One or more processes are dedicated to fault injection → Keeps a clean boundary between the normal behavior of a system and its fault profile, centralized approach
- Common/Finally Blocks: They are executed before and after state-specific code upon process activation → They can also access activation arguments and local variables, per-instance behaviors are possible
- Annotation-Based Injector: CPAL supports annotations to express non-functional properties of a program and isolate them from functional properties → With respect to common/finally, they can also affect state transitions

Summary Table

Mechanism	Fault category: Global state	Activation arguments	Local variables	Control flow
External process(es)	✓			✓
Pre/post conditions	✓	✓	✓	
Annotation-based	✓			✓

Results

- Software fault injection of data errors can be effectively performed at the DSL level
- More limited modeling of code changes is possible, too
- All patterns can be fully automated

Summary Table

Mechanism	Fault category: Global state	Activation arguments	Local variables	Control flow
External process(es)	✓			✓
Pre/post conditions	✓	✓	✓	
Annotation-based	✓			✓

Results

- Software fault injection of data errors can be effectively performed at the DSL level
- More limited modeling of code changes is possible, too
- All patterns can be fully automated

UNIVERSITÉ DU LUXEMBOURG

Summary Table

Mechanism	Fault category: Global state	Activation arguments	Local variables	Control flow
External process(es)	✓			✓
Pre/post conditions	✓	✓	✓	
Annotation-based	✓			✓

Results

- Software fault injection of data errors can be effectively performed at the DSL level
- More limited modeling of code changes is possible, too
- All patterns can be fully automated

UNIVERSITÉ DU LUXEMBOURG

Ongoing Work

Automatic code generation and instrumentation

- Complete the implementation of the fault tolerance and fault injection framework
- Operate only at the DSL level, for modularity and applicability to other languages
- Design an appropriate annotation-based language extension to this purpose
- Consider further fault tolerance and injection mechanisms

Further Reading

Nicolas Navet and Loïc Fejoz.
CPAL: High-level abstractions for safe embedded systems.
In *Proc. of the ACM International Workshop on Domain-Specific Modeling (DSM)*, pages 35–41, October 2016.

Ivan Cibrario Bertolotti, Tingting Hu, and Nicolas Navet.
Model-based design languages: A case study.
In *Proc. 13th IEEE International Workshop on Factory Communication Systems (WFCS)*, pages 1–6, May 2017.

Nicolas Navet, Ivan Cibrario Bertolotti, and Tingting Hu.
Software patterns for fault injection in CPS engineering.
In *Proc. 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–6, September 2017 (to appear).

Tingting Hu, Ivan Cibrario Bertolotti, and Nicolas Navet.
Towards seamless integration of N-Version Programming in model-based design.
In *Proc. 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8, September 2017 (to appear).

THANK YOU FOR YOUR ATTENTION