Modelling Resilience of Data Processing Capabilities of CPS

Linas Laibinis[1] , Dmitry Klionsky[2],
Elena Troubitsyna[1], Anatoly Dorokhov[2],
Johan Lilius[1], Mikhail Kupriyanov[2]

[1]Åbo Akademi University, Finland
[2]St. Petersburg Electrotechnical University (SPbGETU ``LETI"), Russia

MOTIVATION

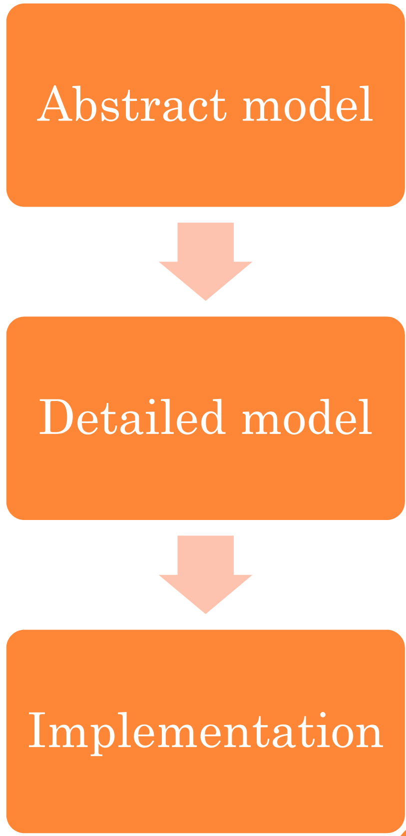
- Modern CPS should process large amount of data with high speed and confidence
  - Need for dynamically scaling architectures
- State-of-practice:
- heuristics regarding the degree of parallelism versus volume ratio
  - The impact of failure on the data processing is hard to predict
- Our aim is to study this aspect
  - via formal modelling of a reconfigurable dynamically scaling systems in Event-B
  - Sensitivity analysis and assessment of the likelihood of successful data processing under different parameters in statistical Uppaal

Talk outline

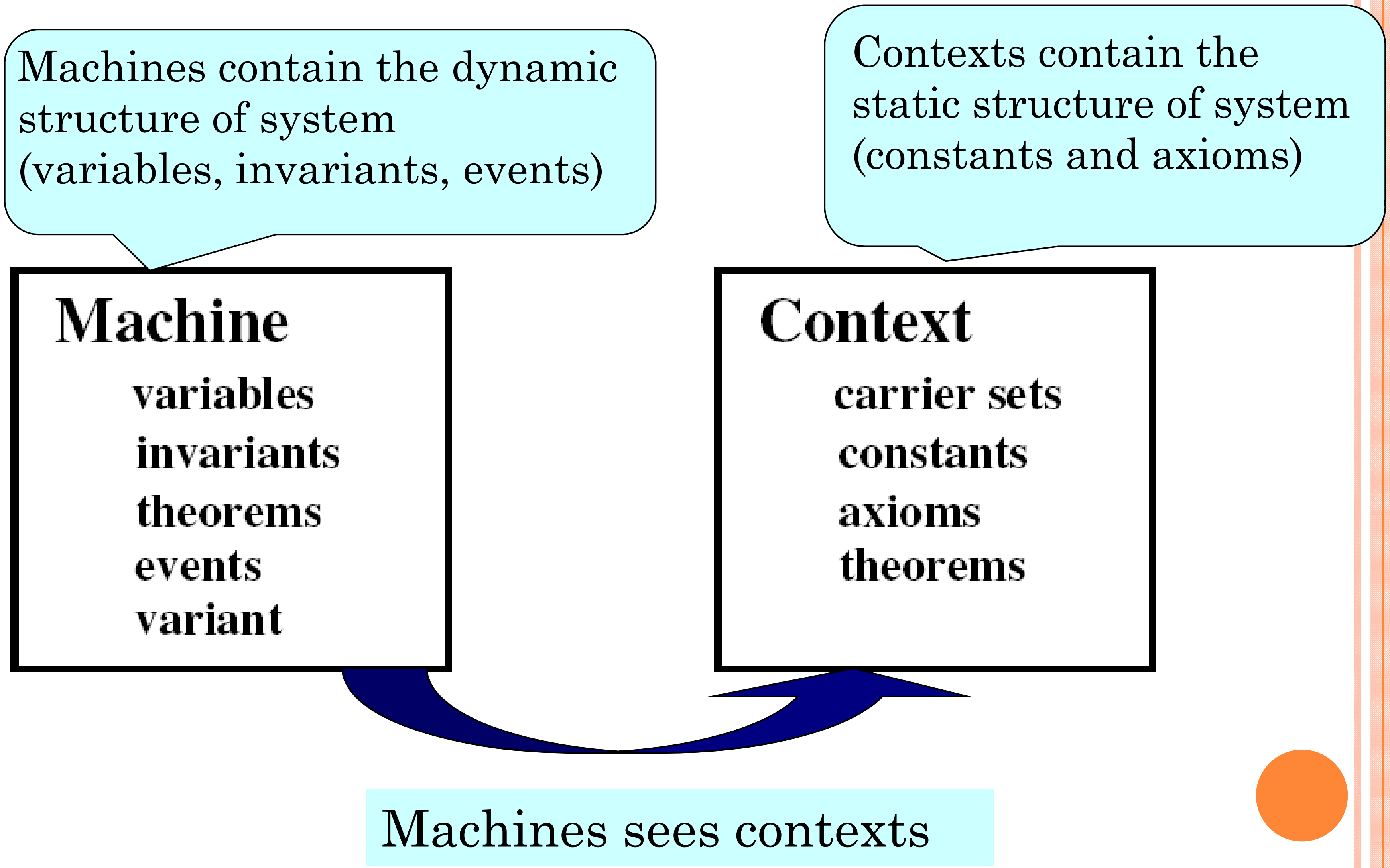
- Event-B
- Modelling reconfigurable systems: refinement strategy
- Quantitative assessment in Uppaal-SMC
- Discussion

Formal correct-by-construction development in Event-B

- Modelling facilitates requirements engineering and architecture derivation
- Explicit representation of fault tolerance, resilience
- Model transformations under resilience constraints: predictability, efficient design space exploration, clean architecture, robustness
- Automated support for formal verification



SYSTEM MODEL IN EVENT B



General form of a specification in Event-B

MACHINE

Machine Name

SETS

Definition of local types

CONSTANTS

Definition of abstract constants

VARIABLES

List of variables

INVARIANT

Typing of variables and other invariant properties of the machine

INITIALIZATION

Assignment of initial values to variables

EVENTS

EventName_1 = …

…

EventName_N = …

END

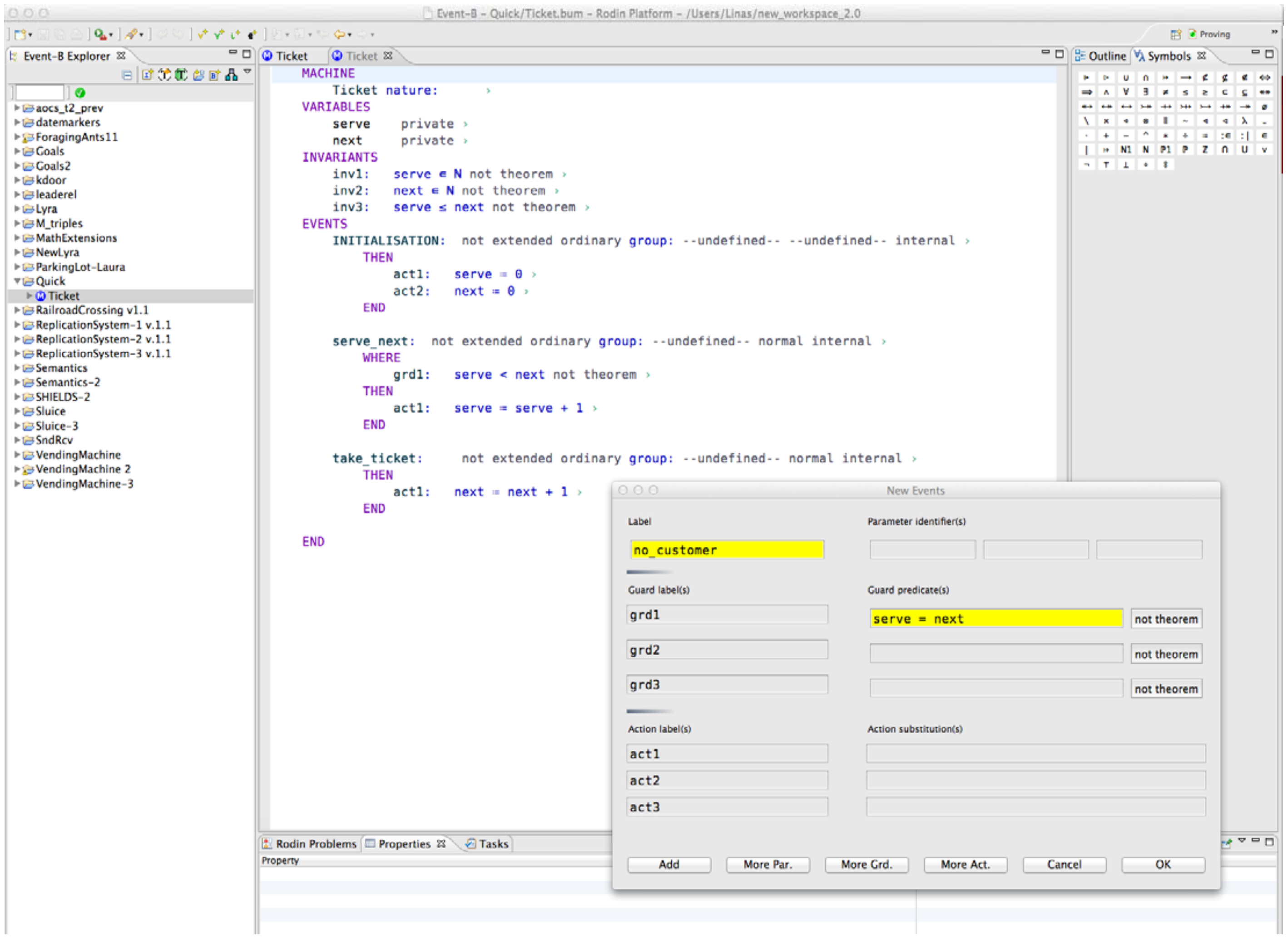
Machine consistency

- Verify that
  - Well-definedness conditions are satisfied
  - Initialization establishes invariant
  - Each event preserves invariant

- Verification is done by proofs

- Tool support – Rodin platform to generate and discard proof obligations

The Rodin platform



AUTOMATED DEVELOPMENT TOOL SUPPORT: RODIN PLATFORM

- Automates incremental formal development by refinement-based model transformation;
  - Supports strong interplay between modelling and verification;
  - Reactive: analysis tools are automatically invoked in the background whenever a change in a model is made
- The platform is extendable by plug-ins extending the Event-B language and verification techniques
- High degree of automation of verification efforts
- Integrated environment for model creation, editing, refinement, verification, animation etc.

Data processing in CPS

- Data processing (sub)-system is an important part of a wide class of CPS
  - Specific characteristics of data processing depend on the nature of CPS

- Typical steps:
  - receiving batches of data,
  - pre-processing them, (e.g., to filter our noise)
  - produce a compact data representation to be used as an input for the control functions of CPS

Case study: Floating oil refinery

  - Modelling and assessment of a multi-channel data processing of acoustic data
- Different modes of system operation
  - significantly varying data volumes to be processed.
- The system relies on dynamic scaling of parallelism to ensure the required performance.
- The pressing demand to improve resilience
  - work on augmenting data processing with fault tolerance.

- Result: complex dynamic behaviour with a tangled control flow and intricate interplay between the dynamic parallelism scaling and recofinguration

DATA PROCESSING: BASIC PROPERTIES

- Timelines and resilience
- Each data batch should be processed by a certain deadline.
- The steps of data processing are computationally-intensive
  - Reliance on parallel execution to meet the required deadlines.
- The volume of data to be processed varies
  - The system dynamically adjusts the degree of parallelism to cope with it
- Due to failures sometimes data processing might fail
  - Guarantee certain probability of success per batch

Scope of modelling

- Ensuring the required data flow between the computational steps of data processing
- Associating specific computational steps with the corresponding processing components
- Orchestrating dynamic parallel execution of the data transformation steps to achieve the adequate degree of parallelisation
- Modelling fault tolerance and reconfiguration strategies that take into account component failures and availability of the computational resources.

REFINEMENT STRATEGY

- to model the required data flow
- associate it with the involved computational components
- Introduce fault tolerance by reconfiguration
- Model not the causes of failures but effect on the execution flow

Initial model

- Abstract representation of cyclic behaviour
- Data processing is done in one atomic step
- Ensure that all the required data transformations are executed (in the required order) and comply with the desired algorithm.
- Individual data transformation steps: abstract functions that take data of one abstract type and return the transformed data belonging to another abstract type.

Modelling processing flow

- The data transformation steps are modelled as the abstract functions in the CONTEXT under the AXIOMS clause
- Sequential step: $Step2 \in FP\_Data \pto LF\_Data$
- Step2 is a partial function that takes the results of the first transformation and produces the result of the consequent transformation.

$$dom(Step2) = ran(Step1)$$

- states that the domain of this partial function is all the data that can be produced by the previous data transformation, modelled by the function Step1.

Modelling processing flow

- Modelling parallel steps:
- Explicit introduction of data partitioning

$$Step3 \in LF\_Data \nrightarrow (\mathbb{N} \nrightarrow W\_Data)$$

- The steps with parallelism can produce or accept the data that are partitioned and can be assigned to distinct components for processing.

- The maximal number of such parallel executions is fully determined by the volume of the received input data.

MODELLING PROCESSING FLOW

- The function *M* on input data and restricting the allowed data partitioning

$$
\begin{array}{l}
\forall idata, lfdata \cdot \; idata \in Input\_Data \setminus \{NO\_DATA\} \;\wedge \\
\qquad\qquad lfdata = Step2(Step1(idata)) \;\Rightarrow \\
\qquad\qquad dom(Step3(lfdata)) = 1 \,..\, M(idata) \\
Max\_M \in \mathbb{N}_1 \\
\forall idata \cdot \; idata \in Input\_Data \Rightarrow M(idata) \leq Max\_M \\
\forall f, x \cdot \; f \in dom(Step4) \wedge x \in dom(Step4(f)) \;\Rightarrow \\
\qquad\qquad (\forall x0 \cdot \; (x \mapsto x0) \in Step4(f) \Leftrightarrow Step4(f)(x) = x0)
\end{array}
$$

FURTHER REFINEMENTS

- 1st refinement: refinement of sequential steps: new events step1 and step2 modelling these data transformations and new variables *outputStep1* and *outputStep2* storing the results of these computations.

- 2nd refinement: refinement of parallel steps: refinement of atomicity of events

FURTHER REFINEMENTS

- To guard against non-termination, we define the following variant expression

$$1 \mathrel{..} M(idata) \setminus dom(outputStep3)$$

Event *step3_partial* $\widehat{=}$
Status convergent
  any
    idx
  where
    grd1 : $fstep2 = TRUE$
    grd2 : $fstep3 = FALSE$
    grd3 : $idx \in 1 \mathrel{..} M(idata)$
    grd4 : $idx \notin dom(outputStep3)$
  then
    act1 : $outputStep3(idx) := Step3(outputStep2)(idx)$
  end
Event *step3* $\widehat{=}$
  when
    grd1 : $fstep2 = TRUE$
    grd2 : $fstep3 = FALSE$
    grd3 : $dom(outputStep3) = 1 \mathrel{..} M(idata)$
  then
    act1 : $fstep3 := TRUE$
  end
END

INTRODUCING FAULT TOLERANCE

- Introducing representation of components
- Explicit definition of the link between the computation and the available components.
- Master process -- an orchestrator – schedules the computations to components
- Reconfiguration.
  - The components change their availability status non-deterministically.
  - Component is unavailable when it is either failed or does not have the computational capacity
- The scheduler detects component unavailability and reconfigure the data processing control flow, i.e., to reassign the failed tasks to the available components.

FAULT TOLERANCE

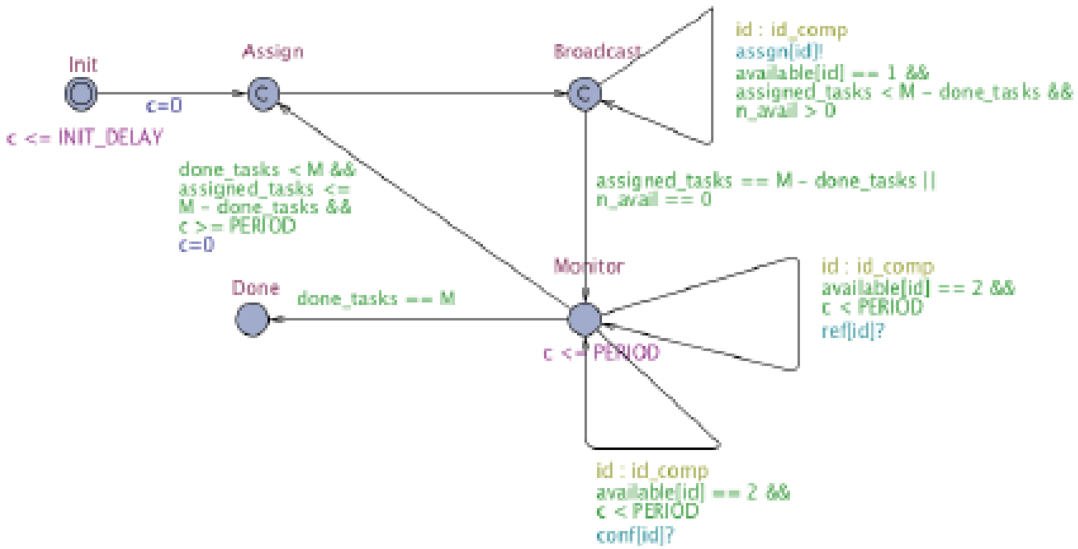
- Ensures that the reconfiguration will be performed when there are the available components

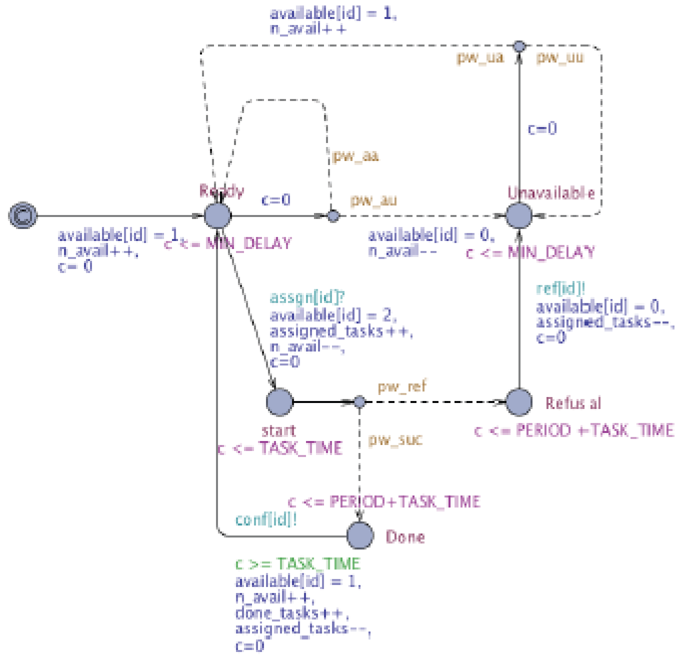
- Reconfiguration delay is tolerable: the system might become congested,
  - Then processing of the batched is aborted and resources are released

NEED FOR QUANTITATIVE ASSESSMENT

- Formal modelling helps to derive reconfigurable dynamically scalable architecture
- Gives assurance regarding correctness of the data flow processing
- Need quantitative assessment of timeliness and data processing success rate

Modelling in Uppaal-SMC





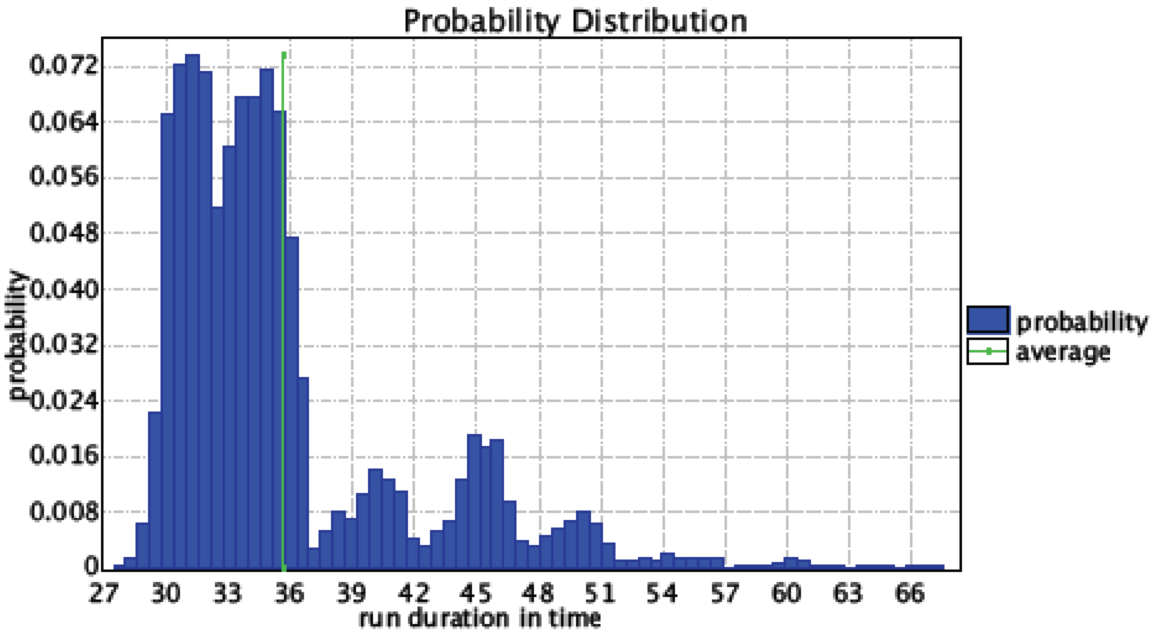
Modelling in Uppaal-SMC

- Verified a number of time reachability properties, considering different value combinations for system parameters. All the verified properties are of the form

$$\Pr[\leq \mathit{time\_bound}](<> \mathsf{Master.Done})$$

- The result is the probability that the Master component eventually reaches the state Master.Done (i.e., the state where all the parallel task calculations are successfully completed) within the given time bound.

PROBABILITY DISTRIBUTION FOR THE CASE (SAMPLE 20000, N = 10)



Runs: 4612 in total, 4609 displayed, 3 remaining.
Probability sums: 0.99935 displayed, 0.000650477 remaining.
Minimum, maximum, average: 27.4211, 67.5414, 35.6614.

Discussion

- We demonstrated how to formally derive a representation of dynamically scaling reconfigurable architecture by refinement in Event-B
- Refinement process allowed us to systematically introduce the reconfiguration mechanisms
- Improve system fault tolerance and resilience against stress load and faults
- An integration with the statistical model checking allowed us to evaluate the likelihood of successful completion of data processing by different deadlines and under diffrent probabilities of failures.