Dependability assurance of autonomous systems: an integrated formal approach

Elena Troubitsyna
KTH – Royal Institute of Technology, Sweden
Dependability

• Dependability is a property of the system to deliver its services in a trustworthy way

• It encompasses a wide set of requirements:
  • safety, reliability, security etc.

• Traditional dependability engineering: the goal is to demonstrate that the design is safe (reliable, secure etc.) under the given (constrained) environment model
Autonomous systems

- An autonomous system is capable of delivering its services in a highly independent way.

- A fully autonomous system can
  - Gain information about the environment
  - Work for an extended period without human intervention
  - Move either all or part of itself throughout its operating environment without human assistance

- Essentially, it is an autonomous mobile agent with a high degree of self-awareness and self-management (and hence, non-determinism)

*But it is also a safety-critical system!*
Formal methods in high-assurance system engineering

- Formal methods are mathematically rigorous techniques for the specification, development, and verification of SW and HW systems
  - Mathematical analysis is typically required for good system design

- Formal methods come in different flavors:
  - Lightweight FM – a formal specification precedes the actual design
  - Correct-by-construction development frameworks: refinement-based development (e.g., Event-B) and formal verification
  - Theorem proving: domain is formalized as a theory and verified by the machine-checked proofs
Event-B is a formal state-based modelling framework based on set theory and first order logic.

System state is defined by a collection of variables (can be functions, relations etc).

The dynamic system behaviour is described in terms of guarded commands (events):

- \textbf{WHEN} \textit{predicate} \textbf{THEN} \textit{assignment} \textit{stimulus} -> \textit{response}

Events define state transitions (can also be non-deterministic).

Model invariant defines a set of allowed (safe) states.
- Each event should preserve the invariant.

### Machine SimpleRobot

### Variables

- \textit{xposition}

### Invariant

\[
\text{xposition} \in \text{NAT} \land L_{Edge} \leq \text{xposition} \leq R_{Edge}
\]

### Initialisation

\[
\text{xposition} := 0
\]

### Events

- \textbf{StepLeft} $\equiv$
  \[
  \text{WHEN} \\text{xposition} \geq L_{Edge} + 1
  \\text{THEN} \\text{xposition} := \text{xposition} - 1
  \]

- \textbf{StepRight} $\equiv$
  \[
  \text{WHEN} \\text{xposition} \leq R_{Edge} - 1
  \\text{THEN} \\text{xposition} := \text{xposition} + 1
  \]

- \textbf{StepAnywhere} $\equiv$
  \[
  \text{WHEN} \ L_{Edge} < \text{xposition} < R_{Edge}
  \\text{THEN}
  \text{xposition} : \in [\text{xposition}-1, \text{xposition}+1]
  \]
Formal modelling and verification in Event-B: Rodin platform

- Rodin platform: Eclipse-based integrated modelling environment

- Automates refinement process
  - 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

- Automated support for strongest evidence of safety – safety invariant

- Support for model checking

- High degree of automation of verification efforts
Correct-by-construction development: refinement

- Abstract specification: defines essential behavior and invariant properties of system
- Refinement transform specification to add new behavior, reduce non-determinism
- The properties of the abstract specification are preserved throughout the entire refinement chain

- For example, we can add a variable $y_{position}$ to our $SimpleRobot$ specification and define movement along y axis.
  - Requires to prove that initial invariant is maintained, i.e., we refined implicit $skip$ statements – superposition refinement
  - We might also decide to replace rectangular coordinates with polar – data refinement
  - We need to define the invariant connecting new and old state space $r \times \cos(\theta) \land y = r \times \sin(\theta)$ and prove that it is always preserved
Iterative model-based development in Event-B

- Each iteration
  - aims at defining and formalising a certain subset of system requirements
  - incorporating a feedback provided by the formalisation into the requirements definition.

- Refinement step: introduction of new variables and events

- Proofs verify that refined model adheres to the abstract model
A system requirement $\text{SysReq}$ is a relation between a set $M$ of monitored variables and a corresponding set $C$ of controlled variables:

$$\text{SysReq} \subseteq M \times C$$

A software requirement $\text{SofReq}$ is a relation between a set $I$ of input variables and a corresponding set $O$ of output variables:

$$\text{SOFREQ} \subseteq I \times O$$

we need to provide satisfaction arguments in the form:

$\{\text{SOFREQ}, \text{ASM}, \text{DOM}\} \vdash \text{SysReq}$

$\text{DOM}$ Domain properties
$\text{ASM}$ Assumptions

Parnas and Madley (1995) Four-variable model
Formal modelling of safety-critical systems

- Formal modelling: avoid design faults
- Fault tolerance: hardware random faults, residual faults are unavoidable,
  - Need to guarantee deterministic behavior in different failure modes
- Modelling failure occurrence and refining according to different failure modes allows us to derive properties preserved under different failure conditions
- Augmenting model with probabilistic data (failure rate) we enable quantitative verification
  - For example, express properties like probability of catastrophic failure within n time units
Achieving Dependable Autonomy

- New challenges:
  - Open and complex operating environment
  - Continuous evolution (e.g., based on learning)
  - Inherent uncertainty – internal (complex failure modes, component interaction) and external (complex operating environment)

- Trustworthy system functioning becomes dependant on new complex factors:
  - Networks: is QoS sufficient for hard real-time safety-critical functions?
  - Security: can data for making safety-critical decisions be trusted?
  - Resources: are the components involved into implementing safety-critical functions have sufficient power level?

- Building an exhaustive model of the environment at design time is unfeasible and hence run-time verification is important

- Safety depends on many factors, hence multi-aspect models are required
Different degrees of uncertainty

• Unforeseen types of hazards
  • We do not know what we do not know, e.g., unforeseen scenarios or feature interactions
• Foreseen types of hazards
  • We do not know for sure, e.g., operational environment, coverage of different situations
• Known hazards
  • Sufficient observability and controllability
Strategic, tactic and active safety

- **Strategic**: plan ahead to maximise safety
  - Safety-aware mission planning

- **Tactic**: monitor and re-plan at run-time
  - Run-time system and environment monitoring and planning

- **Active (or emergency)**: mitigate and remove hazard occurrence
  - Akin human reflexes: hazard detection and default “safety escape”
Modelling and design challenging

• We need to combine design and run-time efforts to monitor safety and resource efficiency and adapt to operating conditions at run-time

• Need for multi-layered dependability management that combines design and run-time safety mechanisms
  • Complex, tangled and hence, requires formal modelling

• Formal modelling
  • Modelling and verification of layered architecture enabling run-time adaptation
  • Specification and verification of safety conditions
  • Design and verification of a safety net to cope with unforeseen hazards and around AI components

• Run-time planning algorithms:
  • Design high performance planning algorithms capable of controlling autonomous agents at run-time in safe and efficient way
Self-adaptive architecture

- We adopt MAPE-K architectural pattern:
  - Cyclic behaviour
  - At each cycle: M-Monitor, A-Analyse, P-Plan, E- Execute over shared K-Knowledge
Adaptive architecture of multi-agent control: system level

**Autonomous mode**

- **M**: Monitor
  - Robot’s positions
  - Obstacle appearance

- **A**: Analyse
  - Changes in safety level

  - No adaptation
  - Adaptation by route recalculation
  - Mode change

- **P**: Plan
  - Route recalculation

- **E**: Execute
  - Broadcast routes to robots

**Controlled mode**

- **M**: Monitor
  - Robot’s positions
  - Obstacle appearance

- **A**: Analyse
  - Dynamic object positions
  - Hazard presence

  - No object (hazard is mitigated)
  - Mode change

  - Hazard is present

- **P**: Plan
  - Collision avoidance manoeuvre for robots in hazardous positions
  - Route recalculation

- **E**: Execute
  - Robots are stopped
  - Collision avoidance manoeuvre is executed
Adaptive architecture of fleet control: agent’s level

- At agent’s level MAPE-K architectural pattern is used to implement “emergency response” – confine damage or mitigate hazard impact

- Safety reflex mechanism is designed to cope with unexpected hostile changes in the environment or mistakes in AI-based planning

- Safety properties are distributed through different architectural layers and have intricate interdependencies
Modelling and verifying multi-agent control architecture

A chain of model refinement:

• Abstract specification: abstract representation of a progress of a mission execution;
• 1st refinement: abstract model of system-level MAPE-K cycle;
• 2nd refinement: introducing abstract behaviour of agents; conditions triggering re-calculating planning and adaptation logic;
• 3rd refinement: introducing model of dynamically emerging hazards and change of modes
• 4th refinement: modelling agent’s MAPE-K loop
• Result: formally verified safety requirement (in our case it was collision avoidance)
Developing planning algorithm

• Formal modelling allowed us to demonstrate safety of proposed architectural solution
• We also modelled unreliable communication and hand-over from failed to functioning agent
• However, we need an algorithm capable of generating route planning for the fleet in run-time
• The main requirement to the algorithm:
  • High performance
  • Minimising resource consumption
  • Maximising safety
• Optimisation problem: solved using AI
Work in progress and challenges ahead (1/2)

- Autonomous systems are connected systems. Hence security is in the picture
  - We are working on formal modelling of safety-security interactions
    - Which safety properties are violated under different attacks?
  - Open challenge: safety in presence of untrusted agents – deriving architectures and protocols

- Deriving run-time safety monitors from system model
  - Monitoring against unknown hazards
  - Safety monitoring in presence of evolution (due to learning)
  - Change-sensitive model verification

- Conditional safety modelling based on QoS
  - Can we define a two-way approach: can safety-critical application reconfigure network to achieve the required QoS?
  - Quantitative rely-guarantee approach
  - Resource negotiations and coordination
Work in progress and challenges ahead (2/2)

- Multi-aspect modelling
  - Resource-explicit modelling
  - Projection of system-level model into different types of models: verification of timing, resources, quantitative dependability guarantees
  - Flexible adaptive architectures

- Integration with simulation platforms
  - Modelling and verifying safety of behaviour trees
  - Verification of different robotic library components

- Fundamental challenge: support for compositionality and defining different abstractions layers
Thank you!