Formal Engineering of Resilient Systems: Achievements and Challenges

Elena Troubitsyna

Åbo Akademi University,

Turku, Finland

Motivation

- Dependability is a degree of trust that we can justifiably place on a computer-based system
- Resilience is a further development of the dependability concept
- Resilience - the ability of a system to deliver services that can be justifiably trusted despite changes
- It encompasses the system aptitude to autonomously adapt to evolving requirements, operating environment changes and/or component failures

Motivation cnt.

- We need scalable resilience-explicit modelling techniques

- Modelling of resilient systems in the FP7 EU Deploy project - *Industrial deployment of system engineering methods providing high dependability and productivity (2008-2012)*

Deploy project

- **Aim:**
- to make major advances in engineering methods for dependable systems through the deployment of formal engineering methods (Event-B)

- Duration: 4 years

- **Deployment partners**

- Bosch (Germany)
- Siemens Transportation (France)
- Space Systems Finland (Finland)
- SAP (Germany)

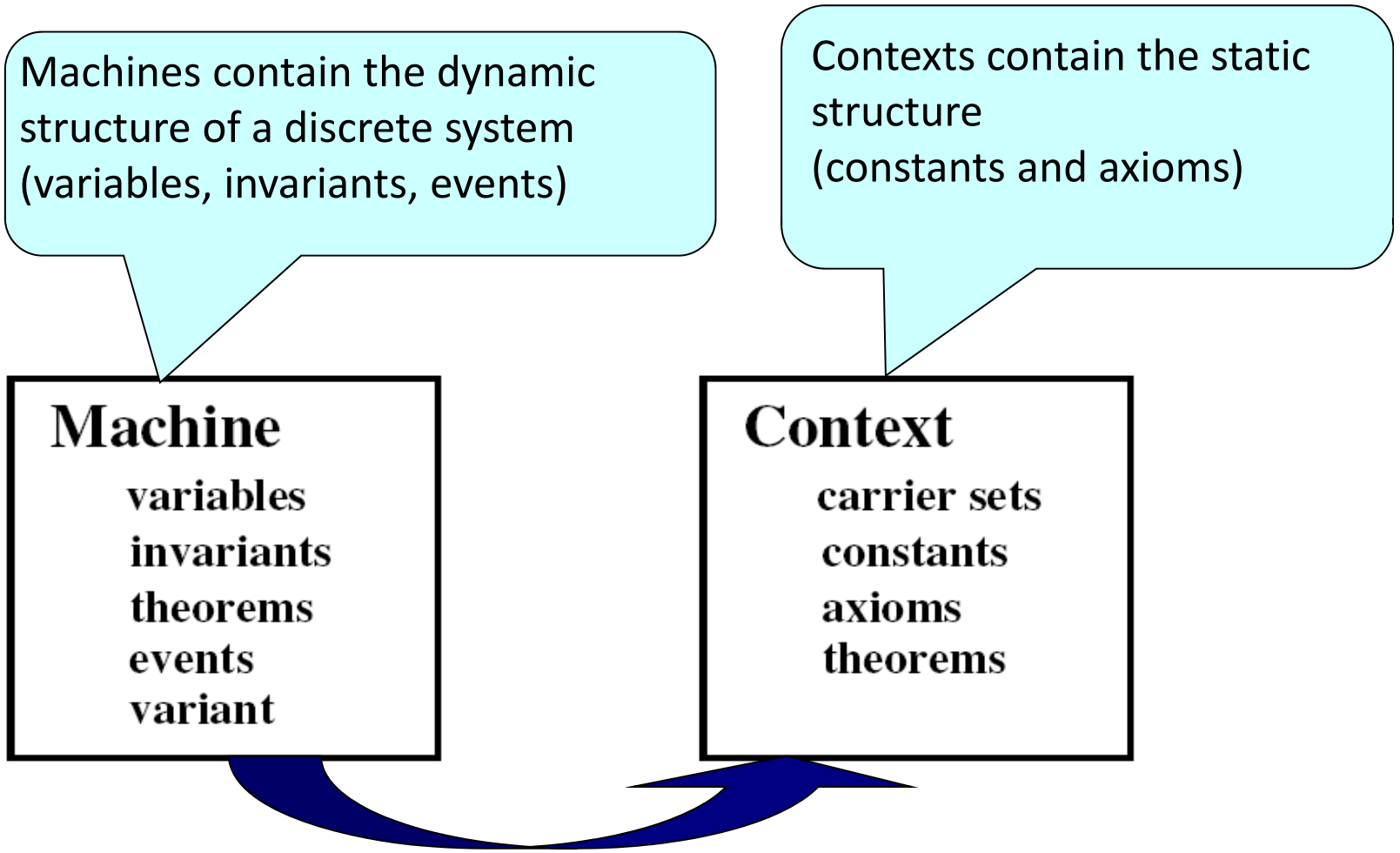
- **Technology providers**

- Åbo Akademi University (Finland)
- ETH Zürich (Switzerland)
- Newcastle University (UK)
- Systerel (France)
- University of Southampton (UK)
- University of Dusseldorf(Germany)

Event B

- Specialisation of the B-Method
- Event B has been successfully used in development of several complex real-life applications
- It adopts top-down development paradigm based on refinement
- Refinement process: we start form an abstract formal specification and transform it into an implementable program by a number of correctness-preserving steps
  - It allows to structure complex requirements
  - Small transformations simplify verification
  - Verification by theorem proving does not lead to state explosion

Abstract System Model in Event B

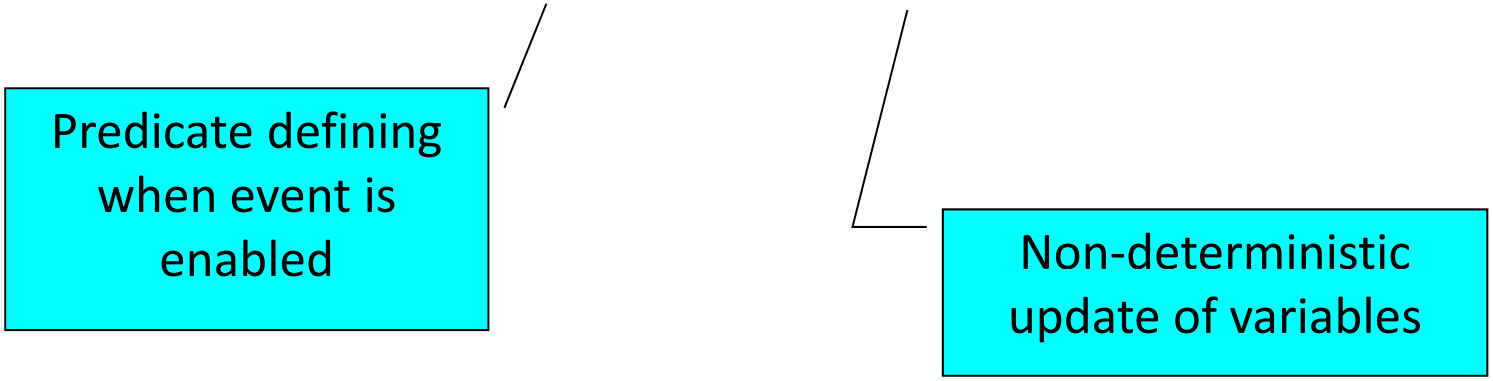


General form of a specification in B

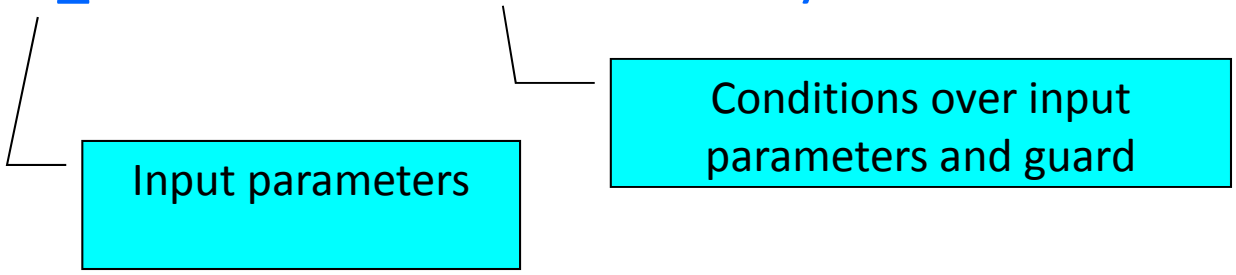
MACHINE
Machine Name
SETS
Definition of local types
CONSTANTS
Definition of abstract constants
VARIABLES
List of variables
INVARIANT
Typing of variables and other invariant properties of the machine
INITIALIZATION
Assignment of initial values to variables
EVENTS
EventName_1 = …
…
EventName_N = …
END

General form of event

WHEN guard THEN body END



ANY local_var WHERE cond THEN body 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 dischard proof obligations
- Verification by theorem proving does not lead to state explosion

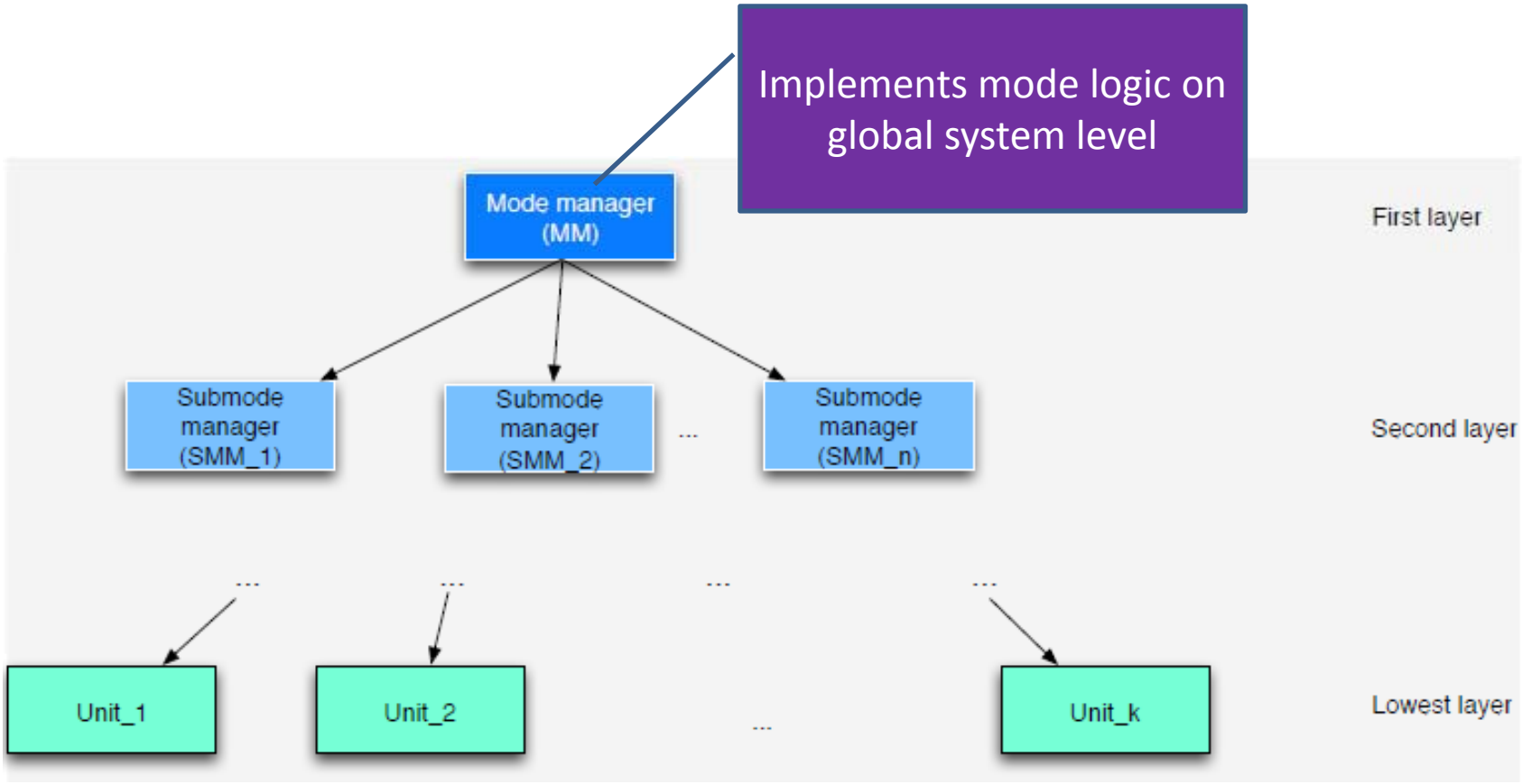
Refinement and modularization

- Allows us to introduce implementation details as well as gradually build system architecture
- Proofs guarantee that externally observable behavior is preserved in the refined model
- Modularization: we model a component via its interface and develop its implementation as separate (formal) development without losing correctness
- The technique for building systems correct-by-construction

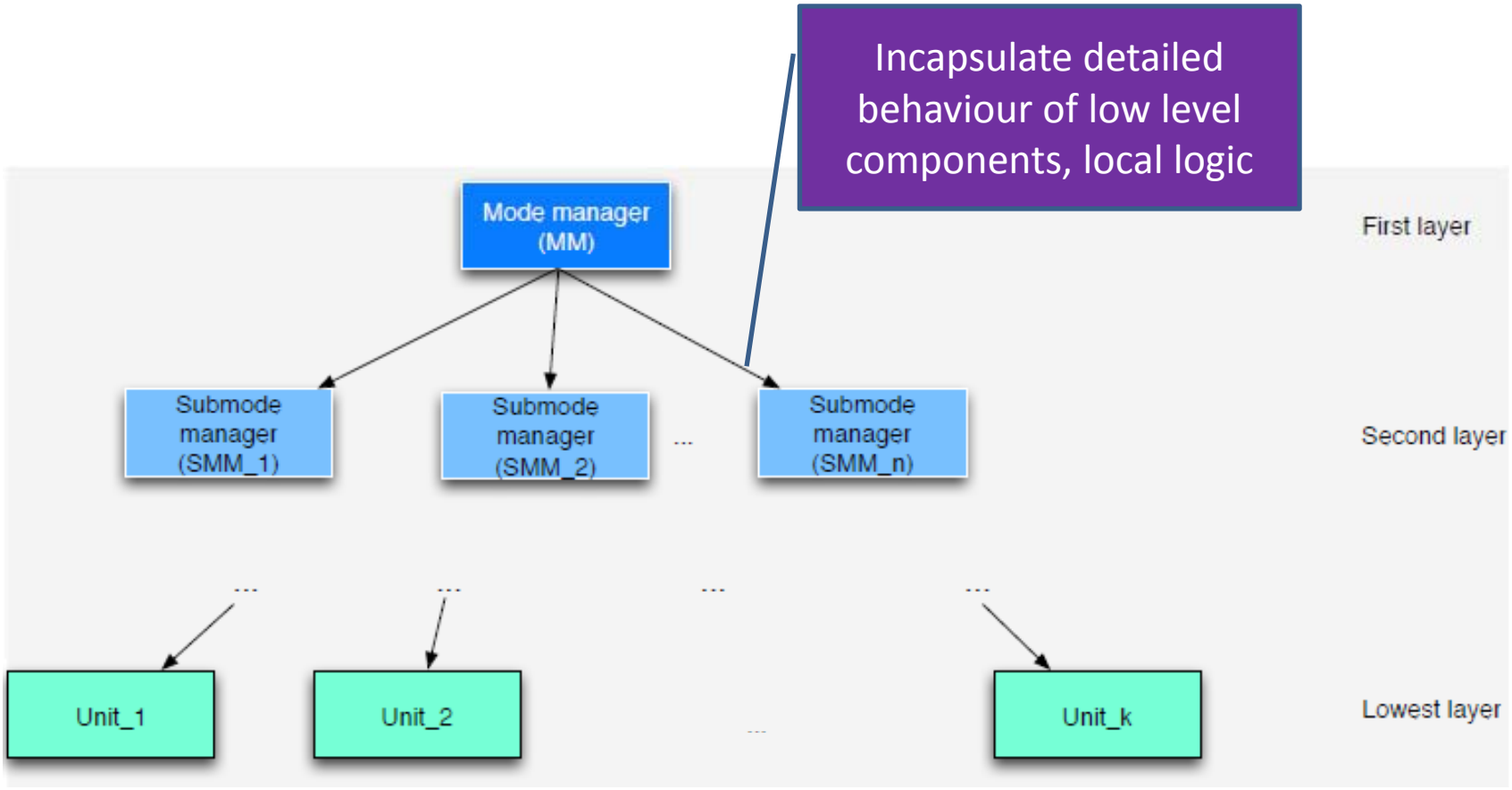
Modes

- Modes -- mutually exclusive sets of the system behaviour
- A widely used mechanism for adapting to changing operating conditions
  - The choice of the operating mode is guided by the operating conditions
- A large class of resilient systems are mode-rich systems

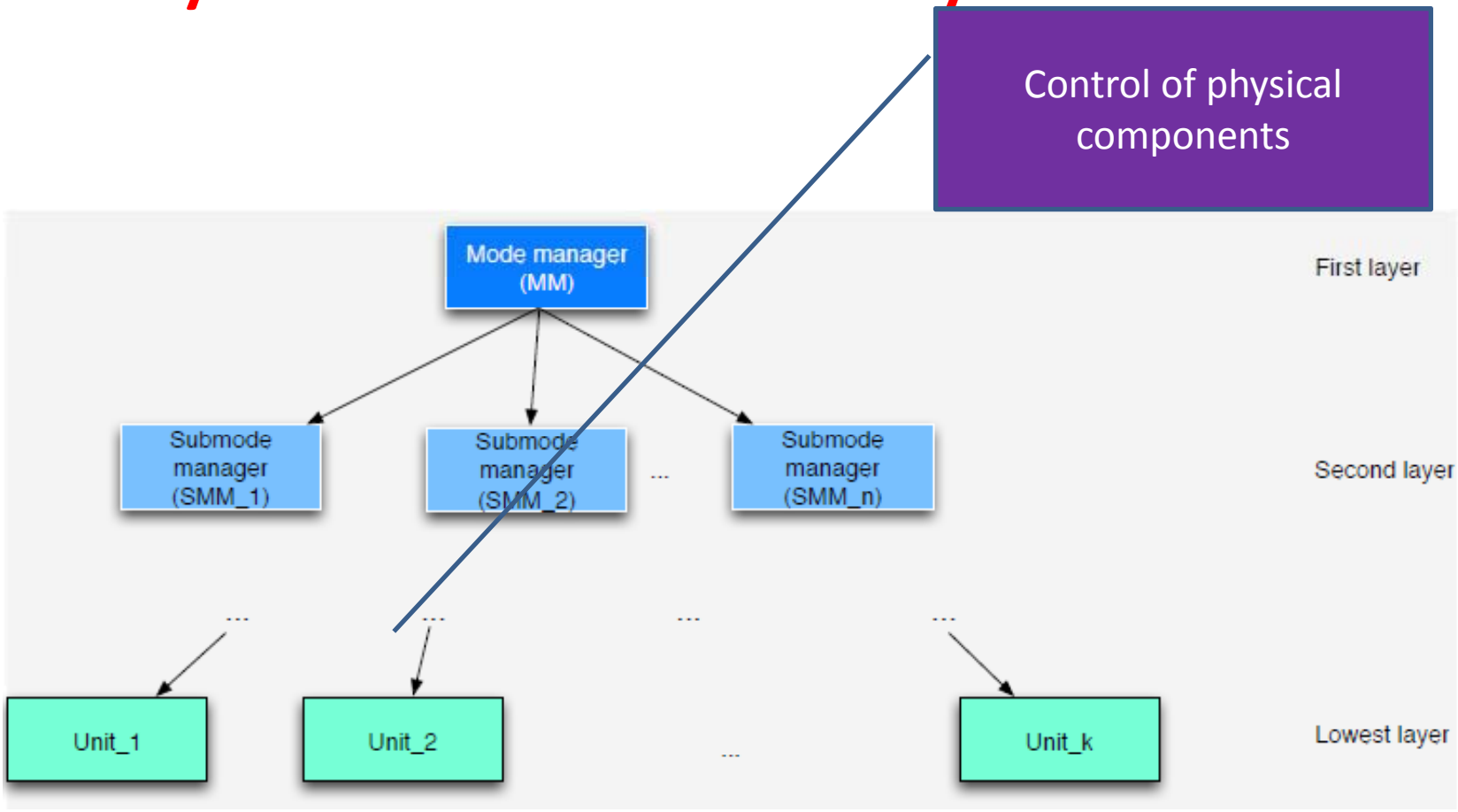
Layered mode-rich systems



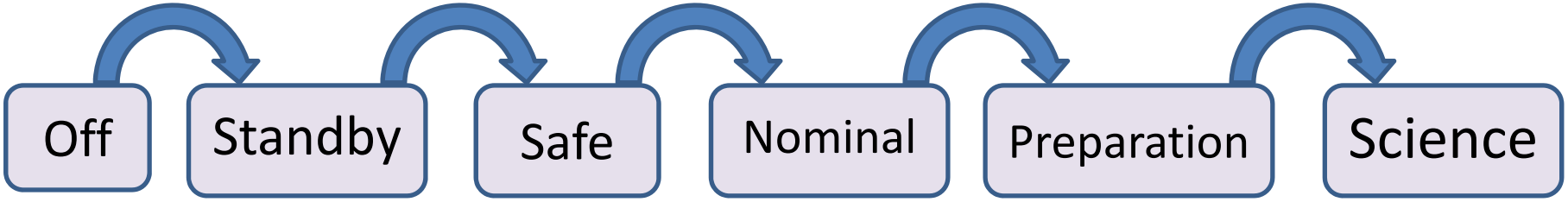
Layered mode-rich systems



Layered mode-rich systems



Attitude and Orbit Control System (AOCS): Global mode logic

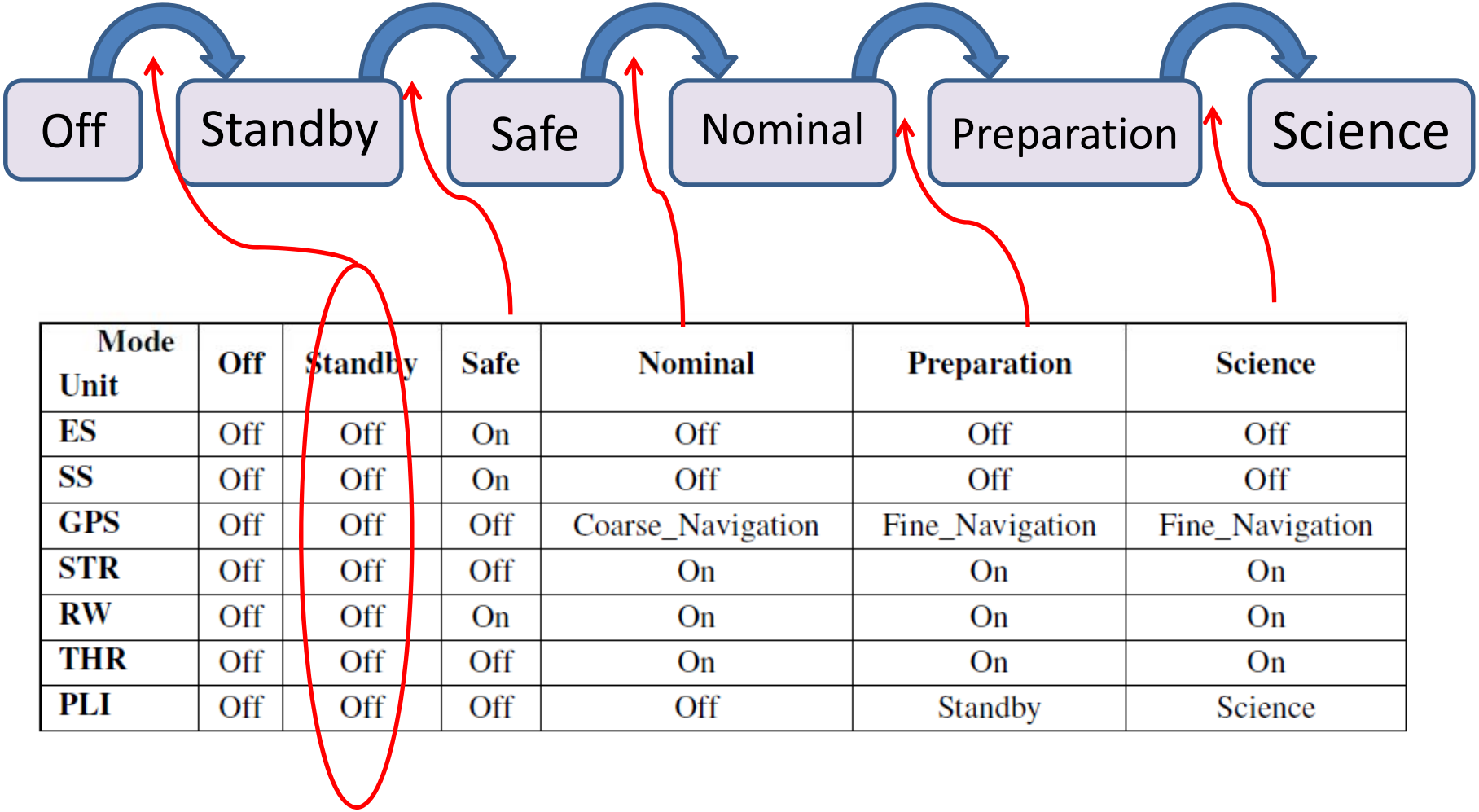


- Off– the satellite is in this mode after system (re)-booting
- Standby mode is maintained until separation from the launcher is completed
- Safe – The satellite acquire stable attitude, which allows the coarse pointing control
- Nominal -- The satellite is trying to reach the fine pointing control, which is needed to use the payload instruments
- Preparation – The payload instrument is getting ready after fine pointing is reached
- Science – the payload instrument is ready to perform its tasks. The mission goal is to reach this mode and stay in it as long as needed

AOCS Components

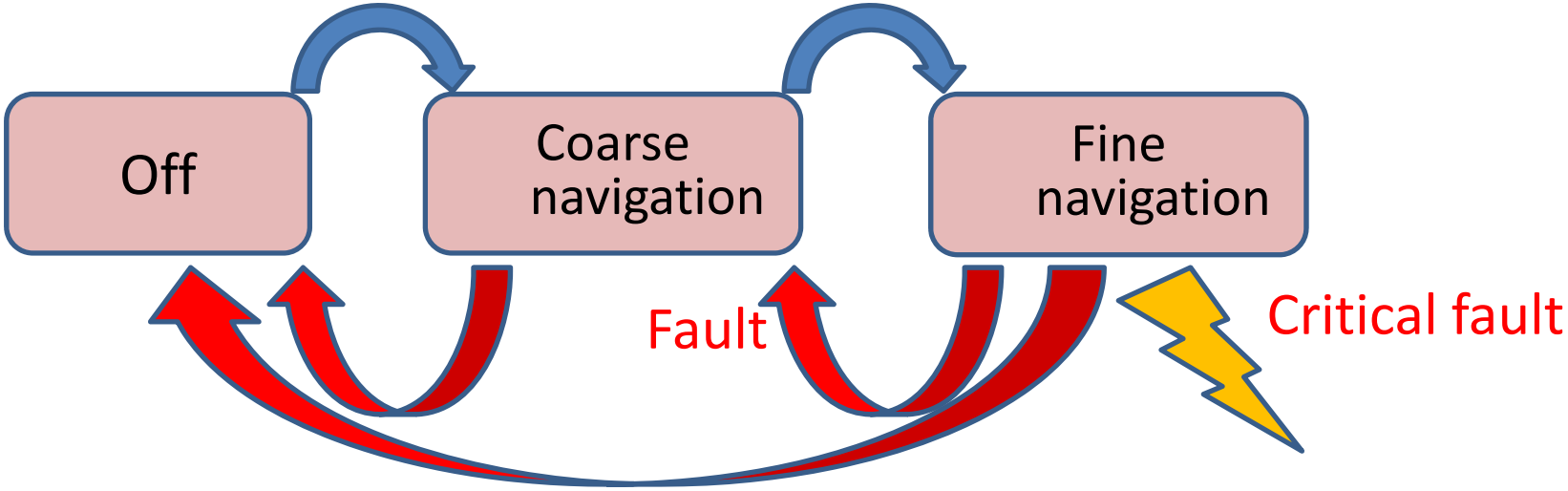
- Four sensors
  - Star tracker, Sun Sensor, Earth Sensor, Global Positioning system
- Two actuators
  - Reaction Wheel and Thruster
- Payload instrument producing mission measurements

AOCS: Mode entrance conditions



Mode / Unit	Off	Standby	Safe	Nominal	Preparation	Science
ES	Off	Off	On	Off	Off	Off
SS	Off	Off	On	Off	Off	Off
GPS	Off	Off	Off	Coarse_Navigation	Fine_Navigation	Fine_Navigation
STR	Off	Off	Off	On	On	On
RW	Off	Off	On	On	On	On
THR	Off	Off	Off	On	On	On
PLI	Off	Off	Off	Off	Standby	Science

Fault occurrence and mode logic

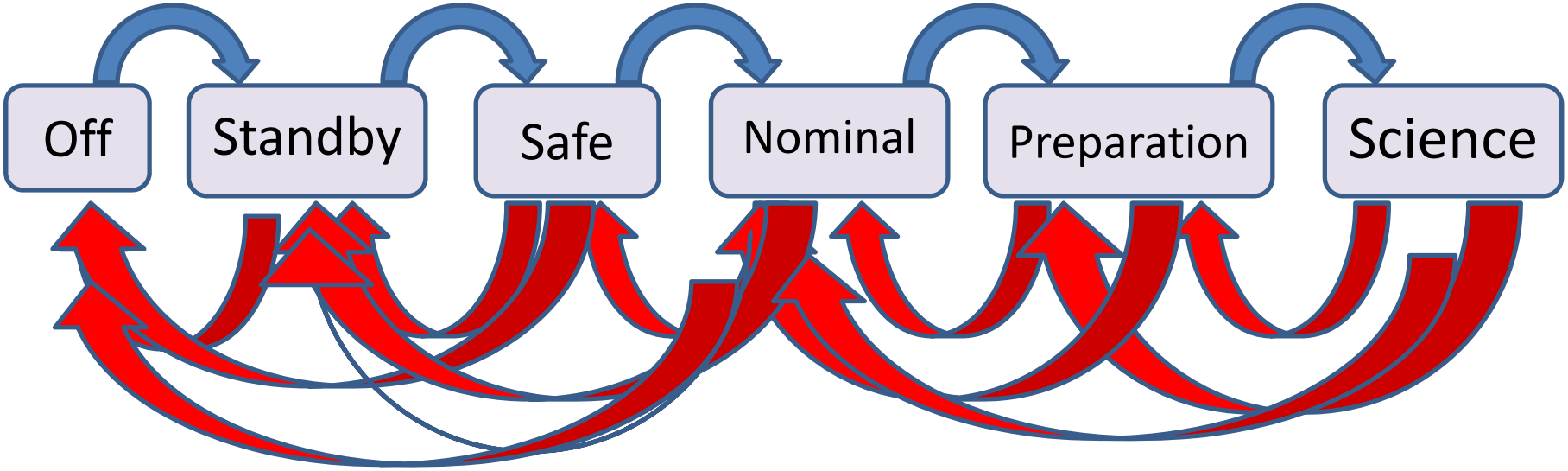


•Resilience mechanism:

• While trying to reach a certain mode a component can fail and roll-back

• In some cases the entire system needs to roll-back

Attitude and Orbit Control System (AOCS): Global mode logic



- Several component might fail at the same time or during roll-back
- Cascading effect
- State explosion problem, very large number of scenarious and hence difficult to test

We need an architectural-level rigorous approach to designing mode-rich systems to handle complexity and guarantee correctness

System structure and behavior

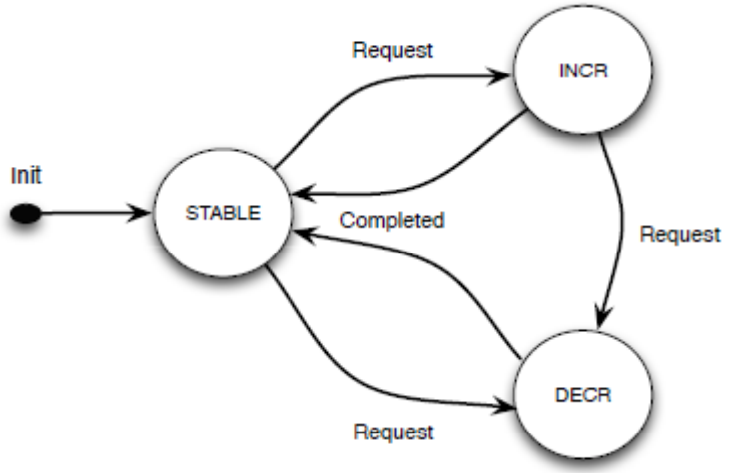
At each cycle MM assesses SMM states by monitoring their modes and detected errors and either

- Initiates a forward transition according to the predefined scenario
- Initiatates backward transition (if error occurred). Target mode depends on error severity
- Completes transition to the target mode and becomes stable (if conditions for entering mode are satisfied and no error occur)
- Maintains the current mode (if neither conditions for entering new mode are satisfied nor error occured)

Mode managing component: behavioural pattern

Introducing component status:

- *last_mode* – last successfully reached mode
- *next_target* – the target mode that a compoent is currently in transition to
- *previous_target* – the previos mode that a component was in transition too (though not necessarily reached it)



$$Stable \triangleq last\_mode = previous\_target \wedge next\_target = previous\_target$$

$$Increasing \triangleq last\_mode = previous\_target \wedge previous\_target < next\_target$$

$$Decreasing \triangleq next\_target < previous\_target$$

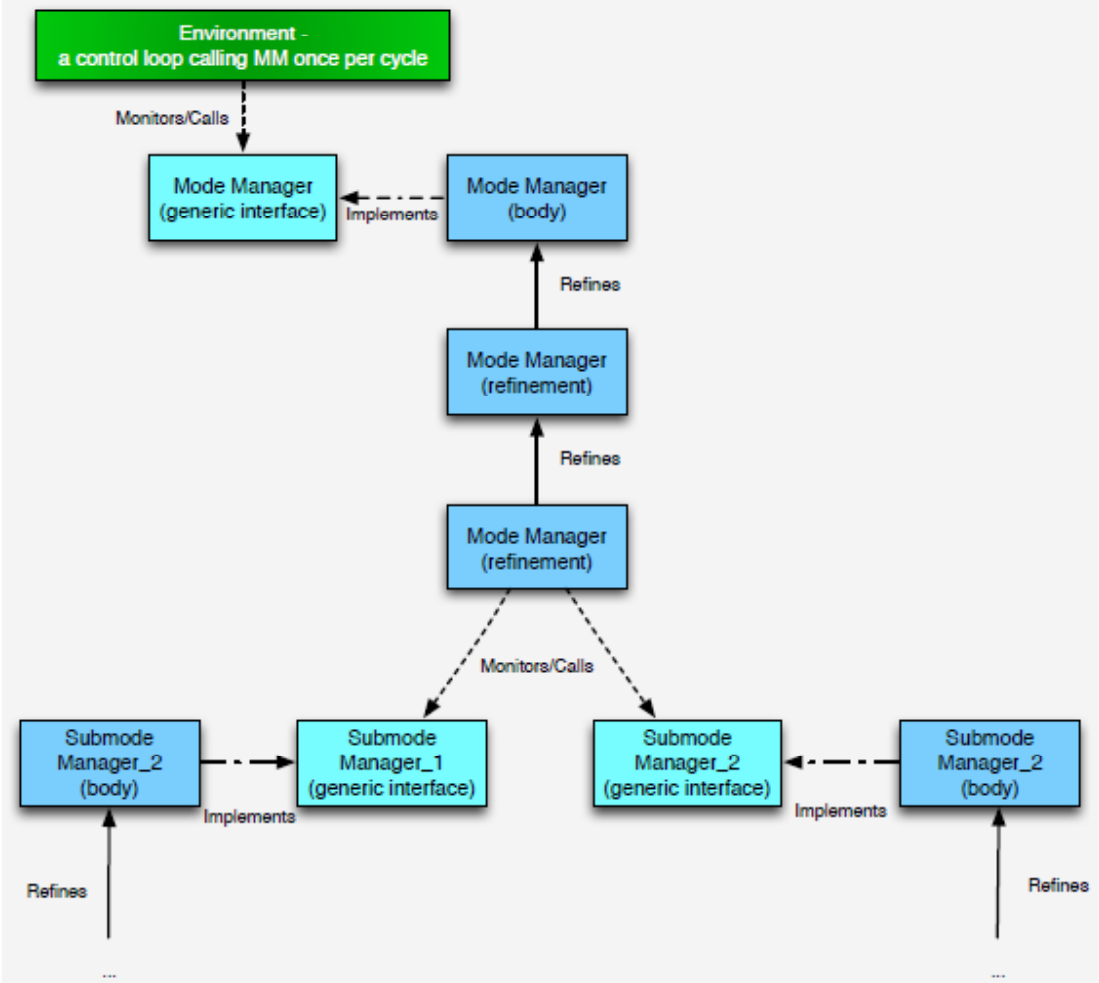
Mode managing component:

- Stable state: decides to initiate a new mode transition to some more advanced mode
- Transition state: monitors states of lower layer components. If at some point mode entry conditions are satisfied for the target mode then complete transition and become stabe
- In both stable and transitional states: monitors lower layer components for the detected error, execute error recovery by setting new target mode if errors are detected

Formal development strategy

- General idea: to define generic interface of mode managing component
- Build the entire system in the top-down fashion by instantiating generic interface and unfolding one layer at the time
- Proof desired properties of model logic as part of refinement verification

Refinement strategy



Proved properties of mode logic

- Unambiguity of mode logic:

$$\forall i, j \bullet M_i \in Modes \ \wedge \ M_j \in Modes \ \wedge \ i \neq j \ \Rightarrow$$
$$Mode\_ent\_cond(M_i) \cap Mode\_ent\_cond(M_j) = \varnothing$$

- A component satisfies mode entry conditions and mode invariant (when it is *Stable)*

$$\forall i \bullet m_i \in Modes \wedge current\_mode = M_i \wedge Stable \ \Rightarrow \ Mode\_ent\_cond(M_i)$$
$$\forall i \bullet m_i \in Modes \wedge current\_mode = M_i \wedge Stable \ \Rightarrow \ Mode\_Inv(M_i)$$

Overview of the approach

- The approach is based on instantiation and refinement of a generic pattern for specifying a mode managing component.
- The overall development process can be seen as a stepwise unfolding of architectural layers. Each such unfolding is accompanied by proving correctness of refinement, which also verifies the mode consistency properties between two adjacent layers.
- Incremental verification allows us to guarantee the global mode consistency, yet avoid checking the property for the whole system architecture at once.
- Positive experience: disciplined development, confidence, control over complexity, traceability

Requirements engineering for mode-rich systems

- In our AOCS example the mode logic was given
- Usually the global mode scenario is known
- But how to derive the fault tolerance part?
  - The rollbacks to degraded modes

Deriving mode logic using FMEA

- The proposed approach: to conduct Failure Modes and Effects Analysis (FMEA) of each particular mode.
- It allows us to
- encapsulate the details of dynamic reconfiguration performed in response to the occurred errors
- arrive at an efficient mechanism for structuring fault tolerance according to the identified architectural layers.

FMEA worksheet fields

Component – name of a component

Failure mode – possible failure modes

Possible cause – possible cause of a failure

Local effects – caused changes in the component behaviour

System effect – caused changes in the system behaviour

Detection – determination of the failure

Remedial action – actions to tolerate the failure

Proposed FMEA worksheet fields

Global mode – name of a global mode

Failure mode – possible failure modes (unit failure)

Possible cause – possible cause of a failure

Local effects – caused changes in the component behaviour

System effect – caused changes in the system behaviour

Detection – determination of the failure

Remedial action – actions to tolerate the failure

The *general rule* of the rollback

- Mode Manager (MM) puts the system to the previous, however as advanced as possible, global mode where the failed unit is in *Off* state.

- All units that should be operational in the chosen degraded mode should be faul free
  - Otherwise, MM should put the system to a global mode where all failed units are in *Off* states.

FMEA worksheet for mode *Nominal*

Global mode	**Nominal**
Failure mode	GPS failure
Possible cause	Primary hardware failure
Local effects	Loss of precision of GPS
System effects	Switch to a degraded mode
Detection	Comparison of received data with the predicted one
Remedial action	If a failure occurs for the first time, then switch the nominal branch of the unit to the mode *Off* and the redundant branch of the unit to the mode *Coarse*. During the reconfiguration between the unit branches maintain the current global mode *Nominal*. If the redundant branch fails, then switch the branch to the mode *Off* and put the system to the previous, the most advanced, global mode where GPS unit is in *Off* state, i.e., to the mode *Safe*. Keep the unit status equals to *Locked* only if one of two branches is in *Coarse* state and there is no ongoing reconfiguration. Otherwise, change the unit status to *Unlocked*.

The results of integrating FMEA into the requirements engineering

- Allows for a systematic derivation of fault tolerance part of mode logic.
- Facilitates formalisation of the required conditions of mode consistency

Resilience in the context of service-oriented systems

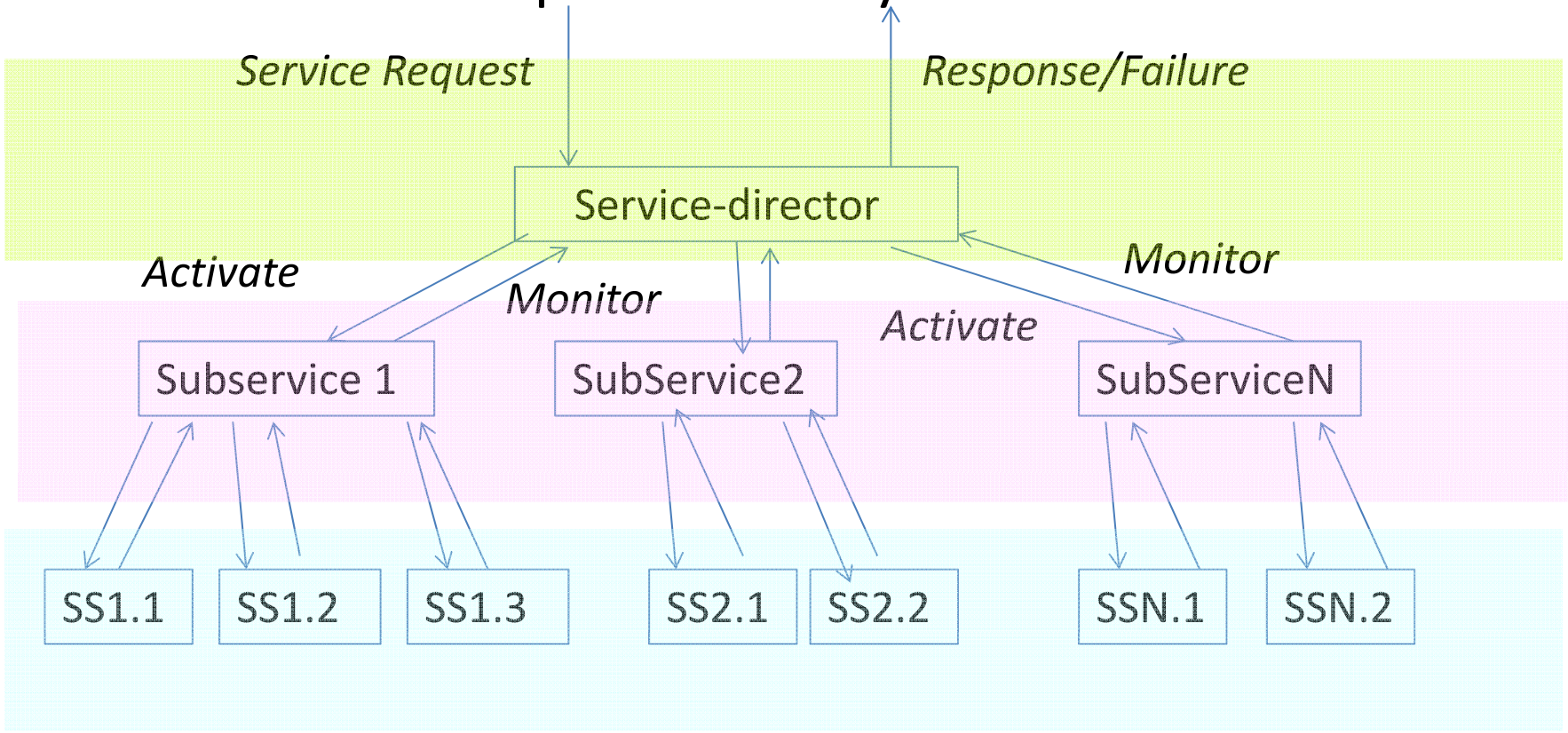
- Service-oriented computing enables rapid building of complex sofware by assembling readily-available services
- Formalisation of Lyra development approach (by Nokia) in Event-B: correctness and agility + fault tolerance and
- Are fault tolernace mechanisms appropriate, i.e., allow us to meet the desired quality of service (QoS) attributes?
- Need for techniques enabling evaluation of QoS attributes at early design stages

Proposed approach

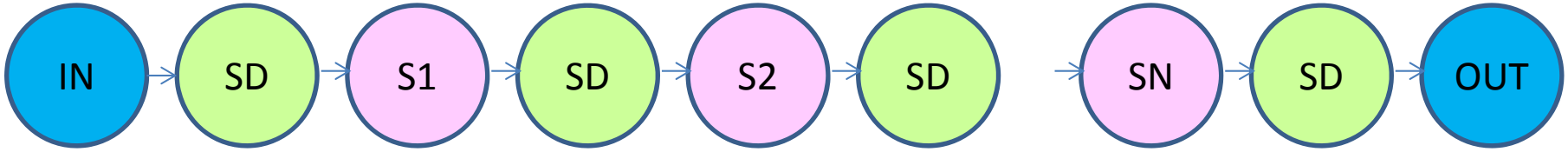
- Build model of dynamic service architecture in Event-B
- Formalise and verify dynamic service behaviour
- Augment Event-B model with stochastic information about rates and durations of the orchestrated services
- Use probabilistic model checking to verify the desired QoS attributes of the resultant Continuous-Time Markov Chain (CTMC)

Service-Oriented Systems (SOS)

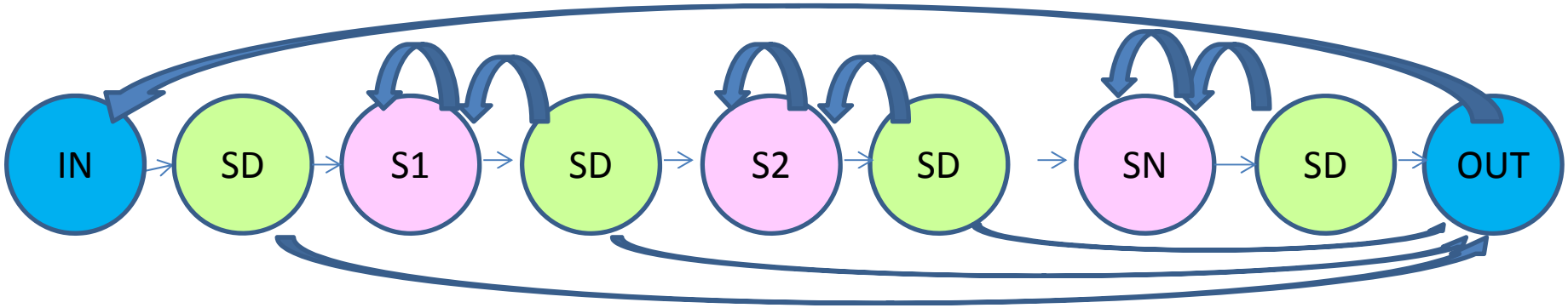
- Services are built by aggregating of lower-layer subservices
- Coordination is performed by *a service-director*



Flow of control



- Services are handled one by one
- After each subservice execution the service director might
  - allow the subservice to continue
  - proceed to the next subservice
  - retry subservice execution
  - abort (the entire service)



Event-B model as transition system

- Let Σ be a state space and, $\mathcal{E}$ a set of events, $\mathcal{I}$ invariant

- Event is defined as

  e = **when** G_e **then** R_e **end**

  can be seen as syntactic sugaring for

$$e(\sigma, \sigma') = G_e(\sigma) \bigwedge R_e(\sigma, \sigma')$$

  - To define Event-B model as a transition system, we define functions *before(e)* and *after(e)*:

$$\text{before}(e) = \{\sigma \in \Sigma \mid \mathcal{I}(\sigma) \wedge G_e(\sigma)\}$$

$$\text{after}(e) = \{\sigma' \in \Sigma \mid \mathcal{I}(\sigma') \wedge (\exists \sigma \in \Sigma \cdot \mathcal{I}(\sigma) \wedge G_e(\sigma) \wedge R_e(\sigma, \sigma'))\}$$

Event-B model as transition system (cnt.)

- The behaviour of any Event-B machine is defined by a transition relation –>

$$\frac{\sigma, \sigma' \in \Sigma \ \wedge \ \sigma' \in \bigcup_{e \in \mathcal{E}_\sigma} \mathsf{after}(e)}{\sigma \rightarrow \sigma'}$$

where $\mathcal{E}_\sigma = \{e \in \mathcal{E} \mid \sigma \in \mathsf{before}(e)\}$ is a subset of events enabled in σ

Formalising dynamic properties

- By defining Event-B specification we can formally define a number of essential dynamic properties of SoS under construction
- Formalisation of requirements can be added as a collection of model theorems

- If mapping between model events and “skeleton” is defined then the process can be automated

- Proving: either within Rodin platform or using external theorem provers

Probabilistic Event-B

- Transforming dynamic service architecture into a Continuous Time Markov Chain
- We aim at verifying time-bounded reachability and reward properties related to a possible abort of service execution
- Properties are specified as Continuous Stochastic Logic (CSL) formulae

Model transformation

- All events are augmented with the information about probability and duration of all the actions
- For the state $\sigma \in \Sigma$ and event $e \in \mathcal{E}$ where $\sigma \in before(e)$ assume that R_e can transform σ to a set of states $\{\sigma_1', \ldots, \sigma_m'\}$
- We augment every such transformation with a constant transition rate

$$\lambda_i \in \mathbb{R}^+,$$

- The sojourn time in state σ is exponentially distributed with parameter $\Sigma\lambda_i$
- Hereby we replace a nondeterministic choice between the possible successor states by the probabilistic choice associated with the exponential race conditions

Event-B model as a probabilistic transition system

- The behaviour of a probabilistically augmented Event-B machine is defined by a transition relation

$$\frac{\sigma, \sigma' \in \Sigma \ \wedge \ \sigma' \in \bigcup_{e \in \mathcal{E}_\sigma} \mathsf{after}(e)}{\sigma \xrightarrow{\Lambda} \sigma'}$$

where $\Lambda = \sum_{e \in \mathcal{E}_\sigma} \lambda_e(\sigma, \sigma')$

Quantitative verification of QoS attributes with PRISM

- What is the probability that at least one service execution will be aborted during a certain time interval?
- What is the probability that a number of aborted services during a certain time interval will not exceed some threshold?
- What is the mean number of served requests during a certain time interval?
- What is the mean number of aborted requests during a certain time interval?
- What is the mean number of failures of some particular subservice during a certain time interval?

Discussion

- Rich experience in modelling resilient systems from the transportation, aerospace and business information system domains

- Two types of approaches:
  - Focusing on creating modelling patterns and guidelines for representing and verifying certain resilience-related behavior
  - Integrating (external) techniques for safety and reliability analysis into the formal development process of Event-B

Challenges

- Scalability in formal modelling
- Powerful automatic tool support

- Event-B and Rodin platform:

  event-b.org
- Deploy project:

  http://www.deploy-project.eu/

Some references

- Alexei Iliasov, Elena Troubitsyna, Linas Laibinis, Alexander Romanovsky, Kimmo Varpaaniemi, Dubravka Ilic, Timo Latvala. Developing Mode-Rich Satellite Software by Refinement in Event-B. *Science of Computer Programming, 2012.*

- Anton Tarasyuk, Elena Troubitsyna, Linas Laibinis. Formal Modelling and Verification of Service-Oriented Systems in Probabilistic Event-B. In Proc. of *International Conference on Integrated Formal Methods, IFM 2012*, Lecture Notes for Computer Science, Springer, 2012.

- Yuliya Prokhorova, Linas Laibinis, Elena Troubitsyna, Kimmo Varpaaniemi, Timo Latvala, Derivation and Formal Verification of a Mode Logic for Layered Control Systems. In: Tran Dan Thu, Karl Leung (Eds.), *Proceedings of the 18th Asia-Pacific Software Engineering Conference (APSEC 2011)*, 49-56, IEEE Conference Publishing Services (CPS), 2011.

Thank you!