X-by-Construction Design Framework for Engineering Autonomous and Distributed Real-time Embedded Software Systems

Tutorial: Simulation-based development of networked avionics systems using the XANDAR toolchain

Tobias Dörr[1], Florian Schade[1], Alexander Ahlbrecht[2]
[1]Karlsruhe Institute of Technology (KIT)
[2]German Aerospace Center (DLR)

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957210.

Agenda

- **Part I:** The software development methodology of XANDAR
  - Motivation + XANDAR project overview
  - Software development approach
  - Simulation and target deployment
  - Safety pattern concept + two sample patterns
- **Part II:** Programming models for concurrent systems
  - Fundamentals + selected programming models
  - Applicability to the XANDAR toolchain
- **Part III:** Live demonstration of XbCgen
  - Overview and introductory examples
  - Application to an avionics use case

Part I

The software development methodology of XANDAR

Motivation

- Designing embedded systems is an error-prone task
- **Flight 501 of Ariane 5 (1996):**
  - Programming error in the control software [1]
  - Failure of the Inertial Reference System during test flight
  - Self-destruction after 37 seconds
- **Remote access to 1.4 million road vehicle (2015):**
  - Vulnerability in the vehicle's head unit [3]
  - Remote reprogramming of a gateway component
  - Full access to a CAN bus of the vehicle

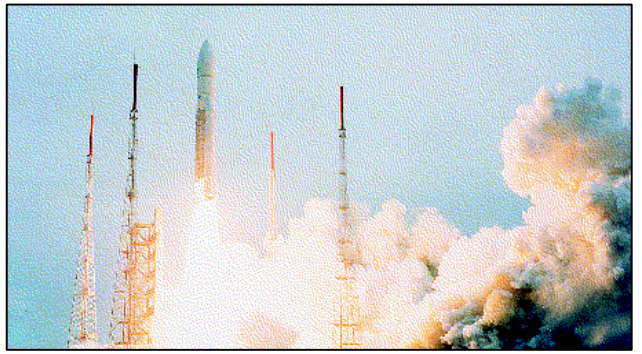


Image source: [2]

Image source: [3]

[1] B. Nuseibeh, **"Ariane 5: Who Dunnit?"**, IEEE Software, vol. 14, no. 3, May-June 1997.
[2] European Space Agency, Bulletin Nr. 89, online: https://www.esa.int/esapub/bulletin/bullet89/dalma89.htm, accessed 2 May 2023.
[3] C. Miller, "**Lessons Learned from Hacking a Car**", IEEE Design & Test, vol. 36, no. 6, December 2019.

Motivation

- **Properties of modern cyber-physical systems** aggravate the design challenge:

Consolidation of functions on multicore platforms, e.g. in the automotive domain

Integration of Artificial Intelligence (AI) algorithms, e.g. for object detection

Functional safety and cybersecurity requirements, e.g. fault tolerance

Continuous interaction with remote entities, e.g. in a cloud-edge setting

- **Conflicting requirements** are particularly difficult to address, for example:

Centralization of automotive on-board architectures [4]

⇒ Lack of physical separation between components
⇒ Potential for timing interferences, fault propagation, …
⇒ Functional safety issues

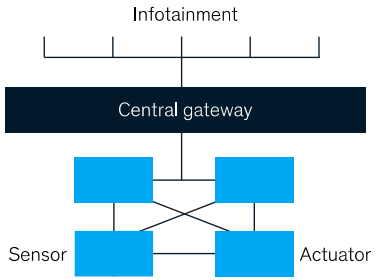


Image source: [4]

[4] O. Burkacky, J. Deichmann, J. P. Stein, "**Automotive software and electronics 2030: Mapping the sector's future landscape**", McKinsey & Company, 2019.

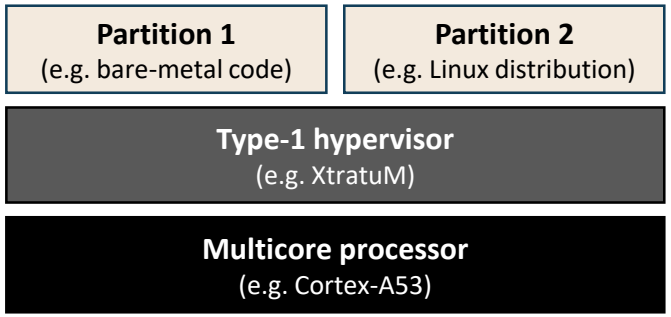
Motivation

- **Methodological gap** between requirements and implementations, e.g.:
  - ISO 26262 [5] specifies Freedom from Interference (FFI) requirements
  - A hypervisor such as XtratuM [6] is a building block that contributes to this goal
  - The correct application of such building blocks can be difficult (timing interferences, …)

ISO 26262 – Functional safety standard for road vehicles (initially published in 2011). Covers the following scope [5]: *"This document addresses possible hazards caused by malfunctioning behaviour of safety-related E/E systems, including interaction of these systems."*

Hypervisor – Software layer providing independent execution environments on a single processor. Hypervisors running directly on the target hardware (without a host OS) are called type-1 hypervisors.

Example of a hypervisor-based system

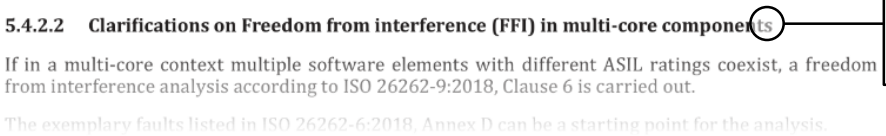


[5] ISO 26262-1:2018, **"Road vehicles — Functional safety — Part 1: Vocabulary"**, Geneva, 2018.
[6] M. Masmano, I. Ripoll, et al., **"XtratuM: a Hypervisor for Safety Critical Embedded Systems"**, 11th Real-Time Linux Workshop, 2009.

Motivation

- **Motivation:** How to apply such building blocks in a provably correct manner?

Excerpt of ISO 26262-11:2018



5.4.2.2 **Clarifications on Freedom from interference (FFI) in multi-core components**

If in a multi-core context multiple software elements with different ASIL ratings coexist, a freedom from interference analysis according to ISO 26262-9:2018, Clause 6 is carried out.

The exemplary faults listed in ISO 26262-6:2018, Annex D can be a starting point for the analysis.

"Techniques such as hypervisors can help to achieve software partitioning […]"

X-by-Construction (XbC) – The *"step-wise refinement process from specification to code that automatically generates software (system) implementations that by construction satisfy specific non-functional properties concerning security, dependability, reliability, or resource/energy consumption, to name but a few"* [7].

- **Key idea:** Apply the X-by-Construction (XbC) paradigm to the design process
  - Auto-generate implementation artefacts that leverage suitable low-level techniques
  - Simplify the post-hoc verification process in a manner comparable to [8]

[7] M. H. ter Beek et al., **"X-by-Construction"**, Leveraging Applications of Formal Methods, Verification and Validation: Modeling, ISoLA '18, Springer, Cham, October 2018.

[8] B. W. Watson, D. G. Kourie, et al., **"Correctness-by-Construction and Post-hoc Verification: A Marriage of Convenience?"**, Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques, ISoLA '16, October 2016.

Overview of the XANDAR project

https://xandar-project.eu

Duration:
01/2021 – 12/2023

Budget:
€ 4.96 million

Project coordinator:
Prof. Jürgen Becker (KIT)

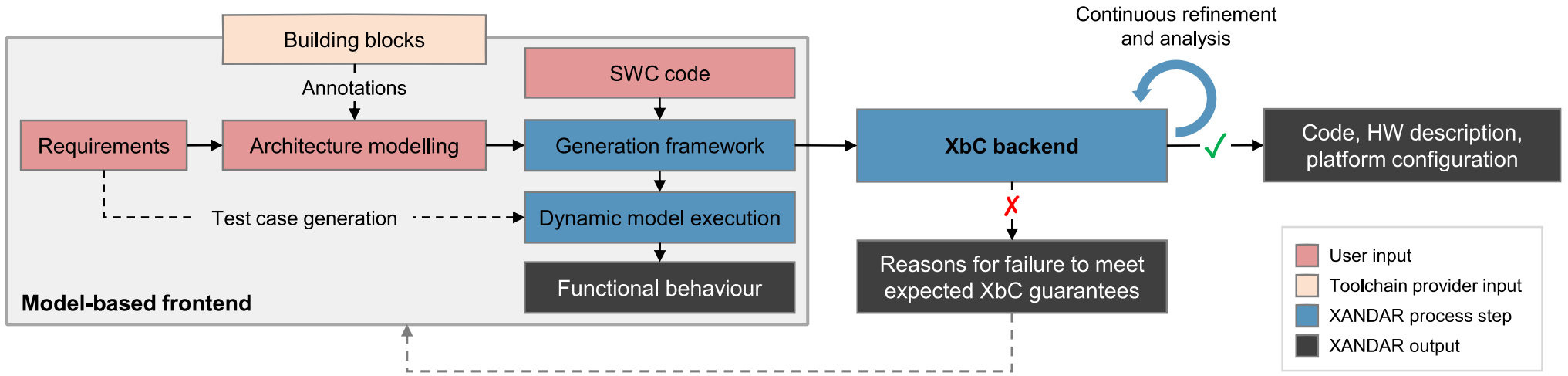
Scientific coordinator:
Prof. Nikolaos Voros (UoP)

Goal: Deliver a mature software toolchain that uses the X-by-Construction paradigm to generate system implementations with guaranteed properties

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957210.

Overview of the XANDAR project

- The toolchain allows users to apply the XANDAR development process [9]
- This process defines a sequence of **iteratively traversed steps**:

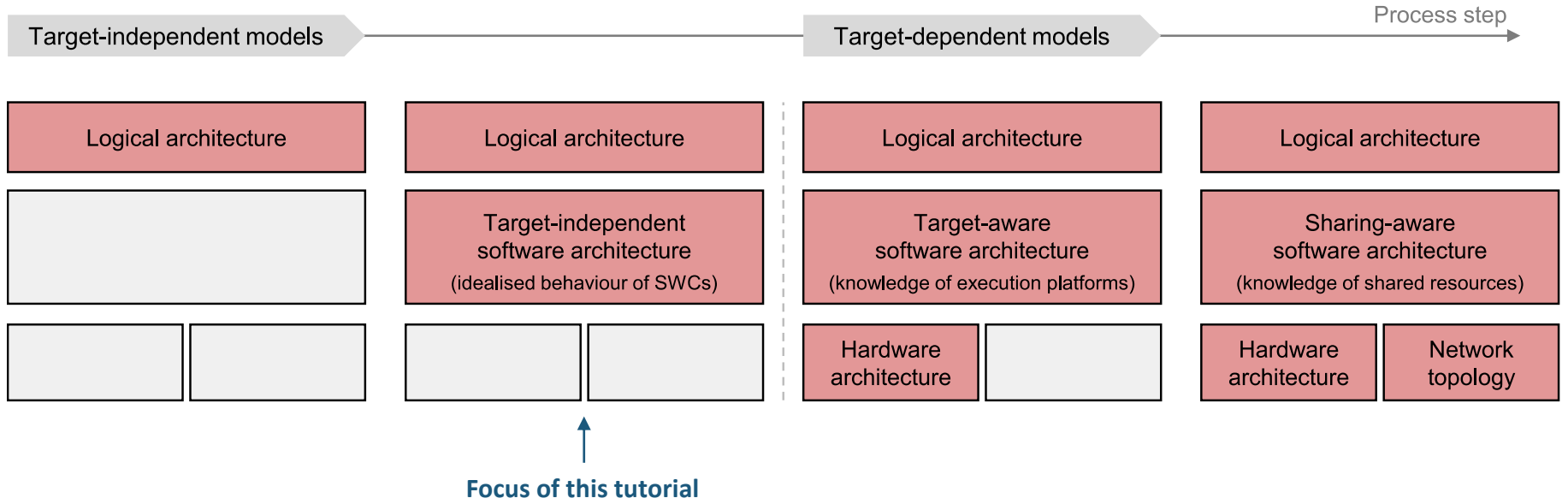


Acronyms: XbC = X-by-Construction, SWC = Software Component, HW = Hardware

[9] L. Masing, T. Dörr, et al. **"XANDAR: Exploiting the X-by-Construction Paradigm in Model-based Development of Safety-critical Systems"**, DATE '22, March 2022.

Overview of the XANDAR project

- With every iteration, the architecture model is refined or extended
- Relevant abstraction levels inspired by the PREEvision [10] layer model
- **Evolution of an architecture model** along the development process:



[10] J. Schäuffele, **"E/E architectural design and optimization using PREEvision"**, SAE 2016 World Congress and Exhibition, 2016.

Overview of the XANDAR project

- **Toolchain evaluated** in a laboratory setup with two use cases:
  - DLR ⇒ Resilient Avionic Architecture for Urban Air Mobility (UAM)
  - BMW ⇒ Autonomous Systems with Integrated Machine Learning Applications
- **Logical architecture** excerpt of DLR's pilot assistance system for UAM [11]:

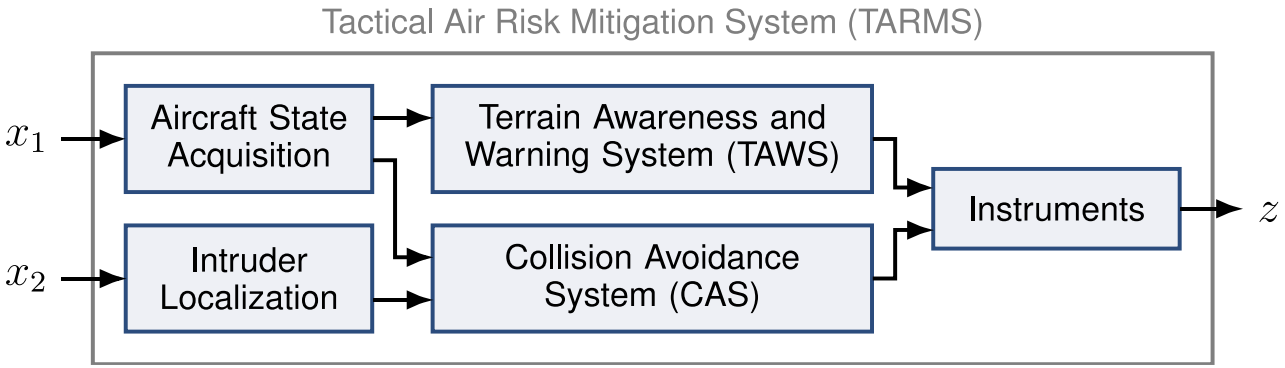
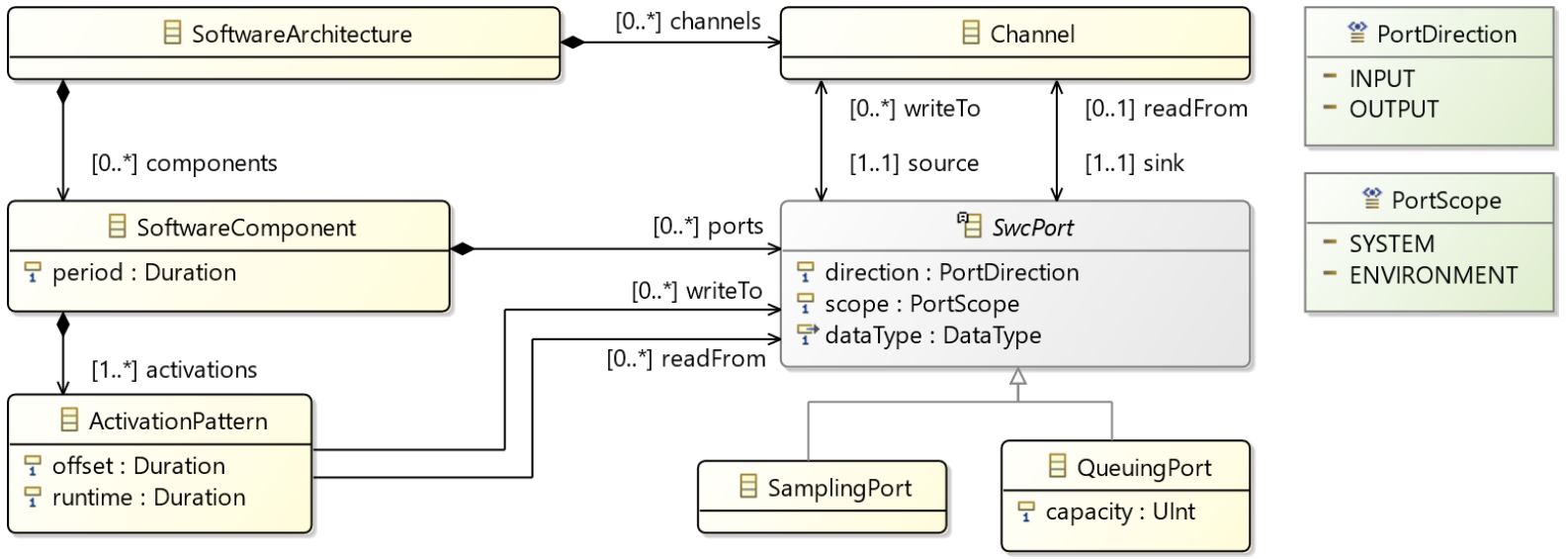


Image source: [12]

[11] J. Athavale, A. Baldovin, et al., **"Chip-Level Considerations to Enable Dependability for eVTOL and Urban Air Mobility Systems"**, DASC '20, October 2020.
[12] T. Dörr, F. Schade, et al., **"A Behavior Specification and Simulation Methodology for Embedded Real-Time Software"**, DS-RT '22, September 2022.

Software development approach

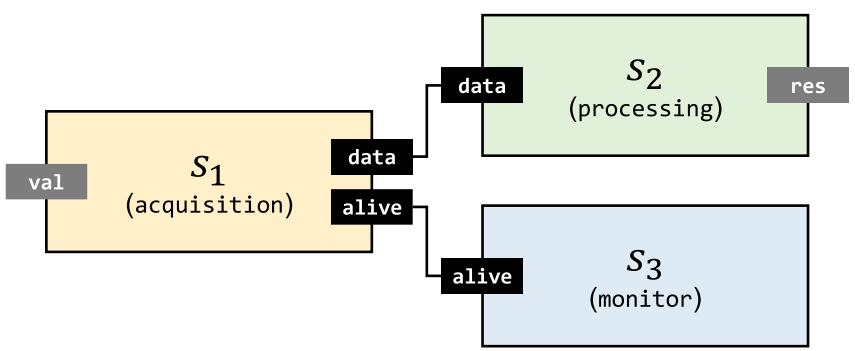
- **Description of the software architecture** by the toolchain user:
  - Software Component (SWC) entities communicating via message passing
  - Logical Execution Time (LET) parameters [13] capture the desired timing behaviour
- **Software architecture metamodel** as class diagram:



[13] T. A. Henzinger, B. Horowitz, C. M. Kirsch, **"Embedded Control Systems Development with Giotto"**, SIGPLAN Not., vol. 36, no. 8, August 2001.

Software development approach

- **Sample network of SWCs** and textual description in JSON5 [14] syntax:



JSON5 representation

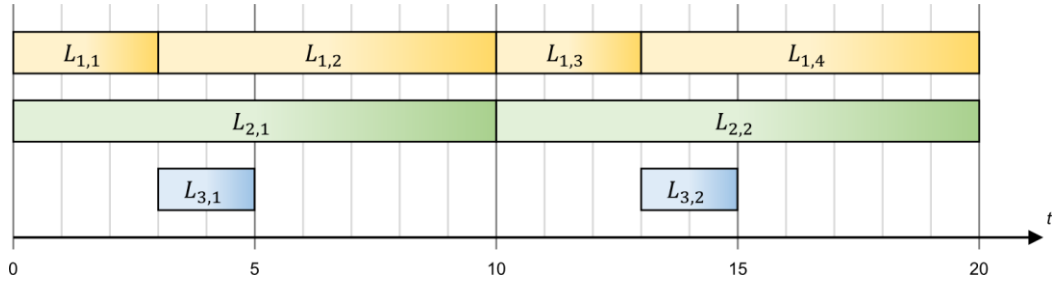
```
acquisition: {
    period: 10,
    activations: {
        check: {
            offset: 0,
            runtime: 3,
            write_to: [ 'alive' ],
        },

        gen: {
            offset: 3,
            runtime: 7,
            write_to: [ 'data' ],
        },
    },

    system_ports: {
        data: { /* … */ },
        alive: { /* … */ },
    },

    env_ports: {
        val: { /* … */ },
    }
}
```

- **LET frame sequence**:



[14] A. Kishore, J. Tucker, **"The JSON5 Data Interchange Format"**, v1.0.0, March 2018.

Software development approach

Logical Execution Time (LET) – *"LET determines the time it takes from reading program input to writing program output regardless of the time it takes to execute the program"* [15].

- **Generation** of SWC code skeletons:

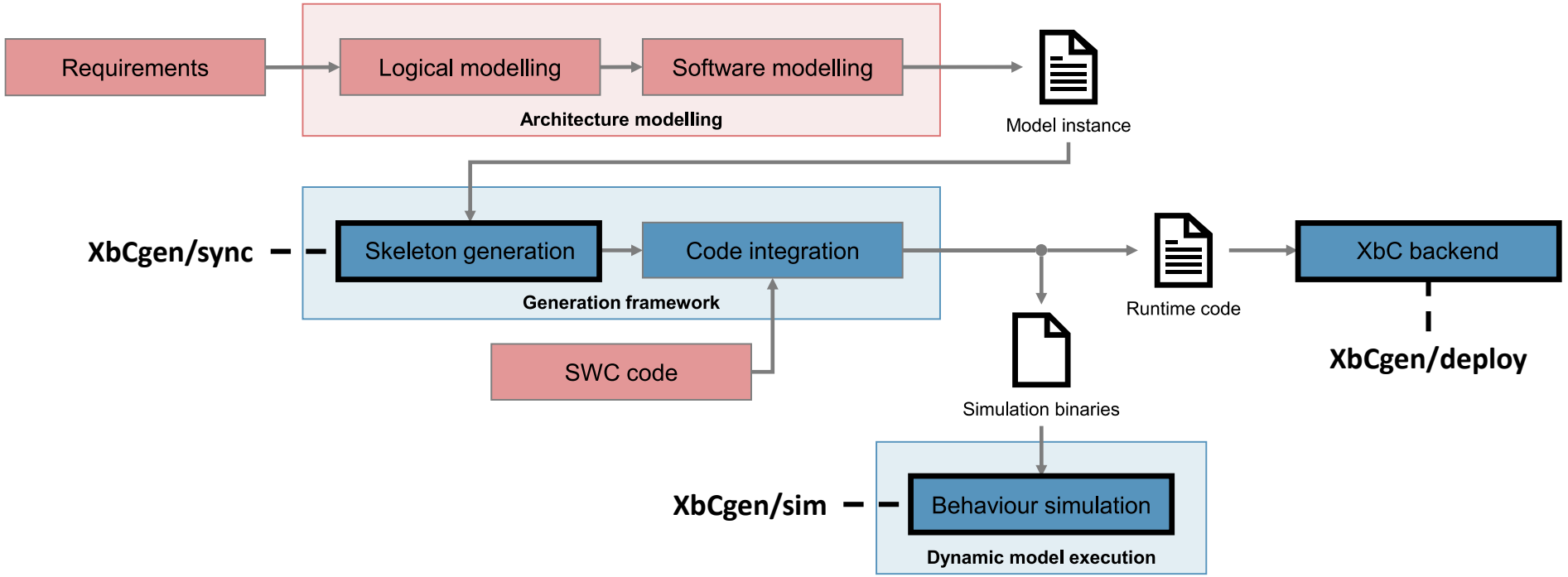
```
// Initialization hook:
void swc_init(void);
// Trigger hook:
void swc_trigger(enum activation_id activation, struct swc_port_map *port);
```

- **Access to input/output ports** via the framework-provided port map
  - Support for structured data structures (based on the Protocol Buffers [16] format)
  - Built-in support for access to input/output controllers of the hardware
- **On the target hardware:** invocation of the trigger hook for each SWC activation

[15] C. M. Kirsch, A. Sokolova, **"The Logical Execution Time Paradigm"**, Advances in Real-Time Systems, Springer, Berlin, Heidelberg, 2012.
[16] Google LLC, **"Google Developers: Protocol Buffers"**, online: https://developers.google.com/protocol-buffers, accessed 2 May 2023.

Simulation and target deployment

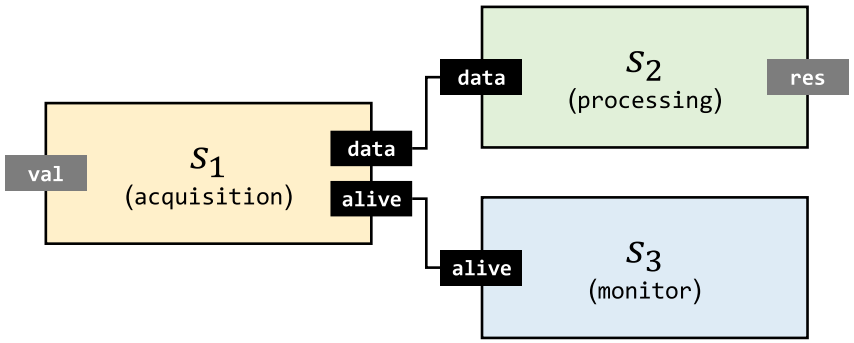
- **Starting point:** Fully deterministic specification of the software behaviour
- **Tool support** provided by the XbCgen framework of XANDAR:



Simulation and target deployment

- **XbCgen/sim:** Generation of execution traces for verification and validation
  - Discrete-event simulation with a user-provided plant/environment model (in Ptolemy II)
  - Automatic synthesis of an executable model (calling SWC simulation binaries)
  - Refer to previous work in [12] for a detailed description

- Sample excerpt of an execution trace:



⇒

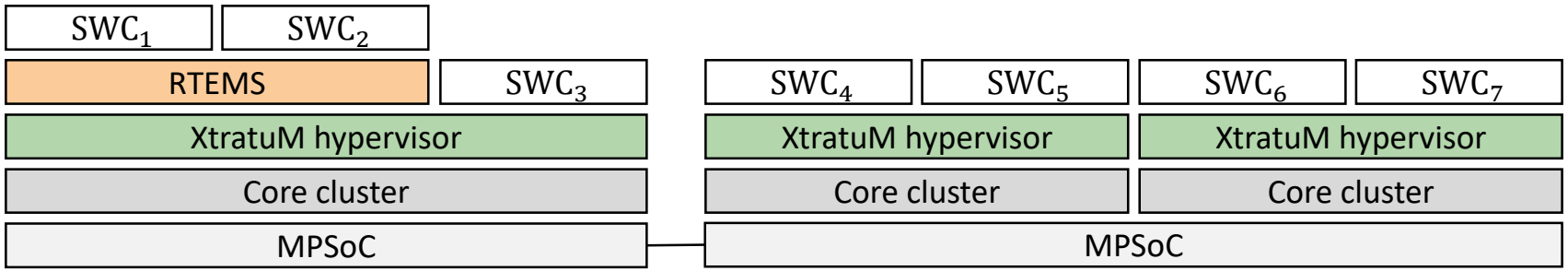
```
acquisition.gen(3..10ms)
 - data = [1566, 1564, 1563]
acquisition.check(10..13ms)
 - alive = true
monitor.main(13..15ms)
processing.main(10..20ms)
 - res = 1564.3333333333333
acquisition.gen(13..20ms)
 - data = [1558, 1557, 1555]
acquisition.check(20..23ms)
 - alive = true
monitor.main(23..25ms)
```

[12] T. Dörr, F. Schade, et al., **"A Behavior Specification and Simulation Methodology for Embedded Real-Time Software"**, DS-RT '22, September 2022.

Simulation and target deployment

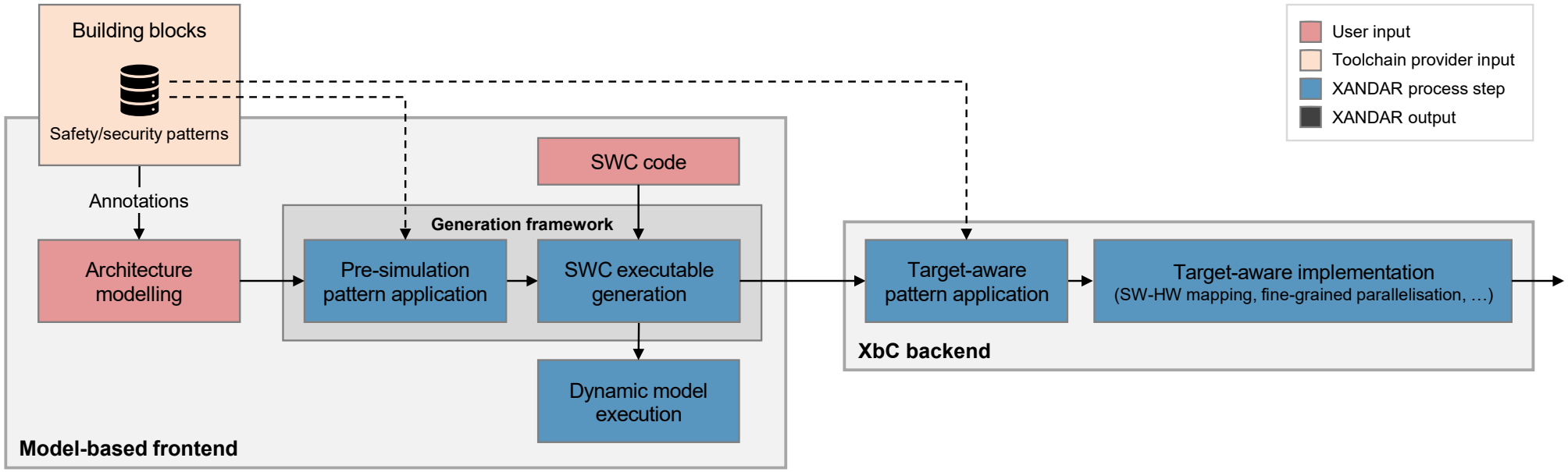
- **XbCgen/deploy:** Automatic generation of implementation artefacts for MPSoCs
- The target runtime environment consists of:
  - A hypervisor layer currently implemented using the XtratuM hypervisor
  - An optional operating system layer (RTEMS, FreeRTOS, Linux, …)

- SWCs are deployed to XtratuM partitions or OS tasks/processes, for example:



Safety pattern concept

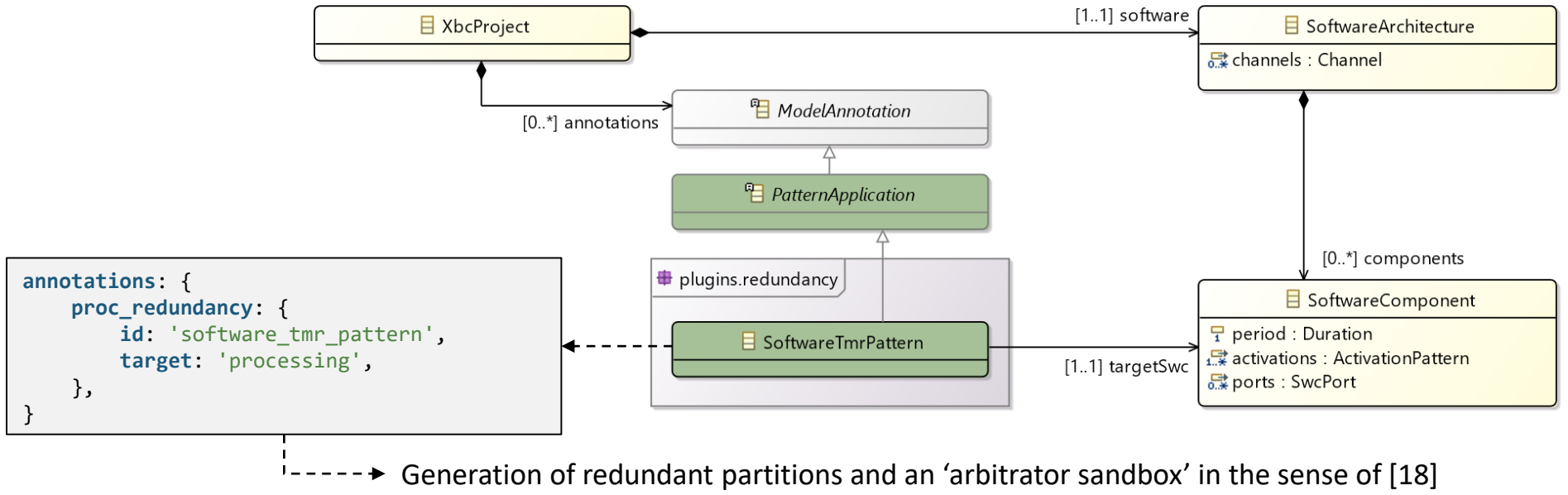
- **XbC pattern library:** Collection of verified design-time procedures for safety and security that can be annotated to system entities [17]
- Automatic implementation of the corresponding mechanisms:



[17] T. Dörr, F. Schade, et al., **"Safety by Construction: Pattern-Based Application of Safety Mechanisms in XANDAR"**, ISVLSI '22, July 2022.

Safety pattern concept

- **Pattern usage**: Explicit specification in the textual model instance (JSON5)
- **Example:** Triple Modular Redundancy (TMR) for fault tolerance

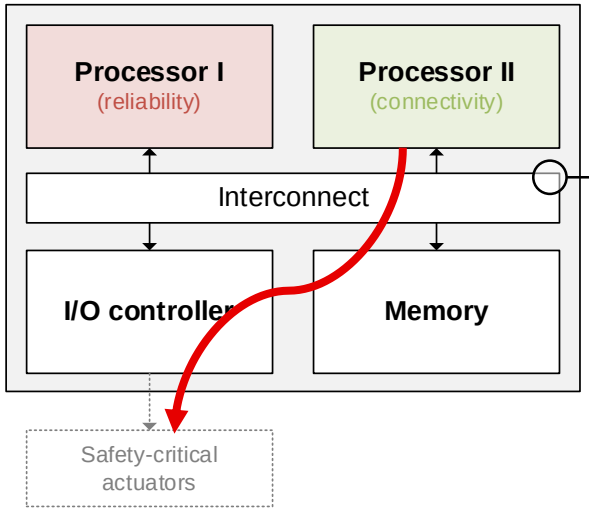


- **Next up:** Information Flow Control (IFC) + runtime logging for AI components

[18] E. Missimer, R. West, Y. Li, **“Distributed Real-Time Fault Tolerance on a Virtualized Multi-Core System”**, OSPERT '14, July 2014.

Information Flow Control (IFC) safety pattern

- **Motivation**: Unintended information flow via shared on-chip resources can lead to the violation of safety/security requirements, for example…



Access protection units (APUs) control transactions on shared interconnects. They are state-of-the art components of modern multiprocessor system-on-chip (MPSoC) devices.

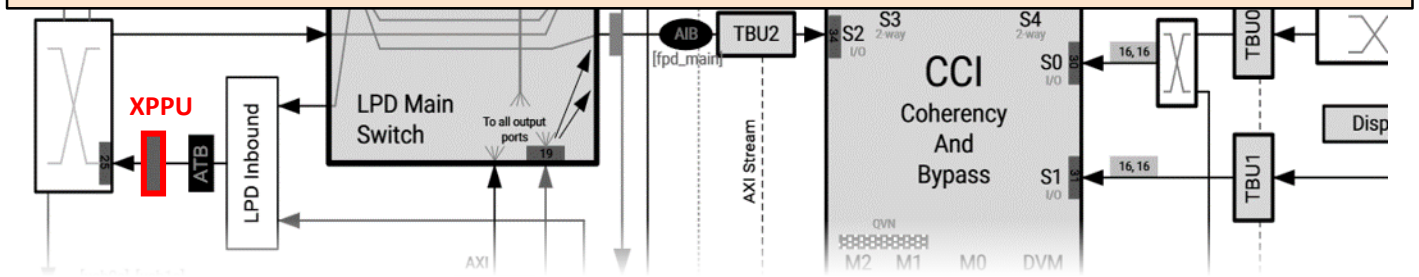


Image source: [19]

- **Concept:** Auto-generate APU configurations that are as prohibitive as possible and compare potentially feasible information flows to an accept list

[19] AMD/Xilinx, **"Zynq UltraScale+ Device: Technical Reference Manual"**, UG1085 (v2.3), September 2022.

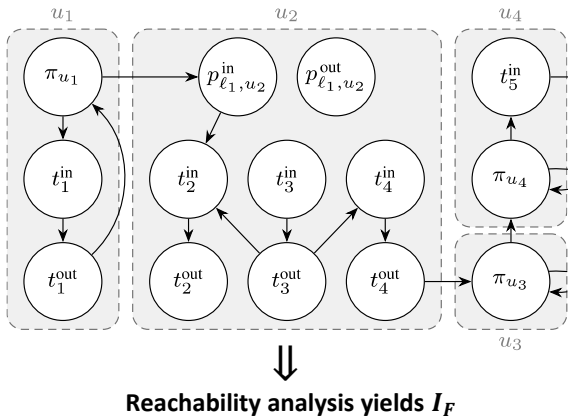
Information Flow Control (IFC) safety pattern

- **Underlying methodology** from the state of the art [20]:
  - Inputs are the system architecture and the desired information flow policy (I_A)
  - Graph-based algorithm to determine all potentially feasible information flows (I_F)

Algorithm 1 G_β construction from a model instance

1: $G_\beta \leftarrow (V, E)$, $V \leftarrow \varnothing$, $E \leftarrow \varnothing$
2: **for all** $\ell \in L : \omega_L(\ell) = 0$ **do**
3: **if** $\exists\, u \in \varphi_L(\ell) : \delta_U(u) = 0$ **then**
4: $V \leftarrow V \cup \{\pi_\ell\}$ ▷ Add link sharing node
5: **for all** $u \in U : \delta_U(u) = 0$ **do**
6: $V \leftarrow V \cup \{\pi_u\}$ ▷ Add unit sharing node
7: **for all** $\ell \in \varphi_L(u) : \omega_L(\ell) = 0$ **do**
8: $E \leftarrow E \cup \{(\pi_u, \pi_\ell), (\pi_\ell, \pi_u)\}$
9: **for all** $u \in U : \delta_U(u) = 1$ **do**
10: **for all** $\ell \in \varphi_L(u)$ **do**

⇒



- **XbC guarantee:** $\mathrm{I_F} \subseteq I_A$ holds for interactions between SWCs, ports, …

[20] T. Dörr, T. Sandmann, J. Becker, **"Model-based configuration of access protection units in networks of heterogeneous multicore processors"**, Microprocessors and Microsystems (MICPRO), vol. 87, November 2021.

Runtime logging for AI components

- Upcoming **normative and legislative constraints on AI components**:
  - Ethics of Connected and Automated Vehicles [21] by the European Commission
  - EASA Concept Paper: First usable guidance for Level 1 machine learning applications [22]

- Selected excerpts focused on the **transparency** of algorithmic decisions:

"User-centred methods and interfaces for the explainability of AI-based forms of CAV decision-making should be developed."

Source: [21]

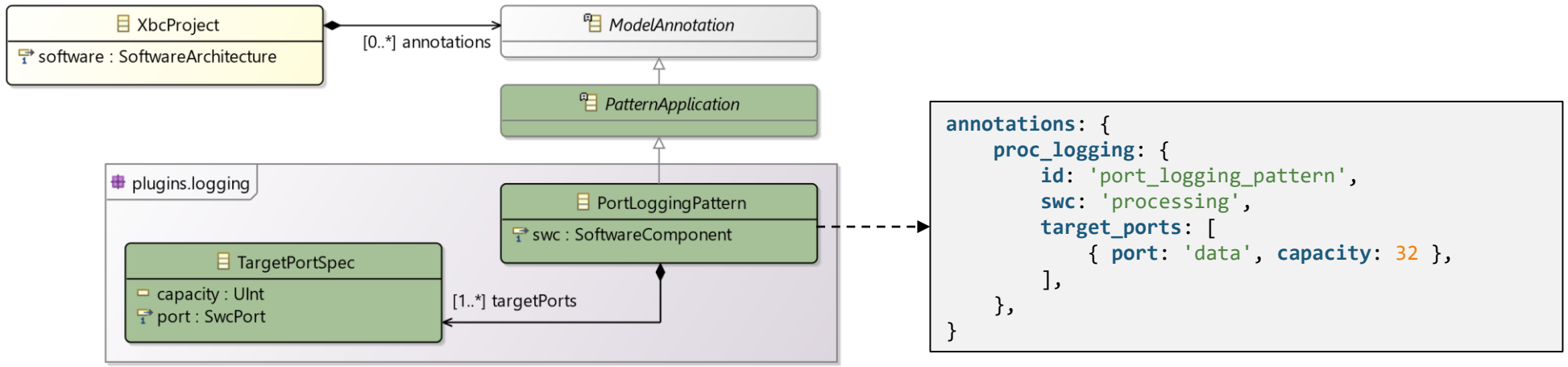
"Did you put adequate logging practices in place to record the decision(s) or recommendation(s) of the AI-based system?"

Source: [22]

[21] European Commission, **"Ethics of Connected and Automated Vehicles: recommendations on road safety, privacy, fairness, explainability and responsibility"**, 2020.
[22] European Union Aviation Safety Agency, **"EASA Concept Paper: First usable guidance for Level 1 machine learning applications"**, issue 01, 2021.

Runtime logging for AI components

- **Library-provided logging support** at the level of SWC ports:



- **XbC guarantee:** Correct synthesis and deployment of logging modules
  - Allocation of a dedicated memory region (incl. IFC pattern compatibility)
  - Non-interference with the LET-based communication model

Part II

Programming models for concurrent systems

Fundamentals

Concurrency – Two or more actions are in progress at the same time [23].
Parallelism – Two or more actions execute simultaneously [23].

Shared memory communication – Concurrent modules communicate via objects in memory [24].
Message passing – Concurrent modules communicate by transmitting messages over a channel [24].

- Cyber-physical systems (CPS) are inherently concurrent [25]
  ⇒ Approaches to achieve **deterministic concurrency** are essential
- Incorrect data sharing is a common source of concurrency issues [26]
  ⇒ Techniques based on **message passing particularly interesting** for CPS applications

[23] C. Breshears, **"The Art of Concurrency: A Thread Monkey's Guide to Writing Parallel Applications"**, O'Reilly Media, Inc., 2009.
[24] R. Miller, M. Goldman, 6.031 — Software Construction, Reading 21: Concurrency, online: https://web.mit.edu/6.031/www/sp22/classes/21-concurrency, 2022.
[25] P. Derler, E. A. Lee, A. Sangiovanni Vincentelli, **"Modeling Cyber-Physical Systems"**, Proceedings of the IEEE, vol. 100, no. 1, January 2012.
[26] P. Koopman, **"A Case Study of Toyota Unintended Acceleration and Software Safety"**, online: https://betterembsw.blogspot.com/2014/09/a-case-study-of-toyota-unintended.html, 2014.

Kahn Process Networks (KPN)

- **Idea:** simple language for parallel programming [27]
  - Processes (*computing stations*) = sequential programs with Algol-like syntax
  - Channels (*communication lines*) = unbounded first-in-first-out queues
  - Asynchronous message passing (nonblocking writes and blocking reads)

- KPN programs are deterministic [28]:

  "Kahn showed that concurrent execution was possible without nondeterminism"

- Expressivity deliberately limited

```
Begin
(1) Integer channel X, Y, Z, T1, T2 ;
(2) Process f(integer  in U,V; integer out W) ;
    Begin integer I ; logical B ;
          B := true ;
          Repeat Begin
(4)         I := if B then wait(U) else wait(V) ;
(7)         print (I) ;
(5)         send I on W ;
            B := ¬B ;
            end ;
    End ;
  Process g(integer in U ; integer out V, W) ;
    Begin integer I ; logical B ;
      B := true ;
      Repeat Begin
```

Image source: [27]

[27] G. Kahn, **"The semantics of a simple language for parallel programming"**, IFIP Congress, 1974.
[28] C. Ptolemaeus (editor), **"System Design, Modeling, and Simulation using Ptolemy II"**, Ptolemy.org, 2014.

Communicating Sequential Processes (CSP)

- **Key principle:** parallel composition of sequential processes [29]
  - Introduced to coordinate/synchronise multiprocessor machine communication
  - Synchronous message passing (blocking writes and blocking reads)

- Part of programming languages such as occam-π [30] and Go [31]:

```
func main() {
    exit := make(chan bool)
    channel := make(chan uint8)

    go consumer(channel, exit)
    go producer(channel)
    <-exit
}
```

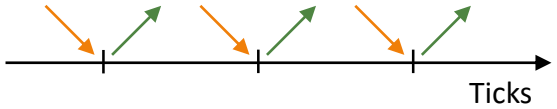
```
func consumer(in <-chan uint8, exit chan<- bool) {
    fmt.Println(<-in)
    exit <- true
}
```

```
func producer(out chan<- uint8) {
    time.Sleep(time.Second)
    out <- 42
}
```


[29] C. A. R. Hoare, **"Communicating Sequential Processes"**, Communications of the ACM, vol. 21, no. 8, 1978.
[30] P. H. Welch, F. R. M. Barnes, **"Communicating Mobile Processes: Introducing occam-pi"**, Communicating Sequential Processes: The First 25 Years, Springer, Berlin, Heidelberg, 2005.
[31] J. Whitney, C. Gifford, M. Pantoja, **"Distributed execution of communicating sequential process-style concurrency: Golang case study"**, Journal of Supercomputing, vol. 75, 2019.

Synchronous languages

- **Key principle:** combination of synchrony and concurrency [32]
  - Examples are Lustre [33] or more recent approaches such as Blech [34]
  - Immediate reactions (simultaneous reads and writes)
  - Temporal semantics based on a global clock

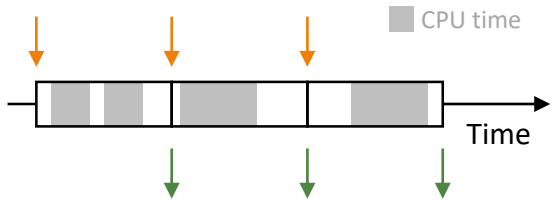


- Strong foundation for formal verification
- Widespread use for safety-critical system design:
  - Lustre is a fundamental part of the SCADE toolset by Esterel Technologies
  - Applications include nuclear power plants and flight control software [32]

[32] A. Benveniste, P. Caspi, et al., **"The synchronous languages 12 years later"**, Proceedings of the IEEE, vol. 91, no. 1, January 2003.
[33] N. Halbwachs, P. Caspi, et al., **"The synchronous data flow programming language LUSTRE"**, Proceedings of the IEEE, vol. 79, no. 9, September 1991.
[34] F. Gretz, F.-J. Grosch, **"Blech, Imperative Synchronous Programming!"**, FDL '18, September 2018.

Logical Execution Time (LET)

- **Origin:** Giotto methodology for embedded control system design [13]
  - Periodic invocation of sequential tasks ⇒ time-triggered framework
  - Reading of input ports at the invocation time
  - Writing of output ports at the end of the period



- Other manifestations of the LET abstraction:
  - xGiotto for event-driven programming [35]
  - Timing Definition Language (TDL), cf. [36]
  - System-level LET for distributed systems [37]
  - The synchronous LET paradigm [38]

[13] T. A. Henzinger, B. Horowitz, C. M. Kirsch, **"Embedded Control Systems Development with Giotto"**, SIGPLAN Not., vol. 36, no. 8, August 2001.
[35] A. Ghosal, T. A. Henzinger, et al., **"Event-Driven Programming with Logical Execution Times"**, Hybrid Systems: Computation and Control, HSCC '04, Springer, Berlin, Heidelberg, 2004.
[36] W. Pree, J. Templ, **"Modeling with the Timing Definition Language (TDL)"**, ASWSD '06, Springer, Berlin, Heidelberg, 2006.
[37] R. Ernst, L. Ahrendts, K.-B. Gemlau, **"System Level LET: Mastering Cause-Effect Chains in Distributed Systems"**, IECON '18, October 2018.
[38] F. Siron, D. Potop-Butucaru, et al., **"The synchronous Logical Execution Time paradigm"**, ERTS '22, June 2022, hal-03694950.

Reactors and Lingua Franca (LF)

- **Concept:** reactors as coordination model for concurrent systems [39]
  - A reactor is composed of **reactions** in a target language (C, Python, Rust, …)
  - Reactions are executed in response to **trigger events** (carrying a value and a tag)
- Lingua Franca (LF) for the development of reactor-based systems:

```
target C;
main reactor Timer {
    timer t(0, 1 sec);
    reaction(t) {=
        printf("Timer expired!\n");
    =}
}
```

"Lingua Franca is a polyglot coordination language for reactive, concurrent, and time-sensitive applications."

www.lf-lang.org

[39] M. Lohstroh, C. Menard, et al. **"Toward a Lingua Franca for Deterministic Concurrent Systems"**, ACM Transactions on Embedded Computing Systems, vol. 20, no. 4, May 2021.

Applicability to the XANDAR toolchain

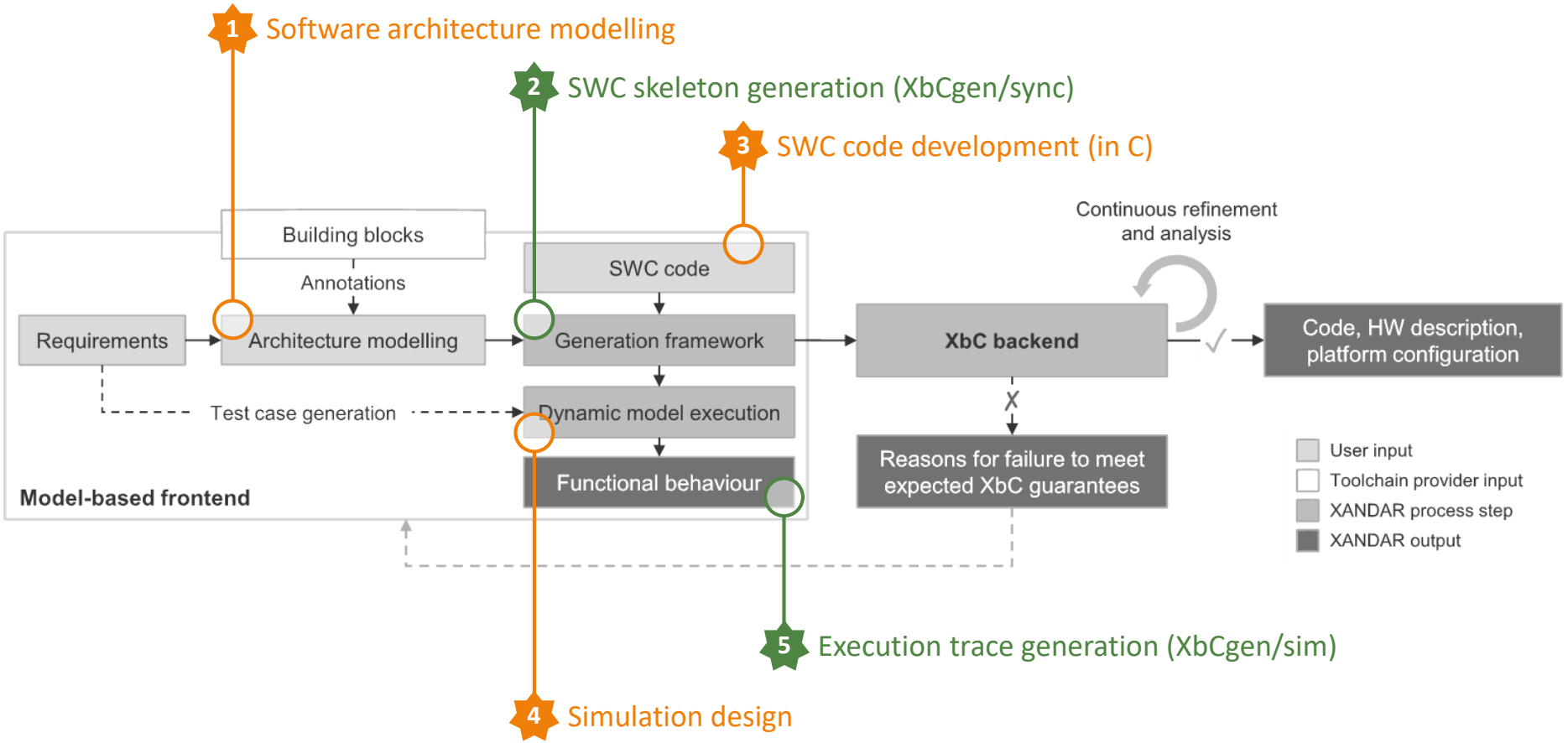
- Compatibility with the full feature set of XANDAR is essential
- **LET has been shown to offer a promising trade-off**
  - Straightforward integration into the overall framework
  - Compatible with the requirements of XANDAR's use cases
- Relaxation of current constraints is future work
- Ideas for steps beyond the current model:
  - Integration of LF, which is a generalisation of LET [40]
  - Add support for the synchronous LET paradigm [38]

[38] F. Siron, D. Potop-Butucaru, et al., **"The synchronous Logical Execution Time paradigm"**, ERTS '22, June 2022, hal-03694950.
[40] E. A. Lee, M. Lohstroh, **"Generalizing Logical Execution Time"**, Principles of Systems Design - Essays Dedicated to Thomas A. Henzinger on the Occasion of His 60th Birthday, July 2023.

Part III

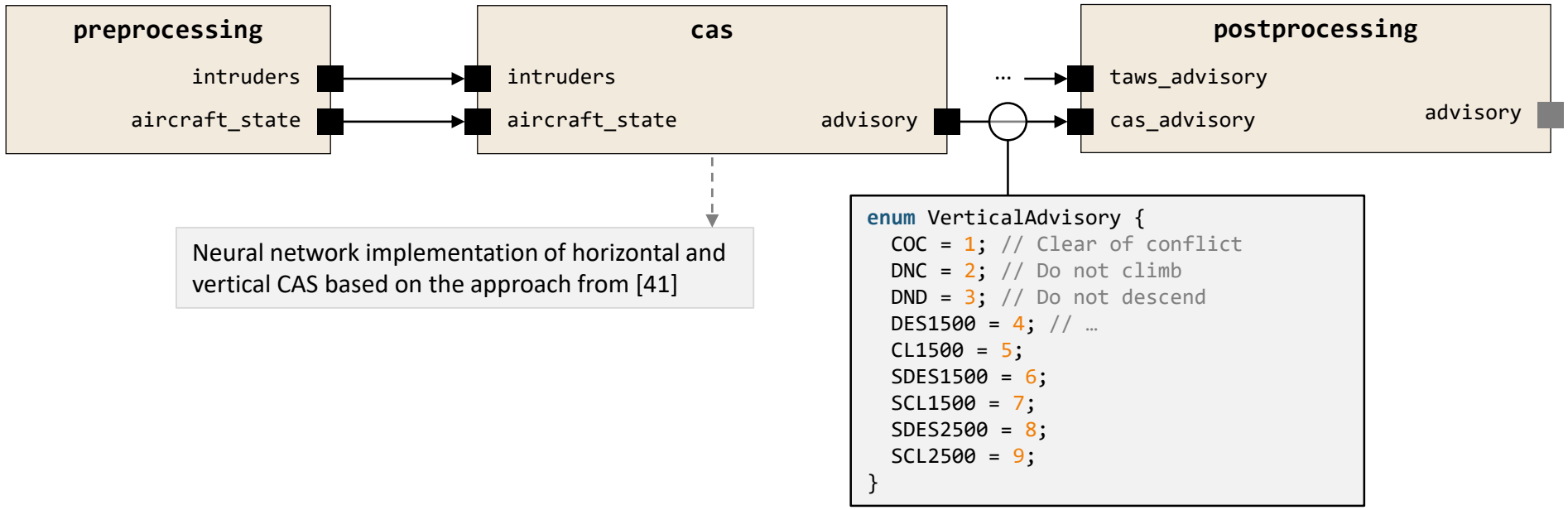
Live demonstration of XbCgen

Overview and introductory examples



Application to an avionics use case

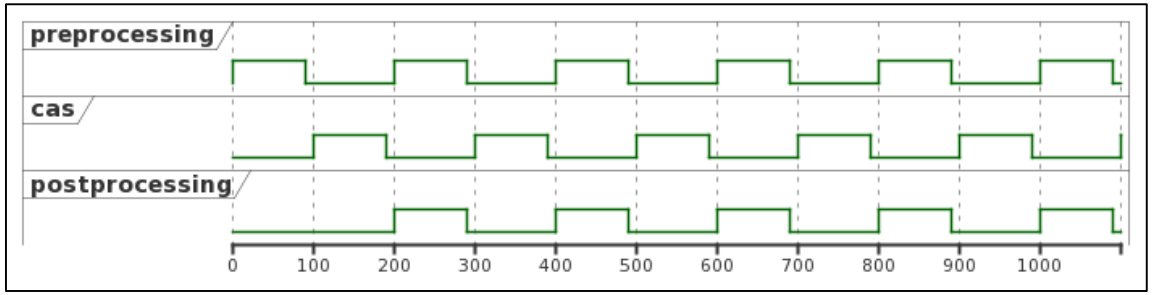
- SWC chain from the **Tactical Air Risk Mitigation System (TARMS)** by DLR:



[41] K. D. Julian, M. J. Kochenderfer, **"Guaranteeing Safety for Neural Network-Based Aircraft Collision Avoidance Systems"**, DASC '19, September 2019.

Application to an avionics use case

- Sample activation pattern specification:



- FlightGear [42] scenarios serve as simulation stimuli:

⇒

```
struct scenario_trace_item scenario_trace[] = {
    { 3,63.99467872,-22.65,500,77.74823761,-0,180,0,0,63.
    { 3.51667,63.99381768,-22.65,500,77.75377655,-0,180,0
    { 4.05,63.9936291,-22.65,500,77.75377655,-0,180,0,0,6
    { 4.55833,63.99344053,-22.65,500,77.75377655,-0,180,0
    { 5.025,63.99327291,-22.65,500,77.75377655,-0,180,0,0
    { 5.50833,63.99310828,-22.65,500,77.75377655,-0,180,0
    { 6.03333,63.99291073,-22.65,500,77.75377655,-0,180,0
    { 6.55,63.99272215,-22.65,500,77.75377655,-0,180,0,0,
    { 7.00833,63.99256052,-22.65,500,77.75377655,-0,180,0
    { 7.53333,63.99237195,-22.65,500,77.75377655,-0,180,0
    { 8.05,63.99218637,-22.65,500,77.75377655,-0,180,0,0,
    { 8.55833,63.99200977,-22.65,500,77.75377655,-0,180,0
```

[42] FlightGear Flight Simulator website, online: www.flightgear.org, accessed 2 May 2023.

Closing

Summary

Summary

- **Software development methodology of the XANDAR toolchain:**
  - Software modelled as a network of SWCs with LET parameters
  - Pre-verified safety/security patterns are annotated by the toolchain user
  - Behaviour simulation + automatic deployment to a type-1 hypervisor on MPSoCs
- **Programming models for concurrent systems:**
  - LET has been shown to be a suitable model for the XANDAR framework
  - Relaxation of current constraints (e.g. by moving to LF) is future work
- **Live demonstration of XbCgen:**
  - Flow from modelling to execution trace generation
  - Application to the DLR use case