

SafeDSE: Joint Exploration of Safety Architecture Patterns, Mapping, and Scheduling for Dataflow Applications With Safety Arguments (Extended Version)

Fatemeh Bahrami and Seyed-Hosein Attarzadeh-Niaki

Abstract

Embedded platforms increasingly host safety-related streaming functions next to less critical ones. Whether a safety architecture pattern, such as a doer/checker pair, protects a function depends on the cores that run its components. Their schedule must meet the throughput requirement, and mixing safety integrity levels (SILs) on one core raises development cost. This report presents SafeDSE, a design space exploration framework for synchronous dataflow applications. One declarative constraint model decides the pattern of each actor, the placement that the fault model requires, the static-order schedule, and the platform, and prices SIL promotion. Guards activate the chosen patterns in a fixed graph that holds all of them, and node potentials encode throughput. An independent verifier re-checks every solution, and a generator turns its checks into evidence of a safety argument in Goal Structuring Notation. On 9 benchmark instances under period bounds, no rule that fixes patterns before mapping was both always feasible and never more expensive under conservative pricing, while joint exploration was. In the generated arguments, 49 to 62 % of the leaves cite verifier checks; the rest are claims a mapping cannot discharge. Optimality within the model was proven for all synthetic graphs with up to 16 actors. This extended version adds the full model, the verifier and generator in detail, the complete catalog and parameters, and a proof of the potential characterization. It also reports every experiment per instance, a mutation analysis of the verifier, and the negative results of the development.

Index Terms

Assurance cases, constraint programming, design space exploration, functional safety, mixed criticality, multiprocessor scheduling, safety architecture patterns, synchronous dataflow.

I. INTRODUCTION

EMBEDDED systems increasingly integrate functions of different criticality on one multiprocessor platform [1]. Streaming functions, which process images, audio, or sensor data, are often specified as synchronous dataflow (SDF) graphs [2] and must sustain a throughput. Under IEC 61508, each safety function has a safety integrity level (SIL), and the techniques and measures required for its software become stricter with the SIL [3], [4]. Experts estimated that moving from SIL 2 to SIL 3 may double the development cost [4]. To integrate such a function, a designer chooses a safety architecture pattern (e.g., a doer/checker pair or a 2-of-2 pair), the cores that run its components, a schedule that meets the throughput requirement, and the cores to buy.

This technical report is the extended version of a shorter paper that has been submitted for publication. It adds the material that the page limit of the shorter version leaves out: full model and verifier details, the complete pattern catalog and parameters, the proof of the potential characterization, all measured results per instance, and the negative results of the development.

The authors are with the Faculty of Computer Science and Engineering, Shahid Beheshti University, Tehran, Iran (e-mail: farnoosh3@gmail.com; h_attarzadeh@sbu.ac.ir). Corresponding author: Seyed-Hosein Attarzadeh-Niaki.

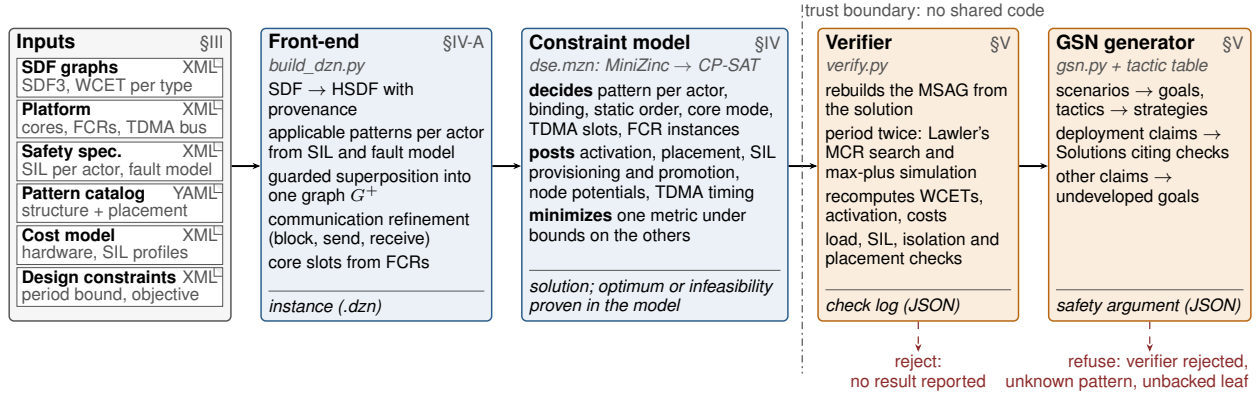


Fig. 1. SafeDSE design flow. The front-end turns all design alternatives into one fixed constraint instance; the model decides and proves the optimum within the model. Only checks of an independent verifier, which shares no code with the model, become evidence in the generated argument; dashed arrows are rejections and refusals.

Each part of this problem has been studied. Design space exploration (DSE) with constraint programming (CP) maps and schedules SDF graphs on multiprocessors under throughput constraints and proves the result optimal [5]–[7], or proves that no mapping meets the requirements [8]. Safety architecture patterns have been cataloged with their structure, tactics, and costs [9]–[11]. Optimization methods have allocated and decomposed SILs to reduce development cost [12]–[14], added redundancy for reliability [15], [16], and selected architectures from templates [17].

However, the choice of a pattern and its deployment depend on each other. A checker covers random hardware faults of its doer's core only if it runs on another core; on the same core, it covers only systematic faults (Section III-B). Each checker or replica adds execution time and often a cycle to the dataflow graph, so a pattern that meets the throughput requirement under one mapping can miss it under another. Actors of different SILs on one core without certified partitioning promote the lower-SIL ones, so saving cores costs development effort. DSE tools take redundancy as an input, and catalogs describe patterns without a platform. The methods that combine both search heuristically [12], [13], [16], separate redundant parts by a fixed rule [14], or choose patterns without a timing model [18]. Safety arguments are built for a given design [19], [20], and a DSE tool delivers no evidence for the claims of the argument. It therefore remains unclear whether pattern choice, placement, scheduling, and SIL cost can be decided in one exact search, and whether the chosen design can carry its own safety evidence.

This report presents SafeDSE, a DSE framework that takes these decisions in one declarative constraint model (Fig. 1). A front-end places every applicable pattern, every communication actor that a remote channel may need, and every core that can be bought in one fixed graph. A MiniZinc [21] model, solved with CP-SAT [22], chooses the pattern of each actor, the core and static order of each node, and the platform, and proves the result optimal within the model. A verifier that shares no code with the model re-checks each solution, and a generator turns the checks into evidence in a safety argument in Goal Structuring Notation (GSN) [23]. On 9 benchmark instances under period bounds, no rule that fixes the pattern before mapping was both always feasible and never more expensive under our conservative pricing. With a lower-SIL doer and priced diverse development, the cost-aware rule reached the joint optimum. Joint exploration was feasible and optimal in all cases.

The contributions are:

- C1) Guarded superposition: patterns with their explicit voters, communication refinement, and platform instances become guarded parts of one fixed graph, so pattern choice, replica placement, static-order scheduling, and platform selection form one CP problem (Section IV-A).
- C2) A placement-aware safety model: a pattern is a structure plus placement relations tied to fault

classes. Each core is provisioned to a SIL under the IEC 61508 rule on mixing SILs or certified partitioning, and a cost profile prices SIL promotion (Sections III-B and IV-C).

- C3) Throughput on a guarded graph: the potential form of cyclic scheduling [24] is posted on the mapping- and scheduling-aware graph under reified binding, order, and activation conditions. A CP solver thus decides mapping, static orders, and period together without a throughput propagator. The encoding excludes deadlock and couples applications that share a core to one period (Section IV-E).
- C4) Evidence-backed safety arguments: an independent verifier computes each period by two disjoint methods and logs every check. A GSN generator cites these records for the deployment claims and refuses to emit an argument with an unsupported leaf (Section V).

Relation to the shorter version. This report extends a shorter paper that has a page limit. It keeps the structure of that paper and restores what was cut. Section II gives the related work in more detail. Section III adds the complete catalog with its scenarios and the cost profiles. Section IV describes every constraint family of the model, including communication timing, the static order, the implied constraints, and the exact variant of the period constraints, and Section A proves the potential characterization in full. Section V lists every verifier record, the tactic table, and the refusal conditions of the generator. Section VI reports every experiment per instance, including the head-to-head with DeSyDe, input variants, a cost-model extension, model size, cross-solver runs, and a mutation analysis of the verifier. Section VII collects the negative results and the defects that the evaluation found in the tool, and Sections B and C list the instances, parameters, and the scripts that produce each table and figure. All numbers come from the same measured data as the shorter paper. The model and the tools are available at <https://github.com/SBU-CPS-Lab/SafeDSE>, and the scripts and data of all experiments at <https://github.com/SBU-CPS-Lab/SafeDSE-experiments>.

II. RELATED WORK

A. Dataflow Mapping and Constraint-Based DSE

Throughput-constrained mapping of SDF graphs has been studied with heuristics and with exact methods. Stuijk et al. [25] allocated resources heuristically on top of exact throughput analysis [26]. Bonfietti et al. posted a throughput constraint for SDF graphs in a CP solver [27] and later mapped and scheduled SDF graphs for maximum throughput with complete CP [5]. DeSyDe [7], [28] extended CP-based exploration to heterogeneous platforms with time-division multiple access (TDMA) communication and to trade-offs of throughput against buffer size and power. Rosvall et al. [6] added throughput propagators and hard constraints for some applications next to best-effort ones. IDeSyDe [29] makes such models modular, and Jordão et al. [8] used it to map avionics functions to integrated modular avionics (IMA) platforms. Their CP model minimizes the number of processing elements and either finds a mapping that meets all requirements or proves that none exists. The redundant part of a flight function is an input. Since the model is optimistic about partitioning, an infeasibility carries over to partitioned designs.

Cyclic scheduling characterizes periodic schedules by node potentials [24]. Hanen and Munier showed that the least period of a uniform graph is the ratio of its critical circuit. Fixing a cyclic order of the tasks on each processor adds uniform constraints (the tie-breaking graph) whose critical circuit gives the best schedule for that order. This construction is the analogue of the static-order and wrap edges of the mapping- and scheduling-aware graph (MSAG) used in this report. Govindarajan et al. [30] put Reiter's characterization into one integer linear program that schedules loops and maps them to function units.

Reliability-aware synthesis adds redundancy during the search. Izosimov et al. [31] chose re-execution and replication for fault-tolerant distributed systems under time and cost constraints. Glaß et al. [15] and Kang et al. [16] replicated or re-executed tasks, and Bolchini and Miele [32] hardened tasks by duplication with comparison or triple modular redundancy. These works search evolutionarily against transient faults. Kang et al. also drop low-criticality task graphs under faults, use passive replication,

derive response-time bounds, and search with SPEA-II. In the CP line, the redundancy of an application is an input; in the reliability line, the search is heuristic and certification cost is not modeled. SafeDSE keeps exact CP search and the MSAG of DeSyDe, makes the pattern a decision, and posts the potential form on the MSAG under reified conditions instead of a throughput propagator (Section IV-E).

B. Safety Patterns, SIL Cost, and Mixed-Criticality Optimization

Pattern catalogs describe safety architectures with their effect on safety, reliability, and cost [9], [10]. Douglass [33] described, among others, the Monitor-Actuator, Homogeneous Redundancy, Heterogeneous Redundancy, and triple modular redundancy patterns, and Koopman and Wagner [11] explained how a high-SIL monitor lets a low-SIL actuator carry a high-SIL function. The designer applies these patterns by hand. Mixed-criticality scheduling, which started from Vestal’s criticality-dependent WCETs [34], solves another problem: it guarantees timing at several levels of assurance [1] and does not choose redundancy. In this report, “different criticality” means different SILs, not Vestal’s sense.

The closest work is by Tămaş-Selicean and Pop [12], [35]. They mapped tasks of different SILs to a partitioned architecture with static-cyclic schedules, assigned tasks to partitions, and sized the partition time slots. Tasks that share a partition are elevated to the highest SIL among them, and high-SIL tasks are decomposed into redundant lower-SIL tasks from a library taken from ISO 26262-9, connected through distributor and collector tasks. They minimized development cost with Tabu Search and cited a certification overhead of 25 to 100 %. Fault tolerance is outside their problem. Our promotion rule (Section III-B) uses the same certification argument, applied per core. HiP-HOPS synthesizes fault trees from architecture models [36], and Parker et al. [13] decomposed and allocated SILs on them with a genetic algorithm and a SIL-dependent cost. Zaheri and Reuss [14] solved the decomposition of automotive SILs (ASILs), mapping, and non-preemptive scheduling as one mixed-integer linear program (MILP) that minimizes development cost and chain latency. Redundant subtasks never share an electronic control unit, by a fixed constraint. ArchEx [17] selected a minimum-cost architecture from a template by MILP under a reliability bound and an additive latency bound. It has an algorithm that is optimal within the error of an approximate reliability algebra and a lazy one that is sound but possibly sub-optimal; its patterns are requirement patterns that generate MILP constraints, not architecture patterns. SafPat [18], implemented in answer set programming and validated by simulated fault injection, recommends patterns, including the hardware separation of replicas, from the fault type. The deployment of the extended architecture is then optimized with the SMT-based synthesis of AutoFOCUS3 [37], which minimizes safety-critical coupling between control units. AutoFOCUS3 also respects core SIL limits with time-triggered schedules, with a binary search on the schedule length over an SMT solver [38]. It places given master and standby replicas on nodes with different power supplies, choosing hot or cold standby from the fault-tolerance and recovery times, without optimization [39]. As Table I shows, none of these works decides the pattern, its placement for a given fault model, and a throughput-feasible schedule in one exact search.

C. Model-Based Safety Argument Generation

GSN [23] is the common notation for safety arguments, and Preschern et al. [10] gave a GSN argument for each pattern. Hawkins et al. [19] linked argument elements to design models so that argument parts can be generated from them. AdvoCATE [20] supports the creation of arguments, the integration of formal methods, the instantiation of argument patterns from structured data, abstraction, queries, and the verification of the argument structure. Resolute [40] discharges claims with analyses of an AADL architecture model through rules written in a domain-specific language, so that the evidence is computed. These tools take the design as given. Assurance 2.0 [41] asks for deductive argument steps, a search for defeaters, and an assessment of confidence. In SafeDSE, the argument describes the design that the exploration chose, and each deployment claim cites a check of an independent verifier (Section V); the search for defeaters stays with the engineer.

TABLE I
COMPARISON WITH RELATED WORK

Approach	Thr.	Exact	Pat.	Place.	Cost	Arg.
DeSyDe [7], [28]	✓	✓	✗	✗	✗	✗
Bonfietti et al. [5], [27]	✓	✓	✗	✗	✗	✗
Jordão et al. [8]	○	✓	✗	✗	✗	✗
Kang et al. [16]	○	✗	○	○	✗	✗
Tămaş-Selicean, Pop [12]	○	✗	○	✗	✓	✗
Parker et al. [13]	✗	✗	○	✗	✓	✗
Zaheri, Reuss [14]	○	✓	○	○	✓	✗
ArchEx [17]	○	✓	○	✗	✗	✗
AutoFOCUS3 [38], [39]	○	✓	○	○	✗	✗
SafPat [18], [37]	✗	○	✓	✓	✗	✗
AdvOCATE [20]	✗	✗	✗	✗	✗	✓
SafeDSE	✓	✓	✓	✓	✓	✓

Columns: the decisions and outputs of the problem in Section III-C. Thr.: ✓ SDF throughput, ○ deadline or latency. Exact: optimum proven, ○ for the deployment only. Pat.: ✓ catalog patterns chosen, ○ redundancy or SIL decomposition only. Place.: ✓ separation from the fault model, ○ fixed rule. Cost: SIL-dependent development cost. Arg.: safety argument produced.

III. PRELIMINARIES AND PROBLEM FORMULATION

A. Application and Platform Models

An application is a synchronous dataflow (SDF) graph [2]: actors fire with fixed rates on channels that may hold initial tokens. Like DeSyDe [7], we map and schedule the unfolded homogeneous SDF (HSDF) graph, in which every node fires once per iteration and records its SDF actor and copy index. Each core runs its nodes self-timed in a static order.

Definition 1 (Period): Let G be an HSDF graph with execution time T_i of node i and tok_{ij} initial tokens on edge (i, j) . The cycle ratio of a cycle C is $\sum_{i \in C} T_i / \sum_{(i,j) \in C} tok_{ij}$, and $MCR(G)$ is the maximum cycle ratio over all cycles of G . The period of G is $\mu = MCR(G)$ and its throughput is $1/\mu$ [26].

The platform offers core types. A core type has a worst-case execution time (WCET) per actor and operating mode, an integer in the benchmark's time unit. A missing entry forbids the binding: without a timing figure, the binding is not certified. It also has the highest SIL to which software on it can be certified, a memory size, and optionally certified partitioning at a cost. Each mode has a price and a power. Cores are bought as cards, each an instance of a card template. The cores on a card share power and clock, so a card is a fault containment region (FCR). Remote channels use a TDMA bus whose slots go to the sending cores [7]. Fig. 2a shows the running example and the FCR templates of the platform, and Table II lists all card templates of the evaluation.

B. Safety Model

The safety specification assigns each SDF actor a a required SIL $SIL_a^{\text{req}} \in \{0, \dots, 4\}$ from a hazard and risk analysis (SIL 0: no requirement) [3]. It fixes a fault model: random hardware faults (H), systematic faults (S) that differ between core types and their toolchains or between algorithms, or both. Design faults of the application software that reappear in every implementation are not in S; they need diverse software development, which SafeDSE does not establish.

A safety architecture pattern [9]–[11], [33] adds components, with edges and initial tokens, to the actor it protects, the owner. The edge that returns a verdict to the owner holds one token, so the fault reaction starts within one period. Each component fires once per iteration and produces on all its outputs; a detected fault changes token values, not the presence of tokens, so the timing analysis stays valid under a fault. Placement relations link the roles by SAME or DIFFERENT core, DIFFERENT_FCR, or DIVERSE (different core types), mostly per fault class. On a platform with a single core type, DIVERSE cannot hold, so heterogeneity is a precondition for any defense against systematic faults.

TABLE II
CARD TEMPLATES OF THE PLATFORMS IN THE EVALUATION

Platform file	Card	Core type	Cores	Max.	Card price	Modes (cycle factor, core price)	$SIL^{\max}$	Part.	Memory
mixed.xml	R-card	cortexR	2	3	120	standard (1, 40); eco (1.3, 40)	4	yes, 15	64000
	M-card	cortexM	4	4	25	standard (1.6, 8); performance (1.2, 11)	2	no	16000
mixed_noiso.xml	R-card	cortexR	2	3	120	standard (1, 40); eco (1.3, 40)	4	no	64000
	M-card	cortexM	4	4	25	standard (1.6, 8); performance (1.2, 11)	2	no	16000
mixed_3type.xml	R-card	cortexR	2	3	120	standard (1, 40); eco (1.3, 40)	4	no	64000
	P-card	ppcE200	2	3	140	standard (1.15, 45)	4	no	48000
	M-card	cortexM	4	4	25	standard (1.6, 8); performance (1.2, 11)	2	no	16000
mixed_4type.xml	R-card	cortexR	2	3	120	standard (1, 40); eco (1.3, 40)	4	no	64000
	P-card	ppcE200	2	3	140	standard (1.15, 45)	4	no	48000
	M-card	cortexM	4	4	25	standard (1.6, 8); performance (1.2, 11)	2	no	16000
mixed_costly.xml	S-card	sparcLEON	2	3	150	standard (1.25, 48)	4	no	48000
	R-card	cortexR	2	3	480	standard (1, 160); eco (1.3, 160)	4	no	64000
	M-card	cortexM	4	4	100	standard (1.6, 32); performance (1.2, 44)	2	no	16000

Used by: mixed.xml: p_rasta; mixed_noiso.xml: f_random_hw, d_*, c_pat, RQ4; mixed_3type.xml: f_sw3, f_both3; mixed_4type.xml: v_nvp; mixed_costly.xml: RQ4 (old Fig. 12). Max.: cards the front-end may instantiate. Cycle factor: multiplier of the nominal execution time in that mode. Part.: certified partitioning and its cost per partitioned core. TDMA bus of the card platforms: 16 slots, at most 8 per core. Prices in cost units; they are illustrative.

The catalog (Table III) gives the SIL range for which a pattern is admitted, the fault classes it covers, a recurring cost, and the tactics and general scenarios used by the argument generator (Section V). Table V lists the remaining fields of each record, and Table VI the scenarios. The SIL range is necessary, not sufficient: SafeDSE computes no failure rates, hardware fault tolerance, or diagnostic coverage. The ceiling of the pattern `none` at SIL 1 follows IEC 61508's expectation of diagnostic coverage from SIL 2 upward; it is the most consequential number of the catalog and therefore lives in the data, where it can be argued with. The 2-of-2 pair runs only while both channels agree, i.e., 1oo2 in IEC 61508 notation [42], [43]. The pattern names are Koopman's; Koopman and Wagner [11] published the doer/checker (monitor/actuator) pair, Douglass [33] has counterparts for four rows, and Armoush [9] for three (Table III). Armoush calls the mixed-SIL doer/checker and the diverse doer/checker homogeneous and heterogeneous duplex patterns, and the low-SIL diverse doer/checker is our combination of two catalog patterns. N-version programming is IEC 61508-7 C.4.4.

In Fig. 2b, the two doer/checker patterns have the same structure and differ only in `DIFFERENT` versus `SAME`, and the fault model selects between them. A pattern is applicable to a if its SIL range contains SIL_a^{req} , it covers the fault model, and the platform has enough core types certifiable at that SIL for its `DIVERSE` groups. An inapplicable pattern is dropped as a candidate; an instance is rejected only if an actor has no applicable pattern left.

If software implements safety functions of different SILs, IEC 61508 requires that all of it is developed to the highest SIL unless their independence is shown [3]. We apply this rule per core and accept certified partitioning as evidence of independence; otherwise, a low-SIL actor next to a high-SIL actor is promoted. This assumes that cores interfere only through the bus, not through shared memory, caches, or I/O [44]. Communication actors are exempt under the black-channel principle [45]: the communicating actors detect corruption, loss, and delay end to end. Whether a transfer is a safety function is a system-level assumption; the front-end offers two alternatives (Section IV-B), and the generated argument states the one used. Development cost is a baseline cost times a multiplier $k(s)$ for SIL s . Three profiles of k , which disagree strongly, come from the estimates that Myklebust et al. collected [4]: their own (default), Klosterman's, and DO-178B ratios with levels D and C as SIL 2 and 3 (Table IV). Klosterman's table has two rows that map to SIL 3; the profile takes the higher one, and the evaluation also uses the lower

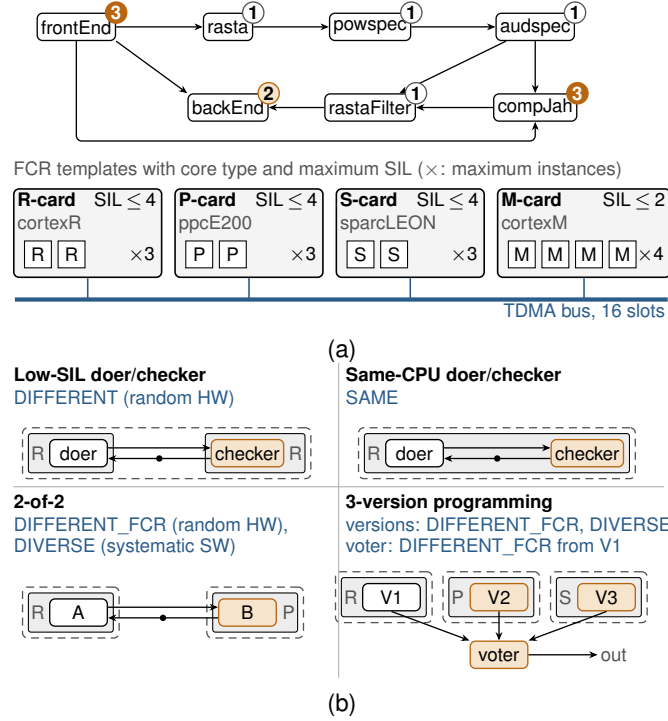


Fig. 2. (a) Running example: RASTA speech front end with the required SIL of each actor (badge), and the FCR templates of the platform. (b) Four catalog patterns with their placement relations (white: owner, shaded: added component, dot: initial token). Both doer/checker patterns have the same structure and differ only in placement, which the fault model selects; the evaluation instances use subsets of the catalog.

TABLE III
PATTERN CATALOG (EXCERPT OF THE DATA FILE)

Pattern	Adds	Placement per fault class	SIL	Covers
none	–	–	0–1	–
Low-SIL D/C	c	H: $o \neq c$	1–2	H
Same-CPU D/C	c	$o = c$	1–2	S
2-of-2	b	H: $o \neq_f b$; S: $o \neq_t b$	3–4	H, S
High-SIL isolated D/C	c	H: $o \neq_f c$	3	H
Mixed-SIL D/C	c_1, c_2	H: $o \neq_f c_1, c_1 \neq_f c_2$; S: $c_1 \neq_t c_2$	3–4	H, S
Dual 2-of-2	b, s_a, s_b	H: $o \neq_f b, s_a \neq_f s_b$, $o \neq_f s_a$	3–4	H
1-of-2 failover	b	H: $o \neq_f b$	0–1	H
Low-SIL diverse D/C	c	H: $o \neq_f c$; S: $o \neq_t c$	1–2	H, S
3-version (NVP) [†]	v_2, v_3, w	H: $\neq_f \{o, v_2, v_3\}$, $o \neq_f w$; S: $\neq_t \{o, v_2, v_3\}$	3–4	H, S

Roles: o owner (the application actor); c, c_1, c_2 checkers; b second channel of the owner's pair; s_a, s_b standby pair; v_2, v_3 versions; w voter. $\neq_c, \neq_f, \neq_t$: different core, FCR, core type; $=$: same core. S is covered through different core types or, for Same-CPU D/C, the checker's algorithm. D/C: doer/checker. NVP: N-version programming. [†] Separate catalog, used in one case study; its voter w is not replicated. Sources: Koopman (names; D/C pair in [11]): all except 3-version (NVP); Douglass [33]: Low-SIL D/C, High-SIL isolated D/C, 1-of-2 failover, Low-SIL diverse D/C; Armouth [9]: Mixed-SIL D/C, Low-SIL diverse D/C, 3-version (NVP).

one.

C. Problem Statement

Given the applications with period bounds, the platform, the safety specification, the catalog, and a cost profile, the problem is to choose a pattern per SDF actor and, per active HSDF node, a core, a static-order

TABLE IV
DEVELOPMENT-COST PROFILES

Profile	SIL 0	1	2	3	4
Myklebust et al.	1.00	1.13	2.25	5.18	9.06
Klosterman	1.00	1.13	1.23	1.70	2.35
Klosterman (lower row)	1.00	1.13	1.23	1.40	1.59
DO-178B	1.00	2.00	3.00	5.00	9.00

Multiplier $k(s)$ of the baseline development cost (100 units per node). SIL 0 and SIL 4 are extrapolated. Lower row: Klosterman's ASIL C row for SIL 3, used only in the price sweep of Section VI-F.

TABLE V
PATTERN CATALOG: STRUCTURE AND ATTRIBUTES OF EVERY RECORD

Pattern	Components (WCET, Δ SIL)	Edges (tokens)	Inputs to / output from	Failure	Rec.	Mult.	Tactics
none	—	—	— / —	none	0	—	—
Low-SIL D/C	c (checker, -1)	$o \rightarrow c$ (0), $c \rightarrow o$ (1)	c / o	active	2	1	SaC, RR
Same-CPU D/C	c (checker, -1)	$o \rightarrow c$ (0), $c \rightarrow o$ (1)	c / o	active	1	1	SaC
2-of-2	b (full, 0)	$o \rightarrow b$ (0), $b \rightarrow o$ (1)	b / o	silent	2	1 (2^d)	RR, Cmp, Vot, DR
High-SIL isolated D/C	c (checker, 0)	$o \rightarrow c$ (0), $c \rightarrow o$ (1)	c / o	silent	2	1	SaC, RR
Mixed-SIL D/C	c_1 (checker, -2), c_2 (checker, -2)	$o \rightarrow c_1$ (0), $o \rightarrow c_2$ (0), $c_1 \rightarrow c_2$ (1), $c_2 \rightarrow c_1$ (1), $c_1 \rightarrow o$ (1)	$c_1, c_2 / o$	silent	3	1	SaC, Cmp, RR, Vot, DR
Dual 2-of-2	b (full, 0), s_a (full, 0), s_b (full, 0)	$o \rightarrow b$ (0), $b \rightarrow o$ (1), $s_a \rightarrow s_b$ (0), $s_b \rightarrow s_a$ (1), $s_a \rightarrow o$ (1)	$b, s_a, s_b / o$	operational	4	1	RR, Cmp, Vot, Rep
1-of-2 failover	b (full, 0)	$b \rightarrow o$ (1)	b / o	operational	2	1	RR, Rep
Low-SIL diverse D/C	c (checker, -1)	$o \rightarrow c$ (0), $c \rightarrow o$ (1)	c / o	active	2	2	SaC, DR, RR
3-version (NVP) [†]	v_2 (full, 0), v_3 (full, 0), w (voter, 0)	$o \rightarrow w$ (0), $v_2 \rightarrow w$ (0), $v_3 \rightarrow w$ (0)	$v_2, v_3 / w$	masking	4	0.4/1	RR, DR, Cmp, Vot

Roles as in Table III. WCET: *full*, the owner's WCET (a replica); *checker* or *voter*, the component's own WCET entry. Δ SIL: the catalog's *sil_offset*, the number of levels by which the doer may be developed below the actor's SIL; the model does not use it (Section VIII). Edges: from, to, and initial tokens. Inputs to: components that receive copies of the owner's input channels; output from: the role whose output leaves the pattern. Failure: failure mode of the protected function. Rec.: recurring cost units. Mult.: development-cost multipliers per component in the catalog (^dwhen the pattern places the component DIVERSE); read but not priced by the model. Tactics after Prescherm et al.: SaC sanity check, Cmp comparison, RR replication redundancy, DR diverse redundancy, Vot voting, Rep repair. [†]Separate catalog.

position, and an implemented SIL. The choice also covers each core's purchase, mode, provisioned SIL, and partitioning, and the TDMA slots and buffer sizes. The placement relations of the chosen patterns and all period bounds must hold, and one metric is minimized under upper bounds on the others. Patterns are not nested: they apply to the actors of the application only, which keeps the superposed graph linear in the catalog size.

TABLE VI
GENERAL SCENARIOS OF THE CATALOG PATTERNS

Pattern	Scen.	Covers	Tactics	General scenario (catalog text)
Low-SIL D/C	SC1	–	Sanity Check	An implausible output of the doer is detected by the checker.
	SC2	H	Replication Redundancy	A random hardware fault in the doer’s processor does not prevent detection, because the checker runs on another processor.
Same-CPU D/C 2-of-2	SC1	–	Sanity Check	An implausible output of the doer is detected by the checker.
	SC1	–	Comparison	A fault in either channel is detected as a disagreement between the two channels.
	SC2	H	Replication Redundancy	A random hardware fault in one channel does not affect the other channel.
	SC3	S	Diverse Redundancy	A systematic fault is not common to both channels.
	SC4	–	Voting	A detected disagreement does not propagate an incorrect result to the consumer.
High-SIL isolated D/C	SC1	–	Sanity Check	An implausible output of the doer is detected by the equally-high-SIL checker.
	SC2	H	Replication Redundancy	A random hardware fault in the doer’s fault containment region does not disable the checker.
Mixed-SIL D/C	SC1	–	Sanity Check	An implausible output of the low-SIL doer is detected by the high-SIL checkers.
	SC2	–	Comparison	A fault in one checker is detected by the other checker’s cross-check.
	SC3	H	Replication Redundancy	A random hardware fault affects at most one of the doer and the two checkers.
	SC4	S	Diverse Redundancy	A systematic fault in one checker is not reproduced by the other checker.
	SC5	–	Voting	A detected disagreement does not propagate an incorrect result to the consumer.
Dual 2-of-2	SC1	–	Comparison	A fault in a channel is detected as a disagreement within its pair.
	SC2	H	Replication Redundancy	A random hardware fault affects at most one of the four channels.
	SC3	–	Repair	The function remains available after the primary pair has failed silent.
	SC4	–	Voting	The failed pair’s result does not reach the consumer once the standby pair has taken over.
1-of-2 failover	SC1	–	Repair	The function remains available after one channel stops.
Low-SIL diverse D/C	SC1	–	Sanity Check	An implausible output of the doer is detected by the checker.
	SC2	H	Replication Redundancy	A random hardware fault in the doer’s fault containment region does not prevent detection.
	SC3	S	Diverse Redundancy	A systematic fault in the doer’s implementation is not reproduced by the checker.
3-version (NVP) [†]	SC1	–	Voting	A fault in any single version is masked by the majority of the remaining versions.
	SC2	H	Replication Redundancy	A random hardware fault affects at most one of the three versions.
	SC3	S	Diverse Redundancy	A systematic fault in one version is not reproduced by the other two versions.
	SC4	–	Comparison	A disagreement among the versions is detected before the result leaves the component.

Covers: the fault class a scenario needs (H random hardware, S systematic); a scenario outside the fault model becomes an undeveloped goal with the fault model as the reason. –: in scope under every fault model. none has no scenario. [†] Separate catalog.

IV. THE SAFEDSE CONSTRAINT MODEL

A. Decision Variables and Guarded Superposition

The model decides a pattern pat_a per SDF actor a , and per node i a core $proc_i$, a successor in that core’s static order, and an implemented SIL sil_i , the systematic capability of its software. Per core slot p , it decides the use, mode, provisioned SIL $csil_p$, and partitioning $part_p$; per application z , a period μ_z ; and globally, the TDMA slots and buffer sizes.

The topology is not a variable. The front-end builds the superposed graph G^+ , which holds every component of every applicable pattern (Fig. 3a). Patterns share a component slot of an actor only if their components use the same WCET entry (a full replica uses its owner’s, a checker its own). Node i carries the set Γ_i of patterns that use it (all patterns for an application node). Patterns are applied before unfolding, because a pattern is a decision about a function, not about one iteration of it, so copy r of a component belongs to copy r of its owner, $o(i)$. A component slot is itself an SDF actor of G^+ ; its

pattern variable is tied to its owner's, which removes one redundant variable and one symmetry per slot. With $a(i)$ the actor of node i (for a component, the owner),

$$\begin{aligned} active_i &\Leftrightarrow pat_{a(i)} \in \Gamma_i, \\ \neg active_i &\rightarrow T_i = 0 \wedge proc_i = proc_{o(i)}. \end{aligned} \quad (1)$$

An inactive node is neutralized, not deleted: it takes no time, stays on its owner's core, and no cost or memory sum counts it. Pinning matters as much as the zero time; an unpinned inactive node could take any core and multiply every solution by the number of cores. The graph thus has a fixed shape, and the order and throughput constraints are posted once. The pattern of each actor is restricted to the applicable ones, which the front-end computes from the SIL ranges, the fault model, and the platform.

The execution time of a computation node follows its binding and mode,

$$T_i = active_i W(i, ctype_{proc_i}, mode_{proc_i}), \quad (2)$$

where W is the WCET table; a missing entry is a forbidden value of $proc_i$. The nested lookup looks expensive, but a flattened encoding with one reified implication per node and core was never faster in the development measurements, so the model keeps the direct form.

A negative result motivated guarded superposition. A variable adjacency matrix over all possible replicas needs a quadratic number of topology variables and an `element` constraint on a variable index for every access. In the early analysis of the tool, it did not solve a five-actor instance within minutes. With a fixed graph, only the selection is a variable, and every guard is a lookup into a constant table.

Explicit voters. Most catalog patterns fold the comparison into the owner (Table V). A pattern with an explicit voter, such as N-version programming, routes the owner's output through the voter. The front-end re-emits every channel that leaves the owner from the voter slot, guarded by the voter patterns. The original channel survives only under the other patterns. An edge into a downstream pattern can then depend on the patterns of two actors (the voter pattern of its source and the input fan-out of its destination), so a guarded edge e carries two conditions,

$$on_e \Leftrightarrow pat_{a_1(e)} \in \Gamma_e^1 \wedge pat_{a_2(e)} \in \Gamma_e^2. \quad (3)$$

An edge with one condition repeats its owner as the second one with an all-true row, so the conjunction degenerates exactly and no sentinel index is needed. Dropping either condition would let the pre-voter result reach a downstream checker. The guarded edge constraints themselves turned out to be redundant: the front-end also writes every pattern channel into the token matrix, which (12) posts unconditionally, and zero execution time of inactive nodes is what neutralizes their edges (Section IV-E). They are kept because this redundancy depends on how the front-end fills the token matrix.

B. Communication Refinement

Communication refinement uses the same mechanism. For each application channel $c = (u, v)$, G^+ holds a block, a send, and a receive actor. They take zero time unless the channel is remote, which the model defines as

$$remote_c \Leftrightarrow active_u \wedge active_v \wedge proc_u \neq proc_v. \quad (4)$$

A remote channel replaces its direct edge by the path through these actors (Fig. 3b); the front-end removes the direct edge from the token matrix, so exactly one of the two forms is posted. A direct edge kept next to the path would be a shortcut past the whole transfer delay. Block and send run on the sending core's network interface and receive in the receiving core's memory, so their placement is derived: $proc_{blk(c)} = proc_{snd(c)} = proc_u$ and $proc_{rcv(c)} = proc_v$.

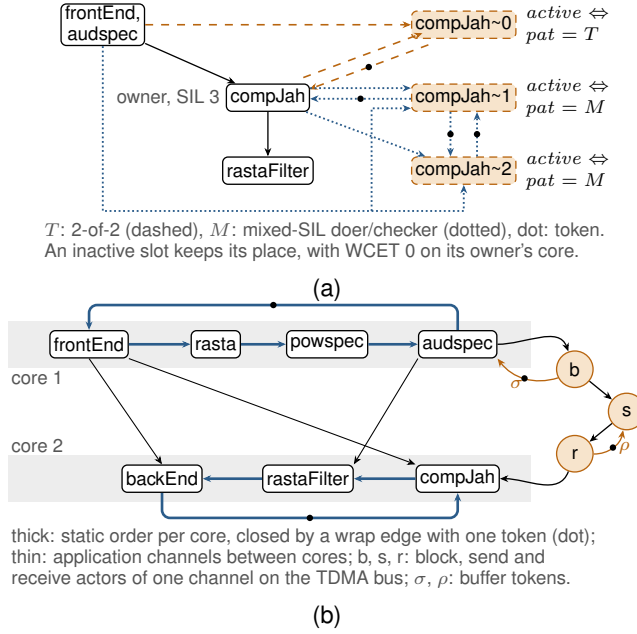


Fig. 3. (a) Guarded superposition of `compJah` under both fault classes: each slot is guarded by the patterns that use it, the graph is fixed, and only pat is a variable. (b) MSAG of a two-core binding of the running example, with one remote channel refined for the TDMA bus. Not drawn: the one-token self-edge of every actor and the refinement of the other remote channels.

The times follow DeSyDe's TDMA model [7]. With S slots per round of cycle length L , a_p slots allocated to the sending core p , and n_c slots needed for the message of c ,

$$\begin{aligned} T_{blk(c)} &= (S - (a_p - 1))L, \\ T_{snd(c)} &= (\rho_c(a_p)S + S)L, \quad T_{rcv(c)} = 0, \end{aligned} \quad (5)$$

where $\rho_c(s) = \lceil n_c/s \rceil - 1$ is the number of full extra rounds. Blocking is the worst-case wait for the core's slot, and sending is the transfer in whole rounds. Receiving takes no time because the token lands in local memory; the receive actor still exists because its buffer closes a cycle. The slots of all cores sum to at most S , a core has slots iff it sends, and it has no more slots than its largest outgoing message needs, a restriction that DeSyDe also posts.

The path of a remote channel $c = (u, v)$ in the MSAG is

$$\begin{aligned} u &\xrightarrow{tok_{uv}} blk \xrightarrow{0} snd \xrightarrow{0} rcv \xrightarrow{0} v, \\ blk &\xrightarrow{\beta_c^s} u, \quad rcv \xrightarrow{\beta_c^r} snd, \end{aligned} \quad (6)$$

with the channel's initial tokens on the first edge, so the pipeline depth is kept. The send and receive buffer sizes $\beta_c^s, \beta_c^r \geq 1$ are the tokens on the back-edges; a local channel has $\beta_c^s = 0$ and $\beta_c^r = 1$. The send buffers of a core must fit its network interface. Because the buffer sizes are variables, their back-edges in (12) multiply two variables, which CP-SAT handles natively.

Communication actors are scheduled on the bus, not in a core's static order, and they are not counted in a core's load. A transfer overlaps computation; it lengthens the latency, and the period only when the bus is the bottleneck. Contention for the bus is already priced into (5), so this exclusion does not make the model optimistic. An early version that put them into the static order reported wrong periods; Section VII returns to it.

Only application channels are refined (`--comm-scope app`). Cross-checks between pattern components are assumed to use a dedicated link; `--comm-scope all` refines them too, and RQ4 measures

the effect (Section VI-F). Whether a transfer is a safety function is set by `--comm-sil`. The default `exempt` leaves communication actors out of the SIL-mixing rule. `inherit` gives each transfer at least its sender's SIL, and `core` subjects it to the rule of the core it runs on, like any software there. A replica channel pair can also be required to leave from different cores, which separates their TDMA slots; on a single bus this is weaker than redundant communication, and the argument says so.

C. Safety and Placement Constraints

A core's SIL is bounded by its type, only a type t with certified partitioning (cp_t) may be partitioned, and every active node meets its actor's required SIL:

$$\begin{aligned} csil_p &\leq SIL_{ctype_p}^{\max}, & part_p &\rightarrow cp_{ctype_p}, \\ active_i &\rightarrow SIL_{a(i)}^{\text{req}} \leq sil_i \leq csil_{proc_i}. \end{aligned} \quad (7)$$

An unused core has $csil_p = 0$ and no partitioning, and an inactive node has $sil_i = 0$. The IEC 61508 rule on mixing SILs becomes, for every active computation node (communication actors are exempt by default),

$$active_i \wedge \neg part_{proc_i} \rightarrow sil_i = csil_{proc_i}, \quad (8)$$

and $csil_p$ equals the highest sil_i of an active, non-exempt node on a used core p . So all nodes on a core without partitioning share that SIL, and a partitioned core pays the partitioning cost of its type. With the baseline development cost b_i and the profile's multiplier k , the development cost and its part spent on promotion (included in C_{dev}) are

$$\begin{aligned} C_{\text{dev}} &= \sum_i active_i b_i k(sil_i), \\ C_{\text{pro}} &= \sum_i active_i b_i (k(sil_i) - k(SIL_{a(i)}^{\text{req}})). \end{aligned} \quad (9)$$

The model stores $100k$ and divides each sum by 100, so all arithmetic stays integral. An input flag forbids promotion: every active node is then implemented exactly at its required SIL (Section VI-C).

For each placement relation of an applicable pattern whose fault class is in the fault model, the front-end emits a tuple (u, v, R, a, Γ) , and the model posts

$$pat_a \in \Gamma \rightarrow R(proc_u, proc_v). \quad (10)$$

Relation R is one of $proc_u = proc_v$ (SAME), $proc_u \neq proc_v$ (DIFFERENT), $fcr_{proc_u} \neq fcr_{proc_v}$ (DIFFERENT_FCR), or $ctype_{proc_u} \neq ctype_{proc_v}$ (DIVERSE). As fcr and $ctype$ are constant arrays, each relation is an `element` constraint. Relations hold per HSDF copy: copy r of a checker avoids copy r of its doer. The set-wise reading (no copy of the checker on a core of any copy of the doer) is stronger and forbids safe interleavings. Under a single fault the pairwise reading suffices: a fault on a core with doer copy $r + 1$ and checker copy r is caught by checker copy $r + 1$ elsewhere.

D. Static Order

Each core runs its computation nodes in one static order, encoded by a successor $succ_i \in \{0, \dots, n\}$ (0: last on its core), a position ord_i , and a head flag h_i :

$$\begin{aligned} succ_i &= j > 0 \rightarrow proc_j = proc_i \wedge ord_j = ord_i + 1, \\ h_j &\Leftrightarrow \nexists i : succ_i = j, & h_j &\rightarrow ord_j = 0, \end{aligned} \quad (11)$$

with all non-zero successors distinct, no node its own successor, distinct positions on one core, and exactly one last node per used core. The wrap edge runs from the last node of a core to its head. This form replaces Rosvall's encoding with a dummy node and a `circuit` constraint [7], a Gencode idiom that MiniZinc decomposes anyway. The anchor $h_j \rightarrow ord_j = 0$ is not cosmetic: without it the positions

float, the head cannot be identified, and the wrap edge, and with it the bound “period at least the core’s load”, silently disappears. A core is used iff an active node is on it, and a used core requires its card. Deadlock is caught by the potentials of Section IV-E, so the positions only have to make each static order acyclic.

E. Throughput by Node Potentials

The MSAG [7] of a solution adds to G^+ a token-free edge from each node to its successor and a one-token wrap edge from the last to the first node of each core. Remote channels add their paths (6), whose back-edges carry the buffer sizes as tokens. Each computation node has a one-token self-edge and the time (2). With an integer potential pot_i per node, the model posts for every MSAG edge (i, j) , with $z(i)$ the application of i ,

$$pot_j \geq pot_i + T_i - tok_{ij} \mu_{z(i)}. \quad (12)$$

Order, wrap, and channel edges are posted under reified conditions on the successor and $remote_c$ variables. One potential is anchored at zero, which removes the free additive constant.

Proposition 1: Fix all decisions except μ and pot . Potentials that satisfy (12) on every MSAG edge exist iff every cycle C of the MSAG satisfies $\sum_{i \in C} T_i \leq \mu \sum_{(i,j) \in C} tok_{ij}$.

Proof: If pot is feasible, summing (12) along C cancels the potentials and yields the inequality. Conversely, weight each edge with $w_{ij} = T_i - \mu tok_{ij}$; no cycle has positive weight. Add a source with a zero-weight edge to every node. The longest-path lengths ℓ_j from it are then finite and satisfy $\ell_j \geq \ell_i + w_{ij}$ on every edge, so $pot = \ell$ is feasible; it is integral for integral T , tok , and μ . As $0 \leq \ell_j \leq \sum_i T_i$, the bounded potential domain of the model holds ℓ after anchoring. Section A gives the proof in full, with its corollaries. ■

Proposition 1 is the linear-programming form of the cycle ratio problem, with the potentials as node distances [46], as in cyclic scheduling [24]. Minimizing μ thus yields the period of Definition 1 without a throughput propagator [6], which a MiniZinc model cannot host; as μ is an integer, the model reports [MCR]. A token-free cycle with positive execution time makes (12) infeasible, so the same inequalities replace separate deadlock machinery, such as Rosvall’s rank constraints or a test for positive throughput. As remote buffers hold a token and every WCET in our inputs is at least one, each token-free cycle of active nodes has positive time, so deadlock is excluded. The encoding needs one variable per node, against a quadratic number for an edge-based residual formulation. For fixed token counts, the inequality is linear; only the buffer back-edges are products of two variables.

Neutralized nodes keep their channels with zero execution time. A token-free cycle through them has zero weight and does not constrain μ . A path through an inactive slot, however, can raise the least μ of a design above its MCR, never below it. A channel of an unselected pattern into an inactive slot, followed by the slot’s static-order edge, is a path that the deployed design does not have (Section A, Corollary 4). The model thus stays sound but can be pessimistic for a design, and under a period bound it can reject a design that meets the bound. We wrote two exact variants. Variant A posts pattern channels only under their guard and changes the verifier in step. Variant B adds a second potential for the static-order and wrap edges, equal to pot on active nodes, and keeps under the guard only the channels whose guard differs from the activity of their endpoints. Both remove the gap on the affected designs but solve more slowly. Variant B served as a cross-check of every RQ2 and RQ4 result (Section VI-C), with a median run-time ratio of 1.09 and a heavy tail at loose bounds; the default model keeps the faster form. For other objectives than the period, the model guarantees only $MCR \leq \mu \leq$ the period bound, and the evaluation fixes the design and minimizes μ afterwards (Section VI-A).

Rosvall et al. [6] gave each application the MCR of its connected MSAG component. Applications that share a core lie in one component and share its period, since the core repeats one static order in which each of their nodes fires once. The model equates their periods pairwise:

$$sh_{y,z} \Leftrightarrow \exists p : on_{y,p} \wedge on_{z,p}, \quad sh_{y,z} \rightarrow \mu_y = \mu_z, \quad (13)$$

where $on_{z,p}$ holds iff an active node of z is on core p . By transitivity, each component gets one period, and its potentials share one time base, which is the condition for summing (12) along a cycle that crosses applications. DeSyDe states the coupling over pairs of actors of different applications, a quadratic number of reified implications. In an early version of SafeDSE, this timed out at 32 actors; the occupancy matrix on makes it $O(|APP|^2 + n|APP|)$ and is equivalent.

Four implied constraints from DeSyDe [7] prune at the root, although the potentials imply them only after search. With the load λ_p of core p over its computation nodes,

$$\lambda_p = \sum_j [proc_j = p] T_j, \quad proc_i = p \rightarrow \mu_{z(i)} \geq \lambda_p. \quad (14)$$

Second, an active node's time is at most its application's period bound. Third, two active nodes of one application whose least WCETs sum to more than its period bound share no core and are not adjacent in a static order. The activity condition matters, because an inactive slot is pinned to its owner's core (Section VII). Fourth, an application uses at least $\lceil \sum_i \min_t W(i, t) / B_z \rceil$ distinct cores, with the sum over its always-active nodes and B_z its period bound. The wrap edge already implies (14), but only through a cycle that the solver must find; stated as a sum, it propagates at once.

F. Further Metrics and Constraints

The memory of the active nodes on a core is bounded by its type's memory,

$$\sum_i [proc_i = p \wedge active_i] m_i \leq M_{ctype_p}. \quad (15)$$

The hardware cost sums the prices of the used cards and of the used cores in their modes, and the power sums the power of the used cores in their modes. A latency bound between a source and a sink actor uses the potentials again: in the periodic regime, the k -th firing of node i starts at about $pot_i + k\mu$, so

$$lat(s, d) = pot_d - pot_s + \rho(s, d) \mu + T_d, \quad (16)$$

with $\rho(s, d)$ the least number of initial tokens on a path from s to d , computed by the front-end. The estimate is exact in the periodic phase but ignores the transient before it, so it serves as a cheap filter; a separate checker simulates the transient, and a reported latency is not a worst-case bound without it.

G. Objectives and Solving

The model computes eight metrics: the worst period, the largest latency, the hardware, development, and total cost, the power, the number of cores, and the promotion cost. One metric is minimized, chosen by a parameter, and every metric has an upper bound, so switching the objective needs no model edit. The total cost

$$C_{tot} = C_{hw} + C_{dev} + C_{part} + C_{pat} \quad (17)$$

adds the price of cards and cores in their modes, (9), the cost of each partitioned core, and the recurring cost of the selected patterns, summed over the actors of the applications only, all in one cost unit.

Symmetry breaking orders only interchangeable cores. Cores of one type on one card are interchangeable, so a value-precedence constraint orders their use. Cores on different cards are not, once FCR relations exist; only whole cards of the same template are, so the model buys a prefix of them and orders successive identical cards by their node membership. Cores of one type that are each alone on their card are interchangeable outright and get value precedence too. The order of Rosvall and Sander [7] on the HSDF copies of an SDF actor (copies ordered by core index, and by position on a shared core) acts only on multi-rate actors, which our pattern instances lack. With per-copy placement relations, it could exclude an optimum in principle; in our probes with multi-rate owners, it pruned designs but moved no optimum.

The model is written in MiniZinc [21] and solved with CP-SAT [22]. The search annotation branches on the binding (first fail, smallest value first), then on the successors, then on the periods (smallest value first). With one worker, CP-SAT follows this annotation; with several, its portfolio does not, which is why the period of a non-period optimum is not pinned to its least value without the extra step of Section VI-A. The model also has a two-step strategy inherited from Rosvall and Sander: step 1 forbids applications to share a core, and its objective value bounds the full model in step 2, which stays complete. With four workers it is no longer needed on the multi-application benchmark (Section VI-B).

V. INDEPENDENT VERIFICATION AND SAFETY-ARGUMENT GENERATION

A. Independent Verifier

A wrong constraint does not fail; it returns a plausible number. A verifier that shares no code with the model therefore re-checks every solution. It reads the front-end’s instance and the solution, recomputes every field that the model derives, and appends one record to a check log per check. No derived field of the solution is taken on trust. Each record has a kind, a pass flag, and a detail. An `input` record checks that every required field is present, so a missing field is a failed record, not a crash. A `wcet` record checks that every node time equals the table entry of its core type and mode, zero when inactive, and that no binding is forbidden. An `order` record checks one acyclic static order per used core, over exactly the nodes bound to it. An `activation` record checks that activation follows the selected pattern, that the pattern is admissible for the actor, and that an actor’s slots share it. The `memory` and `cost` records recompute memory use and the hardware, development, promotion, partitioning, and pattern cost from the instance tables. A `sil_actor` record per actor and per component checks that the implemented SIL is at least the required one and at most the core’s. An `isolation` record per core checks the SIL ceiling of the type, the SIL-mixing rule, and partitioning only on a type that provides it. Each relation instance of the selected patterns gets a `placement` record. Finally, the `period` record checks the period as described next, and the `load_bound` record that it is at least the load of the busiest core.

For the period, the verifier rebuilds the MSAG of the deployed design. It removes inactive nodes and splices each static order around them: consecutive live nodes get a token-free edge, and the last live node a one-token wrap edge to the first. A refined channel that stays local gets back its direct edge with its initial tokens; a remote one gets the path (6), with the tokens on its first edge and the two buffer back-edges. A token-free cycle in this graph is a deadlock and fails the record. For each connected component (applications that share a core share a period, (13)), the verifier computes the MCR twice. The first oracle is Lawler’s parametric search [46]: bisection on μ with a Bellman-Ford test for a positive cycle, then the rational with the smallest denominator in the final interval. The second fires the actors in max-plus time, i.e., it simulates the self-timed execution, until the differences of the start times repeat over a window of up to eight iterations. The two share no reasoning: one is a cycle analysis, the other a simulation. Both must agree, and the reported period must equal $\lceil \text{MCR} \rceil$; for other objectives than the period, we first minimize μ for the fixed design (Section VI-A). If a latency bound is set, the verifier also simulates the transient and flags a worst case above the model’s periodic estimate (16).

The log records what was checked, not what is true. The generator treats a missing record like a failed one, so “nobody looked” differs from “looked and found it fine”. The verifier reads the front-end’s instance, so front-end and specification errors are common to model and verifier, as every argument states in an assumption. It is an unqualified T2 tool in the sense of IEC 61508 [42]. Its first version took WCETs, static orders, activation, and costs from the solution; a mutation analysis (Section VI-F) showed that it then accepted corrupted solutions, and the current version re-derives all of them.

B. Evidence-Backed Pattern Arguments

Preschern et al. [10] attached a GSN [23] fragment to each pattern: its general scenarios are goals, the tactics that achieve a scenario are strategies below it, and IEC 61508 methods realize the tactics. A

scenario that depends on another is nested below it. Their fragments are per pattern and generic; our generator instantiates them for a solved and verified deployment, which is what makes them checkable.

The top goal claims that the deployed architecture meets, for every safety-related actor, the architectural preconditions that the catalog states for its SIL. Its contexts name the standard and the meaning of SIL 0, the hazard analysis as the source of the required SILs, and the catalog with its sources. Further contexts give the applications, the platform with the instantiated cores, the fault model with the excluded fault classes and the meaning of S, and the SHA-256 hashes of instance, solution, and check log. A justification limits the argument to the architecture and its deployment, and three assumptions cover sound WCETs, token preservation, and the shared front-end. A strategy then argues over each actor with $SIL_a^{\text{req}} \geq 1$ and over the platform-wide properties.

Per actor, the goal claims that the deployment meets the catalog’s preconditions for SIL_a^{req} ; no goal claims an attained SIL. Contexts give the HSDF copies with their cores and SILs, and the components of the selected pattern with theirs. A provisioning goal has a Solution for the owner and one per component, each citing a `sil_actor` record. A strategy “by application of” the selected pattern has one goal per general scenario (Table VI) and below it one strategy per tactic. A tactic table, transcribed from Preschern et al.’s appendix, links each tactic to IEC 61508-7 methods and to the placement relations whose records discharge its deployment precondition (Table VII). It links Replication Redundancy to `DIFFERENT_FCR` or `DIFFERENT`, Diverse Redundancy to `DIVERSE`, and value-domain tactics (Sanity Check, Condition Monitoring, Comparison, Voting, Override, Repair) to none. This column is the exact boundary between what a mapping can and cannot establish, and the one to be most suspicious of. A relation listed there that the model does not post would turn a hand-wave into a claim.

For a tactic with relations, a deployment goal claims exactly what the relation establishes. It claims “the replicated components run in different fault containment regions” for `DIFFERENT_FCR`, “on different cores, possibly in one fault containment region” for `DIFFERENT`, and “the diverse components run on different core types” for `DIVERSE`. Its Solutions cite the `placement` records of that relation by index (Figs. 4 and 5). An undeveloped goal next to a replication goal covers the common causes that remain: the interconnect and cross-check link, the shared inputs, the comparison or voting point, and for `DIFFERENT` also the shared card with its power and clock. The goal that an IEC 61508 method realizes the tactic, and is effective for the faults of concern, stays undeveloped with the reason “value-domain or implementation evidence; the mapping cannot establish this” and the candidate methods as context. A `DIVERSE` goal carries the assumption that distinct core types give distinct toolchains and object code, not an independent design. A scenario whose fault class lies outside the fault model becomes an undeveloped goal with the fault model as the reason. A placement relation that no tactic consumes (e.g., `SAME`, which confers no independence) appears as a goal “remaining placement constraints” with its records.

Four platform-wide goals follow. The isolation goal claims that no core hosts software of differing SIL without certified partitioning, also from SIL-0 actors. It is argued by the Barrier tactic with a Solution citing the `isolation` records, under the assumption that cores interfere only through the interconnect; if a core is partitioned, an undeveloped goal claims that the mechanism is certified. The timing goal has two Solutions, the `period` and the `load_bound` records. The development process goal stays undeveloped, and so does the goal that each detected fault brings the function to its safe state within the process safety time, with the verdict tokens of the selected patterns as context. With `--comm-sil_exempt`, an assumption states the black-channel condition for transfers.

Definition 2 (Well-formed argument): Every leaf is a Solution that cites at least one passed record of the check log of the argued solution, or a goal marked undeveloped with a stated reason.

Definition 2 is structural: nothing unchecked appears as checked, but the argument may still be inadequate (Section II-C). The generator’s audit enforces it, including that every cited record exists and passed, and the generator refuses (exit code 3) to emit an argument that violates it. It also refuses if the verifier rejected the solution (exit code 2), if the catalog lacks a pattern selected in the instance

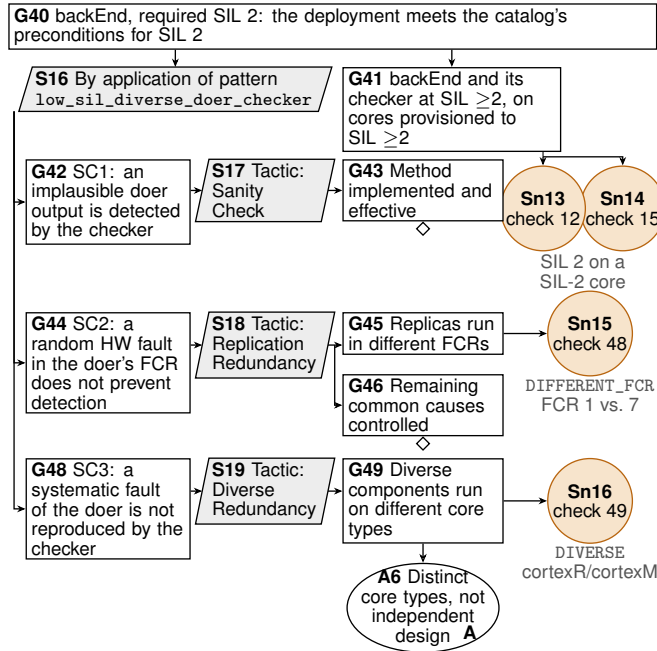


Fig. 4. Excerpt of the argument generated for a verified optimum of the running example under both fault classes: actor *backEnd* (SIL 2, low-SIL diverse doer/checker), ids as generated, texts shortened. Deployment goals claim exactly what their log records establish; the remaining common causes and the method goals (diamonds) stay with the engineer. The method goal is drawn once.

(exit code 4), or if a given fault model contradicts the one it infers (exit code 2). The fault model is inferred, not trusted from a flag. The admissible patterns per actor are a function of the fault model and the required SILs, so the generator tries all three fault models and keeps those that reproduce them. If none or several fit, it names none, and the out-of-scope analysis is unavailable. A wrong flag would otherwise yield a fluent argument that claims coverage the design never had. The catalog check exists because of an observed failure: against a catalog without the NVP record, an early generator version produced a well-formed, audit-passing argument that argued an actor with three-version programming “by development process alone”.

VI. EXPERIMENTAL EVALUATION

A. Setup

All runs used one machine (Intel Core i7-7700HQ, 2.80GHz, 8 logical CPUs, 15.5GiB), Mini-Zinc 2.10.1, and CP-SAT 9.15 with 4 workers, one per physical core, and a time limit of 300s per solve unless stated otherwise. When μ is not the objective, an optimum may report any period between the MCR and the bound (Section IV-E), so we then fix the design and minimize μ in a wrapper model (the period step). Every solution below passed the verifier; a run that did not is left out of every number and named where it matters. Optima are often not unique (on 3 instances, two runs chose different SIL-3 patterns at equal cost), so we report cost, and periods only where the period is the objective.

Table VIII lists the pattern instances. They include the running example on platforms with two to four core types (one with certified partitioning), the fault model changed, two with the TDMA bus, Sobel [7] with a SIL-3 actor, and three applications together; Section B gives their inputs. WCETs on the card platforms are the benchmarks’ nominal times scaled per core mode, and a checker or voter takes 30% of its owner’s. All nodes have the same baseline development cost, and prices are illustrative (Tables II and IV). The joint optimum of every row of Table VIII was solved 5 times. All 270 runs of this kind

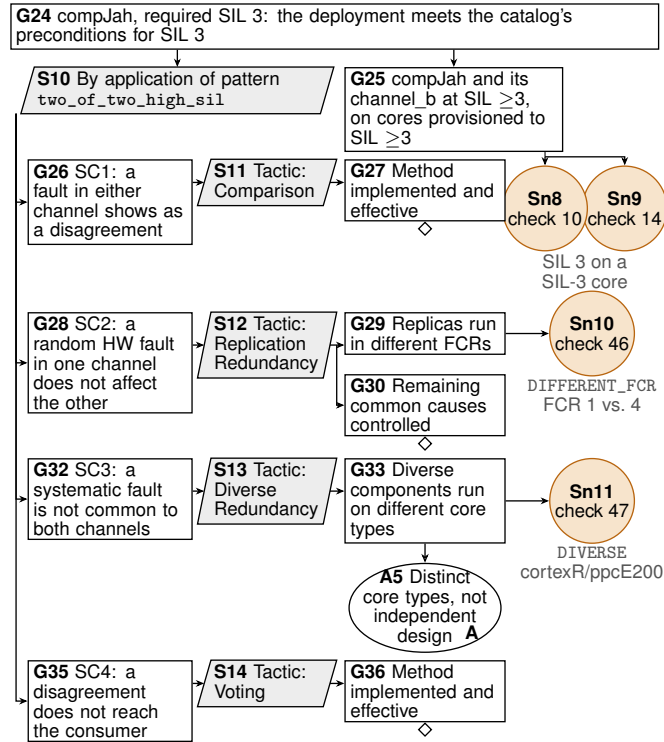


Fig. 5. Second excerpt of the same argument: actor `compJah` (SIL 3, 2-of-2). The replication goal cites the `DIFFERENT_FCR` record and the diversity goal the `DIVERSE` record of the same pair; Comparison and Voting are value-domain tactics and stay undeveloped. Generated by the script of Fig. 4.

returned the same optimum per row, and the run time varied by a median of 9 % and at most 86 % around the median of a group. Front-end, period step, verifier, and generator take at most 1.2 s per instance; the solve dominates (Table IX).

B. RQ1: Comparison With DeSyDe

Without a safety specification, the model only maps and schedules. We ran it and DeSyDe on the files of Experiments 3 and 4 of Rosvall and Sander’s article [7]: four applications on eight identical cores with a TDMA bus (Experiment 3) and the same with a JPEG accelerator (Experiment 4). DeSyDe was built at the version of the article. It needed compile fixes for a current compiler and one fix of undefined behavior: the constructor used by these experiments left the system constraints uninitialized, which failed every scenario at the root. After the fix it reproduced the article’s first result exactly. Both tools had 4 threads and a wall budget of 3600 s per scenario. DeSyDe ran its published configuration, a two-step heuristic with its throughput propagator, and minimizes one application’s period under bounds for the others; SafeDSE minimizes the worst period, and every solution was pinned and verified. The objectives coincide when the optimized application’s period is bounded below by more than every other bound, which holds in 7 of the 11 scenarios; the other 4 (Experiment 3 SoSuRa and the Experiment 4 scenarios with SUSAN) were not compared.

Table X gives the results. SafeDSE reached the article’s period in all 7 scenarios and proved it in 6, including one optimum that the article left open after 23705 s (here 67 s). DeSyDe proved 2 within the budget on our machine. In the other scenarios, its first step mostly ended at its inactivity time-out with the article’s value, and its second step did not finish. On Experiment 4 RaJp, it reached only 1514 (article: 1388). SafeDSE is not faster everywhere: RASTA with JPEG took 18 s against 4 s, and two

TABLE VII
TACTIC TABLE: TACTICS, IEC 61508-7 METHODS, AND DEPLOYMENT RELATIONS

Tactic	Category	IEC 61508-7 methods	Relations	Patterns using it
Sanity Check	Checking	A.1.2 monitoring relay contacts; A.2.7 analog signal monitoring; A.3.1-A.3.3 self-tests; A.4.1-A.4.4 checksums; A.5.1-A.5.5 RAM tests; A.6.1 test pattern; A.9 temporal and logical program monitoring; C.3.3 assertion programming; C.5.3 interface checking	–	Low-SIL D/C, Same-CPU D/C, High-SIL isolated D/C, Mixed-SIL D/C, Low-SIL diverse D/C
Condition Monitoring	Checking	A.1.1 failure detection by online monitoring; A.6.4 monitored outputs; A.9 temporal and logical program monitoring; A.13.1 monitoring	–	–
Comparison	Checking	A.1.3 comparator; A.6.5 input comparison/voting	–	2-of-2, Mixed-SIL D/C, Dual 2-of-2, 3-version (NVP)
Replication Redundancy	Redundancy	A.2.1 tests by redundant hardware; A.2.5 monitored redundancy; A.3.5 reciprocal comparison by software; A.4.5 block replication; A.6.3 multi-channel output; A.7.3 complete hardware redundancy; A.7.5 transmission redundancy	DIFFERENT_FCR, DIFFERENT	Low-SIL D/C, 2-of-2, High-SIL isolated D/C, Mixed-SIL D/C, Dual 2-of-2, 1-of-2 failover, Low-SIL diverse D/C, 3-version (NVP)
Diverse Redundancy	Redundancy	A.7.6 information redundancy; A.13.2 cross-monitoring of multiple actuators; B.1.4 diverse hardware; C.4.4 diverse programming	DIVERSE	2-of-2, Mixed-SIL D/C, Low-SIL diverse D/C, 3-version (NVP)
Voting	Masking	A.1.4 voter; A.6.5 input comparison/voting	–	2-of-2, Mixed-SIL D/C, Dual 2-of-2, 3-version (NVP)
Override	Masking	IEC 61508 fail-safe principle; A.1.3 comparator	–	–
Repair	Recovery	C.3.9 error correction; C.3.10 dynamic reconfiguration	–	Dual 2-of-2, 1-of-2 failover
Degradation	Recovery	A.8 voltage supply error handling; C.3.8 degraded function limitation	–	–
Barrier	Isolation	A.11 separation of energy lines from information lines; B.1.3 separation of safety functions from non-safety functions; B.3.4 modularisation; C.2.8 information hiding/encapsulation; C.3.11 time-triggered architecture	–	–
Simplicity	Avoidance	B.2.1 structured specification; B.3.2 structured design; C.2.7 structured programming; C.4.2 programming language subset	–	–
Substitution	Avoidance	B.3.3 usage of well-proven components; B.5.4 field experience; C.2.10 usage of well-proven/verified software elements; C.4.3 certified tools and compilers	–	–

Methods as listed by Preschern et al. (annex clause of IEC 61508-7 and its title). Relations: the placement relations whose verifier records discharge the tactic's deployment precondition (`deployment_relations` of the tactic table); a tactic without one is realized in the value domain and stays undeveloped.

scenarios took close to the budget. Both searches are nondeterministic, and the one-hour scenarios ran once; the comparison covers only scenarios where the two objectives agree. An uncapped first DeSyDe run of one scenario was still in its second step long after the budget, where only DeSyDe's own six-hour limit applies, so all runs got the same cap.

On the single-application benchmarks of the same article, minimizing the hardware cost under the published period bound gives the optima of Table XI, each proven within a second, and every period meets the published bound. With all four applications together, the optimum has 54 cost units on 6 cores.

TABLE VIII
PATTERN INSTANCES AND JOINT OPTIMA (ALL SOLUTIONS VERIFIED)

Instance	Platform	FM	$ G^+ $	Vars	Cost	Cores	μ^*	t (s)
p_rasta	2T, part.	H	26	11k	3330	2	235	6.0
f_random_hw	2T	H	26	11k	3349	5	235	8.7
d_random_hw ^a	2T	H	18	7k	1857	3	235	6.0
d_systematic_sw ^a	2T	S	14	5k	1846	2	235	3.3
f_sw3	3T	S	18	8k	3365	4	235	5.2
f_both3	3T	H+S	18	8k	3406	5	235	3.2
c_pat	2T, TDMA	H	71	15k	3349	5	235	14.3
pc_sobel ^b	2T, TDMA	H	57	5k	1401	3	333	3.0
v_nvp ^c	4T	H+S	18	9k	5680	6	235	4.9
m_3app ^d	2T	H	44	22k	5665	6	—	44.0

Application RASTA unless noted; nT : n core types; part.: certified partitioning; FM: fault model (H random hardware, S systematic). $|G^+|$: nodes of the superposed graph incl. communication actors; Vars: FlatZinc variables; Cost: proven minimum of C_{tot} of the cost model; μ^* : minimum period; t : median solve time of the cost optimum over 5 runs. ^aSIL 3 actors lowered to SIL 2. ^bSobel on Rosvall’s platform. ^cSeparate catalog with the NVP pattern. ^dSobel (SIL 2), SUSAN (SIL 0), and RASTA, one period each.

TABLE IX
TIME PER PIPELINE STAGE (S)

Instance	Front-end	Solve	Period step	Verifier	Generator
p_rasta	0.14	6.2	0.53	0.09	0.17
f_random_hw	0.14	9.5	0.55	0.09	0.17
d_random_hw	0.16	7.1	0.56	0.10	0.17
d_systematic_sw	0.15	3.5	0.50	0.10	0.17
f_sw3	0.14	5.7	0.46	0.08	0.15
f_both3	0.14	3.4	0.48	0.10	0.16
c_pat	0.15	15.1	1.17	0.19	0.18
pc_sobel	0.15	3.2	0.92	0.10	0.17
v_nvp	0.12	5.6	0.52	0.10	0.14
m_3app	0.15	80.7	0.79	0.12	0.18

Wall-clock seconds of one run of each stage on the joint TOTALCOST instance (4 workers for the solve). Period step: the design fixed and μ minimized (Section VI-A).

TABLE X
HEAD-TO-HEAD WITH DESyDE (PERIODS, TIMES IN S)

Scenario	Article		DeSyDe here			SafeDSE	
	μ	t	μ	t	pr.	μ	t
3: RaJp	2524	10	2524	4	yes	2524	18
3: SoSuJp	2544	2662	2544	3600	no	2544	2829
3: SoRaJp	2544	3124	2544	3600	no	2544	3602*
3: SuRaJp	2544	2562	2544	3600	no	2544	3312
3: SoSuRaJp	3882	1906	3882	2885	yes	3882	31
4: RaJp	1388	279	1514	3600	no	1388	18
4: SoRaJp	1766	23705*	1766	3600	no	1766	67

Experiment: scenario (So Sobel, Su SUSAN, Ra RASTA, Jp JPEG). Article: DeSyDe with 8 threads as published [7]; here: 4 threads, 3600 s wall budget; a time at the budget means the search was stopped with the best value so far. pr.: proven within the budget. * Not proven. SafeDSE times are medians over up to 5 runs for the fast scenarios and one run otherwise.

TABLE XI
SINGLE-APPLICATION BENCHMARKS, MINIMUM HARDWARE COST

Application	Hardware cost	μ	Published bound	t (s)
Sobel	16	384	400	0.5
SUSAN	12	1659	2050	0.5
RASTA	18	543	550	0.6

Rosvall and Sander’s platform and period bounds; μ : largest period over the repetitions (the period of a cost optimum is not unique); t : median time to prove the optimum.

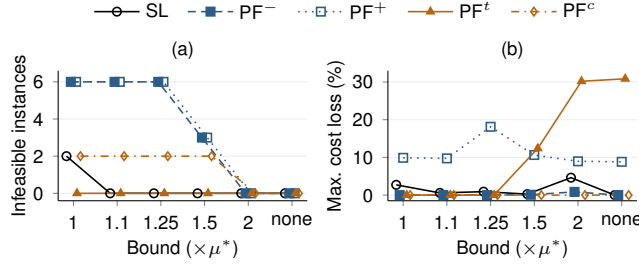


Fig. 6. Decoupled flows against joint exploration per period bound: (a) instances proven to have no feasible design, (b) largest cost loss among the feasible ones (NP, equal to joint, not drawn). No pattern-first rule is both always feasible and never more expensive.

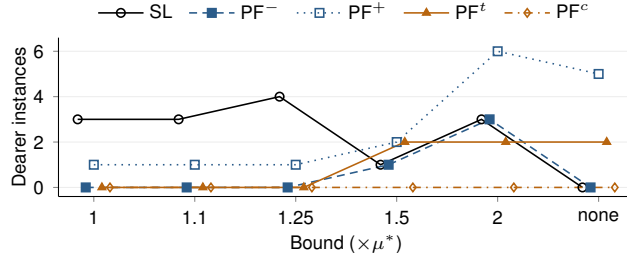


Fig. 7. Instances on which a decoupled flow is feasible but dearer than joint exploration, per period bound (the third count of Table XIII). SL loses cost mostly at tight bounds, the timing-agnostic PF⁺ mostly at loose ones.

The two-step strategy of Section IV-G proved it in 9.1 s (step 1 3.6 s, step 2 5.5 s), and one step in 7.3 s (status OPTIMAL). With four workers, the two steps are thus not needed on this benchmark; with one worker in an early run, only the two-step strategy proved the optimum within the time limit.

C. RQ2: Value of Joint Exploration

We compare joint exploration with decoupled flows, each the joint model plus constraints, so none can be cheaper. Safety-last (SL) maps the instance without patterns at least cost (ties by period, then by core index), fixes the core of every base actor, and leaves patterns, replicas, and platform free. Pattern-first fixes each actor's pattern before mapping. The rules take the fewest components (PF⁻), the widest fault coverage (PF⁺, akin to SafPat [18]), the least added WCET on the fastest type (PF^t), or the least added cost (PF^c), with the catalog order as the last tie-break. No-promotion (NP) forbids promotion. All minimize C_{tot} without a period bound and with the bound $\lceil f\mu^* \rceil$ for $f \in \{1.00, 1.10, 1.25, 1.50, 2.00\}$, where μ^* is the instance's own minimum period. No other tool solves this problem (Table I). A pattern-first flow that backtracks over its pattern choices would be joint exploration by enumeration; the flows here model a design process that fixes the architecture first.

Table XII shows the results without a bound. SL, PF⁻, PF^c, and NP reach the joint optimum on all 9 instances; PF⁺ loses up to 8.8 % and PF^t up to 30.8 %. Without a bound, throughput never binds, and SL can still co-optimize patterns, replicas, and platform after fixing the base actors, since a replica can always go to another core. Under a bound, PF⁻ and PF⁺ are infeasible on 6 instances up to 1.25 μ^* and on 3 at 1.50 μ^* , and PF^c on 2 up to 1.50 μ^* (Fig. 6, Table XIII). PF^t stays feasible but loses up to 30.2 % at 2.00 μ^* , and SL fails on 2 at 1.00 μ^* and loses up to 4.6 %. NP equals joint everywhere. No pattern-first rule is thus both always feasible and never more expensive. Fig. 7 adds how often each flow is feasible but dearer.

TABLE XII
RQ2 WITHOUT A PERIOD BOUND (EXTRA COST OVER JOINT)

Instance	Joint	SL	PF ⁻	PF ⁺	PF ^t	PF ^c	NP	Cores
p_rasta	3330	=	=	=	=	=	=	2
f_random_hw	3349	=	=	+25	=	=	=	5
d_random_hw	1857	=	=	+25	=	=	=	3
d_systematic_sw	1846	=	=	+163	=	=	=	2
f_sw3	3365	=	=	+41	+1038	=	=	4
f_both3	3406	=	=	=	+1038	=	=	5
c_pat	3349	=	=	+25	=	=	=	5
pc_sobel	1401	=	=	=	=	=	=	3
v_nvp	5680	=	=	=	=	=	=	6

No period bound. Joint: proven minimum of C_{tot} . Flows: extra cost over joint (= equal). Cores: of the joint optimum.

TABLE XIII
RQ2 PER INPUT VARIANT AND PERIOD BOUND: INFEASIBLE / DEARER / LARGEST LOSS (%)

Inputs	Bound	n	SL	PF ⁻	PF ⁺	PF ^t	PF ^c	NP
base	none	9	0/0/0.0	0/0/0.0	0/5/8.8	0/2/30.8	0/0/0.0	0/0/0.0
	1.00 μ^*	9	2/3/2.7	6/0/0.0	6/1/9.9	0/0/0.0	2/0/0.0	0/0/0.0
	1.10 μ^*	9	0/3/0.6	6/0/0.0	6/1/9.7	0/0/0.0	2/0/0.0	0/0/0.0
	1.25 μ^*	9	0/4/0.9	6/0/0.0	6/1/18.1	0/0/0.0	2/0/0.0	0/0/0.0
	1.50 μ^*	9	0/1/0.3	3/1/0.1	3/2/10.6	0/2/12.4	2/0/0.0	0/0/0.0
	2.00 μ^*	9	0/3/4.6	0/3/0.9	0/6/9.0	0/2/30.2	0/0/0.0	0/0/0.0
tok2	none	9	0/0/0.0	0/0/0.0	0/5/8.8	0/2/30.8	0/0/0.0	n/r
	1.00 μ^*	9	0/3/2.7	0/1/0.3	0/4/8.0	0/2/28.4	0/0/0.0	n/r
	1.10 μ^*	9	0/3/3.6	0/3/4.6	0/5/7.9	0/2/28.4	0/0/0.0	n/r
	1.25 μ^*	9	0/5/1.3	0/4/1.3	0/6/10.3	0/2/28.4	0/0/0.0	n/r
	1.50 μ^*	8	0/2/4.6	0/0/0.0	0/2/10.1	0/2/30.1	0/0/0.0	n/r
	2.00 μ^*	8	0/0/0.0	0/0/0.0	0/4/8.7	0/2/30.4	0/0/0.0	n/r
chk10	none	9	0/0/0.0	0/0/0.0	0/5/8.8	0/2/30.8	0/0/0.0	n/r
	1.00 μ^*	9	0/2/2.7	6/0/0.0	6/1/11.4	0/0/0.0	2/0/0.0	n/r
	1.10 μ^*	9	0/1/4.8	6/0/0.0	6/1/11.6	0/0/0.0	2/0/0.0	n/r
	1.25 μ^*	9	0/2/1.0	6/0/0.0	6/1/10.6	0/0/0.0	2/0/0.0	n/r
	1.50 μ^*	9	0/2/1.0	3/2/1.0	3/3/10.8	0/2/12.4	2/0/0.0	n/r
	2.00 μ^*	9	0/3/4.6	0/3/0.9	0/6/9.0	0/2/30.2	0/0/0.0	n/r
chk50	none	9	0/0/0.0	0/0/0.0	0/5/8.8	0/2/15.4	0/0/0.0	n/r
	1.00 μ^*	9	2/3/2.7	6/0/0.0	7/0/0.0	2/0/0.0	2/0/0.0	n/r
	1.10 μ^*	9	2/1/0.1	6/0/0.0	6/1/11.3	2/0/0.0	2/0/0.0	n/r
	1.25 μ^*	9	0/2/0.5	6/0/0.0	6/1/18.4	2/0/0.0	2/0/0.0	n/r
	1.50 μ^*	9	0/2/1.3	0/3/1.3	0/5/10.6	0/2/13.7	0/0/0.0	n/r
	2.00 μ^*	9	0/2/4.6	0/0/0.0	0/4/9.0	0/2/15.2	0/0/0.0	n/r

Each cell: instances where the flow is infeasible / feasible but dearer than joint / largest extra cost among the feasible ones in percent (–: none feasible). n : instances with a verified joint optimum at that bound. Inputs: base catalog and WCETs; tok2: two tokens on every verdict edge; chk10, chk50: checker WCETs at 10 % and 50 % of the owner's (base: 30 %). μ^* is the minimum period of each instance under the same inputs. n/r: not run.

Each failure is a proven infeasibility. The 6 instances concerned have a SIL-3 actor, for which PF⁻ and PF⁺ fix 2-of-2 (PF⁻ by catalog order, as the isolated checker ties on its key), and PF^c fixes it where the mixed-SIL pair costs more (Table XIV). The 2-of-2 verdict edge closes a one-token cycle through the owner and a full replica, so the period is at least the sum of their WCETs. This sum is 340 on RASTA, where μ^* is 235, 365 when DIVERSE puts the replica on a slower type, and 512 on Sobel, where μ^* is 333. With a checker, the cycle is shorter. On all RASTA instances, μ^* is the WCET of one actor on its self-edge.

Fig. 8 and Table XV give the least period that each flow reaches under its own restriction. That of PF⁻ and PF⁺ is 1.45 to 1.55 times μ^* on 6 instances, that of PF^c 1.55 times on 2, and that of SL at most 1.09 times (for SL, step 1 minimizes the period too). A lower bound on the period of a pattern-first flow needs no solver: the MCR of the graph that the fixed patterns imply, with every node at its least WCET.

TABLE XIV
PATTERN FIXED BY EACH PATTERN-FIRST RULE (BASE INPUTS)

SIL	FM	PF ⁻	PF ⁺	PF ^t	PF ^c
2	H	Low-SIL D/C	Low-SIL diverse D/C	Low-SIL D/C [‡]	Low-SIL D/C [‡]
2	S	Same-CPU D/C	Low-SIL diverse D/C	Same-CPU D/C	Same-CPU D/C
2	H+S	Low-SIL diverse D/C	Low-SIL diverse D/C	Low-SIL diverse D/C	Low-SIL diverse D/C
3	H	2-of-2 [‡]	2-of-2	High-SIL isolated D/C	High-SIL isolated D/C
3	S	2-of-2	2-of-2	Mixed-SIL D/C	2-of-2
3	H+S	2-of-2 (2); 3-version (NVP) (2)	2-of-2 (2); 3-version (NVP) (2)	Mixed-SIL D/C (2); 3-version (NVP) (2)	2-of-2 (2); 3-version (NVP) (2)

Pattern that each pattern-first rule fixes for an actor of the given required SIL and fault model (FM) of the instance, base inputs; in parentheses the number of actors when an instance set mixes choices (the platform decides which patterns are applicable). SIL 0–1 actors always take none. [‡]The rule’s key ties, and the catalog order decides for at least one actor.

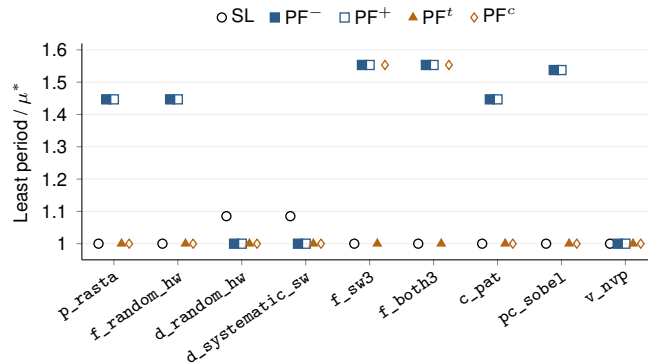


Fig. 8. Least period of each decoupled flow relative to the joint minimum period μ^* (base inputs). The pattern-first rules that fix 2-of-2 for a SIL-3 actor need 1.45 to 1.55 times μ^* ; SL loses period only on the two instances without a SIL-3 actor.

TABLE XV
LEAST PERIOD PER FLOW: INSTANCES ABOVE μ^* , LARGEST RATIO

Inputs	μ^*	SL	PF ⁻	PF ⁺	PF ^t	PF ^c	NP
base	235–333	2, 1.09	6, 1.55	6, 1.55	0, 1.00	2, 1.55	0, 1.00
tok2	235–256	0, 1.00	0, 1.00	0, 1.00	0, 1.00	0, 1.00	0, 1.00
chk10	235–282	0, 1.00	6, 1.82	6, 1.82	0, 1.00	2, 1.55	0, 1.00
chk50	235–384	2, 1.13	6, 1.43	7, 1.43	2, 1.43	2, 1.43	0, 1.00

Least period of each flow under its own restriction (THROUGHPUT), over the 9 instances. Cell: instances whose least period exceeds the joint μ^* , and the largest ratio to μ^* . Range of μ^* over the instances in the second column.

Per actor, it is strengthened by enumerating the core types that satisfy the actor’s SAME and DIVERSE relations. This bound certifies all 149 pattern-first infeasibilities of the base inputs and the variants below (136 without the placement step). In all 45 cases where a pattern-first flow’s least period exceeds μ^* , the bound equals that period exactly (45). So every pattern-first infeasibility holds for the deployed system, independently of the solver.

The result depends on the catalog’s timing and pricing, so we varied both (Table XIII). With two tokens on each verdict edge (tok2, fault reaction within two periods), no flow is infeasible at any bound, and PF^c matches joint exploration everywhere. The remaining gaps are costs, up to 30.4 % for PF^t at 2.00 μ^* . With checkers at 10 % of their owner’s WCET (chk10), the picture is that of the base inputs. With checkers at half their owner’s WCET (chk50), μ^* of the RASTA instances rises, PF^t ties 2-of-2 on added WCET for one actor and picks it, and it fails on 2 instances up to 1.25 μ^* .

Consider a doer one SIL lower than its actor, as the sources allow [11], and priced diverse development

TABLE XVI
THREE APPLICATIONS (m_3app) PER BOUND AND FLOW

Bound	Joint	SL	PF ⁻	PF ⁺	PF ^t	PF ^c	NP
none	5665	+0.4 %	+0.0 %	+0.4 %	+0.0 %	+0.0 %	+0.0 %
1.00 μ_z^*	∞	∞	∞	∞	∞	∞	∞
1.10 μ_z^*	∞	∞	∞	∞	∞	∞	∞
1.25 μ_z^*	5866	∞	∞	∞	+0.0 %	+0.0 %	+0.0 %
1.50 μ_z^*	5852	∞	+0.0 %	+0.0 %	+0.0 %	+0.0 %	+0.0 %
2.00 μ_z^*	5768	∞	+0.0 %	+0.0 %	+0.0 %	+0.0 %	+0.0 %

m_3app . Joint: proven minimum of C_{tot} ; flows: extra cost over joint; ∞ : proven infeasible. μ_z^* : least period of application z alone (Sobel 416, SUSAN 1177, RASTA 235).

(a model variant, Section VI-D). Then PF^c and PF^t reach the joint optimum everywhere, while PF^- and PF^+ stay infeasible as above and lose up to 23.2 and 46.3 % where feasible (Table XIX). Which rule works thus depends on the cost model, while joint exploration was feasible and optimal in every case.

Exactness. Because of the gap of Section IV-E, we re-solved all 1657 RQ2 and RQ4 runs with exact variant B, with the same builds, bounds, objectives, fixings, and workers, and verified every solution. 1578 runs were conclusive in both, and 1577 of them agree in status and value; all 155 infeasibilities, including the 6 SL ones that the cycle bound does not cover, are infeasible under variant B too. The one difference is an SL run in which variant B finds a slightly cheaper design on the same step-1 binding. 79 runs stayed open because variant B reached the time limit, all at loose bounds where the model proves optimality in seconds; in all of them, variant B’s incumbent equals the model’s value. SL depends on which step-1 optimum the solver returns, as step 1 has several optima with different bindings; the canonical tie-break above makes it reproducible.

Three applications. With Sobel (SIL 2), SUSAN (SIL 0), and RASTA on one platform (m_3app), the joint optimum costs 5665 on 6 cores (median 44.0 s, range 37.5 to 63.3 s). Bounds are $\lceil f\mu_z^* \rceil$ per application, with μ_z^* the least period of the application alone (Table XVI). Nothing is feasible for $f \leq 1.10$: alone, each application reaches these periods; together, they compete for the SIL-4 cores. PF^- and PF^+ fail at 1.25, and SL fails at every bound. The variant-B cross-check of the bounded runs agrees on all 21 that are conclusive in both runs, with no better incumbent among the 14 open ones. A longer variant-B run confirmed the joint infeasibility at 1.10 (UNSAT in 994.7 s), so every infeasibility holds for the deployed system.

Promotion. No optimum above promotes an actor, which is why NP equals joint. With the core count bounded below that of the unbounded optimum (Table XVII), every feasible joint optimum promotes (293 to 4550 units), and NP is infeasible in all 19 cases: promotion is what allows consolidation with patterns. One run, m_3app with at most four cores, reached the time limit, and its incumbent fails the verifier by period only (the gap of Section IV-E); it is left out.

D. Model Extensions: Doer SIL, Priced Diversity, Verdict Tokens

The model develops every component and the doer at the actor’s SIL and prices no diverse development; verdict tokens are catalog data. To measure what this conservative approximation costs, we evaluated three extensions in a copy of the model, not in the tool. E1 develops the doer at the actor’s SIL minus the pattern’s `sil_offset`. E2 applies the catalog’s development multipliers per role of the selected pattern, with the diverse multiplier where the pattern places the component DIVERSE. E3 makes the verdict tokens a decision $r \in \{1, 2\}$ per actor under $r\mu \leq \kappa B$, a stand-in for a process safety time κB that no input states. Each solution was verified by the unmodified verifier on an instance into which the decisions were projected. With neutral data, the copy reproduces the stored optima, and E3 with $\kappa = 2$ reproduces the two-token catalog variant on all 40 runs with the same bound.

TABLE XVII
PATTERN INSTANCES WITH A BOUNDED CORE COUNT

Instance	Cores	k	Joint	Promotion	NP
p_rasta	2	1	∞	–	∞
f_random_hw	5	4	3634	293	∞
		3	3919	586	∞
		2	5506	2206	∞
d_random_hw	3	1	∞	–	∞
		2	2297	448	∞
		1	∞	–	∞
d_systematic_sw	2	1	2286	448	∞
f_sw3	4	3	3805	448	∞
		2	5530	2206	∞
		1	∞	–	∞
f_both3	5	4	3659	293	∞
		3	3944	586	∞
		2	5531	2206	∞
c_pat	5	1	∞	–	∞
		4	3634	293	∞
		3	3919	586	∞
pc_sobel	3	2	5506	2206	∞
		1	∞	–	∞
		2	2608	1215	∞
v_nvp	6	1	∞	–	∞
		5	5933	293	∞
		4	6218	586	∞
m_3app	6	3	7805	2206	∞
		2	∞	–	∞
		5	6108	448	∞
		4	t/o	–	∞
		3	10191	4550	∞
		2	∞	–	∞

Cores: of the unbounded joint optimum. k : at most k cores. Joint: proven minimum of C_{tot} , Promotion: its C_{pro} ; NP: the no-promotion flow. ∞ : proven infeasible; t/o: time limit, incumbent not verified (Section VIII).

TABLE XVIII
MODEL EXTENSIONS AGAINST THE MODEL (JOINT OPTIMA)

Extension	Rows	Cheaper	Dearer	Pat.	Time
E1: doer one SIL lower	54	48 (1.9–19.6)	0	15	1.29
E2: priced diversity	54	6 (6.6–7.0)	9 (4.8–37.0)	10	1.03
E1 and E2	54	38 (3.1–18.5)	4 (5.8–9.8)	11	1.26
E3, $\kappa = 1$	45	0	0	3	1.02
E3, $\kappa = 1.5$	45	2 (8.9–11.5)	0	5	1.13
E3, $\kappa = 2$	45	28 (0.1–22.2)	0	15	1.33

Joint optima in a model copy against the model, on the 9 RQ2 instances, without a bound and at the bounds of Section VI-C. Cheaper/Dearer: rows and the range of the change in percent. Pat.: rows whose selected patterns change. Time: median ratio of the solve time. E3: verdict tokens $r \in \{1, 2\}$ per actor with $r\mu \leq \kappa B$.

TABLE XIX
DECOUPLED FLOWS UNDER THE EXTENDED MODELS

Model	SL	PF [−]	PF ⁺	PF ^t	PF ^c
E1+E2, inf.	2/0/0/0/0	6/6/6/3/0	6/6/6/3/0	0/0/0/0/0	0/0/0/0/0
loss (%)	5.9	23.2	46.3	0.0	0.0
E3, $\kappa = 1$, inf.	2/0/0/0/0	6/6/6/3/0	6/6/6/3/0	0/0/0/0/0	2/2/2/2/0
loss (%)	4.6	0.9	18.1	30.2	0.0
E3, $\kappa = 1.5$, inf.	2/0/0/0/0	6/5/5/0/0	6/5/5/0/0	0/0/0/0/0	2/2/2/0/0
loss (%)	13.1	0.9	18.1	30.2	0.0

inf.: infeasible instances at 1.00/1.10/1.25/1.50/2.00 μ^* ; max. loss: largest extra cost over the extended model's joint optimum.

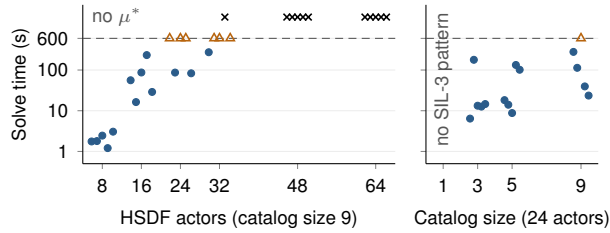


Fig. 9. Solve time on synthetic instances, one mark per graph (counts in Table XX). Filled: proven optimal; open: time limit reached; cross: no period bound μ^* found, so no cost solve. Beyond 16 actors, most searches reach a limit.

Table XVIII summarizes the joint optima. E1 makes 48 of 54 optima cheaper, by up to 19.6%; E2 makes 9 dearer, by up to 37.0% (diverse 2-of-2 and checker), and 6 cheaper (the NVP voter’s multiplier is below one). Together, the patterns of 11 optima change, at 1.26 times the median solve time. E3 with $\kappa = 1$ changes nothing, since two tokens need $\mu \leq B/2$; with $\kappa = 2$, 28 optima become cheaper, by up to 22.2%, at 1.01 times the time of the fixed two-token catalog. Under E1 and E2 together, the mixed-SIL doer/checker (on S and H+S) and the isolated checker (on H) become the cheapest and the lightest SIL-3 choice. So PF^c and PF^t reach the joint optimum on every instance and bound, while PF^- and PF^+ keep their infeasibilities (Table XIX). The cost beyond solve time is in the claims. E1 needs a goal that a lower doer SIL is acceptable, E3 needs a process safety time per function, and each needs the verifier’s requirement and token semantics changed in step.

E. RQ3: Scalability

We generated consistent, single-rate SDF graphs with the random generator of SDF3 [47], 5 per size from 8 to 64 actors, with WCETs for 3 core types. The WCETs are the generator’s own execution times per processor type; a type it leaves out for an actor forbids that binding. The generator has no seed option, so the generated graphs are stored in the artifact; reproducibility comes from them. Each instance has 8 core slots in FCRs of one or two cores (each type spans at least two FCRs, so `DIFFERENT_FCR` can hold), fault model H, and a quarter of its actors each at SIL 3 and SIL 2. A throughput solve of at most 300 s gives the best period μ^* it finds as the period bound; the cost solve then has 600 s. A second sweep varies the catalog size at 24 actors, with prefixes of one fixed pattern order, so the catalogs are nested.

Fig. 9 and Table XX show where exact search stops. All graphs with 8 and 16 actors are proven optimal (medians 1.8 s and 56.1 s). With 24 actors, 2 of 5 are proven (median 84.9 s) and 3 reach the time limit. With 32 actors, 1 is proven, 3 reach the limit, and for 1 the throughput solve finds no period in time, so there is no cost solve; with 48 and 64 actors, this holds for all graphs. μ^* is proven for 4, 2, and 0 of the graphs with 16, 24, and 32 actors. Only 1 timed-out cost solve has a verified incumbent (gap 12.2%); the two 32-actor timeouts fail the verifier by period (Section IV-E), so their gaps are not reported.

Model size limits as much as search (Fig. 10). Flattening the throughput model takes 8.3 s at 24 actors, 61.9 s at 48, and 139.0 s at 64. There, it writes 252 MB of FlatZinc with 259k variables, the compiler needs up to 1588 MB, and flattening alone uses almost half of the throughput budget. The instances of Table VIII flatten in at most 2.4 s to 5k to 22k variables.

The catalog size matters less. With only the pattern `none`, the front-end rejects all 5 instances, as no SIL-3 actor can be served. With 3, 5, and 9 patterns, 5, 5, and 4 of 5 instances are proven, with medians of 13.3, 18.2, and 76.5 s, and the FlatZinc grows from 14786 to 38572 variables.

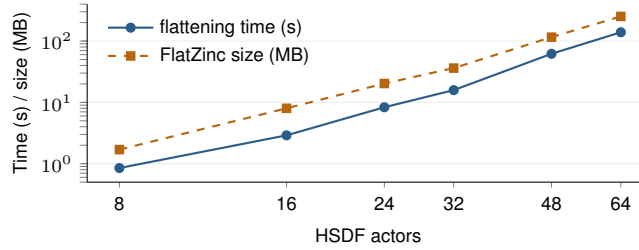


Fig. 10. Cost of flattening the throughput model of the synthetic graphs (median per actor count, logarithmic axes): compile time and FlatZinc size both grow steeply with the number of HSDF actors.

TABLE XX
RQ3 ACTOR SWEEP: OUTCOME, SOLVE TIME, AND MODEL SIZE

Actors	Graphs	μ^* found	μ^* proven	Proven	Time limit	No μ^*	Median (s)	Flatten (s)	FlatZinc (MB)	Memory (MB)
8	5	5	5	5	0	0	1.8	0.8	2	44
16	5	5	4	5	0	0	56.1	2.9	8	86
24	5	5	2	2	3	0	84.9	8.3	20	173
32	4	4	0	1	3	1	273.7	15.8	36	305
48	—	0	0	—	—	5	—	61.9	116	736
64	—	0	0	—	—	5	—	139.0	252	1588

Actor sweep, full catalog. μ^* found/proven: the throughput solve (300 s) found or proved a period. Proven / Time limit: cost solve (600 s) proved optimal / reached the limit. No μ^* : no period found, hence no cost solve. Median: solve time of the proven graphs. Flatten, FlatZinc: median compile time and FlatZinc size of the throughput model; Memory: largest peak memory of the compiler. Graphs: graphs that built (—: all).

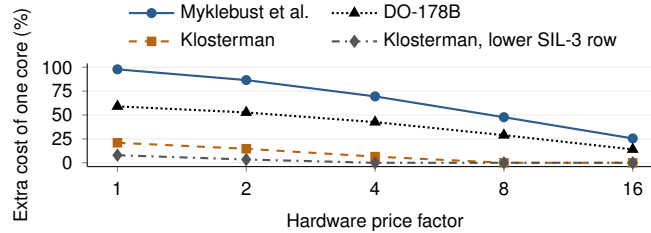


Fig. 11. Extra cost of the cheapest one-core design of the running example without patterns over its optimum, per cost profile and hardware price factor. Only the flat profiles reach zero (one core optimal).

F. RQ4: Sensitivity and Generated Arguments

Cost profile and hardware price. For Fig. 11, we scaled all hardware prices by 1 to 16 and solved the running example without patterns, unbounded and with at most three, two, and one cores (Table XXI). With Klosterman’s flat profile, one core is optimal from factor 8 (4 with his lower SIL-3 row). With the steep profiles of Myklebust et al. and DO-178B, the optimum keeps 3 cores, while the extra cost of one core falls from 98 to 26 % and from 59 to 14 %. The promotion cost of the one-core design depends on the profile, not on the factor (1913, 275, 125, and 1400 units). Fig. 12 shows the same trade-off at factor four over the core bound: the profile sets the price of consolidation, not only the cost level. Before the Klosterman profile took the higher of the two SIL-3 rows, its optimum at factor four used one core; with the higher row it uses 3, and with the lower row 1. A flip of the architecture between profiles therefore rests on one row of a published table. With the original prices and the default profile, the promotion cost of the optimum with at most three, two, and one cores is 0, 293, and 1913 units; the price of each further step grows steeply.

Fault model. The fault model changes placement, not structure. With random hardware faults, the optimum of the SIL-2 variant uses 3 cores, since each checker must run on another core; with systematic

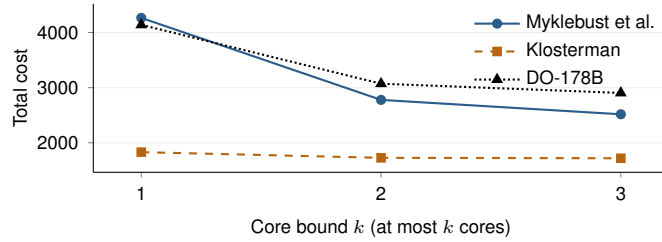


Fig. 12. Total cost of the optimum with at most k cores, running example without patterns at four times the hardware prices. The steep profiles pay heavily for the third consolidation step; Klosterman's profile hardly at all.

TABLE XXI
HARDWARE PRICE SWEEP PER COST PROFILE (TOTAL COST)

Profile	$\times$	Opt. (cores)	$k \leq 3$	$k \leq 2$	$k \leq 1$	Pro.
Myklebust et al.	1	1914 (3)	1914	2199	3786	1913
	2	2115 (3)	2115	2392	3946	1913
	4	2517 (3)	2517	2778	4266	1913
	8	3321 (3)	3321	3550	4906	1913
	16	4929 (3)	4929	5094	6186	1913
Klosterman	1	1116 (3)	1116	1148	1350	275
	2	1317 (3)	1317	1341	1510	275
	4	1719 (3)	1719	1727	1830	275
	8	2470 (1)	2470	2470	2470	275
	16	3750 (1)	3750	3750	3750	275
Klosterman (lower row)	1	1056 (3)	1056	1065	1140	125
	2	1257 (3)	1257	1258	1300	125
	4	1620 (1)	1620	1620	1620	125
	8	2260 (1)	2260	2260	2260	125
	16	3540 (1)	3540	3540	3540	125
DO-178B	1	2301 (3)	2301	2493	3660	1400
	2	2502 (3)	2502	2686	3820	1400
	4	2904 (3)	2904	3072	4140	1400
	8	3708 (3)	3708	3844	4780	1400
	16	5316 (3)	5316	5388	6060	1400

RASTA without patterns on `mixed_noise.xml`, all hardware prices $\times$ the factor. Opt.: the unbounded optimum of C_{tot} and its cores; $k \leq n$: the optimum with at most n cores; Pro.: C_{pro} of the one-core optimum. All proven and verified.

faults, a checker may share its doer's core, and the optimum needs 2. The NVP case study is optimal in 5.5 s on 6 cores.

Communication. On the two instances with the bus, an ideal bus changes neither cost nor μ^* (Table XXII), so TDMA decides nothing there. The communication assumptions matter more. Without the black-channel exemption, the cost rises by 57 and 86 %, while subjecting transfers to their core's rule changes nothing. With cross-checks on the bus, μ^* rises by 19 and 11 % and the superposed graph grows several times. So the exemption decides cost, and the dedicated-link assumption decides throughput. The minimum period of Sobel with a SIL-3 pattern and TDMA, 333, stays within the published bound of 400.

Generated arguments. In the argument for random hardware faults, 16 of 31 leaves (52 %) are Solutions that cite the check log; under both fault classes, 19 of 39 (49 %), and for NVP, 33 of 53 (62 %) (Table XXIII). The others are method, common-cause, fault-reaction, process, and partitioning goals. The share measures what the deployment discharges, not adequacy. No argument has a scenario out of scope: none of the three optima selects a pattern with a scenario outside its fault model, which an earlier tie between equal-cost patterns had hidden.

Other solvers. We ran the Gecode, Chuffed, and HiGHS back ends bundled with MiniZinc 2.10.1,

TABLE XXII
COMMUNICATION VARIANTS

Variant	c_pat			pc_sobel		
	$ G^+ $	C_{tot}	μ^*	$ G^+ $	C_{tot}	μ^*
TDMA bus (default)	71	3349	235	57	1401	333
ideal bus	26	3349	235	15	1401	333
--comm-sil inherit	71	5246	277	57	2608	333
--comm-sil core	71	3349	235	57	1401	333
--comm-scope all	392	3349	279	270	1401	369

C_{tot} : proven minimum of the total cost; μ^* : minimum period (separate THROUGHPUT solve). *inherit*: communication actors take the sender's SIL; *core*: they fall under the SIL-mixing rule of their core; *scope all*: pattern cross-checks are refined for the bus too.

TABLE XXIII
STATISTICS OF THE GENERATED ARGUMENTS

	H	H+S	NVP (H+S)
elements	117	151	165
goals	41	54	54
undeveloped goals	15	20	20
Solutions	16	19	33
leaves	31	39	53
Solutions among leaves (%)	52	49	62
records in the check log	42	51	71
DIVERSE records	0	3	7
scenarios out of scope	0	0	0

Arguments generated for the verified TOTALCOST optima of *f_random_hw* (H), *f_both3* (H+S), and *v_nvp*.

TABLE XXIV
CROSS-SOLVER RUNS ON THE INSTANCES OF TABLE VIII

Instance	CP-SAT (4)	Gecode	HiGHS	Chuffed
p_rasta	3330	–	–	err.
f_random_hw	3349	–	–	err.
d_random_hw	1857	–	2585	err.
d_systematic_sw	1846	–	✓	err.
f_sw3	3365	–	5470	err.
f_both3	3406	–	4349	err.
c_pat	3349	–	–	err.
pc_sobel	1401	–	✓	err.
v_nvp	5680	✓	6330	err.
m_3app	5665	–	–	err.

Second column: the proven optimum of C_{tot} with 4 workers. Other columns: one thread, 300 s. ✓: proven optimal and equal; a number: best verified incumbent at the limit; –: no solution; err.: the solver stopped with an error.

with one thread and 300 s (Table XXIV). Gecode and HiGHS proved only 1 and 2 of the 10 instances, and HiGHS found verified incumbents on 6 in all. Chuffed stopped with an error on all of them, as it cannot handle a product of variables whose sign is not fixed. CP-SAT with one worker proved 8, each equal to the four-worker optimum. So three optima are confirmed by a second solver, and the verifier is the main independent check.

Mutation analysis. We corrupted 421 verified optima on 10 instances with 15 operators (Table XXV). Each mutant breaks one requirement. The fields that follow from the change (WCETs, core SILs, FCR use, costs) are repaired, and every period is set to the ceiling of the mutated deployment's MCR, so the period check cannot catch a mutant by accident. The exact variant B of the model serves as the oracle; it rejects all of them. We also solved the same instances with 18 model mutants, each without one constraint family (163 runs, Table XXVI); 108 solutions violate the original model, in 14 families;

TABLE XXV
CORRUPTED SOLUTIONS PER OPERATOR: REJECTED BY THE VERIFIER

Mutation (what it breaks)	Mutants	First	Current
a placement relation of the selected pattern	32	32	32
a node's SIL below its requirement	50	50	50
a core's provisioned SIL below a node's	39	39	39
a core SIL above its type's ceiling	21	21	21
two SILs on an unpartitioned core	23	23	23
a reported period below the MCR	12	12	12
partitioning on a type without it	23	0	23
a WCET that differs from the table	50	0	50
a static order across cores	19	0	19
two static orders on one core	30	2	30
a cyclic static order	30	2	30
a component of the selected pattern inactive	32	0	32
an inadmissible pattern	30	0	30
a required field missing	20	10	20
a misreported cost	10	0	10
all	421	191	421

Corrupted copies of verified optima on 10 instances; every mutant violates the model (exact variant B as the oracle). First / Current: mutants rejected by the first verifier and by the current one.

the other families never changed a solution. The model alone is not a sound oracle: it rejects 3 correct deployments through the gap of Section IV-E.

The current verifier rejected all 421 corrupted solutions and all 108 violating mutant solutions, without a false alarm, and its intended check fired for every corrupted solution; it also accepts all 4113 verified solutions that the experiments stored. The first verifier, which took WCETs, static orders, activation, and costs from the solution, rejected only 191 and 79. It missed every mutant that changes only a WCET, the order structure, activation, the admissible pattern, partitioning capability, or a cost. The generator then emitted an argument for each of these 230 corrupted solutions. Where it rejected a mutant of these families, the period or deadlock check fired, not a check of the broken constraint.

The generator refused every rejected solution, every catalog mismatch (17 of 17), and every contradicting fault model (18 of 18). On the NVP instance, the inference is ambiguous, so a given fault model has nothing to contradict (3 cases, not counted). It emitted an argument for all 10 uncorrupted solutions. Of 300 mutations of generated arguments in 6 kinds, the first audit caught 250; it missed all 50 Solutions that cite a record index past the end of the log. The current audit, which requires every cited record to exist and to have passed, catches 300.

TABLE XXVI
MODEL MUTANTS PER CONSTRAINT FAMILY

Constraint family removed	Runs	Viol.	First	Current
placement relations, (10)	10	9	9	9
node $SIL \geq$ requirement, (7)	10	10	10	10
node $SIL \leq$ core SIL , (7)	10	0	0	0
core $SIL \leq$ type ceiling, (7)	10	7	7	7
SIL -mixing rule, (8)	10	10	10	10
partitioning capability, (7)	10	9	0	9
WCET follows binding, (2)	10	10	8	10
channel edges of (12)	10	7	7	7
static-order edges of (12)	10	9	9	9
wrap edges of (12)	10	6	6	6
implied load bound, (14)	10	0	0	0
activation, (1)	10	10	7	10
admissible pattern	10	10	1	10
one static order per core, (11)	10	0	0	0
order within one core, (11)	10	9	3	9
memory bound, (15)	10	0	0	0
remote-channel path, (6)	2	1	1	1
period coupling, (13)	1	1	1	1
all	163	108	79	108

One copy of the model per family, the family disabled; optima of the 10 instances (TOTALCOST, 120s). Viol.: solutions that violate the original model (exact variant B as the oracle). First / Current: violating solutions rejected by the first and the current verifier.

VII. LESSONS AND NEGATIVE RESULTS

A. Failure Modes During Development

The recurring lesson of building SafeDSE is that constraint models and their tooling fail silently. Almost every failure below produced a plausible, well-formed output rather than an error, and was caught only by an independent check or by reading the output instead of the exit code. This is the reason for the verifier of Section V.

Silent under-constraint. In MiniZinc, `=` binds tighter than `\ /`, so an assignment of a disjunction without parentheses leaves the variable free whenever the second disjunct holds. A disjunction placed outside a quantifier instead of inside it once collapsed the safety design space of all actors to two points: either every actor took one option or every actor took the other. The solutions looked reasonable and came from a trivially small space; per-actor pattern variables with per-actor guards, the root of guarded superposition, came out of this failure. A static order encoded as an open chain, without the wrap edge, silently dropped the bound “period at least the core’s load” and reported a graph with many actors on two cores at full pipeline rate. A direct edge of a remote channel kept next to its communication path was a shortcut past the whole transfer delay, which made the bus free. The same shape reappeared with explicit voters, where a surviving direct edge let the output bypass the voter (Sections IV-A and IV-B).

Wrong modeling intuitions. Communication actors were first placed in the static order of their core and counted in its load, which gave wrong periods for the communication benchmarks. A transfer overlaps computation and only lengthens the period when the bus is the bottleneck. Three optimization intuitions did not survive measurement. Symmetry breaking was not the bottleneck, a flattened encoding of the WCET lookup was slower than the nested one, and tying the activity of communication actors to the remote flag made solving slower. The standing rule became to measure first, and to build the deliberately unsound variant first to bound what an optimization can gain. Pre-unfolded HSDF benchmarks also lose the provenance of each node’s SDF actor, which the copy-order symmetry needs, so multi-rate SDF input is preferred.

Wrong results from tooling. A killed installation left a MiniZinc binary that existed and crashed, and every solve returned no solution; it looked like an unsatisfiable model. Instances renamed or dropped from

the build script survived in the output directory and were solved against a newer model. A cross-solver sweep printed nothing for minutes because it skipped timeouts silently, which hid that on the pattern instances no second solver answered at all (Section VI-F).

Fluent but false output. The worst failures were arguments that passed every structural check and were false. One was an argument for an actor with three-version programming that argued it “by development process alone” because the catalog lacked the pattern. The other was the possibility of a wrong fault-model flag yielding claims of coverage the design never had. Both are now refusals (Section V-B). Pre-checks were also weaker than they looked. A diversity check that counted all core types passed on a platform whose third type was capped below the required SIL. Its first fix fired too eagerly and rejected an instance whose other patterns were placeable. A satisfiability pre-check that does not fire is not evidence of satisfiability, and one that fires too eagerly is worse than none.

B. Defects Found by the Evaluation

Running the experiments of this report on the tool found further defects (Table XXVII). Each was found by a disagreement between two independent computations: model and verifier, two oracles, two runs, or a result and its expected value. Each defect was fixed in the tool, and every affected experiment was re-run; the results in this report are from the fixed tool. Some of them changed results that a first draft of the evaluation had reported. Before the slot-sharing fix, a checker that shared a slot with a full replica was priced as a replica, which raised the minimum period of the RASTA pattern instances. Before the no-promotion fix, the NP flow activated every pattern slot and appeared to lose heavily against joint exploration. Before the fix of an implied constraint, tight period bounds made feasible instances infeasible. And an argument statistic depended on a tie between equal-cost patterns. The last case is the general lesson of the evaluation. A quantity of a non-unique optimum (its period, its pattern, an argument derived from it) is not a property of the instance, so the evaluation reports only proven objective values, or fixes the tie explicitly.

TABLE XXVII
DEFECTS FOUND WHILE RUNNING THE EVALUATION, AND THEIR FIXES

Part	Defect	Found by	Resolution
Model	Implied constraint on heavy node pairs ignored activity, so an owner could not share a core with its own inactive slot; the front-end's least core count also counted slots	tight bounds made instances with a verified design infeasible	guarded by the activity of both nodes; core count from always-active nodes
Model	Recurring pattern cost counted once per slot of an actor	reading the model for the paper	summed over application actors only
Model	No-promotion rule also bound inactive slots, which then had to be active	every NP solution activated all slots	rule guarded by activity; NP re-run, now equal to joint
Model	A path through an inactive slot can raise a design's least period above its MCR	verifier rejected two time-limited incumbents by period	documented; exact variants kept as patches and used as a cross-check (Section IV-E)
Front-end	Slots shared by position between patterns, so a checker took a replica's WCET and the reverse	a slot whose role and WCET type differed	slots shared only by equal WCET type; all RQs re-run
Front-end	Owner type of a replica not found for a multi-rate owner	building multi-rate synthetic graphs	lookup in the owner's application, ignoring the copy suffix; regression test
Front-end	<code>sil_offset</code> read as raising a component above its actor's SIL, against the sources	a review of the safety semantics	every component at the actor's SIL; the offset may only lower the doer (unused by the model)
Verifier	A zero-weight token-free cycle through inactive slots reported as deadlock	solver and verifier disagreed on an optimum	checks the deployed MSAG, inactive nodes removed, orders spliced
Verifier	Local refined channel lost its direct edge; wrong tokens on a remote path	reading the verifier against the model	edges as in the model
Verifier	Required the integer period to equal a fractional MCR	a correct design failed verification	compares with $\lceil \text{MCR} \rceil$
Verifier	Took WCETs, orders, activation, and costs from the solution	mutation analysis (Section VI-F)	re-derives every field; six new record kinds
Generator	Named the first oracle Karp, though it is Lawler's search	reading the oracle	texts corrected
Generator	Audit accepted an undeveloped goal without a reason, and a citation past the end of the log	reading the audit; mutation analysis	both rejected
Generator	Claimed an attained SIL; cited <code>DIFFERENT</code> records for separate FCRs	a review of the safety semantics	one claim per relation, common-cause goals, catalog preconditions instead of SIL
Catalog	Klosterman's SIL 3 took the lower of two table rows; the NVP multiplier key matches no role	reading the source; the extension study	higher row (lower one as a variant); key harmless, as its default equals the stated value
Harness	SL depended on which step-1 optimum the solver returned	two runs of the same SL row differed	canonical step 1 (ties by period, then core index)
Harness	More than one thread started eight workers; without intermediate solutions, a time-limited run reported no incumbent	process monitor; a run with no solution at the time limit	worker count passed explicitly; intermediate solutions always on

Each fix is a commit of the tool's history with a regression test where one applies; the affected experiments were re-run afterwards (Section C).

VIII. DISCUSSION AND LIMITATIONS

The results hold within the model and under its assumptions. Timing assumes sound WCETs and self-timed execution in one static order per core, without preemption or time-triggered tables, so applications that share a core need a common period. Replicas and checkers always fire, so a detected fault changes token values, not the schedule. The latency constraint is a periodic-phase estimate, not a worst-case bound; the verifier simulates the transient and flags a larger worst case. The fault model leaves out transient, timing, omission, and interconnect faults, common causes across FCRs, latent faults, and degraded modes. SafeDSE computes no failure probabilities, and `DIVERSE` means distinct core types, not an independent design.

The model develops the doer at the actor's SIL, although the sources allow a lower one [11], and does not price diverse development. In a model variant (Section VI-D), the lower doer SIL made joint optima up to 19.6% cheaper, priced diversity up to 37.0% dearer, and both together took 1.26 times the median solve time. The cost profiles are our reading of published estimates; that the profile can flip the architecture rests on one row of Klosterman's table (Section VI-F). Cross-checks between pattern components are assumed to use a dedicated link, and transfers are treated as non-safety under the black-

channel principle; both assumptions are stated in every generated argument, and RQ4 shows that each decides a result.

Because inactive slots keep their channels, the least period of a design can exceed the MCR of its deployment (Section IV-E), so the model can reject a design that meets a bound. This occurred in time-limited incumbents (synthetic graphs with 32 actors and `m_3app` with at most four cores) and in one proven safety-last design under the two-token variant; no reported result is affected. The exact variant B confirmed every infeasibility and all but one conclusive RQ2 and RQ4 result, but solves more slowly. With one thread, Gecode and HiGHS proved only 1 and 2 of the 10 instances of Table VIII, and Chuffed stopped with an error, so the verifier is the main independent check. The pattern instances come from three small applications, and exact search reaches its limit at a few tens of HSDF actors (Section VI-E).

On the tooling side, the fault-model inference is ambiguous on small catalogs, such as the NVP case study, where all three fault models reproduce the admissible patterns; the generator then names none and cannot list out-of-scope scenarios. The generator emits one argument per instance without GSN modules or away goals, which becomes large for several applications. Platforms with multi-core cards solve more slowly than ones with single-core cards, because value precedence is sound only within a card. Finally, a per-actor “not promotable” attribute would let a designer forbid promotion for selected actors only, between the global switch and free promotion.

A faster exact formulation, priced pattern-dependent SILs, verdict tokens as decisions under a process safety time, a declared interconnect with routed cross-checks, GSN modules, and industrial case studies are future work.

IX. CONCLUSION

Whether a safety architecture pattern protects a function and fits its period depends on where its components run and how they are scheduled. SafeDSE decides the pattern, its deployment, and the platform in one constraint model over a fixed superposed graph and proves the optimum within that model. On the benchmarks, the value of joint exploration lay in feasibility under a throughput bound: which pattern-first rule works depends on catalog timing and pricing, and joint exploration needs no such rule. Verifier checks discharge the argument’s claims on placement, SIL provisioning, isolation, and timing; claims on detection, common causes, fault reaction, and the development process remain with the engineer.

APPENDIX A

PROOF OF THE POTENTIAL CHARACTERIZATION

Fix the binding, the static orders, the pattern selection, the TDMA allocation, and the buffer sizes. The MSAG is then a fixed digraph $M = (V, E)$ with integer node times $T_i \geq 0$ and integer tokens $tok_{ij} \geq 0$ on each edge. Self-edges and wrap edges hold one token, order edges none, and buffer back-edges the buffer sizes. Within one connected component, all periods are equal by (13), so one μ applies to every edge of the component, and the argument is per component.

Proposition 2 (Proposition 1, full form): A vector pot with $pot_j \geq pot_i + T_i - tok_{ij} \mu$ for all $(i, j) \in E$ exists iff every directed cycle C of M satisfies $\sum_{i \in C} T_i \leq \mu \sum_{(i,j) \in C} tok_{ij}$. If T , tok , and μ are integers, a feasible pot exists iff an integral one does.

Proof: (Only if.) Let $C = (i_0, i_1, \dots, i_{m-1}, i_0)$. Summing the m inequalities of the edges of C gives $\sum_k pot_{i_{k+1}} \geq \sum_k pot_{i_k} + \sum_k T_{i_k} - \mu \sum_k tok_{i_k i_{k+1}}$, indices modulo m . Every node of C appears once as head and once as tail, so the two potential sums are equal, and $0 \geq \sum_{i \in C} T_i - \mu \sum_{(i,j) \in C} tok_{ij}$.

(If.) Set $w_{ij} = T_i - \mu tok_{ij}$. By assumption, every cycle has weight $w(C) \leq 0$. Add a node s with an edge (s, j) of weight zero to every $j \in V$; s has no incoming edge, so no new cycle arises. For every j , let ℓ_j be the largest weight of an s - j walk. It is finite: removing a cycle from a walk does not decrease its weight, as $w(C) \leq 0$, so the maximum is attained by a simple path, and there are finitely

many. Further, $\ell_j \geq 0$ because of the edge (s, j) . For each edge $(i, j) \in E$, a best s - i walk followed by (i, j) is an s - j walk of weight $\ell_i + w_{ij}$, hence $\ell_j \geq \ell_i + w_{ij}$, i.e., $pot = \ell$ satisfies every inequality. If T , tok , and μ are integers, every w_{ij} and every ℓ_j is an integer. Shifting all potentials by a constant keeps them feasible, so the model's anchor of one potential at zero loses nothing, and as $0 \leq \ell_j \leq \sum_i T_i$, the bounded domain of the potentials in the model holds ℓ . ■

Corollary 1 (period). For fixed T and tok , the feasible μ are exactly those with

$$\mu \geq \max_{C: tok(C) > 0} T(C)/tok(C) = \text{MCR},$$

provided every token-free cycle has $T(C) = 0$. The least real μ is the MCR, and the least integer μ is $\lceil \text{MCR} \rceil$, which the model reports.

Corollary 2 (deadlock). A token-free cycle with $T(C) > 0$ makes the system infeasible for every μ . A deadlocking design is thus excluded by the same inequalities, without separate deadlock machinery.

Corollary 3 (zero-weight cycles). Token-free cycles with $T(C) = 0$ satisfy the condition with $0 \leq 0$ and do not constrain μ . In SafeDSE, they arise only through neutralized nodes. A verifier that checks the graph with all nodes would flag them as deadlock; the verifier therefore checks the deployed graph (Section V-A).

Corollary 4 (relation to the deployed graph). Remove the inactive nodes and splice each static order around them. Every cycle of this deployed graph maps to an MSAG cycle with the same total time and tokens. A spliced order edge $a \rightarrow b$ becomes $a \rightarrow s \rightarrow b$ with $T_s = 0$ and no tokens on either edge, and a spliced wrap edge keeps its token. Hence the model's least μ for a design is never below the deployed graph's MCR, and neutralization cannot make the period optimistic. The converse does not hold in general. A channel of an unselected pattern into an inactive slot, followed by the slot's order edge, is an MSAG path without a counterpart in the deployed graph. A cycle through it can raise the least μ above the deployed MCR. Removing the channel edges at inactive endpoints gives exactly the deployed MCR on the designs where this was observed, which is what exact variant B achieves inside the model (Section IV-E).

Corollary 5 (other objectives). When μ is not the objective, the inequalities give only $\text{MCR} \leq \mu \leq$ the period bound. The reported μ equals $\lceil \text{MCR} \rceil$ only if the search assigns it its least value; the evaluation enforces this by the period step (Section VI-A), and the verifier checks it.

APPENDIX B INSTANCES AND PARAMETERS

All pattern instances except Sobel and `m_3app` use the RASTA speech front end with the required SILs of Fig. 2a: `frontEnd` and `compJah` at SIL 3, `backEnd` at SIL 2, and all other actors at SIL 1. Table XXVIII lists their other inputs. Every time value is an integer in the benchmark's unit. On the card platforms, the WCET of an application actor is its nominal execution time in the benchmark file times the cycle factor of the core mode (Table II), rounded. The WCET of a checker or a voter is 30 % of its owner's nominal time times the cycle factor, and a full replica has its owner's WCET. These WCET tables are placeholders generated by the tool, not measurements of a real platform. Sobel on Rosvall's platform uses the published WCETs. The synthetic graphs of RQ3 take the generator's execution times per processor type. Every node has the same baseline development cost (Table IV); card and core prices are illustrative (Table II); the recurring cost units of each pattern are in Table V. All costs are in one unitless cost unit.

APPENDIX C ARTIFACT MAP

Every number of this report comes from CSV files that the experiment scripts wrote on the machine of Section VI-A. The script `make_macros.py` turns them into the macros used in the text and in the

TABLE XXVIII
INPUTS OF THE PATTERN INSTANCES

Instance	Application(s)	Platform file	Fault model	Bus	Remarks
p_rasta	RASTA	mixed.xml	H	ideal	certified partitioning on the R-card
f_random_hw	RASTA	mixed_noiso.xml	H	ideal	–
d_random_hw	RASTA	mixed_noiso.xml	H	ideal	SIL 3 actors lowered to SIL 2
d_systematic_sw	RASTA	mixed_noiso.xml	S	ideal	SIL 3 actors lowered to SIL 2
f_sw3	RASTA	mixed_3type.xml	S	ideal	–
f_both3	RASTA	mixed_3type.xml	H+S	ideal	–
c_pat	RASTA	mixed_noiso.xml	H	TDMA	–
pc_sobel	Sobel	Rosvall's	H	TDMA	get_pixel at SIL 3, others SIL 1; period bound loosened for the cost runs
v_nvp	RASTA	mixed_4type.xml	H+S	ideal	separate catalog with the NVP pattern
m_3app	Sobel, SUSAN, RASTA	mixed_noiso.xml	H	ideal	Sobel at SIL 2, SUSAN at SIL 0; one period per application

All instances use the catalog `patterns.yaml` unless noted and the cost profile of Myklebust et al. RQ4 also uses the running example without patterns on `mixed_noiso.xml` (cost profiles, price sweep) and on `mixed_costly.xml` (Fig. 12). The variants of RQ2 replace only the catalog (`tok2`) or the WCET tables (`chk10`, `chk50`).

TABLE XXIX
EXPERIMENTS, DRIVERS, DATA, AND WHERE THEY APPEAR

Experiment	Driver	CSV (experiments/results/)	Used in
RQ1 benchmarks	<code>harness.py rq1</code>	<code>e1_parity</code>	Table XI
RQ1 DeSyDe	<code>desyde_cmp.py</code> , <code>desyde/build.sh</code>	<code>e7_desyde</code> , <code>e7_safedse</code>	Table X
RQ2 flows	<code>harness.py rq2</code> , <code>rq2tight</code> , <code>rq2thr</code> , <code>rq2var</code> , <code>rq2reps</code> , <code>pftable</code>	<code>e2_joint</code> , <code>e2_tight</code> , <code>e2_thr</code> , <code>e2_var</code> , <code>e2_reps</code> , <code>e2_pfchoice</code>	Figs. 6 to 8, Tables XII to XV
RQ2 certificates	<code>certify_pf.py</code>	<code>e2_cert</code>	Section VI-C
RQ2 exactness	<code>exact_model.py</code> , <code>exact_check.py</code>	<code>e2_exact</code> , <code>e2_exact_sl</code>	Section VI-C
Three applications, promotion	<code>breadth.py multi</code> , <code>patcons</code>	<code>e5_multi</code> , <code>e5_multi_exact</code> , <code>e5_multi_exact_long</code> , <code>e4_patcons</code>	Tables XVI and XVII
Model extensions	<code>modelext_build.py</code> , <code>modelext.py</code>	<code>e8_modelext</code>	Tables XVIII and XIX
RQ3	<code>gen_synthetic.py</code> , <code>harness.py rq3</code> , <code>breadth.py flat</code>	<code>e3_scal</code> , <code>e3_flat</code>	Figs. 9 and 10, Table XX
RQ4 sensitivity	<code>harness.py rq4</code> , <code>breadth.py price</code> , <code>comm</code>	<code>e4_sens</code> , <code>e4_price</code> , <code>e4_comm</code>	Figs. 11 and 12, Tables XXI and XXII
RQ4 arguments	<code>gsn_stats.py</code>	<code>e4_gsn</code> , <code>gsn/*.json</code>	Table XXIII, Figs. 4 and 5
Pipeline, solvers	<code>breadth.py pipe</code> , <code>xsolver</code>	<code>e4_pipe</code> , <code>e4_xsolver</code>	Tables IX and XXIV
Mutation analysis	<code>mutation.py</code>	<code>e6_mut</code> , <code>e6_oracle</code> , <code>e6_recheck</code>	Tables XXV and XXVI

Drivers are in `experiments/`; each is resumable and writes one CSV row per run. The catalog, scenario, tactic, profile, and platform tables are generated from the tool's data files; the GSN excerpts by `make_gsn_figure.py` from the stored arguments.

data files of the plots; `make_report.py` writes the tables and plot data that only this report uses. A row enters a number only if it is optimal (or proven infeasible) and passed the verifier. Table XXIX lists, per experiment, the driver, its CSV files, and where the results appear. The tool itself (model, front-end, verifier, generator, regression tests, and data) is in the artifact repository (<https://github.com/SBU-CPS-Lab/SafeDSE>). The experiment scripts, input variants, generated synthetic graphs, CSV files, and the generated tables and plot data are in the experiments repository (<https://github.com/SBU-CPS-Lab/SafeDSE-experiments>).

ACKNOWLEDGMENT

The authors used the generative AI tools Claude (Anthropic), DeepSeek, and Gemini (Google) to test and refactor the software and to improve the text of the paper and of this report. The authors reviewed all results of this use and take full responsibility for the content.

REFERENCES

- [1] A. Burns and R. I. Davis, “A survey of research into mixed criticality systems,” *ACM Computing Surveys*, vol. 50, no. 6, pp. 1–37, 2018.
- [2] E. A. Lee and D. G. Messerschmitt, “Synchronous data flow,” *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, 1987.
- [3] IEC, “Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 3: Software requirements,” IEC 61508-3:2010, ed. 2.0, 2010.
- [4] T. Myklebust, T. Stålhane, and B. Haugset, “Software development cost related to different SILs in an agile development environment,” in *Proc. 33rd Int. System Safety Conf. (ISSC)*, San Diego, CA, USA, 2015.
- [5] A. Bonfietti, M. Lombardi, M. Milano, and L. Benini, “Maximum-throughput mapping of SDFGs on multi-core SoC platforms,” *Journal of Parallel and Distributed Computing*, vol. 73, no. 10, pp. 1337–1350, 2013.
- [6] K. Rosvall, N. Khalilzad, G. Ungureanu, and I. Sander, “Throughput propagation in constraint-based design space exploration for mixed-criticality systems,” in *Proc. 9th Workshop Rapid Simulation and Performance Evaluation: Methods and Tools (RAPIDO)*, Stockholm, Sweden, 2017, pp. 1–8.
- [7] K. Rosvall and I. Sander, “Flexible and tradeoff-aware constraint-based design space exploration for streaming applications on heterogeneous platforms,” *ACM Transactions on Design Automation of Electronic Systems*, vol. 23, no. 2, pp. 21:1–21:26, Nov. 2017.
- [8] R. Jordão, M. Becker, I. Sander, and I. Söderquist, “Design space exploration for safe and optimal mapping of avionics functionality on IMA platforms,” in *Proc. IEEE/AIAA 42nd Digital Avionics Systems Conf. (DASC)*, 2023, pp. 1–9.
- [9] A. Armoush, “Design patterns for safety-critical embedded systems,” Ph.D. dissertation, RWTH Aachen University, Aachen, Germany, 2010.
- [10] C. Preschern, N. Kajtazovic, and C. Kreiner, “Building a safety architecture pattern system,” in *Proc. 18th Eur. Conf. Pattern Lang. Programs (EuroPLoP)*. Irsee, Germany: ACM, 2013, pp. 1–55.
- [11] P. Koopman and M. Wagner, “Challenges in autonomous vehicle testing and validation,” *SAE International Journal of Transportation Safety*, vol. 4, no. 1, pp. 15–24, 2016.
- [12] D. Tămaş-Selicean and P. Pop, “Design optimization of mixed-criticality real-time embedded systems,” *ACM Transactions on Embedded Computing Systems*, vol. 14, no. 3, pp. 1–29, 2015.
- [13] D. Parker, M. Walker, L. S. Azevedo, Y. Papadopoulos, and R. E. Araújo, “Automatic decomposition and allocation of safety integrity levels using a penalty-based genetic algorithm,” in *Recent Trends in Applied Artificial Intelligence (IEA/AIE 2013)*, ser. Lecture Notes in Artificial Intelligence, vol. 7906. Springer, 2013, pp. 449–459.
- [14] D. Zaheri and H.-C. Reuss, “ASIL-decomposition based resource allocation optimization for automotive E/E architectures,” in *Proc. IEEE Intelligent Vehicles Symp. (IV)*, 2025, pp. 1496–1503.
- [15] M. Glaß, M. Lukasiwycz, T. Streichert, C. Haubelt, and J. Teich, “Reliability-aware system synthesis,” in *Proc. Design, Automation and Test in Europe Conf. (DATE)*, 2007, pp. 1–6.
- [16] S.-h. Kang, H. Yang, S. Kim, I. Bacivarov, S. Ha, and L. Thiele, “Static mapping of mixed-critical applications for fault-tolerant MPSoCs,” in *Proc. 51st ACM/EDAC/IEEE Design Automation Conf. (DAC)*, 2014, pp. 1–6.
- [17] P. Nuzzo, N. Bajaj, M. Masin, D. Kirov, R. Passerone, and A. L. Sangiovanni-Vincentelli, “Optimized selection of reliable and cost-effective safety-critical system architectures,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 2109–2123, 2020.
- [18] Y. Gil Dantas, T. Munaro, C. Carlan, V. Nigam, S. Barner, S. Fan, A. Pretschner, U. Schöpp, and S. Tverdyshev, “A toolchain for synthesizing and validating safety architectures,” *SN Computer Science*, vol. 4, 2023, art. no. 335.
- [19] R. Hawkins, I. Habli, D. Kolovos, R. Paige, and T. Kelly, “Weaving an assurance case from design: A model-based approach,” in *Proc. IEEE 16th Int. Symp. High Assurance Systems Engineering (HASE)*, 2015, pp. 110–117.
- [20] E. Denney and G. Pai, “Tool support for assurance case development,” *Automated Software Engineering*, vol. 25, no. 3, pp. 435–499, 2018.
- [21] N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck, and G. Tack, “MiniZinc: Towards a standard CP modelling language,” in *Principles and Practice of Constraint Programming (CP 2007)*, ser. Lecture Notes in Computer Science, vol. 4741. Springer, 2007, pp. 529–543.
- [22] L. Perron and F. Didier, “CP-SAT, version 9.15,” Google OR-Tools. [Online]. Available: https://developers.google.com/optimization/cp/cp_solver/, 2026.
- [23] Assurance Case Working Group, “Goal structuring notation community standard, version 3,” Safety-Critical Systems Club, Tech. Rep. SCSC-141C, May 2021. [Online]. Available: <https://scsc.uk/gsn>
- [24] C. Hanen and A. Munier, “A study of the cyclic scheduling problem on parallel processors,” *Discrete Applied Mathematics*, vol. 57, no. 2–3, pp. 167–192, 1995.

- [25] S. Stuijk, T. Basten, M. C. W. Geilen, and H. Corporaal, “Multiprocessor resource allocation for throughput-constrained synchronous dataflow graphs,” in *Proc. 44th ACM/IEEE Design Automation Conf. (DAC)*, 2007, pp. 777–782.
- [26] A. H. Ghamarian, M. C. W. Geilen, S. Stuijk, T. Basten, B. D. Theelen, M. R. Mousavi, A. J. M. Moonen, and M. J. G. Bekooij, “Throughput analysis of synchronous data flow graphs,” in *Proc. 6th Int. Conf. Application of Concurrency to System Design (ACSD)*, 2006, pp. 25–36.
- [27] A. Bonfietti, M. Lombardi, M. Milano, and L. Benini, “Throughput constraint for synchronous data flow graphs,” in *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR)*, ser. Lecture Notes in Computer Science, vol. 5547. Springer, 2009, pp. 26–40.
- [28] K. Rosvall and I. Sander, “A constraint-based design space exploration framework for real-time applications on MPSoCs,” in *Proc. Design, Automation and Test in Europe Conf. (DATE)*, 2014, pp. 1–6.
- [29] R. Jordão, M. Becker, and I. Sander, “IDeSyDe: Systematic design space exploration via design space identification,” *ACM Transactions on Design Automation of Electronic Systems*, vol. 29, no. 5, pp. 1–45, 2024.
- [30] R. Govindarajan, E. R. Altman, and G. R. Gao, “A framework for resource-constrained rate-optimal software pipelining,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 11, pp. 1133–1149, 1996.
- [31] V. Izosimov, P. Pop, P. Eles, and Z. Peng, “Design optimization of time- and cost-constrained fault-tolerant distributed embedded systems,” in *Proc. Design, Automation and Test in Europe (DATE)*, 2005, pp. 864–869.
- [32] C. Bolchini and A. Miele, “Reliability-driven system-level synthesis for mixed-critical embedded systems,” *IEEE Transactions on Computers*, vol. 62, no. 12, pp. 2489–2502, 2013.
- [33] B. P. Douglass, *Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems*. Boston, MA, USA: Addison-Wesley, 2002.
- [34] S. Vestal, “Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance,” in *Proc. 28th IEEE Real-Time Systems Symp. (RTSS)*, 2007, pp. 239–243.
- [35] D. Tămaş-Selicean and P. Pop, “Design optimization of mixed-criticality real-time applications on cost-constrained partitioned architectures,” in *Proc. 32nd IEEE Real-Time Systems Symp. (RTSS)*, 2011, pp. 24–33.
- [36] Y. Papadopoulos, M. Walker, D. Parker, E. Rüde, R. Hamann, A. Uhlig, U. Grätz, and R. Lien, “Engineering failure analysis and design optimisation with HiP-HOPS,” *Engineering Failure Analysis*, vol. 18, no. 2, pp. 590–608, 2011.
- [37] T. Terzimehić, S. Barner, Y. Gil Dantas, U. Schöpp, V. Nigam, and P. Ke, “Safety-aware deployment synthesis and trade-off analysis of Apollo autonomous driving platform,” in *Proc. IEEE 20th Int. Conf. Software Architecture Companion (ICSA-C)*, 2023, pp. 309–316.
- [38] S. Voss and B. Schätz, “Deployment and scheduling synthesis for mixed-critical shared-memory applications,” in *Proc. 20th IEEE Int. Conf. Workshops Eng. Computer Based Systems (ECBS)*, Scottsdale, AZ, USA, 2013, pp. 100–109.
- [39] K. Becker, B. Schätz, M. Armbruster, and C. Buckl, “A formal model for constraint-based deployment calculation and analysis for fault-tolerant systems,” in *Proc. Int. Conf. Software Engineering and Formal Methods (SEFM)*, ser. LNCS, vol. 8702, 2014, pp. 205–219.
- [40] A. Gacek, J. Backes, D. Cofer, K. Slind, and M. Whalen, “Resolute: An assurance case language for architecture models,” in *Proc. ACM SIGAda Annu. Conf. High Integrity Language Technology (HILT)*, Portland, OR, USA, 2014, pp. 19–28.
- [41] R. Bloomfield and J. Rushby, “Assurance 2.0: A manifesto,” in *Proc. 29th Safety-Critical Systems Symp. (SSS)*, M. Parsons and M. Nicholson, Eds. York, UK: Safety-Critical Systems Club, 2021, pp. 85–108.
- [42] IEC, “Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 4: Definitions and abbreviations,” IEC 61508-4:2010, ed. 2.0, 2010.
- [43] —, “Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 6: Guidelines on the application of IEC 61508-2 and IEC 61508-3,” IEC 61508-6:2010, ed. 2.0, 2010.
- [44] European Union Aviation Safety Agency, “AMC 20-193: Use of multi-core processors,” Annex I to ED Decision 2022/001/R (AMC-20 Amendment 23), Jan. 2022.
- [45] IEC, “Industrial communication networks – profiles – Part 3: Functional safety fieldbuses – general rules and profile definitions,” IEC 61784-3:2021, ed. 4.0, 2021.
- [46] A. Dasdan, “Experimental analysis of the fastest optimum cycle ratio and mean algorithms,” *ACM Transactions on Design Automation of Electronic Systems*, vol. 9, no. 4, pp. 385–418, 2004.
- [47] S. Stuijk, M. Geilen, and T. Basten, “SDF³: SDF for free,” in *Proc. 6th Int. Conf. Application of Concurrency to System Design (ACSD)*, 2006, pp. 276–278.