Architecting Self-Adaptive Systems with Learned and Symbolic Components

Nicolas Schuler[0009−0009−6688−9416]

KASTEL
Karlsruhe Institute of Technology (KIT), Germany
nicolas.schuler@kit.edu

Abstract. AI-enabled self-adaptive systems increasingly combine learned components with symbolic components. Learned components, such as perception, prediction, or language models, handle uncertain, high-dimensional inputs, while symbolic components, such as planners, monitors, policies, and runtime constraints, provide explicit structure and checkable decision logic. However, the architectural boundary at which learned outputs become symbolic facts, constraints, or actions often remains underspecified. Scores, labels, embeddings, or language fragments may cross component boundaries without explicit assumptions about calibration, freshness, ontology, or policy compliance, forcing downstream components to rely on implicit interpretations. This dissertation investigates how such *learned-symbolic boundaries* can be specified, monitored, and evolved as first-class architectural seams in self-adaptive systems. The proposed abstraction is the *seam contract*: a connector-level specification consisting of typed boundary objects, versioned constraint bundles, runtime monitors, and decision traces. The research follows a design-science methodology and uses a Goal–Question–Metric evaluation to assess expressiveness, runtime assurance, and bounded revalidation in a simulated perception-to-decision pipeline and a policy-governed LLM assistant. At this stage, the framework and evaluation plan are defined; next steps are refining the contract model, implementing *seamKIT*, and instantiating both scenarios.

Keywords: Software Architecture, AI, Neuro-symbolic Architectures

1 Introduction

AI-enabled self-adaptive systems increasingly combine *learned components* (software elements whose behavior is induced from data, such as perception, prediction, or language models) with *symbolic components* (elements governed by explicit rules, logic, or constraints, such as planners, monitors, and policies) [6, 3]. Perception models classify objects, prediction models estimate future states, and language models interpret user intent or recommend actions. Reliable self-adaptation, however, depends on explicit goals, policies, planners, monitors, safety rules, and runtime constraints. This dissertation addresses the architectural challenges that emerge when these learned and symbolic parts interact.

Consider an autonomous mobile system operating in a shared space; a learned perception component may detect an ambiguous object and output object probabilities, location, velocity, and timestamp information. A downstream planner or runtime

monitor then determines whether the system should slow down, stop, reroute, or request human assistance. In such scenarios, the critical issue is not only the accuracy of the perception result but also the assumptions that symbolic components may legitimately make when using it.

In many system architectures, this hand-off remains underspecified. Learned outputs may traverse component boundaries as labels, scores, embeddings, strings, or ad hoc data structures, while downstream components must decide whether and how to rely on them. As a result, runtime failures may arise at component interfaces rather than within the components themselves.

Problem Statement: At *learned-symbolic hand-offs*, where learned outputs are consumed as symbolic facts, constraints, or actions, AI-enabled self-adaptive systems often leave *contract-relevant assumptions* implicit. These include assumptions about calibration, freshness, ontology, policy compliance, and permitted fallback behavior. This ambiguity complicates the assignment of assurance responsibilities to specific connectors or components and can necessitate broad revalidation when models, policies, constraints, or downstream symbolic components change. The issue is fundamentally *architectural rather than solely algorithmic*. Enhancing an individual model does not clarify what downstream components may assume about its outputs, and the addition of a runtime monitor does not, by itself, define the contract governing the hand-off from perception or interpretation to planning and action. Therefore, this dissertation treats learned-symbolic boundaries as *first-class architectural concerns* whose assumptions, runtime evidence, and impact on evolution must be specified explicitly.

The research is organized around three **RQs**:

> **RQ1 – Specification.** How can requirements, policies, uncertainty assumptions, and domain assumptions be represented as modular contracts at learned-symbolic boundaries?
>
> **RQ2 – Runtime assurance.** How can such contracts be monitored at runtime and linked to fallback or mitigation behavior to support controlled and auditable adaptation?
>
> **RQ3 – Evolution.** How can self-adaptive systems add, replace, or modify learned and symbolic components while localizing the required retesting, conformance checking, and model or policy revalidation?

The anticipated contribution is an architectural method centered on hand-off contracts, defined as connector-level specifications of learned-symbolic hand-offs using typed boundary objects, versioned constraint bundles, runtime monitors, and decision traces. The dissertation evaluates whether such contracts make assumptions explicit, support runtime governance, and localize revalidation during architectural evolution.

2 Related Work

The research problem is situated at the intersection of software architecture, machine-learning-enabled systems, neuro-symbolic artificial intelligence, and self-adaptive systems. Collectively, these areas contribute explicit interfaces and connectors, diagnoses of ML-specific coupling, hybrid computational mechanisms, and runtime

adaptation and assurance techniques. However, they remain fragmented regarding learned-symbolic hand-offs and do not yet provide an architecture-level account of contract specification, runtime monitoring, and localized revalidation.

Software architecture and interface specification. Foundational work in software architecture emphasizes decomposition, information hiding, separation of concerns, and explicit connectors [9, 1]. These principles motivate stable component boundaries as a basis for maintainability and evolution. Architectural description languages and contract-based design, including assume-guarantee and behavioral contracts, provide mechanisms for specifying component interactions, assumptions, and constraints [8, 2]. However, these approaches are most directly applicable when externally visible behavior can be described through functional interfaces, protocols, or comparatively stable component contracts. Learned components complicate this view because their outputs may be uncertain, dependent on training and operating distributions, subject to drift, or valid only under specific statistical conditions. The unresolved architectural question is therefore how learned-symbolic boundaries should be represented when uncertainty, statistical validity, and runtime adaptation become part of the interface itself.

Machine-learning-enabled systems and technical debt. Research on ML technical debt shows that ML-enabled systems often exhibit hidden coupling, glue code, unstable dependencies, and the "changing anything changes everything" effect [10]. This line of work further highlights weak abstraction boundaries and architecture-level technical debt in ML-enabled systems. It motivates the central problem addressed in this dissertation: learned components are difficult to evolve when their assumptions, dependencies, and downstream obligations are not isolated. However, technical-debt research primarily diagnoses these risks and proposes engineering practices for managing them. It does not by itself define a connector-level specification that makes learned-symbolic assumptions explicit, supports runtime monitoring, and localizes revalidation when learned or symbolic components change.

Neuro-symbolic AI and orchestration frameworks. Neuro-symbolic AI integrates learned models or representations with explicit symbolic structures such as rules, logics, or knowledge bases [3, 7]. This work demonstrates that learned and symbolic mechanisms can be combined at the computational level. A related but distinct line of work on prompt- and tool-orchestration frameworks decomposes AI application behavior into steps such as retrieval, prompting, tool invocation, validation, and memory management [5]. These approaches are valuable for constructing hybrid AI pipelines, but they often focus on task-level integration or execution flow rather than architecture-level contract specification. As a result, assumptions governing interactions between steps may remain embedded in prompts, wrappers, validators, or task-specific code. In the context of self-adaptive systems, these approaches do not by themselves provide the durable architectural specifications needed for runtime assurance and system evolution.

Self-adaptive systems and runtime assurance. Self-adaptive systems research provides established architectural models for runtime adaptation, including feedback loops, runtime models, and MAPE-K-style monitors, analyzers, planners, and executors [4]. Adjacent assurance techniques, including runtime verification, shielding, and assurance cases, provide mechanisms for checking execution against expected

properties, constraining unsafe actions, and documenting assurance arguments during operation. However, these approaches usually require the monitored properties, managed interfaces, and responsibility boundaries to be explicit. In hybrid learned-symbolic systems, the main challenge is that the assumptions governing learned outputs and their downstream symbolic use are often underspecified. This dissertation, therefore, treats runtime assurance not only as a monitoring problem but also as an *architectural specification problem* at the learned-symbolic seam.

Taken together, prior work provides strong building blocks, but not yet a unified architecture-level treatment of learned-symbolic hand-offs in self-adaptive systems. What remains underdeveloped is a method for treating such hand-offs as first-class architectural seams whose assumptions can be specified, whose conformance can be monitored, and whose revalidation impact can be localized. This gap motivates the seam-contract abstraction developed below.

3 Research Method

This dissertation follows a design-science research approach because the intended contribution is an engineered architectural artifact rather than only an empirical characterization of existing systems. The artifact under development consists of a seam-contract model, an architectural method for deriving and using such contracts, and a prototype toolchain supporting specification, runtime monitoring, and change-impact analysis for learned-symbolic boundaries in self-adaptive systems. The work is organized into four iterative activities: problem characterization, objective definition, artifact construction, and evaluation. These activities are iterated as evaluation results sharpen the problem characterization, design objectives, and artifact.

Problem characterization. The first activity characterizes recurring learned-symbolic boundary problems in self-adaptive systems, using related work and two planned scenarios as problem sources. The guiding hypothesis is that assurance and evolution problems often emerge when learned outputs cross architectural boundaries without explicit assumptions, validity conditions, guarantees, and responsibility assignments. The two scenarios are a simulated perception-to-decision pipeline and a policy-governed enterprise assistant based on an LLM. They are used to elicit boundary information, failure modes, and change cases that the seam-contract model must represent, including exchanged data fields; uncertainty, confidence, and freshness information; semantic labels or ontology references; active policies and validity constraints; permitted actions and fallback behavior; and evidence for audit or debugging.

Objective definition. The second activity operationalizes the problem characterization into four design objectives for the seam-contract artifact.

O1 – Specifiability: make learned-symbolic assumptions, validity conditions, and responsibility assignments explicit at architectural boundaries.
O2 – Adaptability: support runtime checking of these assumptions and link violations to configured fallback or mitigation behavior during adaptation.
O3 – Auditability: record decision-relevant evidence, including contract versions and checked constraints, in a form suitable for audit and debugging.

O4 – Evolvability: identify the contracts, monitors, tests, and components that require revalidation when models, policies, constraints, or symbolic components change.

O1 addresses specification (RQ1), O2 and O3 address runtime assurance and auditability (RQ2), and O4 addresses evolution and localized revalidation (RQ3).

Artifact construction. The third activity constructs and refines the seam-contract model, the associated architectural method, and a research prototype toolchain, provisionally named *seamKIT*. The prototype is intended to support three operations: deriving seam contracts from requirements, policies, interface schemas, domain assumptions, validity conditions, and responsibility assignments; generating or configuring runtime monitors that check boundary conformance, link violations to fallback or mitigation behavior, and emit decision-trace evidence; and identifying which contracts, monitors, tests, and components require revalidation after changes to models, policies, constraint bundles, or symbolic components. The artifact is refined through scenario instantiations so that deficiencies in contract expressiveness, monitorability, or change-impact analysis feed back into the model and method.

Evaluation. The evaluation uses the Goal–Question–Metric approach to derive criteria from the research questions and design objectives. It combines analytical checks over contract instances with scenario-based evaluation in the two planned systems.

For RQ1, the evaluation assesses whether the contract model can express assumptions, validity conditions, constraints, and responsibility assignments at learned-symbolic boundaries. The questions ask whether the model represents scenario-derived boundary assumptions, supports internal and cross-component consistency checks, and remains understandable during structured architecture walkthroughs. Evidence will include: (M1) assumption coverage by category, (M2) unresolved or ambiguous assumptions, (M3) detected inconsistencies, (M4) qualitative feedback on contract understandability.

For RQ2, the evaluation assesses whether seam contracts support runtime monitoring, mitigation, and auditability at learned-symbolic boundaries. It distinguishes directly monitorable assumptions from assumptions requiring offline or statistical evidence. The questions ask whether monitors detect boundary violations before or when downstream components consume boundary objects, whether violations trigger configured mitigation behavior, whether decision traces record the relevant evidence, and what overhead is introduced relative to the underlying learned components and how it scales. Evidence will include: (M1) violation-detection precision and recall by category, (M2) mitigation correctness, (M3) latency and throughput overhead relative to ordinary schemas and application-specific checks, (M4) and trace completeness and fidelity for contract versions, checked constraints, evidence, and decision status.

For RQ3, the evaluation assesses whether seam contracts can identify and localize the required revalidation scope during system evolution. The questions ask whether representative change cases, including model replacement, threshold updates, policy changes, constraint-bundle changes, and symbolic-component changes, can be mapped to an explicit and justified affected frontier. These cases include both contract-preserving and contract-breaking changes, including schema-preserving model replacements that may still alter error distributions. Evidence will include:

(M1) affected-artifact precision and recall against a manually established expected frontier, (M2) frontier size relative to ordinary schemas, wrappers, validators, and application-specific checks, (M3) regressions escaping localized validation.

Together, the analytical and scenario-based parts triangulate evidence about contract expressiveness, runtime assurance, and evolution support. The analytical part applies consistency and change-impact checks over seam-contract instances, while the scenario-based part instantiates contracts, violations, and representative change cases in the two planned systems. The evaluation is not intended to establish statistical generality, but to assess whether the abstraction is expressive, monitorable, auditable, and useful for localizing revalidation in representative learned-symbolic settings.

4 Proposed Solution and Expected Results

The proposed solution is an architectural method for making learned-symbolic hand-offs explicit at the connector level in self-adaptive systems. The central abstraction is the *seam contract*: a connector-level specification for a learned-symbolic hand-off in which learned outputs are consumed as symbolic facts, constraints, or actions. A seam contract defines the boundary objects that may cross the hand-off, the assumptions, validity conditions, constraints, and responsibility assignments that govern their use, how runtime violations are linked to configured fallback or mitigation behavior, and which evidence is recorded for audit, debugging, and later revalidation.

Conceptually, a seam contract consists of four elements. First, a typed boundary-object schema specifies the fields and structured metadata exchanged at the hand-off. Unlike an ordinary interface data type, it may include uncertainty-related, provenance, and semantic metadata, such as confidence scores, timestamps, model or calibration identifiers, and ontology references. Second, a versioned constraint bundle specifies the active rules, policies, safety conditions, calibration assumptions, freshness constraints, validity conditions, and responsibility assignments that govern the use of boundary-object instances. Versioning makes policy, domain-rule, or validity-condition changes explicit architectural changes. Third, a runtime monitor checks boundary-object instances against the active constraint bundle during operation and signals violations to configured fallback or mitigation behavior. Here, monitoring is an engineering mechanism for boundary-conformance and policy checking that may also apply runtime verification to individual constraints where formal properties exist. Fourth, a decision trace records the contract version, checked constraints, observed evidence, triggered mitigation, and produced decision status for audit, debugging, and later revalidation. These elements can build on established techniques: schema validation for boundary objects, policy and rule engines for constraint bundles, runtime verification and drift or calibration checks for monitors, and documentation and provenance artifacts for boundary metadata and decision traces.

The method applies seam contracts across the lifecycle: architecture design, runtime operation, and system evolution. During architecture design, seam contracts help elicit and document assumptions, validity conditions, constraints, and responsibilities at learned-symbolic hand-offs. During runtime operation, monitors check boundary-object instances against the active constraint bundle, trigger configured fallback or mit-

igation on violations, and emit decision traces. During system evolution, contract versions and dependency links identify and justify the revalidation frontier: the contracts, monitors, tests, and components needing revalidation after a model, policy, constraint-bundle, or symbolic-component change. Localized revalidation is justified only when contract-relevant assumptions remain unchanged, or changes are explicitly versioned and propagated through dependency links. Distributional shifts concealed by an unchanged schema may still require offline or statistical evidence and broader validation.

The dissertation is expected to yield five research results. ER1 is a seam-contract model for learned-symbolic hand-offs in self-adaptive systems, consisting of boundary-object schemas, versioned constraint bundles, runtime monitors, and decision traces. ER2 is a derivation method that maps requirements, policies, domain assumptions, validity conditions, and responsibility assignments to seam-contract elements. ER3 is a runtime monitoring and mitigation mechanism that checks boundary-object instances against active constraint bundles, triggers configured fallback or mitigation behavior, and emits decision traces. ER4 is an evolution mechanism that uses contract versions and dependency links to identify and justify the revalidation frontier after changes to models, policies, constraint bundles, or symbolic components. ER5 is analytical and scenario-based evidence about the extent to which seam contracts express scenario-derived assumptions and validity conditions (RQ1), detect boundary violations and produce complete and faithful traces (RQ2), and identify a justified revalidation frontier relative to ordinary interface schemas, wrappers, validators, and application-specific checks (RQ3).

The problem framing, seam-contract concept, design objectives, and evaluation strategy are established; the next steps are to refine the contract model, implement seamKIT, instantiate the two scenarios, and evaluate against the GQM criteria.

5 Critical Reflection

The central challenge is to make seam contracts expressive enough for assurance, auditability, and localized revalidation, yet lightweight enough for realistic architectures. The reflection focuses on four open design issues: seam placement, specification precision and monitorability, localized revalidation, and evaluation credibility.

The first issue concerns where learned-symbolic hand-offs should be made explicit. Too few seam contracts leave critical assumptions implicit and allow change impact to propagate across components; too many create specification, monitoring, maintenance, and performance overhead. In some end-to-end learned designs, tight statistical coupling may even be intentional. The research must therefore develop criteria for justified seams, including safety or policy relevance, volatile policies or domain rules, decision-relevant uncertainty, auditability requirements, and anticipated component replacement.

The second issue concerns specification precision and monitorability. Traditional contracts express preconditions, postconditions, protocols, and invariants, but learned components also introduce assumptions about operating distributions, calibration, confidence thresholds, data freshness, drift, and context coverage. A useful seam-contract style must capture these assumptions without creating false precision or

excessive modeling effort. It should distinguish directly monitorable properties, such as timestamps or explicit policy constraints, from assumptions that require statistical tests, historical evidence, offline validation, or revalidation obligations, such as calibration, distributional validity, or semantic alignment.

The third issue concerns localized revalidation during system evolution. Seam contracts may identify an affected revalidation frontier, but the frontier must not be minimized at the cost of missed behavioral changes. A stable boundary-object schema and unchanged constraint bundle may justify localized revalidation for some contract-preserving changes, but they are insufficient when behavior behind the seam changes. For example, replacing a perception model may preserve the output schema while changing calibration, error distributions, or context-specific performance. The research must therefore determine which evidence, dependency links, and contract-version changes justify localized revalidation and which changes require broader offline, statistical, or system-level evaluation.

The fourth issue concerns evaluation credibility. The approach should be compared with realistic alternatives, such as model-centric pipelines and orchestration based on ordinary schemas, wrappers, validators, prompts, or application-specific checks. The central question is not whether seam contracts eliminate uncertainty, but whether they make relevant assumptions explicit, detect monitorable violations, produce faithful traces, and identify justified revalidation frontiers. Feedback is sought on seam placement, specification precision, credible evaluation scenarios, and whether a later ML-based extension should compile learned-model specifications directly into seam contracts, monitors, or evidence obligations.

References

1. Allen, R., Garlan, D.: A formal basis for architectural connection. ACM Trans. Softw. Eng. Methodol. **6**(3), 213–249 (Jul 1997). https://doi.org/10.1145/258077.258078
2. Benveniste, A., et al.: Contracts for system design. Found. Trends Electron. Des. Autom. **12**(2–3), 124–400 (Mar 2018). https://doi.org/10.1561/1000000053, https://doi.org/10.1561/1000000053
3. Garcez, A.d., Lamb, L.C.: Neurosymbolic AI: The 3rd wave. Artificial Intelligence Review **56**(11), 12387–12406 (2023). https://doi.org/10.1007/s10462-023-10448-w
4. Kephart, J., Chess, D.: The vision of autonomic computing. Computer **36**(1), 41–50 (2003). https://doi.org/10.1109/MC.2003.1160055
5. Khattab, O., et al.: Dspy: Compiling declarative language model calls into self-improving pipelines. CoRR **abs/2310.03714** (2023). https://doi.org/10.48550/ARXIV.2310.03714
6. Lewis, G.A., et al.: Software Architecture and Machine Learning (Dagstuhl Seminar 23302). Dagstuhl Reports **13**(7), 166–188 (2024). https://doi.org/10.4230/DagRep.13.7.166
7. Manhaeve, R., et al.: Deepproblog: Neural probabilistic logic programming. In: NeurIPS. vol. 31. Curran Associates, Inc. (2018)
8. Meyer, B.: Applying 'design by contract'. Computer **25**(10), 40–51 (1992). https://doi.org/10.1109/2.161279
9. Parnas, D.L.: On the criteria to be used in decomposing systems into modules. Commun. ACM **15**(12), 1053–1058 (Dec 1972). https://doi.org/10.1145/361598.361623
10. Sculley, D., et al.: Hidden technical debt in machine learning systems. NeurIPS **28** (2015)