Ref. Ares(2026)10188876 - 29/10/2026



Deliverable Data	
Deliverable number	D5.4 (D15)
Deliverable name	Report describing an integrated adaptive optics software architecture
Work Package	WP5
Lead WP/deliverable beneficiary	Patrick Audebert (LULI) / deliverable beneficiary: GSI
Type and dissemination level	Report, public
Deliverable status	
Submitting author	Jonas Ohland
Verified (WP leader)	Patrick Audebert
Approved (Coordinator)	Vincent Bagnoud
Due date of deliverable	31.10.2026

Funded by the European Union

This project has received funding by the European Union's HORIZON-INFRA-2022-TECH-01 call under grant agreement number 101095207

Table of Contents

About THRILL 2

Executive Summary 3

External Resources & Permanent Digital identifier 4

Main Report 5

Appendix: COALA 17

Disclaimer

This document is part of the deliverables from the project THRILL, which has received funding from the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the granting authority can be held responsible for them.

About THRILL

The THRILL project deals with providing new schemes and devices for pushing forward the limits of research infrastructures (RI) of European relevance and ESFRI landmarks. To do so, the project partners have identified several technical bottlenecks in high-energy high-repetition-rate laser technology that prevent it from reaching the technical readiness level required to technically specify and build the needed devices, and guaranteeing sustainable and reliable operation of such laser beamlines at the partnering RIs. Advancing the technical readiness of these topics is strategically aligned with the long-term plans and evolution of the ESFRI landmarks FAIR, ELI (-BL) and Eu-XFEL, and RI APOLLON, bringing them to the next level of development and strengthening their leading position.

The project is focused and deliberately restricted to three enabling technologies, which require the most urgent efforts and timely attention by the community: high-energy high-repetition-rate amplification, high-energy beam transport and optical coating resilience for large optics. To reach our goals, the major activity within THRILL will be organized around producing several prototypes demonstrating a high level of technical readiness. Our proposal is addressing not yet explored technical bottlenecks - such as transport over long distances of large-aperture laser beams via relay imaging using all-reflective optics - and aims at proposing concrete steps to increase the performances and effectiveness of the industrial community through the co-development of advanced technologies up to prototyping in operational environments.

The project is not only pushing technology, it is also offering an outstanding opportunity to train a qualified work force for RIs and industry. With this in mind, the structure of THRILL promotes synergetic work, fast transfer to industry and integrated research activities at the European level. Access to the RIs will be granted as in-kind contribution.

Executive Summary

Adaptive Optics (AO) systems in high-energy lasers span from semi-static optimization to timing-critical real-time stabilization. Such systems are often found in the form of self-sufficient specialist AO systems, which run in parallel to the facility-wide control systems. A conditional integration problem arises when an AO procedure needs optical context, information, permitted action or operational authority across their usual boundaries. This motivates a product-independent software architecture that allocates responsibilities, identifies the needed context and interfaces, and permits only the control relationships justified by the selected AO integration use case.

This technical report describes the development of a rule-based framework for such a software system. A suitable architecture is expressed as four proposed broadly applicable requirements: qualified ownership, physically meaningful control context, bounded integration interfaces, and case-appropriate control relationships.

Their completeness and representative cross-facility generality remain unvalidated as specialist-native and facility-native solutions remain legitimate under feasible circumstances. Where neither is suitable, we propose the COntroller for Adaptive Laser Architecture (COALA) approach: an optional framework for building a subsystem with an assigned coordination role between the expert subsystems and the facility-control system. We illustrate this approach via a combined real-time AO + steering + semi-static integrated-AO arrangement, while using an exploratory COALA implementation to provide insight and evidence to this approach.

This work has been done by the WP5 team of THRILL at GSI and Ecole Polytechnique/LULI in collaboration with the SCExAO team of the Subaru telescope in Hawaii, illustrating the interdisciplinary link and synergy between AO systems for high-power lasers and astronomy.

External Resources & Permanent Digital identifier

The results have been summarized in a format that is aimed at reaching a larger scientific audience. Therefore, the results have been made available as a report located on a repository with a digital identifier for later referencing.

Report DOI: 10.5281/zenodo.23691048

Exploratory repository:

Link: https://git.gsi.de/phelix/coala/coala_exploratory
Referenced commit: **eed495d6**
Referenced tag: **THRILL_D5.4**
License: Apache 2.0

In the following, we added a version of the report dated Oct. 29th 2026.

Integrating Adaptive Optics into High-Energy Laser Systems: A Responsibility-Based Architecture

Jonas B. Ohland[1,*], Leon Dauerer[1], Mostafa Hassan[1],
Nathalie Lebas[2], Elyes Chalali[2], Vincent Déo[3,4],
Olivier Guyon[3,5], Dimitris Papadopoulos[2], Vincent Bagnoud[1,6,7]

[1] Plasma Physics/PHELIX, GSI Helmholtz Center for Heavy Ion Research GmbH, Darmstadt, Germany

[2] Laboratoire d'Utilisation des Lasers Intenses, École Polytechnique, Palaiseau Cedex, France

[3] Subaru Telescope, National Astronomical Observatory of Japan, National Institute of Natural Sciences, Hilo, HI, USA

[4] Optical Sharpeners SAS, La Ciotat, France

[5] Steward Observatory, University of Arizona, Tucson, AZ, USA

[6] Institute for Applied Physics, Technical University of Darmstadt, Germany

[7] Helmholtz Institute Jena, Germany

[*]Corresponding author: j.b.ohland@gsi.de

This work is licensed under a "CC BY-SA 4.0" license.

29 October 2026

Abstract

Adaptive Optics (AO) in high-energy laser systems spans semi-static optimization and timing-critical real-time stabilization, which are often expressed in the form of capable specialist AO systems parallel to facility-wide control systems. A conditional integration problem arises when an AO procedure needs optical context, information, permitted action or operational authority across their usual boundaries. This motivates a product-independent software architecture that allocates responsibilities, identifies the needed context and interfaces, and permits only the control relationships justified by the selected AO integration use case.

This work took on the development of a rule-based framework for such a software system. A suitable architecture is expressed as four proposed broadly applicable requirements: qualified ownership, physically meaningful control context, bounded integration interfaces, and case-appropriate control relationships.

Their completeness and representative cross-facility generality remain unvalidated as specialist-native and facility-native solutions remain legitimate under feasible circumstances. Where neither is suitable, we propose the COntroller for Adaptive Laser Architecture (COALA) approach: an optional framework for building a subsystem with an assigned coordination role between the expert subsystems and the facility-control system. We illustrate this approach via a combined real-time AO + steering + semi-static integrated-AO arrangement, while using an exploratory COALA implementation to provide insight and evidence to this approach.

1 Introduction

High-energy laser facilities support high-energy-density experiments, including laser-driven shock-compression studies and inertial-confinement-fusion research [1, 2]. Their scientific value depends on the beam delivered to the experiment as well as nominal pulse energy. The former is often controlled via Adaptive Optics (AO), where especially slow, semi-static, commercial AO systems were employed for beam optimization in the past. Recently, Real-Time Adaptive Optics (RTAO) has received increasing attention in order to increase the reproducibility of beam quality across shots [3, 4]. This is especially important for lasers with slow repetition rates, often on the hour-scale, making each delivered shot operationally significant [3].

Semi-static AO measures and corrects a wavefront during setup, periodically, or before a shot, and may hold the Deformable Mirror (DM) command afterwards. It remains useful for static and slowly changing aberrations. RTAO on the other hand continuously rejects dynamic disturbances through a timing-critical loop; Apollon Real-Time Adaptive Optics (ARTAO) demonstrated this operating mode in a high-power-laser beamline [4]. Both operation modes hold their own challenges: semi-static work requires well-defined and reproducible expert procedures, while RTAO raises demands in timing, stability and machine safety. While these challenges have been partly addressed in terms of system integration and automation in singular facilities, these approaches have not yet reached a maturity level where automated, reliable beam control is consistently available across facilities worldwide. This is what we refer to as the *integration problem*.

The gaps that are created by this integration problem become particularly visible when a selected AO procedure needs information or action from more than one responsible system and no single responsibility owner already covers all required optical knowledge, access, authority, performance, and fitness. Here, specialist subsystems like fast AO loops do not have the overview required to solve the problems at hand, while equipping the facility-control system with the logic to orchestrate the envisioned procedures may require in-depth knowledge and engineering levels unfeasible for many facilities. The architectural question is therefore how to assign the responsibilities and define bounded exchanges while preserving qualified local control and established supervision.

In this work, we establish the requirements for a suitable software architecture and identify some possible solutions for this problem. We use the term *software architecture* in order to encapsulate the proposed allocation of functions to responsible systems, the information and

permitted requests at their interfaces, the authority retained by each system, and a possible concrete software realization. Our assessment is based on published AO and control-system literature [3–8], material from a dedicated workshop held in Oxford in 2025, including a live survey, and the experimental findings of the ARTAO project [4].

In Section 2, we begin with identifying the existing capabilities in beam control and its integration, deriving the technology gaps that are present with respect to the maturity of integration approaches. This analysis leads to the identification of a few exemplary use cases for a deeper level of integration in Section 3, from which we derive four core requirements for a feasible beam control integration framework. These ultimately lead to a proposed approach of a dedicated COntroller for Adaptive Laser Architecture (COALA), which we describe in Section 4, accompanied by an exploratory implementation discussed in Sections 5 and 6, with a detailed documentation of the exploratory COALA implementation in Annex A. Finally, we conclude this study in Section 7.

2 AO in laser operation: existing capability and the remaining integration gap

2.1 Existing control capabilities

Nowadays, beam control is mainly distributed along two realms: specialist AO control and the facility control system.

Specialist AO control. Dedicated AO controllers already perform demanding expert functions. Semi-static AO is often controlled via capable (yet only loosely integrated) commercial software tools, combined with the expert knowledge of AO operators.

The RTAO demonstration at Apollon in France used the open-source real-time-control framework Compute and Control for Adaptive Optics (CACAO), developed for high-rate AO processing. Its published implementation uses modular processes and shared image streams, supports CPU/GPU computation at kilohertz rates, and includes specialist functions such as response-matrix acquisition, command-channel combination, timing calibration, and telemetry access [5].

These results demonstrate a capable class of expert AO subsystems, taking local responsibility with powerful tools and dedicated knowledge. They are deliberately no orchestration tools beyond that. The architectural consequence for integration is therefore that qualified specialist functionality needs to be preserved rather than relocated or degraded.

Facility control. Large experimental facilities commonly employ distributed control or Supervisory Control and Data Acquisition (SCADA) infrastructure. Hardware and data are exposed through control-system interfaces so that operators and automation can observe state, issue commands, sequence procedures, and coordinate devices. The degree and location of automation vary markedly between facilities. Experimental Physics and Industrial Control System (EPICS) and TANGO are established examples of distributed device monitoring and control [6, 7]. Karabo provides a further example in which device control, data acquisition, analysis, feedback, and facility-scale coordination are integrated at the European XFEL [8]. These frameworks are examples of the general capability class and can legitimately own even substantial coordination when it has the knowledge, interfaces, authority, and performance needed by the selected laser case.

The starting point is therefore two capable domains. An expert beam-control subsystem may own a demanding correction loop; facility control may own device infrastructure, machine state, operator interaction, sequencing, and established protection.

2.2 Reported integration practice

In January 2025, the LPA Special Workshop on Intelligent Systems was held in Oxford, UK. Over the course of a week, topics such as machine learning based control and advanced optimization schemes for laser particle accelerators were discussed, which inherently involves system integration. As a part of this event, we organized the *THRILL Adaptive Optics Integration Workshop* specifically concerned with adaptive optics in lasers, automation, beam control, and integration practice. Its programme contained facility contributions, and a companion survey recorded practitioner observations. Table 1 summarizes contrasting arrangements reported in the programme and survey. Since the response set was small, self-selected, and question-dependent, while abstracts did not use a common validation method, the material supports qualitative examples, not field-wide prevalence or consensus.

Table 1: Qualitative Oxford observations and their integration relevance. Rows are workshop-programme or survey observations, interpreted here for architectural use.

Topic	Examples or observations reported	Relevance to integration
AO implementation style	Facility-native control, commercial/vendor systems, custom or mixed implementations, and dedicated high-rate subsystems	The implementation landscape is heterogeneous; no single ownership pattern can be assumed.
Programmatic access	Direct facility AO control, API-based access, local custom software, and vendor- or specialist-operated tools	The integration surface differs between systems and may be bounded by the access actually available.
Routine operation	Specialist involvement, operator training, manual optimization, and differing degrees of automation	Capability does not by itself guarantee repeatable routine use.

This operational friction is especially relevant to conventional or semi-static optimization. A facility may have an effective wavefront sensor and deformable mirror yet still rely on an expert to interpret residual aberrations, choose an optimization sequence, adjust references, or decide when a result is acceptable. Training and graphical interfaces can help without eliminating iterative judgment or trial-and-error.

RTAO creates a different emphasis. Its timing-critical loop requires an implementation qualified for the needed behavior; the specialist subsystem is one such demonstrated class. The system-level question is how that loop remains physically valid and operationally coordinated as the surrounding laser state changes. The following focused ARTAO case makes that distinction concrete.

2.3 ARTAO as a focused case study

The ARTAO system was reported as the first implementation of RTAO in a high-power laser. It used an off-spectral pilot beam for continuously probing the wavefront, a Shack-Hartmann Wavefront Sensor (WFS), CACAO on a Real-Time Computer (RTC), and a piezoelectric DM upstream of the last two amplifiers in the Apollon beamline [4]. During short runs it produced strong correction relative to the wavefront-sensor reference.

During extended closed-loop operation, however, the loop became unstable after roughly 30 minutes due to lateral misregistration. Manual adjustment of a steering mirror outside the DM conjugation plane between the DM and WFS realigned the DM image and restored stability.

A separate experiment deliberately applied deformable-mirror strokes roughly an order of magnitude larger than those used in normal correction. After propagation to a non-conjugate near-field

image plane, the beam showed a corresponding fluence imprint [4]. The authors treated this as a possible concern for hot spots and pulse-energy limits in that optical arrangement. This test establishes a non-local consequence: an actuator state that improves or imposes a wavefront at one plane can alter the intensity distribution at another. The experiment did not deliver amplified shots with the loop closed because the observed risks had not been resolved [4].

The main takeaway from these experiments is that locally successful correction with good local error metrics is not inherently sufficient to qualify and guarantee indefinite stable operation, leading to the need for a more integrated approach, e.g. including non-conjugate diagnostics and control units.

2.4 What remains unresolved at system level

The ARTAO observations separate two conditions that an integrated laser case may need to evaluate:

- **Local optical validity** means that the measurement, reference, registration or beam position, and actuator command remain related as assumed by the controller.
- **Propagated-beam acceptability** means that the beam remains acceptable at the downstream or intermediate plane relevant to the experiment or equipment. A local wavefront metric may inform both assessments, but it cannot by itself establish propagated-beam acceptability.

This distinction also clarifies the safety boundary. A local AO controller can evaluate limits and state available inside its responsibility. Established protection and interlocks remain authoritative; operator authorization and shot permission remain facility responsibilities, based on the machine state. At the same time, some safety-relevant assessments may nevertheless require combined optical and system information. For example, the AO registration state together with steering range, or a correction command together with an intermediate-plane diagnostic. The specific criteria, thresholds, emergency actions or facility-specific safety logics remain case-specific engineering and validation work.

Routine operability is a distinct concern, constrained by applicable protection functions. Selected beam control operations should be expressible as well-defined and reproducible programmatic procedures at a level suitable for non-specialist operators or facility automation. In some cases the required procedure may remain entirely within facility control or the specialist AO system. In others, it may require information or actions spanning the specialist loop, additional diagnostics and actuators, facility state and protection, and the routine operating procedure.

The evidence therefore converges on a conditional **cross-boundary control problem**. Briefly summarized, the starting point can be described as follows:

- Capable expert AO systems exist;
- Capable facility-control systems exist;
- Practical arrangements are heterogeneous;
- Semi-static optimization can retain manual and specialist friction;
- RTAO can depend on registration while producing consequences outside the locally sensed plane.

Selected AO procedures may consequently require information or action across established control boundaries. The main question is therefore: Who should own such a functionality, and what must be preserved when it crosses those boundaries?

3 From responsibility to a general integration architecture

3.1 Define the AO integration use case and allocate responsibility

Software alone cannot determine a correct or acceptable optical action. Optical and control engineers first define the AO integration use case: diagnostic and correction planes, references and registration, propagation relationships, actuator limits, permitted interventions, machine conditions, timing needs, and relevant beam-acceptance conditions. This definition identifies functions before assigning their implementation. A responsible subsystem needs the knowledge, permitted authority, behavior, performance, and evidence of fitness appropriate to each assigned function. Fitness can rest on established use, analysis, tests, bounded access, and maintainability as appropriate.

The resulting product-independent architecture has three roles: (1) a specialist subsystem for functions requiring specialist correction or reconstruction; (2) facility control for owned devices, state, operator authorization, sequencing, and protection; and (3) an optional assigned coordination subsystem for a genuinely cross-boundary procedure. One system may cover more than one role when it is fit to do so.

Figure 1 applies that architecture to one combined use case: A specialist RTAO subsystem corrects a localized beam state. Facility control owns a non-conjugate steering mirror between the wavefront sensor and deformable mirror for near-field alignment and a slower, large-aperture integrated AO near the end of the chain, as well as device and protection functions. A bounded coordinator could evaluate selected RTAO, beam-position, slow-AO, and facility-state observations, then request a permitted steering or slow-correction adjustment. For instance, a slow actuator might take over a sustained offset so that a fast corrector retains useful range; the exact offloading criterion, optical references, timing, and authorized request path belong to the use case.

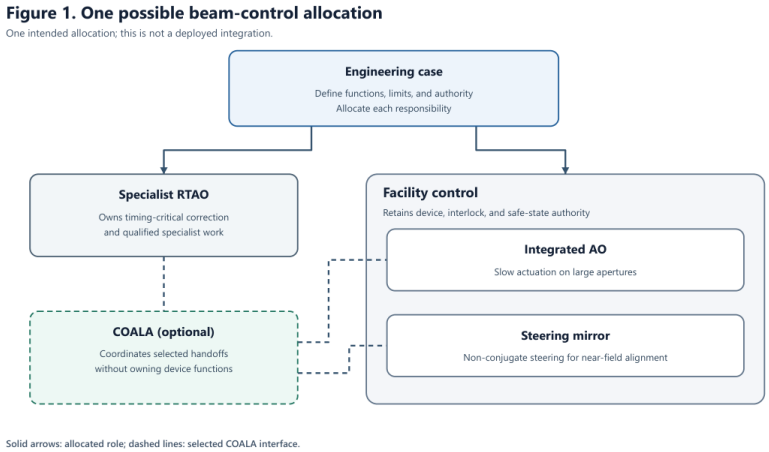


Figure 1: One illustrative beam-control allocation, reproduced from the COALA documentation in Annex A. This is a simple example of a combined arrangement and does not correspond to an existing deployment. Solid arrows express responsibility-allocation; dashed interfaces indicate possible exchanges, protocol, command semantics, or authority transfer. Facility control retains device, interlock, shot-permission, and safe-state authority; the text and Table 2 describe one possible interface interpretation.

Table 2 makes information, permitted requests, and retained authority distinct. Here "owner" denotes the subsystem that was assigned a function. Engineering/project authority approves the use case, interfaces, allocation, and fitness basis; facility operating authority retains device operation, protection, interlocks, shot permission, and safe-state behavior. The coordinator may issue only explicitly authorized requests. Beam acceptability is a separate optical condition that may inform facility decisions but does not confer protection authority.

Table 2: Illustrative responsibility and interface allocation for Figure 1. Proposed design interpretation; exact access, limits, validity, timing, and safe response require engineering for the selected facility. The table is illustrative rather than exhaustive.

Boundary or role	**Information, permitted request, and retained authority**
Specialist RTAO subsystem	*Information:* Selected WFS/DM status, correction or registration indicators, and limits. *Permitted request:* A reference adjustment if its interface and qualified behavior permit it. *Retained authority:* Timing-critical correction, specialist control decisions, and local limits.
Facility-owned steering	*Information:* Steering state. *Permitted request:* - (Not specified at this boundary in this illustration.) *Retained authority:* Device command acceptance and operating limits.
Facility-owned slow/static integrated AO	*Information:* Relevant wavefront, mirror state, and slow-correction status. *Permitted request:* - (Not specified at this boundary in this illustration.) *Retained authority:* Integrated-AO actuation, device limits, and facility operating procedure.
Optional coordination subsystem	*Information:* Selected observations plus machine mode and permission state. *Permitted request:* Requests explicitly allocated to its procedure; reports of unacceptable conditions. *Retained responsibility:* Steering- and offloading calculation.
Engineering/project authority	*Information:* Defines the AO use case, optical criteria, proposed interfaces, allocation, and fitness evidence. *Permitted decision:* Approves or declines the procedure design. *Retained authority:* Engineering approval and project accountability for the proposed integration.

3.2 Proposed broadly applicable requirements

In this section, we derive four requirements via inference from the evidence provided so far. Table 3 records the main transitions and Table 4 states the resulting proposed requirements.

Table 3: Observation → engineering inference → proposed requirement as stated in Table 4.

Observation and evidence status	Engineering inference	Proposed requirement
Specialist AO and facility platforms show capable functions in published examples [5–8].	Reassigning a function could lose qualified performance or authority; an additional subsystem needs its own fitness basis.	**R1 Qualified ownership**
In ARTAO, instability was induced by mis-registration; a separate large-stroke test produced a fluence imprint after propagation [4].	A local wavefront metric alone may not establish control stability and beam acceptability at another plane.	**R2 Physically meaningful control context**; **R4 Case-appropriate control relationships**
The Oxford survey reports heterogeneous ownership, access, and operating arrangements.	A universal vendor interface or default procedure owner cannot be assumed.	**R1 Qualified ownership**; **R3 Bounded integration interfaces**
Published specialist composition [5] and qualitative Oxford-reported arrangements show possible complexity; ARTAO proposes a future fast/slow pair [4], and multi-corrector investigations are ongoing.	The architecture must express the relationships selected by optical engineering, while allowing simpler cases.	**R4 Case-appropriate control relationships**

Table 4: Four proposed broadly applicable AO-integration requirements. These are generalized, product-independent design requirements inferred from Table 3 with no claim of completeness.

Requirement	Architectural obligation in a selected use case
R1 Qualified ownership	Assign each function to a subsystem with the knowledge, authority, behavior, performance, and fitness basis it needs.
R2 Physically meaningful control context	Aggregate the minimal set of measurements to possess complete, physically meaningful information. Cross existing boundaries where necessary.
R3 Bounded integration interfaces	Exchange only required observations, state, and permitted requests or commands. Keep each owner's assigned decisions and facility protection authority explicit.
R4 Case-appropriate control relationships	Represent the selected sensor, process, actuator, and propagation relationships, including multiple elements where justified, without imposing scale and complexity.

These requirements define the responsibility and interface architecture. In Figure 1, R1 keeps the WFS/DM loop on a qualified specialist path and facility devices under facility control. R2 makes registration, steering state, and any relevant propagation observation part of the coordinator's interpretation. R3 restricts its interaction with both systems to approved data and requests. R4 permits the fast/slow and alignment relationships only if that laser case requires them.

4 COALA as an optional software-architecture realization

4.1 Selecting a responsible implementation

The four requirements themselves do not dictate where a function runs. Using existing systems can remain a viable solution: a specialist-native solution is legitimate when the specialist AO subsystem has the required optical and facility access, authority, performance, and fitness, while a facility-native solution is equally legitimate when facility control can supply the optical logic and maintain it. A separate coordination subsystem on the other hand is justified when the selected procedure benefits from a bounded additional responsibility and its interfaces and maintenance can be supported.

COALA is a reusable software *framework* being explored for that third option. A possible deployed COALA-based subsystem could execute an explicitly assigned coordination function. Engineering/project authority approves the use case, interfaces, and fitness basis; facility operating authority retains device operation, protection, interlocks, shot permission, and safe-state behavior, while existing systems retain their assigned functions. The Figure 1 COALA box is one example of how such a conditional subsystem could be deployed.

Reusing an existing framework for this type of subsystem may reduce repeated engineering if functions, descriptions, and bounded adapters transfer usefully between facilities, making functionalities available to facilities that do not possess the resources to implement a fully fledged custom solution.

At the same time, a COALA-based solution also adds a software and maintenance boundary: interfaces must be built and kept compatible with specialist and facility systems; the coordinating subsystem needs its own lifecycle, failure behavior, ownership, and qualification; and extra message paths may affect timing or validity. These costs must be weighed against a specialist-native or facility-native implementation for the actual use case.

4.2 COALA-specific goals and conceptual components

The framework exploration has five goals subordinate to the four general requirements:

1. **Modularity** seeks reusable units for control, analysis, monitoring, calibration, or procedures.
2. **Configurable composition** seeks to select and arrange those units for a facility-specific use case.
3. **Extensibility** seeks a route for accepting custom functions to account for unforeseen scenarios.
4. **Interoperability** seeks bounded contact with specialist and facility systems.
5. **Shared implementation effort** seeks inspectable, reusable building blocks that could reduce repeated work across facilities.

These are goals; Section 5 assesses the implementation evidence for each.

Figure 2 shows the intended COALA software-component relationship. Structured descriptions would select tasks, parameters, and connections. A hosting workflow would resolve and arrange tasks and manage their intended lifecycle. Tasks are reusable units with selected inputs and outputs. If they are realized as microservices, Inter-Process Communication (IPC) would exchange messages between separately running participants. Bounded adapters would translate selected observations and permitted requests at interfaces supplied by existing systems. The existing specialist controllers, facility devices, and protection retain their intended functionality.

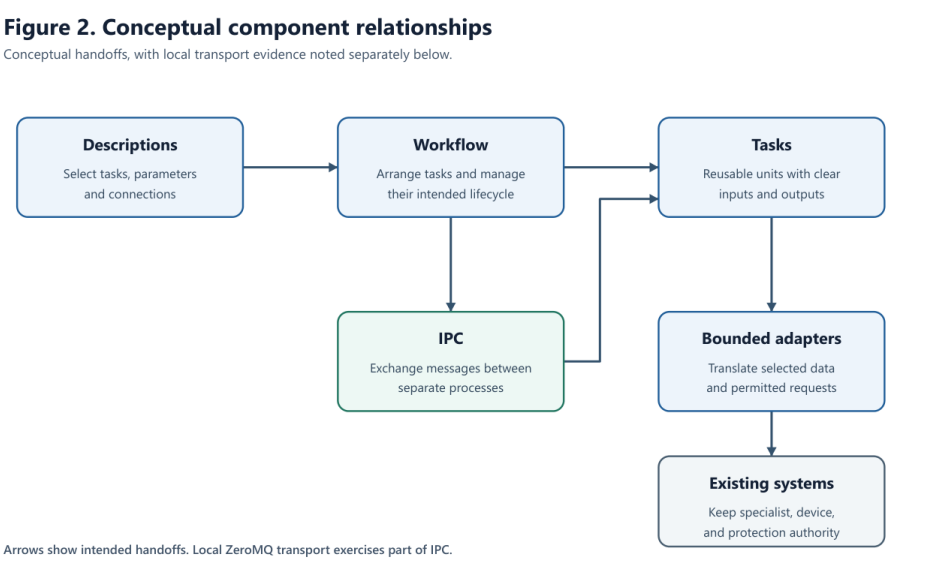


Figure 2: Conceptual COALA software-component relationships, reproduced from the COALA documentation in Annex A. Arrows denote intended handoffs.

5 Exploratory implementation and current maturity

The COALA repository[1] and its documentation (see Annex A for a rendered copy) make several architecture ideas inspectable. The documentation features an implementation-status page that clarifies the state of the exploratory implementation at the time of writing this report, such as identifying production modules, declared mechanisms, current breaks, and absent functions.

Table 5 uses four evidence levels:

- **Intended** means the documentation and Figure 2 describe a relationship.
- **Source-defined** means classes or methods implement a mechanism, without establishing an assembled runnable path.
- **Locally exercised** means a particular check observed behavior under its stated conditions.
- **Absent or unvalidated** means the inspected materials do not establish a required path or representative fitness. The direct transport and selected parsing checks are the strongest local observations; no representative AO/facility case was exercised.

[1]The code referenced in this report is publicly available under the Apache License 2.0 license at:
Repository: `https://git.gsi.de/phelix/coala/coala_exploratory`
Tag: `THRILL_D5.4`
commit: `eed495d6`
inspected: `26.10.2026`

Table 5: Inspected COALA implementation mapped to the four general requirements and five COALA goals. Status applies only to the inspected repository state. Module names and documentation pages identify the evidence.

Architecture requirement or COALA goal	Inspected evidence and status
R1 Qualified ownership	**Intended:** The architecture page of the documentation assigns qualified specialist and facility functions before COALA coordination. **Unvalidated:** no deployed responsibility allocation, timing qualification, or protection authority demonstration.
R2 Physically meaningful control context	**Source-defined:** coala_manifest._models, _loader, and resolvers describe typed data and task/workflow references. **Absent:** specific implementations of use cases.
R3 Bounded integration interfaces	**Source-defined:** coala_ipc defines envelopes, addresses, ZeroMQ transport, and an abstract transport seam, as described in the messaging page of the documentation. **Absent:** representative AO-device, control-system, or facility-protection adapters.
R4 Case-appropriate control relationships	**Source-defined but incomplete:** manifest/workflow forms can express tasks and connections through a YAML format, as described in the description page of the documentation; coala_workflow sketches host and wiring roles. The description generations and launch interfaces do not form one running description-to-workflow path.
Modularity	**Source-defined:** coala_tasks includes generic task hooks and illustrative subclasses. **Absent:** A reusable AO-function library has not yet been implemented.
Configurable composition	**Source-defined but incomplete:** typed loading and reference resolution exist; incompatible model generations, YAML, exports, and launch/wiring joins prevent a supplied configuration-to-running-workflow path.
Extensibility	**Source-defined:** CoalaTask hooks and generic subclasses illustrate additions. A representative facility extension and its integration effort are **unvalidated**.
Interoperability	**Locally exercised only for direct transport:** ZmqTransport is used for directed and published ZeroMQ envelopes. Cross-machine or control-system interoperability and adapters are **unvalidated or absent**.
Shared implementation effort	**Hypothesis:** inspectable source and documentation could support reuse. No comparative implementation effort, adoption, maintenance saving, or settled license basis is established.

6 Open architecture choices, evidence gaps, and outlook

The architecture leaves the exact coordination scope and implementing subsystem to the AO integration use case. Optical topology, relevant planes and references, accessible vendor and facility interfaces, acceptable timing and data validity, and the permitted response to a failed or unacceptable condition are case-specific choices. A simple semi-static procedure might be implemented entirely in facility control, while a registration-sensitive RTAO arrangement might require a distinct qualified path similar to the depiction in Figure 1.

Several issues require evidence beyond the current report. The four proposed requirements need assessment across representative, contrasting facilities to test their completeness and generality.

COALA's internally competing description routes and incomplete host/workflow assembly need resolution before its component model can be judged in use. Actual device and control-system interfaces, vendor access, timing, validity conditions, operator operation, and combined optical/protection criteria must be established for a selected case. The relative cost and maintainability of specialist-native, facility-native, and COALA-based implementations also remain unknown and must likely be judged on a case-to-case basis.

Based on this exploratory work, the newly founded Adaptive Laser Architecture Development and INtegration (ALADIN) program [9] will continue prototype development and validate several use cases on the dedicated Laser Adaptive Optics Testbed for Stabilization Experiments (LAOTSE). The immediate architectural result is a bounded way to assign AO functions and exchanges while preserving incumbent responsibilities; its use across facilities and the framework's practical reuse benefit await further evidence.

7 Conclusion

AO integration in laser systems begins with capable specialist and facility systems. The cross-boundary problem arises conditionally when a selected procedure requires physically meaningful information or permitted action across their responsibilities. The product-independent software architecture developed here assigns functions by knowledge, authority, behavior, performance, and fitness; distinguishes observations from requests and retained decisions; and permits the optical control relationships the use case needs.

Four requirements express that architecture: qualified ownership, physically meaningful control context, bounded integration interfaces, and case-appropriate control relationships. They are proposed as broadly applicable, while completeness and representative cross-facility generality remain unvalidated. The combined RTAO, steering, and slow integrated-AO allocation shows one possible realization, including a possible fast/slow relationship. Specialist-native and facility-native implementations remain legitimate outcomes.

COALA is an optional framework from which a bounded coordination subsystem might be built. Its exploratory implementation approach features descriptions, hosting workflows, tasks and IPC, and adapters, illustrating a possible realization path with different degrees of completeness or maturity over the explored components. Whether COALA reuse justifies its additional integration and maintenance boundary remains an open question to be answered on a case-to-case basis.

Acknowledgements

This work, including the ARTAO project, were funded in the framework of the Technology for High-Repetition-rate Intense Laser Laboratories (THRILL) project by the European Union's HORIZON-INFRA-2022-TECH-01 call under grant agreement number 101095207.

The ALADIN project has received funding from the German Federal Ministry for Research, Technology and Space (BMFTR) within the framework of "Fusion 2040 – Research on the Way to a Fusion Power Plant" under grant agreement No. 13F1023.

The CACAO software package is supported by NSF award 2410616.

Use of Generative AI Tools

OpenAI GPT-5.6 and GPT-6 were used to assist with structuring author-provided material, as well as language and grammar polishing. The authors reviewed, revised, and independently verified all AI-assisted content and take full responsibility for the manuscript. The tools were not used to generate scientific data or analyses.

References

[1] O. A. Hurricane and M. C. Herrmann. High-energy-density physics at the national ignition facility. *Annual Review of Nuclear and Particle Science*, 67:213–230, 2017. doi: 10.1146/annurev-nucl-101916-123227.

[2] Lawrence Livermore National Laboratory. High energy density science. National Ignition Facility and Photon Science, LLNL-WEB-812720. URL https://lasers.llnl.gov/science/hed-science.

[3] J. Zhu et al. Analysis and construction status of SG-II 5pw laser facility. *High Power Laser Science and Engineering*, 6:e29, 2018. doi: 10.1017/hpl.2018.23.

[4] J. B. Ohland, N. Lebas, V. Deo, O. Guyon, F. Mathieu, P. Audebert, and D. Papadopoulos. Apollon real-time adaptive optics: astronomy-inspired wavefront stabilization in ultraintense lasers. *High Power Laser Science and Engineering*, 13:e29, 2025. doi: 10.1017/hpl.2025.16.

[5] O. Guyon et al. Adaptive optics real time control with the Compute and Control for Adaptive Optics (CACAO) software framework. In *Proceedings of SPIE*, volume 11448, 2020. doi: 10.1117/12.2562822.

[6] L. R. Dalesio et al. The Experimental Physics and Industrial Control System architecture: past, present, and future. *Nuclear Instruments and Methods in Physics Research A*, 352: 179–184, 1994. doi: 10.1016/0168-9002(94)91493-1.

[7] A. Götz, E. Taurel, J. L. Pons, et al. TANGO — a CORBA based control system. In *Proceedings of ICALEPCS 2003*, pages 220–222, 2003. URL https://proceedings.jacow.org/ica03/papers/MP705.pdf.

[8] S. Hauf et al. The Karabo distributed control system. *Journal of Synchrotron Radiation*, 26: 1448–1461, 2019. doi: 10.1107/S1600577519006696.

[9] J. B. Ohland et al. AO for laser fusion — the challenges of controlling hundreds of high power laser beams. In *Proceedings of SPIE*, volume 14150, page 141500W, 2026. doi: 10.1117/12.3107897.

Annex A — COALA Documentation

The following pages reproduce the COALA documentation assembled for THRILL Deliverable 5.4. The Code reference and Local exercise chapters are not included in this annex. The online documentation may change over time.

COALA
COntroller for Adaptive Laser Architecture

Overview | Concepts and architecture | Descriptions and workflows | Messaging and processes | Code reference | Implementation status | Local exercise

DOCUMENTATION OVERVIEW

Coordinating advanced beam control across systems

Advanced beam-control procedures may span several subsystems: a measurement from one informs a decision, which can lead to an action permitted in another. COALA (COntroller for Adaptive Laser Architecture) is envisioned as reusable software for coordinating these handoffs.

Adaptive Laser Architecture (ALA) brings laser beam-control components into a unified framework. COALA is envisioned to provide ALA-relevant functions for tasks that cross established system boundaries. Similar needs recur across facilities, so prepared functions and reusable control blocks could help COALA take ownership of a clearly assigned coordination role. The component owners keep their own control functions.

The repository is an exploratory prototype. Local messaging mechanisms can be exercised, while the complete illustrated workflow and representative adaptive-optics (AO)/facility adapters are not executable end to end.

One possible beam-control arrangement

The idea behind COALA is to first define the engineering case, then assign each function to a system with the necessary knowledge, authority, timing, and evidence of fitness. COALA would take a bounded coordination role when bringing those functions together calls for an additional owner.

Illustrative design scenario: a specialist, stand-alone real-time adaptive-optics (RTAO) controller handles timing-critical wavefront correction early in the system. At the same time, the laser control system owns both a steering mirror in a non-conjugate plane for near-field alignment and an integrated, large-aperture AO system at the end of the laser chain for the correction of static aberrations. COALA could coordinate observations, evaluations, and permitted requests involving all three components through bounded interfaces without taking ownership of the subsystems themselves. This enables advanced schemes like actuator offloading and misregistration correction. RTAO keeps its correction loop; facility control keeps the devices, interlocks, and safe-

13

state authority. The arrangement illustrates intended coordination, not a runnable integration supplied by this repository. Figure 1 shows the responsibility boundaries.

Figure 1. One possible beam-control allocation

One intended allocation; this is not a deployed integration.

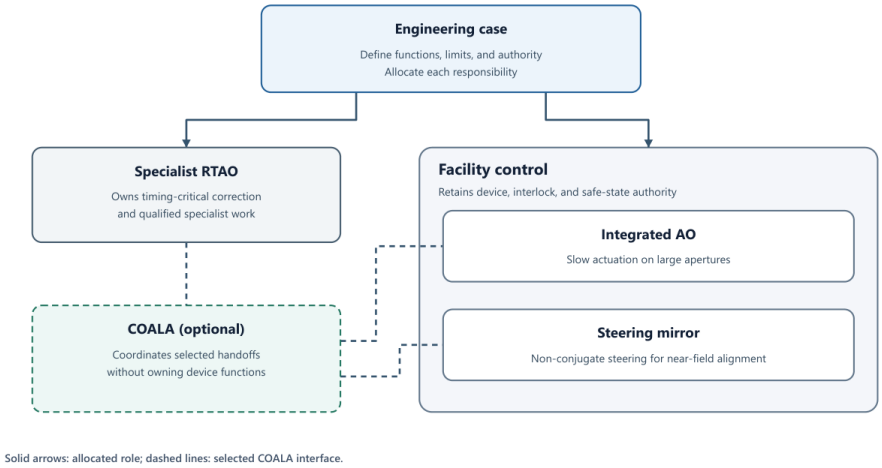


Figure 1. One possible beam-control allocation. This is an illustrative allocation, not a deployed integration. Open the full-size figure.

Figure in words

Engineering defines and allocates the case. Specialist RTAO retains timing-critical correction. Facility control retains device, interlock, and safe-state authority over integrated AO for slow large-aperture actuation and a steering mirror for non-conjugate near-field alignment. COALA may coordinate selected information and requests with all three through bounded interfaces, without owning their individual functions.

The explored building blocks

A **task** represents one reusable unit of work. A **workflow** arranges tasks for a procedure. A **description** selects tasks, parameters, and relationships; a manifest is the explored structured form. **Inter-process communication** (IPC) supports exchange between separately running tasks in the exploratory implementation. An **adapter** would give a task a bounded interface to an existing system. The concepts and architecture chapter follows these relationships one handoff at a time.

Continue reading

Understand the architecture

Learn the roles of tasks, workflows, descriptions, IPC, and adapters through the illustrative sequence.

Follow the implementation

Trace descriptions, tasks, workflows, and their current integration boundary.

Try local messaging

Run a bounded transport exercise in a fresh environment.

Check implementation status

See what works locally, what is incomplete, and what remains a design choice.

COALA
COntroller for Adaptive Laser Architecture

Overview | Concepts and architecture | Descriptions and workflows | Messaging and processes | Code reference | Implementation status | Local exercise

CONCEPTS AND ARCHITECTURE

From a defined responsibility to cooperating functions

COALA is envisioned as a framework for assembling a case-specific coordination role from reusable units of work. This chapter explains the intended relationships.

Responsibility | Tasks and workflows | Descriptions | Messages and IPC | Adapters | Whole picture

1. Define the responsibility

Begin with the engineering case: what must be observed, decided, or requested; which information is available; what limits apply; and who is allowed to act. Allocate each function to a system with the required knowledge, authority, timing, and evidence of fitness. COALA becomes a candidate only if a bounded coordination responsibility remains and reusable integration work would help.

For a timing-critical example, a specialist real-time adaptive-optics (RTAO) controller can keep the correction loop. Facility control can keep devices, machine state, interlocks, and safe-state authority. If a slower procedure needs selected information from both, COALA could own the agreed coordination and make requests through their permitted interfaces.

2. Give each unit of work a clear job

A **task** is the explored unit for one reusable function. As an example, an observation task obtains a diagnostic value; an evaluation task compares it with a case-specific condition; a request task asks an existing system to perform an allowed action. The task boundary helps separate these jobs and their inputs and outputs.

16

A **workflow** describes how tasks cooperate for a procedure. It can say that an observation supplies an evaluation, and that the evaluation may lead to a request. A host or coordinator would manage the tasks' lifecycle and their connections. The order and permission to act must come from the engineering case, not from the mere presence of a connection.

> **Illustrative sequence**
>
> 1. Observe: receive a selected diagnostic value from an existing owner.
> 2. Evaluate: compare it with the condition defined for this case.
> 3. Request: send an allowed request to an existing owner, which retains authority over its action.
>
> This sequence explains roles and handoffs. The repository does not supply the AO/facility adapters or a working end-to-end launch path for it.

3. Describe the arrangement

A **configuration** provides the choices for a particular use: which task types to select, their parameters, and how they connect. A **manifest** is a structured description of those choices and related data shapes. In this repository, YAML is the explored representation.

In the intended path, a loader reads descriptions, resolves references, runs validation checks and gives a host the information to assemble the workflow. Current typed models and sample YAML make parts of that idea inspectable, but the available description generations and workflow examples do not form one executable configuration-to-task path in the current state. Treat any arrangement shown here as a design illustration.

4. Exchange information between running functions

Since the majority of large scale lasers is running on distributed SCADA systems, a COALA workflow may need to span several machines because the beam-control components it coordinates are distributed. The exploratory implementation therefore follows a microservice approach so tasks could be placed where needed while retaining a consistent boundary around their logic. This has the side effect that microservices can be implemented in any programming language, as long as they respect the defined interfaces.

Once tasks run in separate processes, they need a way to communicate. **Inter-process communication** (IPC) can carry task messages locally on one machine. Across machines, a different communication adapter could carry corresponding messages, potentially using an

existing SCADA system's communication capability where appropriate. The intended task interface would allow that adapter to change without substantial rewriting of task logic.

A direct message addresses a particular recipient; publish/subscribe broadcasts an update to subscribers. In the illustrative arrangement, a selected diagnostic update could be published, while a permitted action request could be directed to its existing owner. The engineering again determines the choices what needs to be sent and how.

The explored software gives messages structured envelopes, addresses, and transport paths. Queues can hold work before dispatch; lifecycle and logging mechanisms help a host observe participants. The local ZeroMQ transport demonstrates the envisioned directed and publish/subscribe exchange.

5. Meet existing systems at bounded interfaces

An **adapter** would translate between a COALA task's selected inputs or requests and an existing system's approved interface. It should expose only what the case needs and preserve the existing owner's decisions, timing duties, and protection authority. An adapter therefore provides a case-specific integration boundary without tranferring ownership.

The explored software architecture

Figure 2 follows the proposed handoffs: descriptions inform workflow coordination; a workflow connects tasks; IPC carries messages among separately running participants; adapters would meet existing systems. It depicts intended architecture, with a separately noted local transport mechanism, rather than a working assembled application.

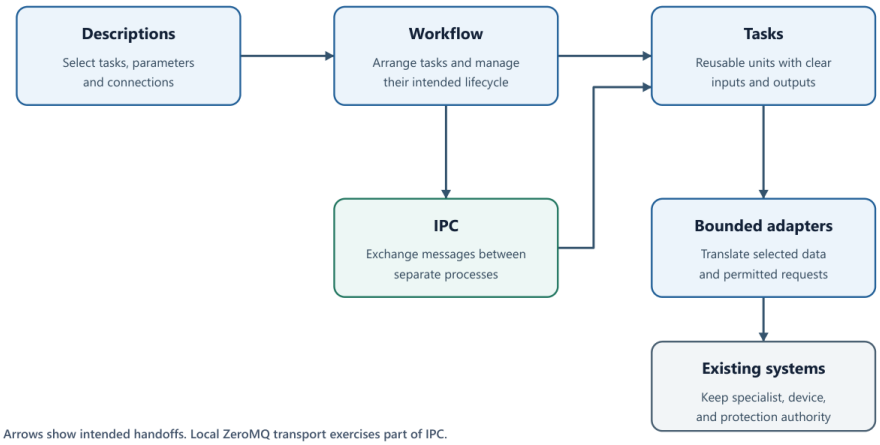


Figure 2. Conceptual component relationships. The IPC transport can be exercised locally; the complete path and AO/facility adapters are not supplied as a running integration. Open the full-size figure.

Figure in words

A description chooses task types, parameters, and connections. Workflow coordination would assemble and manage those tasks. IPC supports message exchange between separate processes. Bounded adapters would pass selected information or requests to specialist AO and facility control, whose own responsibilities remain with them. The full chain is an intended relationship; the transport is a locally exercised part.

Where the ideas appear in code

The names follow the concepts: `coala_manifest` explores typed descriptions and loading, `coala_workflow` explores task and workflow coordination, `coala_tasks` contains generic examples, and `coala_ipc` contains message and transport mechanisms. The `workflows` directory contains illustrative YAML arrangements. These parts are evidence of the design, not a supported end-to-end authoring recipe. Continue into descriptions, tasks, and workflows or messaging and process coordination; the code reference maps every production module and symbol to source.

Previous: overview

Next: descriptions and workflows

COALA
COntroller for Adaptive Laser Architecture

Overview | Concepts and architecture | Descriptions and workflows | Messaging and processes | Code reference | Implementation status | Local exercise

SUBSYSTEM GUIDE

Descriptions, tasks, and workflows

A description names a unit of work and its data; a workflow arranges such units for a procedure. COALA envisions a host that resolves descriptions, starts tasks, and connects their exchanges. This guide follows the repository's exploratory implementation of those roles and distinguishes source-defined mechanisms from intended assembly.

Describe | Resolve | Task and host | Two model routes | Current boundary

1. Name the work and its data

In the repository's typed-description route, a `TaskManifest` records a task name, optional inheritance, execution command and input/output definitions. Inputs and outputs can use schema fields. Like that, tasks can be implemented in any programming language, as long as they remain executable and respect the interfaces defined in the manifest. A `WorkflowManifest` names task references, nested workflows, connections, and tunnels. These manifest objects describe contracts and relationships, but neither the work nor the logic requried for execution.

`ManifestLoader` reads `manifest_directories` from YAML configuration, scans `*.manifest.yaml`, and validates each file according to its declared `manifest_type`. It collects accepted task, workflow, and data-structure models. Invalid files are reported and skipped, so a populated collection does not mean every file passed validation.

2. Resolve references before a host uses them

Figure 3 separates YAML discovery and validation from the subsequent resolution of description references.

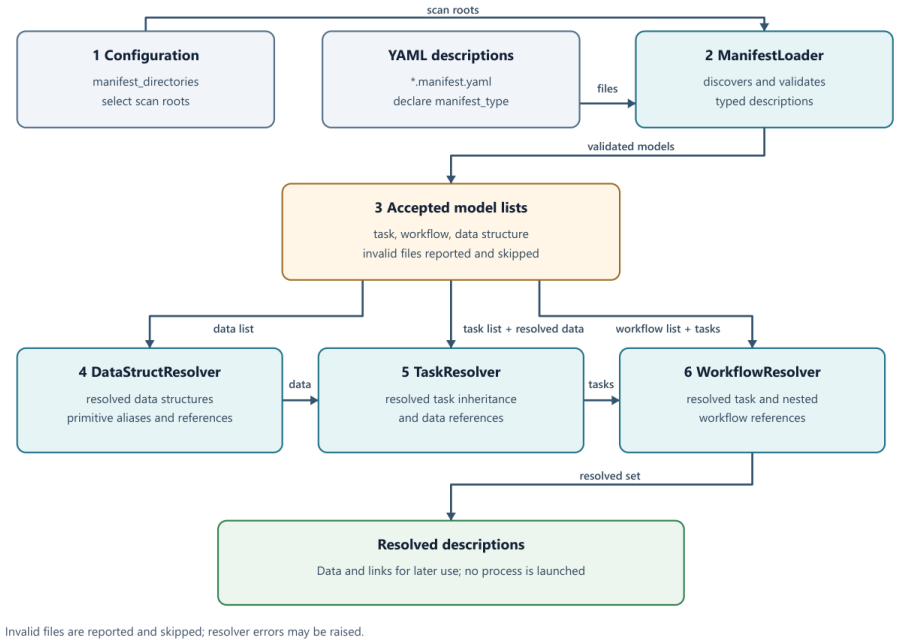
The dependency order in this route is data structures, tasks, then workflows. `DataStructResolver` expands reusable field definitions; `TaskResolver` flattens task inheritance

20

and data references; `WorkflowResolver` attaches task and nested-workflow references. Their constructors perform resolution and can raise for duplicate names, missing references, or cycles. The result is resolved description data from which the actual workflow can be constructed, launched and wired.

Figure 3. From YAML descriptions to resolved objects

Typed-description loader and resolver dependencies.



Invalid files are reported and skipped; resolver errors may be raised.

Figure 3. From YAML descriptions to resolved objects. Open the full-size SVG. The arrows reflect the typed-description classes and their inputs.

> **Read the resolution path:** Configuration supplies directories to `ManifestLoader`, which validates and collects descriptions. `DataStructResolver` expands reusable structures. `TaskResolver` resolves task inheritance and data references. `WorkflowResolver` attaches task and nested-workflow references. Loader validation failures are reported and skipped; resolver failures are raised. This route does not yet form a complete configuration-to-execution path.

3. Assemble host and task roles

`CoalaTask` is a base for task-process hooks such as socket setup, commands, post-launch work, and termination. It constructs a `WorkflowTaskAPI` for task-side messages and status. The clock,

integrator, and dummy are illustrative and test task subclasses: they show reusable hooks and data handling in this exploratory implementation.

`WorkflowHostAPI` constructs and holds a `TaskManager` as `self.tasks`, passing the same `CoalaIPC` instance. `WorkflowHostAPI.add_task` supplies the host address and delegates to `TaskManager.add_task`. The manager assigns a task UUID and `TaskConfig` address, requests `CoalaIPC.launch_process`, then stores the configuration by UUID. That launch request is present in source; the contracts around it do not yet form an executable workflow assembly.

The manager also provides `get_task_conf` and `send_queue_command_to_task`. The forwarding method checks that a target UUID has a stored task configuration, then calls `CoalaIPC.send_message` with that UUID. The manager does not perform the transport send or resolve a router address for each command. `ServiceRegistry.register_service` connects a known UUID to its router address; `ZmqTransport.send_direct` selects the corresponding outbound socket by destination UUID. A registered transport peer is therefore a separate requirement from a TaskManager record.

At deployment or startup, `WorkflowWiringManager` is intended to derive connection and tunnel requirements from a selected workflow description and a nickname-to-UUID map. Its methods can record addresses reported later and calculate instructions for participants. The source does not establish a complete lifecycle for creating the manager, delivering every instruction, or updating a running workflow. `CoalaNode` sketches top-level assembly.

Figure 4 brings these intended roles together. Solid links denote source-defined calls or construction. Dashed links denote architectural handoffs for which the current code has no complete path.

Figure 4. Intended workflow assembly and task participation

Host preparation, task participation, and intended startup wiring.

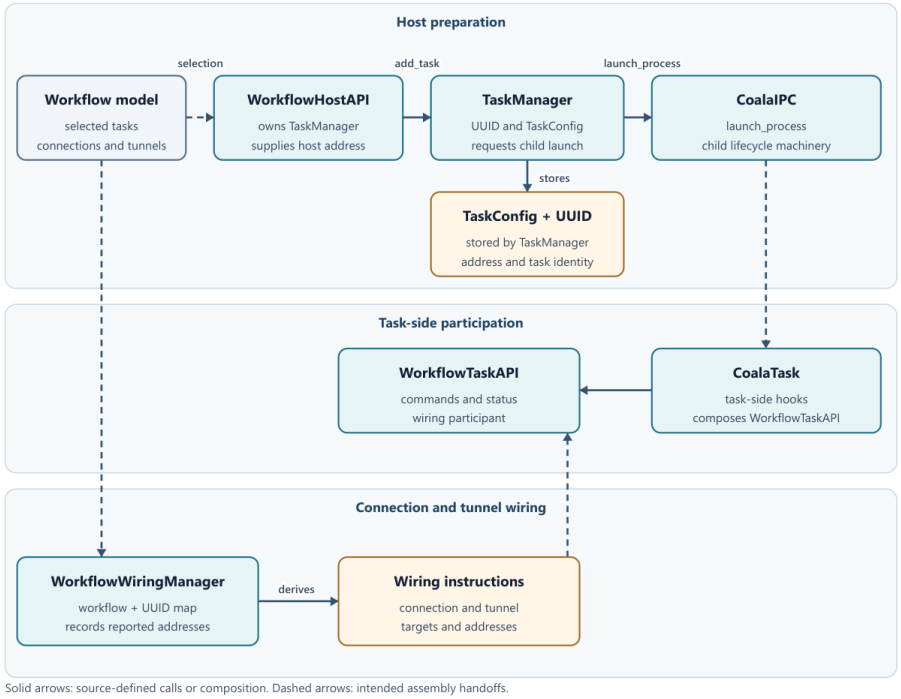


Figure 4. Intended workflow assembly and task participation. Open the full-size SVG. Dashed links mark intended handoffs, not executable calls.

> **Read the assembly view:** A resolved workflow would select task configurations. `WorkflowHostAPI` owns `TaskManager`, whose `add_task` prepares a UUID and requests `CoalaIPC.launch_process`. A launched task is intended to participate through `CoalaTask` and its composed `WorkflowTaskAPI`. `WorkflowWiringManager` can derive connection and tunnel instructions from the workflow and reported addresses. The upward dashed link from derived instructions to the task-side API represents intended delivery. The current manager assembly method is commented out, and the launch and task configuration interfaces disagree, so these dashed joins remain intended relationships.

4. Locate the coexisting description routes

The repository also contains `ManifestManager` with `coala_manifest.task_manifest_models` and `coala_manifest.workflow_manifest_models`. `coala_node.py` uses these socket-oriented

shapes. They are distinct from the `coala_manifest._models` data-schema types used by the loader and resolvers above. The implementation-status page records the unresolved route choice and compatibility limits.

Current use boundary

The repository does not yet provide one executable description-to-workflow-launch route. Consult implementation status for the blocking interfaces, and the linked class reference for individual method behavior. The next chapter follows the lower-level messaging and process mechanisms.

Previous: concepts and architecture

Next: messaging and processes

COALA
COntroller for Adaptive Laser Architecture

SUBSYSTEM GUIDE

Messaging and process coordination

The repository's exploratory IPC layer connects independently running participants. The present ZeroMQ transport exchanges directed and published messages locally; the larger facade adds queues, handlers, registration, retries, logging, and process-lifecycle machinery.

Messages Transport Facade flow Processes Boundaries

1. Give an exchange an address and meaning

`MessageEnvelope` carries source and destination UUIDs, a topic, payload, priority, message ID, timestamp, and optional reply/ACK metadata. A `ServiceAddress` binds a UUID to direct and publisher addresses. `IPCSettings` selects a transport class, local and optional parent address, logging, queue capacity, and retry defaults.

2. Carry directed and published messages

`TransportInterface` defines bind, connect, send, publish, poll, stop, and statistics operations. The sole concrete implementation is `ZmqTransport`. It uses ZeroMQ ROUTER/DEALER sockets for directed exchange and PUB/SUB sockets for topics. A participant binds local inbound and publisher addresses, connects to peers, sends envelopes, and polls for received envelopes. The verified local exercise uses this class directly, without the facade.

The transport interface is a useful seam for another communication adapter. The intended architecture allows a distributed workflow to use local IPC on one machine and another communication adapter across machines, potentially using existing control system communication mechanisms.

3. Follow the IPC facade's message routes

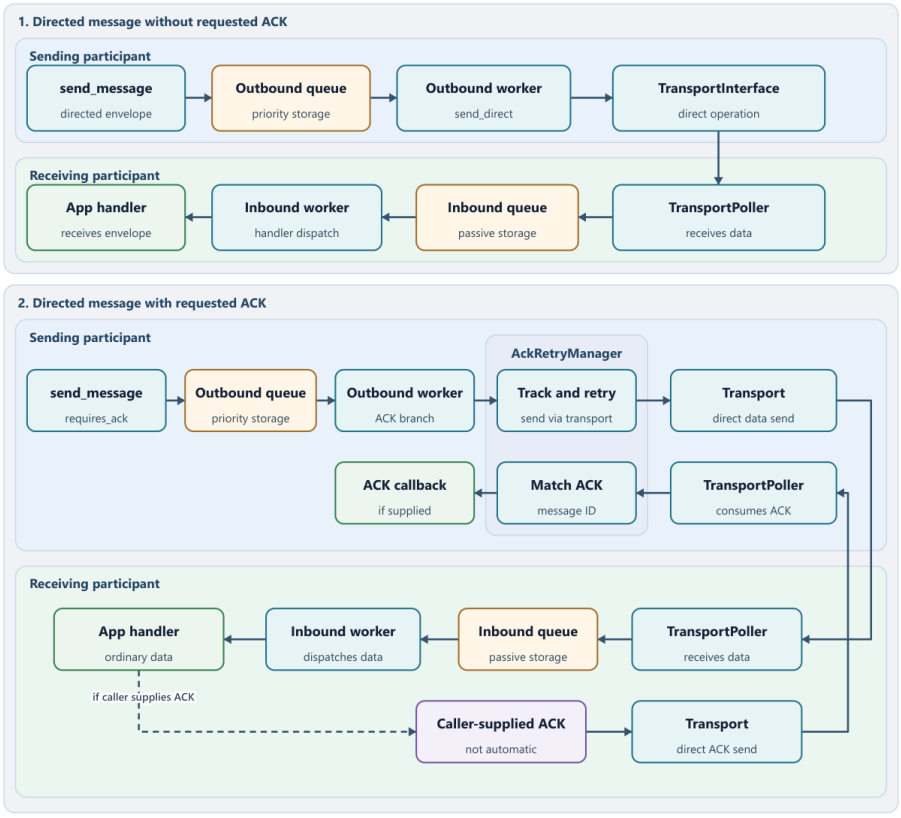
`CoalaIPC` constructs the transport, passive priority queues, outbound worker, poller, inbound worker, and handler dispatcher. Its constructor starts the workers and binds local transport addresses. A participant here is any IPC service, not necessarily a workflow task.

Figure 5 separates four routes. Each gray scenario contains its own participant areas: blue for sending or publishing, green for receiving or subscribing. The first scenario follows a directed message without requested ACK; the second repeats that route with sender-side ACK tracking and a separate, optional ACK return.

1. **Directed, no ACK:** `send_message` enqueues an `OutboundItem`. `OutboundQueueWorker` calls `transport.send_direct`. The receiver's poller enqueues non-ACK data; the inbound worker and dispatcher invoke a matching application handler.
2. **Directed, ACK requested:** The same outbound queue and worker are used, but the worker calls `AckRetryManager.send`. It tracks the message and sends through transport, retrying by its schedule. If the receiving application or another caller explicitly constructs and sends an ACK envelope through transport, it returns to the sender's poller, which gives it to `AckRetryManager.handle_inbound` before the ordinary inbound queue. The receiver does not automatically create that ACK, so tracking does not establish reliable delivery.
3. **Self-directed:** `send_message` detects its own UUID and enters the local inbound queue directly; it skips the ordinary outbound queue and transport.
4. **Published:** `publish` calls `transport.publish` directly. A subscribed participant's poller receives the publication and follows the non-ACK inbound path to dispatch. Publication bypasses the sender's outbound priority queue.

Figure 5. Four routes through the IPC facade

Four distinct routes through sending, receiving, and dispatch.



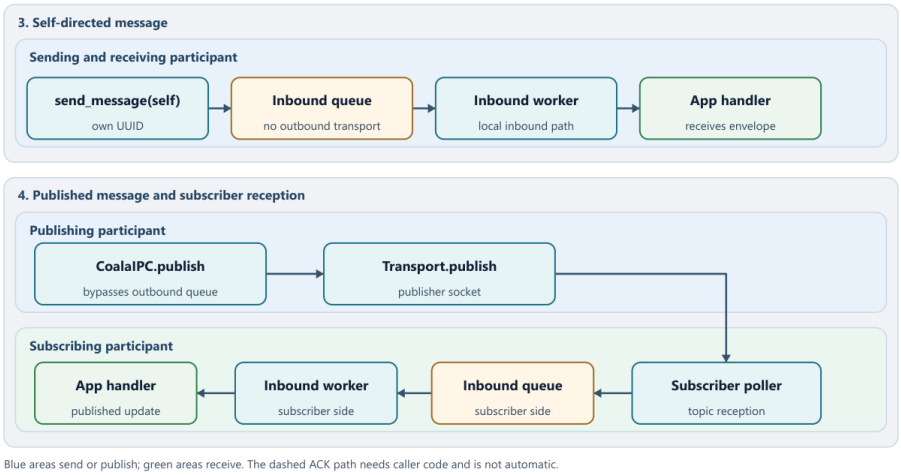


Figure 5. Four routes through the IPC facade. Open the full-size SVG. Pale yellow queue boxes store items; blue-green nodes are active workers or transport operations. Participant backgrounds identify ownership consistently across scenarios.

> **Read the four routes:** For ordinary directed data, the sender's `CoalaIPC.send_message` feeds the outbound queue, worker, and transport interface; the receiver's poller, inbound queue, worker, and dispatcher reach the application handler. Requested ACK tracking branches at the outbound worker into AckRetryManager. An ACK must be constructed and sent explicitly by receiver-side caller code; the figure marks that conditional action with a dashed arrow. It travels through the receiving participant's transport to the sender's poller, then the same retry manager matches its message ID. It is not dispatched as ordinary application data. A self-directed message goes straight to its own inbound queue. A publication goes from `CoalaIPC.publish` to transport and reaches a subscriber's poller and inbound path; neither alternative uses the sender's ordinary outbound queue.

4. Coordinate child registration and readiness

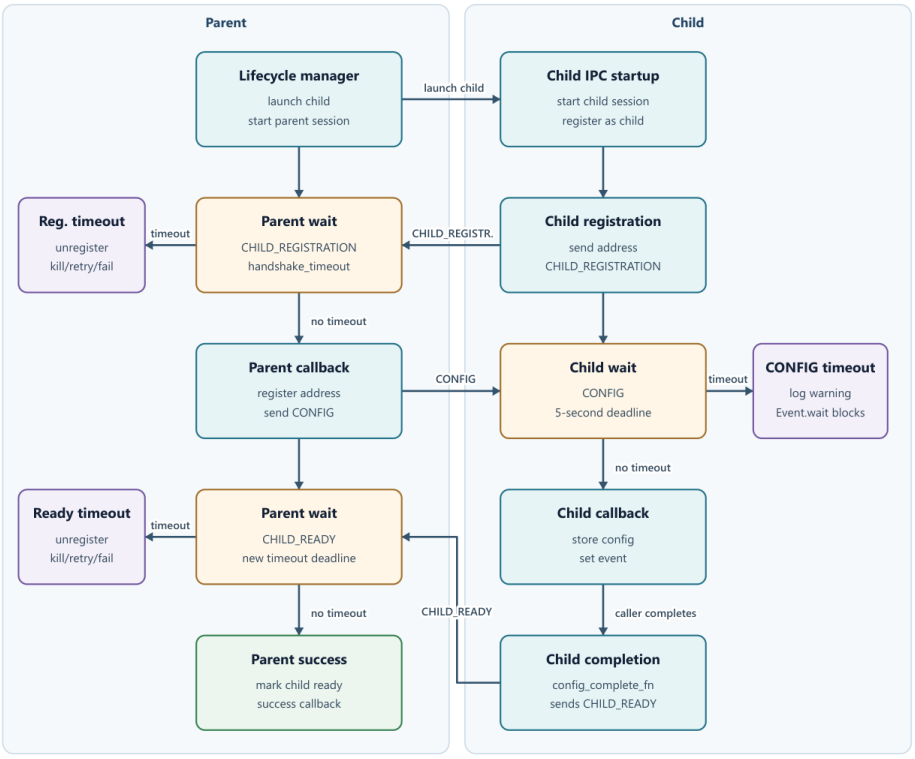
`ServiceLifecycleManager` arranges child launch and registration. `ChildProcessManager` stores child records and launches subprocesses. `HandshakeFlowBuilder` constructs parent and child sessions; `HandshakeManager` routes messages and checks deadlines. `ServiceRegistry` records the child's reported address and connects the parent's transport.

Figure 6 follows registration and readiness, with an alternative local timeout branch attached to each timed wait. The parent starts a timed wait for `CHILD_REGISTRATION` after launching the child. The child sends that message and waits for `CONFIG` under its own session deadline. On matching

registration, the parent session advances to a second timed step for `CHILD_READY`; its callback registers the address and sends `CONFIG`. Receipt of configuration stores it and releases the child's registration event; a caller must invoke the `config_complete_fn` supplied by `get_launch_info` to emit `CHILD_READY`.

Figure 6. Parent and child registration, readiness, and timeout

Registration and readiness, with local timeout branches from each timed wait.



Timeout branches are local callbacks, not messages. The handshake does not establish an end-to-end workflow launch.

Figure 6. Parent and child registration, readiness, and timeout. Open the full-size SVG. All named messages use the `PROC_HANDSHAKE` topic. The parent launch, registration, and readiness arrows and the child configuration branch show local actions. Timeout boxes do not send messages to the other side.

> **Read the handshake branches:** The parent launch handler starts a session whose first timed step awaits `CHILD_REGISTRATION`. The child registration method starts its own timed `CONFIG` step and sends registration. The parent's registration callback calls `register_service` and sends configuration. The child callback stores it and sets its event. The parent then awaits `CHILD_READY`; an explicit child completion callback sends it, and parent success marks the child ready and invokes any supplied success callback. If either parent wait expires, the parent session's failure

> callback unregisters the child and can kill/retry or report final failure. If the child waits too long for `CONFIG`, its timeout logs and its no-op final-failure callback does not release the separate blocking registration event. One side's timeout does not itself notify the other side.

The handshake mechanism is present, but the workflow task-launch path still has incompatible interfaces. See implementation status for the reader-visible effect. The figure omits shutdown and recovery beyond the source-defined callbacks.

Current use boundary

Local directed and publish/subscribe transport exchange has been exercised. The receiving framework does not automatically send an ACK merely because a message requests one, so retry tracking must not be described as completed reliable delivery. Queue overflow may drop messages under its configured policy. The available checks do not establish distributed operation, timing guarantees, recovery behavior, or suitability for facility controls. See implementation status for the cross-system view and the complete code reference for all helpers.

Previous: descriptions and workflows

Next: code reference

COALA
COntroller for Adaptive Laser Architecture

Overview Concepts and architecture Descriptions and workflows
Messaging and processes Code reference Implementation status
Local exercise

This documentation was generated by OpenAI GPT-6 Sol from the COALA code base. Human review has focused on the conceptual explanations; the code-level documentation has received less review.

IMPLEMENTATION STATUS

What the repository demonstrates today

The repository is an exploratory implementation of COALA. The categories below describe what a reader can infer from the current source and bounded checks, without turning a design illustration into a runnable integration claim.

Locally exercised mechanisms

ZeroMQ transport

The direct transport exercise starts three local processes, exchanges directed and published envelopes, prints statistics, and checks disconnect methods. The fresh-environment recipe reproduces this bounded check. It does not exercise the complete IPC facade, queue workers, workflow execution, or facility interfaces.

Selected parsing utilities

Previous focused checks imported `coala_ipc` and queried the type-conversion matrix. The current loader can accept some present-model fixtures. These checks establish local mechanisms only; they do not validate the whole configuration-to-task chain.

Incomplete integration

Description to workflow

The `coala_manifest._loader` and its resolvers exist, but primitive type loading references the wrong location, several repository workflows use fields from another model generation, and `coala_workflow` imports names that `coala_manifest` does not export. A reader

Launching and wiring

Host, task, wiring, and child-lifecycle classes describe the path. `TaskManager.implement_workflow` is commented out; task addition does not return the UUID its caller expects; process command and parser arguments disagree. CoalaNode leaves the base-workflow add

31

cannot take the supplied YAML through one documented, executable workflow path. See descriptions and workflows.

call commented out. Treat their APIs as exploratory source, not a launch quick start.

ACK and retry

Outbound retry tracking and inbound explicit ACK handling are implemented, but the receiving framework does not automatically generate the requested ACK. Do not infer reliable delivery from the presence of `requires_ack` or retry settings. See message flow.

Absent functionality in this repository

No representative AO or facility-control adapter is supplied, and no complete example coordinates specialist RTAO, integrated AO, and steering-mirror systems. The figures in the overview and architecture chapter are conceptual. No verified distributed/SCADA communication adapter, timing guarantee, safety validation, or deployment procedure is provided.

Open architectural choices

The repository keeps `coala_manifest._models` data-schema descriptions alongside the socket-oriented `coala_manifest.task_manifest_models` and `coala_manifest.workflow_manifest_models` routes. The code does not settle whether one should replace the other or how to bridge them. Transport portability beyond ZeroMQ and the exact adapter boundary for a facility case also remain design choices. The responsibility rule still applies: the engineering case assigns functions by knowledge, authority, timing, and evidence of fitness before COALA takes an agreed coordination role.

Where to go next

Read the description and workflow guide or messaging guide for code relationships, then follow their links into the module and symbol reference.

Code reference

Next: local exercise