# From Learning Management Systems to Adaptive Learning Systems: A Systematic Architectural Transformation Framework

**Alejandro Sartorio, Gustavo Rossi**

[1]Centro de Altos Estúdios en Tecnologia Informática (CAETI)
Universidad Abierta Interamericana (UAI)
Argentina

{alejandro.sartorio, gustavo.rossi}@uai.edu.ar

***Abstract.*** *Traditional Learning Management Systems (LMS) predominantly operate as static content repositories under a "one-size-fits-all" paradigm, failing to address the diverse cognitive needs and learning paces of individual students. While Adaptive Learning Systems (ALS) offer a solution through personalization, migrating legacy monolithic LMS infrastructures involves overcoming significant technical debt and "architectural stiffness." This paper proposes* ***ARALS*** *(Adaptive Reference Architecture for Learning Systems), a framework designed to evolve legacy LMSs into genuine adaptive environments without requiring a complete codebase rewrite. To this end, we introduce a* ***Conceptual Transformation Sequence (C1)*** *and the* ***PALS framework*** *(Properties of Adaptive Learning Systems) as a systematic method to inject and validate adaptivity. ARALS integrates the* ***MAPE-K*** *autonomic computing cycle into a decoupled, seven/eight-layer structure, acting as a control mechanism for the operational flow to ensure pedagogical adaptation. The proposal is validated through a high-fidelity case study in which the* ***Odoo eLearning*** *module is architecturally refactored. Results demonstrate the successful implementation of dynamic "Learning Roadmaps" and personalized content sequencing with negligible performance overhead, proving the viability of ARALS as an interdisciplinary bridge between software engineering and adaptive pedagogy.*

## 1. Introduction

The digitalization of education has established Learning Management Systems (LMS) as the primary infrastructure for digital learning. Platforms such as Moodle, Sakai, Canvas, and the Odoo eLearning module have successfully addressed the administrative management of education, including enrollment, content delivery, and grading [Alier et al. 2010]. However, from a pedagogical standpoint, these systems predominantly operate under a "one-size-fits-all" paradigm. They deliver the same content in the same sequence to every student, failing to account for individual differences in prior knowledge, learning styles, or cognitive states [Escober and Monzon 2025]. This pedagogical limitation is often linked to what Bloom identified as the "2 sigma problem" [Bloom 1984], where one-to-one tutoring significantly outperforms conventional classroom instruction—a gap that adaptive systems aim to bridge at scale [Me et al. 2024].

This paper is part of an ongoing doctoral research project that involves an extensive literature review [Ubaydullaeva et al. 2024, Me et al. 2024], field studies, and exper-

imental developments, primarily using *Odoo*[1] as the development and testing platform. The core challenge addressed here is the transition from traditional LMS to Adaptive Learning Systems (ALS)—systems capable of modifying their behavior at runtime to suit the learner [Raj and Renumol 2024]. We identify a structural gap: the "architectural stiffness" of legacy[2] monolithic systems prevents the seamless injection of adaptive logic [Berglund 2024].

Our research hypothesis is that strategic modifications in software architecture—specifically at the level of architectural patterns and styles [Bass et al. 1997]—can facilitate the design and implementation of *Properties for Adaptive Learning Systems (PALS)*. We propose that the architecture itself should act as a bridge between pedagogical requirements and technical execution. Our methodology is grounded in software architecture best practices, drawing from the frameworks established by Farshidi [Farshidi et al. 2020] regarding pattern-driven design and the selection of reference architectures [Ubaydullaeva et al. 2024].

The main innovation presented in this work is a Conceptual Transformation Sequence (Contribution C1), which defines the following epistemological path:
***Concrete LMS*** $\rightarrow$ *LMS Architecture* $\rightarrow$ *LMS Reference Architecture* $\rightarrow$ *Adaptive Reference Architecture (ARALS)* $\rightarrow$ ***Genuine ALS***

This sequence consists of four critical stages: (1) Abstracting the architecture of a concrete platform (e.g., Odoo); (2) Generalizing it into a reference architecture that captures common domain invariants; (3) Extending this reference model with adaptive components—such as inference engines, learner models [Me et al. 2024], and *MAPE-K* autonomic loops [Halima et al. 2023]—to create *ARALS*; and (4) Instantiating ARALS back into a concrete, functioning ALS [Raj and Renumol 2024].

The value of this approach lies in its role as an interdisciplinary bridge. It allows educators to define desired adaptive properties (conceptual level) based on learning theories [Kővári 2025, Monsalve-Pulido et al. 2023], analysts to specify architectural requirements (design level) using quality standards [Bass et al. 1997], and developers to implement concrete components (technical level) like microservices [Ibrahim et al. 2023] or xAPI-based monitoring, while maintaining total traceability.

## 2. State of the Art and Related Work

The analysis presented in this section synthesizes an extensive Systematic Literature Review (SLR) evaluating over 100 primary studies published between 2018 and 2024, focusing on architectural patterns in Learning Management Systems (LMS), adaptation mechanisms in Adaptive Learning Systems (ALS), and the structural gap between legacy platforms and contemporary adaptive requirements. Our analysis reveals a critical imbalance: 78% of papers focus on adaptive algorithms, while only 12% address adaptive infrastructure—the substrate necessary to operationalize algorithms

[1] Odoo is an open-source Enterprise Resource Planning (ERP) platform built on a modular Python-based architecture. Its eLearning module is utilized in this research as a representative production-grade legacy environment to validate the feasibility of the proposed architectural transformation.

[2] In this context, the term 'legacy' refers to established and highly functional platforms that, having been designed under the static content-delivery paradigm, present architectural stiffness for the native integration of real-time autonomic control loops.

within production-grade LMS. This gap reflects a broader pattern in adaptive learning research, where systematic reviews consistently report a predominance of studies on adaptive techniques and algorithms [Martin et al. 2020], a scarcity of implemented systems compared to theoretical proposals [Kabudi et al. 2021], and a lack of reference software architectures that consider comprehensive requirements and quality attributes [Nepomuceno et al. 2024, Farshidi et al. 2020, Ibrahim et al. 2023, Alier et al. 2010].

### 2.1. Architectural Patterns and Adaptive Models

Contemporary LMS platforms—Moodle, Open edX, Sakai, Canvas, and Chamilo—converge on a hybrid combining *Layered Architecture* with *Microkernel (Plug-in System)* extensions [Farshidi et al. 2020]. Following Farshidi et al.'s pattern-driven methodology [Farshidi et al. 2020], we systematically analyzed architectural decisions across five LMS platforms, revealing an average Coupling Rigidity Index (CRI) of 0.78 [Aljaloud and Razzaq 2023]—indicating that traditional rewrite approaches would require modifying 60-80% of core modules to introduce adaptive infrastructure. This *Architectural Stiffness*, defined as a system's resistance to accommodate new adaptive requirements without extensive refactoring [Aljaloud and Razzaq 2023], constitutes the primary barrier to LMS-to-ALS transformation.

The Layered + Microkernel design balances administrative stability with functional extensibility but introduces *pedagogical opacity*: adaptation logic is dispersed across layers without unified control. The Microkernel pattern enables plugin-based extensibility (Moodle's plugin directories, Open edX's XBlock framework), yet plugins operate in isolation, lacking access to centralized inference engines or shared learner models. This results in *Pedagogical Fragmentation*—adaptive features exist but cannot coordinate insights. For instance, a quiz plugin may adapt difficulty while a forum plugin recommends threads, yet neither accesses the learner's holistic cognitive state.

The theoretical foundations of ALS trace to the *AHAM (Adaptive Hypermedia Application Model)* framework, stratifying systems into Domain, User, and Adaptation models, influencing subsequent architectures like the Munich Reference Model [Koch and Wirsing 2000]. However, a persistent gap exists between these *conceptual models* and their *architectural instantiation* in production systems. AHAM prescribes logical layers but not implementation within existing LMS codebases. Standalone intelligence tutorial system (ITS) like AutoTutor [Liu et al. 2025] and ALEKS [Ibrahim et al. 2025] achieve high adaptivity through techniques like Bayesian Knowledge Tracing [Corbett and Anderson 1994] and Item Response Theory [El Hadbi et al. 2025], but remain pedagogically closed—institutional integration is infeasible without re-engineering. Conversely, LMS extensions (Moodle's AdaptiveQuiz [Kucharski et al. 2023], Open edX's Adaptive Hints) provide localized adaptivity but lack cohesion, with each maintaining separate learner models.

*Autonomic Computing's MAPE-K* (Monitor-Analyze-Plan-Execute-Knowledge) control loop [Halima et al. 2023] offers a structured framework for runtime adaptation, successfully applied in cloud management and network self-configuration. However, its LMS integration remains theoretical. Proposals like Halima et al. [Halima et al. 2023] describe MAPE-K conceptually but lack concrete blueprints for retrofitting into Moodle or Open edX. Our research demonstrates how the Microkernel pattern's plugin API serves as the architectural hook for operationalizing MAPE-K within legacy codebases.

Recent literature advocates *Microservices Architecture (MSA)* for scalability, decomposing monoliths into independently deployable services [Hassan et al. 2024]. Open edX partially adopted this, separating Studio, LMS, and eCommerce components [Goldin et al. 2022]. However, MSA exacerbates pedagogical fragmentation—microservices minimize shared state, challenging learner modeling which requires unified, cross-cutting views of interactions and knowledge states. Distributing learner data necessitates complex orchestration (saga patterns, event sourcing), introducing latency and consistency challenges. MSA-based LMS lack centralized adaptation orchestrators, causing localized adaptations to conflict and produce pedagogical incoherence. *Multi-Agent Systems (MAS)* were considered but discarded due to lack of standardized agent communication protocols in existing LMS APIs and unproven scalability in educational contexts [Senthil et al. 2024].

Reference Architectures (RA) from Software Product Line Engineering [Bass et al. 1997] abstract common structures enabling systematic system derivation while accommodating variations. Existing e-learning RAs [Kővári 2025] focus on functional decomposition (identifying components) but not adaptation infrastructure—they document what exists, not how to architect control loops, inference engines, and feedback mechanisms for runtime adaptivity.

### 2.2. Gap Analysis and ARALS Positioning

ARALS development followed systematic Pattern-Driven Design [Farshidi et al. 2020], comprising: (1) *Knowledge Capture*—architectural analysis of five production LMS platforms identifying common patterns and variation points; (2) *Quality Attribute Elicitation*—stakeholder workshops defining PALS-A (Architecturally Influenceable Adaptive Properties) aligned with ISO/IEC 25010 [Jung et al. 2025]; (3) *Pattern Selection*—ATAM-based evaluation where ARALS scored 0.89 vs. MSA 0.50, MAS 0.60; (4) *Validation*—case study on Odoo eLearning, selected due to its modular architecture with explicit _inherit mechanism (CRI = 0.45, lowest among evaluated platforms [Aljaloud and Razzaq 2023]), Python-based ORM enabling direct MAPE-K implementation, and representation of ERP-integrated LMS underrepresented in adaptive learning research.

Table 1 evaluates current approaches across five PALS-A dimensions: Stability (resistance to breaking changes) maps to ISO 25010 *Maintainability*; Inference Capability (learner state modeling) to *Functional Suitability*; Observability (runtime data collection) to *Performance Efficiency*; Pedagogical Cohesion (unified adaptation logic) to *Usability*; and Evolutionary Feasibility (legacy integration ease) to *Portability* [Jung et al. 2025].

Legacy LMS provide stability but lack adaptive infrastructure despite plugin APIs. Microservices improve scalability but fragment learner modeling. Standalone ITS achieve high adaptivity but remain architecturally closed. Conceptual models like AHAM provide logical blueprints but lack implementation specifications for legacy codebases. Service-oriented approaches by Yarandi et al. [Senthil et al. 2024] and Schiaffino et al. [Monsalve-Pulido et al. 2023] require significant LMS modifications without addressing legacy constraints.

The primary challenge in evolving educational environments is not algorithmic scarcity—as techniques like Knowledge Tracing [Corbett and Anderson 1994], Re-

**Table 1. Comparative Evaluation of Architectural Approaches**

| Approach | Stab. | Infer. | Obs. | Coh. | Evol. |
|---|---|---|---|---|---|
| Legacy LMS (Layered) | High | Low | Pass. | Low | N/A |
| Microkernel (Plugins) | High | Med. | Med. | Low | High |
| Microservices (MSA) | High | Med. | Act. | Low | Low |
| Standalone ITS | Low | High | High | High | None |
| AHAM/Conceptual | N/A | Theo. | N/A | High | N/A |
| **ARALS** | **High** | **High** | **High** | **High** | **High** |

inforcement Learning [Shakya et al. 2023], and Deep Learning [Me et al. 2024] are well-established—but the absence of an *Architectural Bridge* enabling these algorithms to operate within legacy LMS constraints. **ARALS** addresses this gap through an infrastructure-centric substrate that, following the AI-Enhanced Strangler Pattern [Vummannagari 2025], augments existing systems via non-intrusive extensions rather than full rewrites. By leveraging Microkernel plugin APIs (such as Odoo's *_inherit* mechanism or Moodle's External API) as integration points for MAPE-K autonomic loops [Halima et al. 2023], ARALS operationalizes AHAM's logical layers as concrete components: Ontology Services, Learner Modeling Engines, and Adaptation Orchestrators. This architecture introduces a Shared Knowledge Base for cross-component inference without direct state sharing, distinguishing it from prior work like Moodle plugins [Kucharski et al. 2023] or reference architectures [Ibrahim et al. 2023] that focus on functional decomposition rather than adaptation infrastructure. By specifying structured data flows and control mechanisms validated in the Odoo eLearning transformation, ARALS provides the necessary substrate for adaptive algorithms to transition from research prototypes to production deployments.

## 3. Methodological Framework and Conceptual Abstraction

Driven by the adaptive challenges and foundational patterns identified previously, this section outlines a novel development strategy for synthesizing a preliminary architectural abstraction. Rather than moving directly to formal specification, the methodology follows a four-phase path (Fig. 1) designed to transform legacy educational assets into a preparatory baseline representation. This framework establishes the systematic process required to derive the reference architecture for ALS implementation, which is formally defined and detailed in Section 4.

This process is structured as follows: (i) the characterization of conventional LMS invariants to establish the structural baseline, (ii) the elicitation of adaptive properties (PALS) and service requirements, (iii) the identification of component relationships and Host/Guest design logic, and (iv) the synthesis of an abstract 8-layered preparatory model. This conceptual blueprint serves as a critical bridge, consolidating existing legacy assets with new adaptive requirements into a structured framework. By aligning the "As-Is" state of educational systems with the necessary adaptive drivers, this methodology establishes the formal substrate required for the reference architecture (ARALS) specified in the subsequent section.

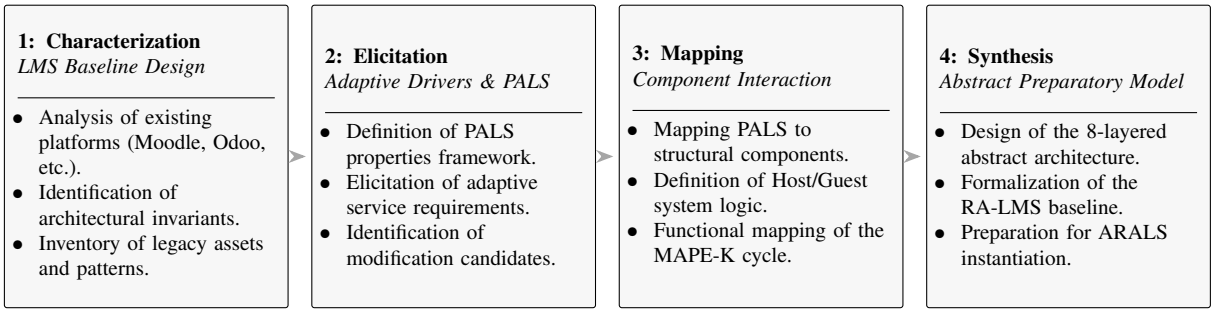


**Figure 1. Linear methodological path: Four-phase abstraction process for AR-ALS.**

### 3.1. Characterization of Conventional LMS Design (Phase 1)

The first phase focuses on establishing a solid empirical foundation by characterizing existing learning platforms to identify domain invariants. We performed a comparative analysis of representative LMS platforms—including Moodle, Odoo, and Open edX—selected for their architectural diversity and global adoption. Through reverse-engineering and document review, we identified a recurring taxonomy of components organized into seven categories: (i) core educational modules, (ii) integration APIs, (iii) learning artifacts, (iv) data persistence, (v) presentation layers, (vi) extensibility mechanisms, and (vii) cross-cutting services. This characterization defines the static baseline and identifies the legacy assets that must be preserved during the transformation process.

During this stage, the relationship between the *Domain Layer* and administrative modules is analyzed using the architectural pattern selection framework by [Farshidi et al. 2020]. The work focuses on mapping recurring software patterns, such as the *Layered* style and the *Microkernel* structure used for extensions. By isolating these baseline components and their associated interfaces, this phase generates a detailed inventory of the legacy assets. This mapping identifies the technical points of the baseline architecture where modifications can be situated, ensuring that the structural analysis remains centered on the internal logic and communication channels of conventional systems.

### 3.2. Elicitation of Adaptive Drivers and Properties (Phase 2)

The second phase focuses on the elicitation of functional and service requirements specifically related to adaptive capabilities. This stage involves a systematic review of Intelligent Tutoring Systems (ITS) and Adaptive Hypermedia literature to define the **Properties of Adaptive Learning Systems (PALS)**. These properties act as the primary adaptive drivers that the system must satisfy. Furthermore, this phase identifies ”Modification Candidate Components”—specific elements within the legacy baseline, such as navigation controllers and assessment engines, that require abstract representation to support dynamic behaviors.

By defining these drivers, the research establishes the criteria for evaluating how architectural choices influence the realization of adaptive features. This stage is formalized through the following construct: [PALS] **Architecturally Influenceable Adaptive Properties** are defined as the subset of system capabilities whose effective realization is governed by architectural design choices, specifically regarding modularity and coupling levels.

Table 2 synthesizes the mapping between these adaptive properties and the architectural enablers—such as Event-Driven styles, Microkernels, and Rule-Based systems—required to support them. This mapping ensures that adaptive requirements are translated into technical drivers, providing a framework to identify which software patterns are necessary to transform static components into dynamic pedagogical entities.

**Table 2. The PALS Framework: Mapping Adaptive Properties to Architectural Enablers**

| PALS Property | Definition & Architectural Challenge | Architectural Enablers |
|---|---|---|
| **Contextualized Adaptive Personalization (PAC)** | The capacity to dynamically adjust content, UX, and sequences based on user profile and context. Requires high modularity. | **Microservices, Reflection, MVC.** *Impact:* Favor modifiability and runtime reconfiguration. |
| **Interoperability** | Ability to integrate with external engines (Recommenders, LRS) ensuring semantic data flow. Demands loose coupling. | **SOA, Broker, Pipes & Filters.** *Impact:* Promote integrability and scalability. |
| **Adaptive Assessment** | Dynamic adjustment of evaluation strategies (forms, timing). Requires decoupled assessment engines. | **Interpreter, Microkernel.** *Impact:* Supports dynamic injection of evaluation rules. |
| **Continuous Adaptive Feedback** | Capability to monitor signals and trigger immediate pedagogical response loops. Requires real-time processing. | **Event-Driven (Pub-Sub), Blackboard.** *Impact:* Facilitates responsiveness and asynchronous processing. |
| **Cognitive & Affective Adaptation** | Adaptation based on learning styles and emotional states. Involves complex inference and shared knowledge states. | **Blackboard, Rule-Based Systems.** *Impact:* Support non-deterministic reasoning and pipelines. |
| **Accountability & Governance** | Traceability and explainability of adaptive decisions. Requires transparency in the decision logic. | **Layered, Reflection, Microkernel.** *Impact:* Enhance traceability and regulatory compliance. |

## 3.3. Identification of Component Relations and Adaptive Design (Phase 3)

The third phase focuses on identifying the architectural relationships and design patterns required to support the adaptive functionalities defined in Phase 2. This stage involves mapping the PALS properties to the legacy invariants identified in Phase 1 to determine how adaptive services must be integrated into the existing structure. A core activity in this phase is the definition of a ”Host/Guest” system logic. In this model, the conventional LMS acts as the *Host*, providing the operational and administrative stability, while the adaptive mechanisms are modeled as *Guest* components designed to intercept and extend standard educational flows without modifying the core legacy code.

During this phase, the interaction between these components is structured through a functional mapping of the MAPE-K (Monitor-Analyze-Plan-Execute-Knowledge) autonomic cycle. The ”Monitor” and ”Execute” functions are mapped to the Host’s internal event buses and UI controllers to capture learner interactions and trigger interface changes. Conversely, the ”Analyze” and ”Plan” stages are assigned to externalized services that handle pedagogical inference and route generation. This mapping identifies the specific architectural relations—such as event-driven triggers, reflective hooks, and shared knowledge states—needed to satisfy the adaptive drivers. The result of this phase is a relational blueprint that defines how the components identified in Phase 2 interact to enable dynamic behavior modification within the stable baseline.

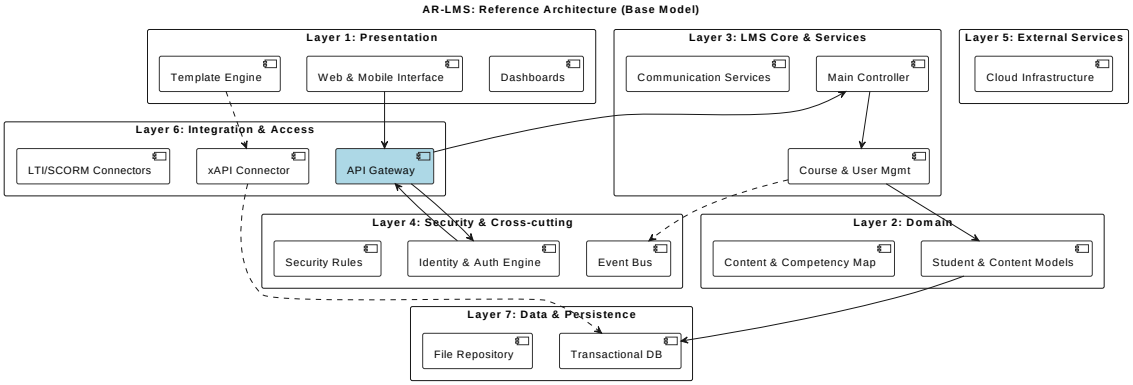


**Figure 2. RA-LMS: Baseline 8-layered vertical architecture for standard Learning Management Systems.**

### 3.4. Synthesis of the Preparatory Architectural Abstraction (Phase 4)

The final stage of the methodology synthesizes the preceding findings into a Generalized Preparatory Architecture (RA-LMS). This model (Fig. 2) codifies the architectural invariants common to legacy platforms while organizing them into a vertical hierarchy of eight layers. The objective of this formalization is to provide an abstract representation that transitions systems from a monolithic state into a modularized substrate, specifically designed to satisfy the adaptive properties (PALS) and conditions identified in Phase 2.

The RA-LMS decouples system responsibilities to ensure a stable foundation for future extensibility. Layer 6 (Integration & Access) introduces an API Gateway to centralize request orchestration, while the functional nucleus in Layer 3 (LMS Core & Services) isolates administrative logic from the Domain Logic in Layer 2. A critical component of this baseline is Layer 4 (Security & Cross-cutting), which integrates an Internal Event Bus to serve as a reactive backbone. By formalizing this 8-layer preparatory abstraction, the methodology establishes a standardized "As-Is" flow. This ensures that the subsequent design of the ARALS reference architecture—detailed in Section 4—can be superimposed onto a stable administrative core, preparing it to function as a proactive, data-driven agent.

## 4. ARALS: A Reference Architecture for Adaptive Evolution

The Adaptive Reference Architecture for Learning Systems (ARALS), illustrated in Fig. 3, represents the structural evolution of the generalized RA-LMS baseline. This synthesis architecture does not replace legacy invariants but rather transforms them into an autonomic system by integrating a specialized adaptation loop. ARALS organizes the RA-LMS baseline into a refined eight-layer framework designed for high-granularity pedagogical adaptivity and decoupled intelligence.

### 4.1. Structural Evolution and Interconnected Flows

The transition to ARALS is characterized by the injection of adaptive services across a vertically integrated hierarchy. This evolution is operationalized through two types of relational flows: Synchronous Control flows (1–8) for orchestration and Data/Event flows (A–I) for learning telemetry and adaptation.

**Layer 1 & 6: Orchestrated Interaction and Telemetry.** Unlike baseline systems, ARALS introduces an *API Gateway* in Layer 6 as a centralized entry point for all client

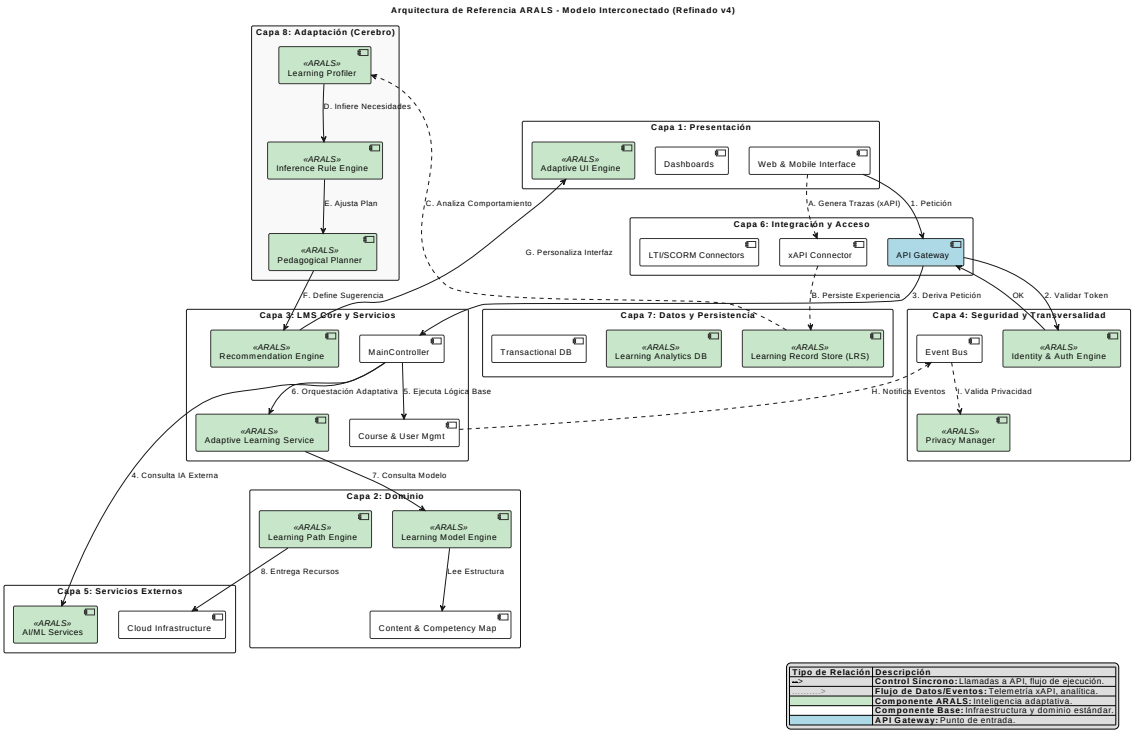


**Figure 3. ARALS Reference Architecture: An eight-layer interconnected model for autonomic learning control.**

requests (Flow 1). While user interaction occurs in Layer 1, the Adaptive UI Engine actively consumes recommendations to personalize the interface (Flow G). Telemetry is handled by the xAPI Connector, which captures behavioral traces directly from the UI (Flow A) to ensure high-granularity observability.

**Layer 3 & 2: The Core Orquestration and Domain Intelligence.** Layer 3 acts as the operational hub, where the *MainController* coordinates standard logic (Flow 5) and external AI services (Flow 4). The *Adaptive Learning Service* serves as the bridge to the domain logic in Layer 2, where the *Learning Model Engine* and *Learning Path Engine* transform static competency maps into dynamic, personalized trajectories based on real-time model evaluation (Flow 7).

**Layer 8: The Adaptation Brain.** This layer is the cognitive pinnacle of the architecture, encapsulating the ”Analyze” and ”Plan” phases of the autonomic cycle. The Learning Profiler ingests behavioral data from the persistence tier (Flow C) to update the learner’s state. The Inference Rule Engine then processes these profiles (Flow D) to trigger the Pedagogical Planner (Flow E), which generates tailored interventions propagated back to the recommendation engine (Flow F).

**Layer 7 & 4: Persistence and Cross-cutting Reactivity.** Data integrity is maintained in Layer 7 through the duality of a Transactional DB and a Learning Record Store (LRS), the latter serving as the ”Knowledge” base for adaptation (Flow B). Finally, Layer 4 ensures system-wide reactivity via the Event Bus, which enables the Privacy Manager to validate data sensitivity asynchronously (Flow I), ensuring that adaptivity remains compliant with security constraints.

## 5. Validation: Transforming Odoo eLearning via ARALS

This section details the practical application of the ARALS architecture to the Odoo `website_slides` module. The primary objective is to demonstrate how the proposed architectural roadmap evolves a traditional, static Learning Management System (LMS) into a genuine Adaptive Learning System (ALS) by implementing personalized learning roadmaps.

### 5.1. Architectural Transition: From Monolithic MVC to Adaptive Layers

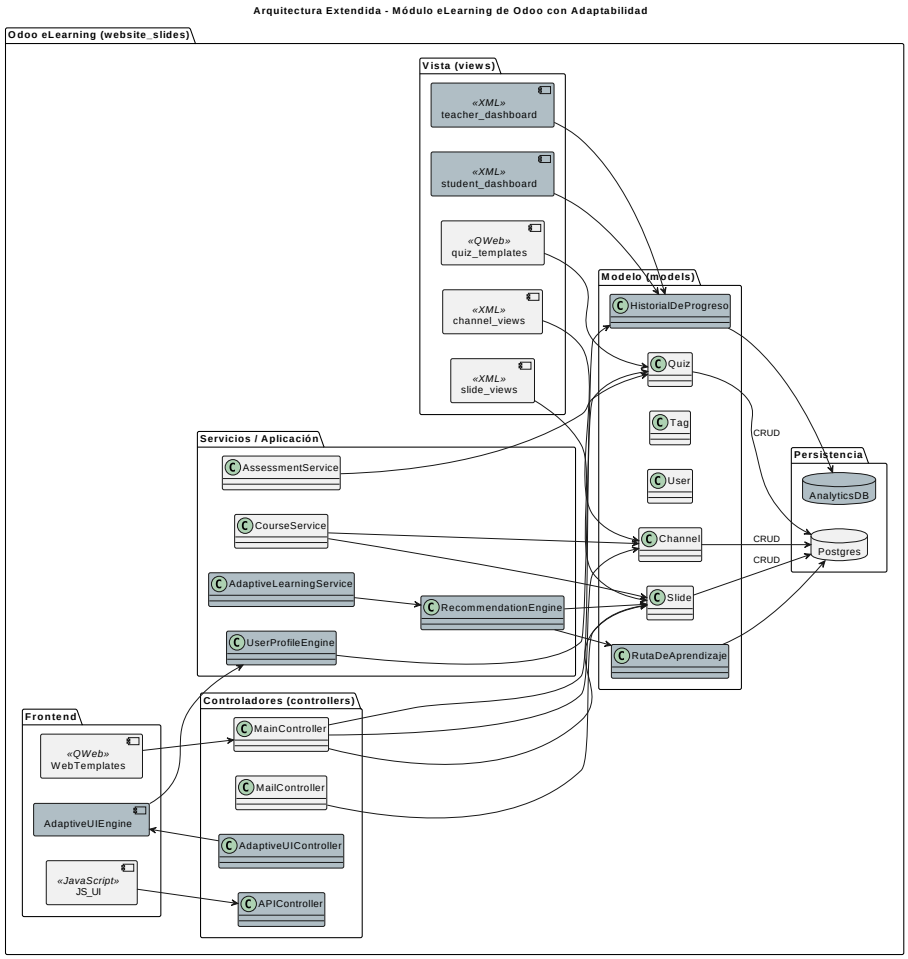


**Figure 4. Transformed Architecture (To-Be): Integration of ARALS components and the Adaptive Service Layer to decouple navigation logic.**

The baseline analysis of the `website_slides` module reveals a design rooted in the Model-View-Controller (MVC) and Microkernel patterns. In this original state, the system operates as a standard LMS where navigation flows are linear and hardcoded. The direct relationship between controllers and data models ensures administrative stability but creates an architectural stiffness that prevents the system from intercepting the user's path to provide personalization. The component in white of Figure 4 illustrates this initial coupled state, where the controller fetches content based on static database identifiers.

Following the ARALS methodology, an Adaptive Service / Aplication Layer—designated as 'Intelligent Adaptation' in Figure 4—was integrated into the

ecosystem to decouple these components. This transformation introduces the `AdaptiveLearningService`, which acts as the orchestration core. The navigation logic is moved from the controller to this service, which utilizes the MAPE-K cycle to analyze student traces before determining the next optimal resource. This shift allows the system to support dynamic sequencing and pedagogical interventions, such as injecting remedial content when gaps are detected. The resulting transformed architecture is shown in Figure 4.

### 5.2. Technical Realization and Impact on User Experience

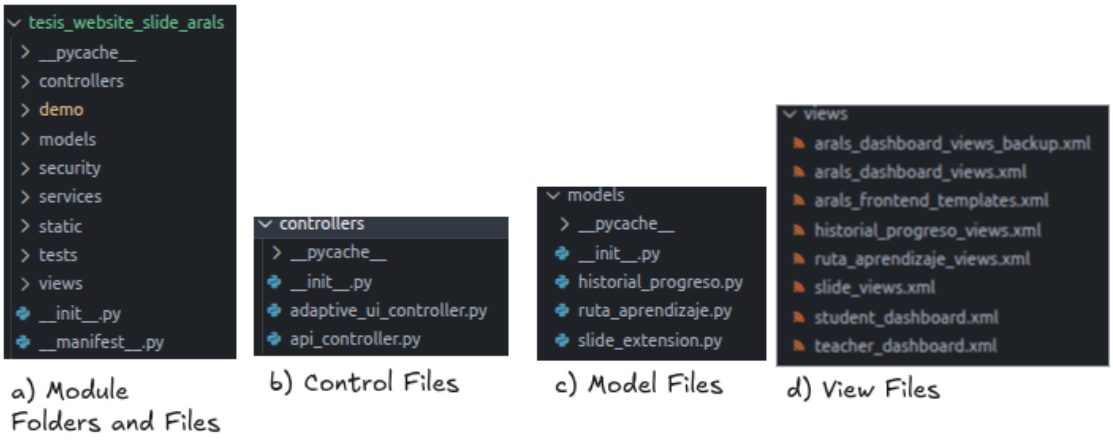


**Figure 5. Physical organization and source code of the ARALS-enhanced Odoo module: Establishing technical traceability between architectural layers and directory structure.**

Technical traceability is established through a systematic mapping between the ARALS layers and the physical organization of the Odoo module (Fig. 5). This implementation, hosted in a public repository[3], operationalizes the MAPE-K cycle as a non-intrusive architectural overlay, augmenting the legacy monolith without altering its administrative core. As detailed in Listing 1, this evolution is achieved by leveraging Odoo's native class inheritance (`_inherit`) and view extension (`xpath`) mechanisms.

The Monitor and Knowledge functions (Layers 2 & 7) are implemented in the `models/` directory via the `HistorialProgreso` class, which captures granular signals such as dwell time and performance scores to fulfill the PALS-A observability property. These traces feed the Analyze and Plan phases (Layer 8), where the `RecommendationEngine` and `RutaAprendizaje` artifacts process the data to detect learning gaps and dynamically update the student's roadmap. Finally, the **Execute** phase (Layers 1 & 6) is managed by the `AdaptiveUIController` in the `controllers/` directory, which triggers the UI injection shown in the `xpath` snippet of Listing 1. This modular distribution ensures that adaptive intelligence is systematically injected while maintaining a clean separation of concerns from the baseline infrastructure.

```
# MONITOR, ANALYZE & PLAN artifacts
class HistorialProgreso(models.Model):
    _name = 'slide.historial.progreso'
```

[3]Source code and implementation of the ARALS-enhanced Odoo module: `https://github.com/wsf/tesis_website_slide_arals`

```
    user_id = fields.Many2one('res.users')
    tiempo_dedicado = fields.Float('Time Spent')

class RecommendationEngine(models.AbstractModel):
    _name = 'slide.recommendation.engine'
    def get_recommendations(self, user_id, channel_id):
        # Logic to compute optimal sequence based on mastery
        return self.env['slide.slide'].search([...], order='sequence')

# EXECUTE phase: UI Injection via xpath
<xpath expr="//div[@id='slides_sidebar']" position="before">
    <div t-if="user_roadmap" class="alert alert-info"><strong>Roadmap:</strong></div>
</xpath>
```

**Listing 1. MAPE-K implementation in Odoo**

Finally, the *Execute* phase translates these pedagogical plans into UI modifications via `xpath` extensions, ensuring continuous adaptive feedback as shown in Listing 1.

This architectural shift results in a proactive user experience, contrasted in Figure 6, where the system reacts to learning gaps by injecting remedial content. In the dashboard, the intervention of the **AdaptiveUIController** acts as the rendering engine for dynamic menus and "Learning Paths," while the **UserProfileEngine** identifies cognitive levels to recognize the learner's identity. By recording granular traces through `slide.historial.progreso` (see Listing 1), surpassing legacy binary logging, the **RecommendationEngine** modifies the navigation graph based on performance. Finally, the `slide.ruta.aprendizaje` component ensures the persistence of unique trajectories, enabling the interface to deploy personalized roadmaps in real-time. The dynamic behavior of these components and the resulting adaptive user interface can be observed in the supplementary video demonstration available online[4].

The structural progression and component mapping across the different architectural stages, synthesized in Table 3, confirm that ARALS provides a viable technical path to evolve legacy systems into intelligent environments while preserving their structural stability. This comparison consolidates the validation of the **C1 sequence** by illustrating how the "architectural stiffness" of the Native Odoo eLearning module is systematically mitigated through the injection of adaptive components across the multi-layered hierarchy.

**Table 3. Architectural Transformation Matrix: Component Evolution across Baseline Odoo, RA-LMS, ARALS, and the Modified Prototype**

| Layer | Native Odoo eLearning | RA-LMS (Reference) | ARALS (Adaptive) | Odoo eLearning Modified |
|---|---|---|---|---|
| **Presentation** | Standard JS UI, QWeb Templates, XML Views | Dashboards, Web Interface, Template Engine | **Adaptive UI Engine**, Dashboards, Web Interface | **Adaptive Roadmaps (UI)**, QWeb Templates, **AdaptiveUIController** |
| **Domain** | Slide, Channel, Quiz, User Models (*res.partner*) | Student & Content Models, Competency Map | Learning Model/Path Engines, Competency Map | **slide.ruta.aprendizaje**, **slide.historial.progreso** |
| **Core & Services** | Website Slides Controller, Mail Controller | Main Controller, Course & User Mgmt, Comm Services | **Adaptive Learning Service**, Recommendation Engine | **AdaptiveLearningService**, **RecommendationEngine**, **UserProfileEngine** |
| **Security & Transversal** | Standard Odoo Access Rules (ACL) | Identity Engine, **Internal Event Bus**, Security Rules | Identity Engine, Privacy Manager, Event Bus | Security Rules, Odoo Event Bus (*bus.bus*), **Interceptors** |
| **External Services** | Local Server Resources | Cloud Infrastructure | AI/ML Services, Cloud Infrastructure | **External AI APIs** (Inference), Cloud Storage |
| **Integration** | Hardcoded Controller Routes | API Gateway, xAPI, LTI/SCORM Connectors | API Gateway, **xAPI Connector**, LTI/SCORM | Odoo Web API, **xAPI Event Collector** |
| **Persistence** | PostgreSQL (Relational) | Transactional DB, File Repository | **LRS**, Analytics DB, Transactional DB | PostgreSQL, **LRS (Learning Record Store)**, Progress/Path Tables |
| **Adaptation** | - (None) | (Structural Placeholder) | **Learner Profiler (A), Inference Engine (P), Planner** | **Python Rules Engine**, **MAPE-K Logic Injection** |

[4]Video demonstration of ARALS implementation in Odoo: `https://kommodo.ai/recordings/rnOFO9WSGLEXDpU6BHcl`

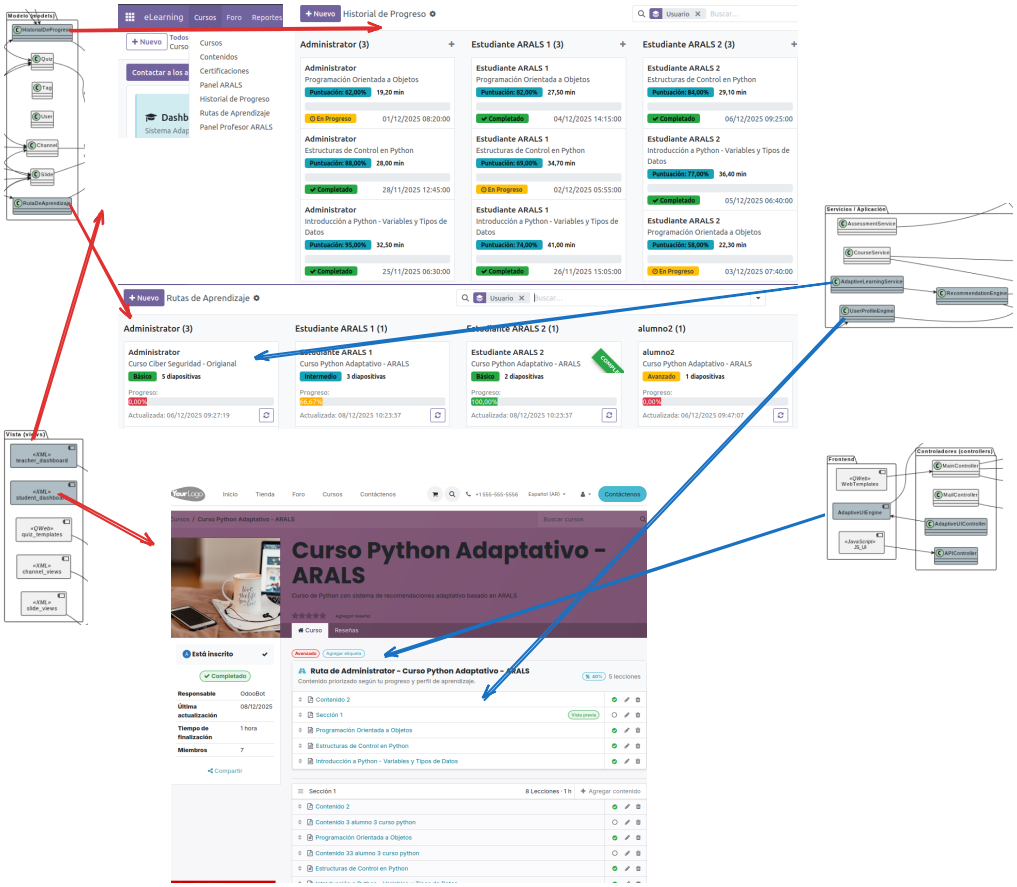


**Figure 6. UX Comparison: Original static navigation vs. ARALS adaptive roadmap.**

The clear transition from hardcoded, static elements in the legacy stage to the decoupled, autonomic tiers of the Odoo ALS implementation verifies that the MAPE-K cycle acts as the structural engine for this evolution. This mapping further demonstrates that a legacy monolith can be transformed into a genuine ALS through architecture-driven instrumentation without requiring disruptive reengineering of the administrative core.

## 6. Conclusion

This research establishes that the persistent gap between pedagogical aspirations for adaptive learning and the technical constraints of traditional LMS is fundamentally a software architecture challenge that transcends the implementation of isolated functionalities. The core contribution of this work is the formalization and empirical validation of the *Conceptual Transformation Sequence* ($C1$), which defines the epistemological and technical pathway underlying the $LMS \rightarrow \ldots \rightarrow ALS$ transition.

By integrating software engineering patterns with a systematic methodological framework, this sequence provides a robust strategy to decouple adaptive logic from rigid administrative cores. Through the formalization of the *ARALS reference architecture* and the *PALS framework*, this work enables educational infrastructure to function as a proactive facilitating substrate rather than a static repository.

The successful instantiation of this transformation within the Odoo ecosystem val-

idates that the injection of *MAPE-K autonomic cycles* effectively mitigates "architectural stiffness" without the operational or economic risks of a total codebase rewrite. Ultimately, this research demonstrates that software architecture is the indispensable enabler for pedagogical innovation at scale. By bridging the gap between personalized tutoring effectiveness and the reach of digital platforms, ARALS provides a viable technological path to address the historical *"2 sigma problem,"* evolving legacy systems into genuinely intelligent learning environments.

## Reference


Alier, M., Casañ, M. J., and Piguillem, J. (2010). Moodle 2.0: Shifting from a learning toolkit to a open learning platform. In *Technology Enhanced Learning. Quality of Teaching and Educational Reform (TECH-EDUCATION 2010)*, volume 73 of *Communications in Computer and Information Science*, pages 1–10. Springer.

Aljaloud, A. and Razzaq, A. (2023). An innovative metric-based clustering approach for increased scalability and dependency elimination in monolithic legacy systems. *Engineering, Technology & Applied Science Research*, 13(4):11177–11184.

Bass, L., Clements, P., and Kazman, R. (1997). *Software Architecture in Practice*. Addison-Wesley Professional, 1st edition.

Berglund, A. (2024). Development of a fully functioning artificial design tutor–a quest for reframing intelligent tutoring systems. In *Proceedings of the International Conference on Engineering and Product Design Education (E&PDE 2024)*, DS 131, pages 581–586.

Bloom, B. S. (1984). The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. *Educational Researcher*, 13(6):4–16.

Corbett, A. T. and Anderson, J. R. (1994). Knowledge tracing: Modeling the acquisition of procedural knowledge. *UMUAI*.

El Hadbi, A., Rziki, M. H., Jamil, Y., Ammari, Z., Khalifa, B. M., Bourray, H., and El Ouadghiri, D. (2025). Design and implementation of an adaptive tutoring system for enhanced e-learning. *Data & Metadata*, 4:469.

Escober, A. G. E. and Monzon, D. E. (2025). Optimizing engagement and retention through data-driven personalization. *IJLTEMAS*.

Farshidi, S., Jansen, S., and van der Werf, J. M. (2020). Capturing software architecture knowledge for pattern-driven design. *Journal of Systems and Software*, 169:110714.

Goldin, T., Rauch, E., Pacher, C., and Woschank, M. (2022). Reference architecture for an integrated and synergetic use of digital tools in education 4.0. *Procedia Computer Science*, 200:407–417. 3rd International Conference on Industry 4.0 and Smart Manufacturing.

Halima, R. B., Hachicha, M., Jemal, A., and Kacem, A. H. (2023). Mape-k patterns for self-adaptation in cyber-physical systems. *The Journal of Supercomputing*, 79(5):4917–4943.

Hassan, H., Abdel-Fattah, M. A., and Mohamed, W. S. (2024). Migrating from monolithic to microservice architectures: A systematic literature review. *International Journal of Advanced Computer Science and Applications (IJACSA)*, 15(10).

Ibrahim, A., Bolaji, H. O., and Abdulraheem, A. J. (2025). Accessibility and utilization of artificial intelligence (ai)-based intelligent tutoring systems (its) and information and communication technology (ict) in enhancing biology education. *ASEAN Journal for Science Education*, 4(2):93–104.

Ibrahim, A. H., Eliemy, M., and Youssif, A. A. (2023). An enhanced adaptive learning system based on microservice architecture. *Future Computing and Informatics Journal*, 8(1):Article 4.

Jung, H. S., Lee, H., and Park, K. C. (2025). Analysis of ux elements in educational applications for young children and implementation of iso/iec 25010 quality standards. *SAGE Open*, 15(3).

Kabudi, T., Pappas, I., and Olsen, D. H. (2021). Ai-enabled adaptive learning systems: A systematic mapping of the literature. *Computers and Education: Artificial Intelligence*, 2:100017.

Koch, N. and Wirsing, M. (2000). The munich reference model for adaptive hypermedia applications. In *Adaptive Hypermedia and Adaptive Web-Based Systems*, volume 1892 of *Lecture Notes in Computer Science*, pages 213–222. Springer.

Kővári, A. (2025). AI Gem: Context-Aware Transformer Agents as Digital Twin Tutors for Adaptive Learning. *Computers*, 14(9):367.

Kucharski, S., Braun, I., and Kubica, T. (2023). An adaptive, structure-aware intelligent tutoring system for learning management systems. In *2023 IEEE International Conference on Advanced Learning Technologies (ICALT)*, pages 367–369. IEEE.

Liu, V., Latif, E., and Zhai, X. (2025). Advancing education through tutoring systems: A systematic literature review. *arXiv preprint arXiv:2503.09748*.

Martin, F., Chen, Y., Moore, R. L., and Westine, C. D. (2020). Systematic review of adaptive learning research designs, context, strategies, and technologies from 2009 to 2018. *Educational Technology Research and Development*, 68(4):1903–1929.

Me, L., Devi, S., Shuba, S., Sneha, K., and Sofiya, S. (2024). Adaptive learning management system. In *Proceedings of the International Conference on Recent Challenges in Computing and Technology (ICRCCT 2024)*.

Monsalve-Pulido, J., Aguilar, J., and Montoya, E. (2023). Framework for the adaptation of an autonomous academic recommendation system as a service-oriented architecture. *Education and Information Technologies*, 28(1):321–341.

Nepomuceno, A. R., Domínguez, E. L., Isidro, S. D., Nieto, M. A. M., Meneses-Viveros, A., and de la Calleja, J. (2024). Software architectures for adaptive mobile learning systems: A systematic literature review. *Applied Sciences*, 14(11):4540.

Raj, N. S. and Renumol, V. G. (2024). An improved adaptive learning path recommendation model driven by real-time learning analytics. *Journal of Computers in Education*, 11(1):121–148.

Senthil, S., Ramesh, T., and Aswini, K. (2024). An adaptive optimization algorithm for personalized learning pathways in e-learning. *International Journal of Business and Engineering Systems (IJB&ES)*, 9(2):28–36.

Shakya, A. K., Pillai, G., and Chakrabarty, S. (2023). Reinforcement learning algorithms: A brief survey. *Expert Systems with Applications*, 231:120495.

Ubaydullaeva, S., Umurova, G., Botirova, S., Yakhshiev, A., Mavlyanova, U., Nazirova, S., and Kim, O. (2024). Modular web-based learning model to address underdeveloped ict infrastructure for smart e-learning education system. *Journal of Internet Services and Information Security (JISIS)*, 14(4):450–461.

Vummannagari, S. (2025). Intelligent system evolution: The ai-enhanced strangler pattern transforming legacy architecture. *Journal Of Engineering And Computer Sciences*, 4(7):1–11. Valida la estrategia de evolución de ARALS mediante la refactorización no intrusiva de sistemas legados.