Achieving Zero-Effort, Quasi-Zero Cost Integration with RTDC, and Application-Aware AI

Stephane H. Maes[1]

February 19, 2026[2]

Abstract

Enterprise AI is currently facing a massive spending problem. Companies are pouring billions into foundation models and infrastructure, yet 95% of these projects never make it out of the testing phase. The issue isn't the AI itself; the problem is how we try to force modern, probabilistic models to work with rigid, decades-old business systems. Most engineering teams rely on manual coding, and fragile API wrappers to connect the two. It is a cycle that drains budgets, creates blind spots, and breaks constantly.

This paper takes a completely different approach. Instead of bolting a generic AI chatbot, or an AI agent, onto the outside of an application, Real-Time Discovery and (self) Coding (RTDC) engine embeds directly into your existing stack. It works as an autonomous digital workforce that actively scans an enterprise systems, understands the enterprise underlying business rules, and writes its own integration code on the spot. This function of an Application-Aware AI platform completely removes the need for manual data mapping, giving AI teams instantaneous enterprise integration with zero manual effort, and at a quasi-zero cost.

RTDC integrations is designed for AI use cases, but it can also be used for traditional enterprise system integration situations.

With RTDC, forward deployed engineering teams, can be significantly replaced, or complemented, with a forward deployed team of AI agent workers performing the tasks of RTDC for application-aware AI.

Zenera product offering is an example of RTDC on application-aware agentic AI platform. There are other platforms that provide more limited variations of the idea.

---

1. Introduction

[1] shmaes.physics@gmail.com

[2] *Text added on April 11, 2026: This is a public version of the original Zenera white paper. Some of the secret sauce, and confidential information has been removed. More details can be found in other references [a3,a7-a12,a16-a19], and in particular full technical and implementation details are in [a7], and derived pending applications. This version has been published on April 11, 2026. The original is available with the right credentials on the Zenera web site [a3]. Text (noted so), and references (in the text), introduced after February 22, 2026, are in italic. Captions are also in italic but came from the original white paper.*

Cite as: Stephane H. Maes, (2026), "Achieving Zero-Effort, Quasi-Zero Cost Integration with Zenera RTDC, and Application-Aware AI", https://zenodo.org/doi/10.5281/zenodo.19655878, https://shmaes.wordpress.com/2026/04/18/achieving-zero-effort-quasi-zero-cost-integration-with-zenera-rtdc-and-application-aware-ai/, February 19, 2026. (https://osf.io/vz6jt/files/bfsr9)

The enterprise software market is facing a massive contradiction. Companies are pouring an estimated $30 billion to $40 billion a year into Generative AI infrastructure, buying up foundation models, advanced compute clusters, and expensive enterprise licenses. Yet, the return on this investment is practically invisible for most [1,2,6].

The State of AI in Business 2025 report published by MIT NANDA surveyed over 300 deployments and found a hard truth: 95% of enterprise AI pilots fail to make it to production or deliver a measurable financial return [1,2,6]. We call this the GenAI Divide. A tiny fraction of companies, i.e., about 5%, are seeing rapid revenue acceleration, while everyone else is stuck permanently in the pilot purgatory [2].

Note that this isn't an issue with the AI models themselves. Token limits, even if annoying, and algorithmic intelligence are fine. A big part of the problem is a structural failure in how we integrate these models into the business. The first wave of enterprise AI tried to bolt generic conversational chatbots, and Co-pilots, onto the periphery of deeply complex, 20-year-old legacy applications. It didn't work. Recent experience with These disconnected tools operate in a vacuum. They don't understand the specific workflows of a business, and they force users to constantly switch contexts between their core systems and an isolated chat window. These and later agentic efforts also stall often because of the challenges to access complete and up to date context across an enterprise heterogeneous environment, typically by AI teams, unfamiliar with these environments, while the rest of the IT and enterprise teams have lost or never mastered the details that are actually required.

To get past the 95% failure rate, the underlying architecture has to change. We have introduced a new category of (agentic) AI platform and tools: Application-Aware AI [3,9-*14,19-23*], driven by a Real-Time Discovery and (self) Coding (RTDC) engine, transactional memory, model of constraints and a meta-agent [9,*10,19,21*]. Rather than acting as a static assistant waiting for prompts, the platform embeds directly into the application stack, with models of constraints, explainability and 100% accuracy. It acts as an autonomous digital workforce that can discover system schemas, infer business logic, and write its own integration code on the fly (and interacts with the developer or the user when needed, building its own UI on the fly to do so). The result is a completely new pattern for enterprise integration. It is one that requires zero manual mapping, operates at quasi-zero cost, and bridges the gap between modern AI and legacy business physics. Developed with AI, for AI, it can also provide quasi real-time effortless traditional enterprise integrations.

Some companies like Palantir, and initiatives like IFS Nexus Black, build on human heavy teams of forward deployed engineers to deploy enterprise AI. With RTDC on an enterprise application-aware agentic AI platform, most of the work of these teams can be replaced by AI agent workers performing the discovery and self-coding. Note that we are not talking of consulting companies efforts to automate with AI requirements to design steps. We are talking of automating the actual coding and deployment of these. Of course, any meta data including any automated collection of requirements and generation of design done by these company can be ingested by the RTDC and self-coding on an application-aware platform.

The constraints brought in by the application-aware platform [*9,10,21*], ensure that AI performance can be brought to 100%, therefore becoming essentially independent of the underlying LLMs [3], including (Self) coding, which has

[3] This leads us to argue that current investment levels, including upcoming IPOs, may not warrant the amount of money involved, and call for a stronger focus on enterprise agentic AI offerings, decoupled from the underlying LLM technology [*24*]. This is not just a personal expert opinion, but it is dictated also by the need of many companies, and organizations, to use AI on-premise or in air-gapped mode. These require being able to replace frontier public models with models able to run on-premise, or air-gapped, like open-source / open weight LLM, without degradation of the performances, something that Zenera has validated in actual deployments at major customers who switched for example between OpenAI ChatGPT and Gemini [3].

its own SLDC (Software Development Lifecycle) constrained by the platform, and therefore addressing better [*10,13,14*] all the typical issues with AI coding assistant and vibe coding, as discussed in [15-*18*] and references therein.

2. The Deployment Paradox and the Stitching Trap

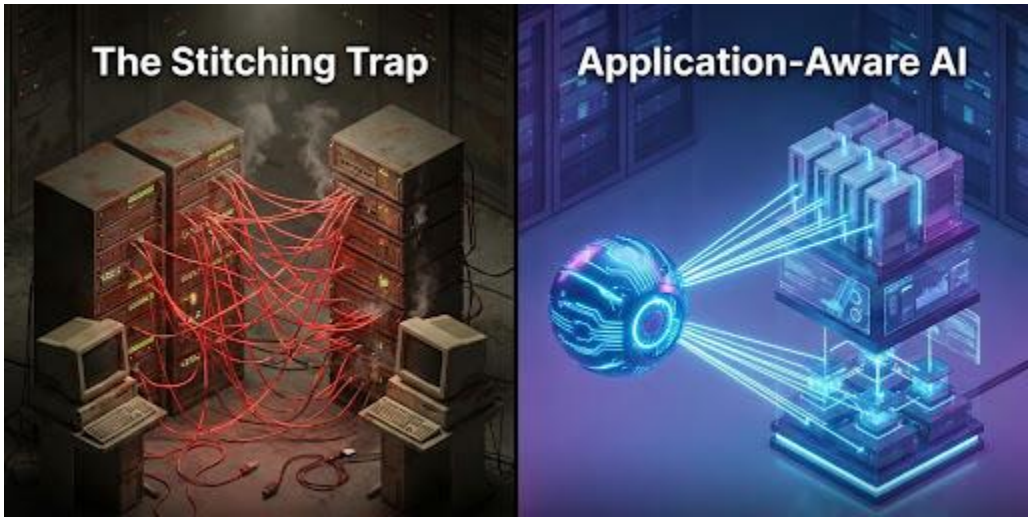

Figure 1: : A visual comparison showing the brittle, manual nature of traditional API wrappers (The Stitching Trap) versus the fluid, embedded intelligence of Application-Aware AI. Traditional manual integration often leads to a Stitching Trap, i.e., a complex and fragile web of custom API wrappers that are difficult to maintain, and may not be well known or understood by the AI developers, worse they may not exist. In contrast, Application-Aware AI embeds intelligence directly into the application stack, creating a more fluid and resilient system. It does so by discovery (crawling, scanning APIs, data repositories and meta data like documentation, manuals, enterprise architecture systems, etc.) and AI coding any missing or ill-defined APIs (and missing logic or UI).

The primary roadblock keeping organizations trapped in pilot purgatory is something we call the Stitching Trap. Most enterprise IT environments are a messy mix of modern microservices, proprietary ERPs, legacy mainframes, and unstructured document repositories. These legacy systems run on strict, deterministic logic. That rigid logic clashes entirely with the probabilistic nature of modern LLMs.

When leadership demands AI integration, engineering teams usually default to the only tool that they know, i.e., manual integration. They try to hand-code custom API wrappers to stitch the new AI to the old system, that they may barely know or that depend on vendors' data. This approach breaks down quickly due to three main bottlenecks:

- The API Bottleneck: Most legacy systems don't have the clean RESTful, or GraphQL APIs that modern AI agents need to function. Building these wrappers from scratch forces teams to reverse-engineer old codebases, trapping them in an 18-to-36-month R&D cycle. By the time the wrapper is ready, the business requirements or the AI models have changed, rendering the work obsolete.

- Schema Opacity and Context Vacuum: Enterprise data schemas, especially in highly customized third-party ERPs, are often opaque and poorly documented, or even missing and implemented, for example, with PL-SQL with projections changing with the context[4]. The engineers tasked with deploying the AI rarely understand the nuanced context of the legacy system. Trying to manually map these endpoints without deep contextual understanding leads directly to high error rates, data corruption, and, again, delays.
- Maintenance Debt: Even when a team manages to successfully stitch an AI agent to a legacy backend, the result is incredibly rigid. These Static Agents are tightly bound to the exact schema present on deployment day. The moment a vendor updates the ERP or an internal team tweaks a database table, the integration shatters. Maintaining these brittle connections consumes roughly 35% of the initial project investment every single year, turning skilled engineers into full-time infrastructure mechanics.

Furthermore, the integration is often left to the AI team handling the project. Their timelines, tools and skills Handicapper them to address to above. Then, involving other teams, makes costs and delays balloon, and dooms very early on the reputation of the project and AI: it is disregard even before a first working implementation exists.

3. The Silver Tsunami and Tribal Knowledge

The technical flaws of manual integration are currently colliding with a major demographic shift. The enterprise and industrial sectors are getting hit by the Silver Tsunami, the mass retirement of the Baby Boomer generation, and highly seasoned technical experts[5].

When these experts leave, they take decades of undocumented Tribal Knowledge with them. This is the unwritten, intuitive wisdom that keeps poorly documented legacy systems running. It's knowing exactly why a specific database column was configured a certain way during a merger ten years ago, or knowing the precise workaround for an unpatched flaw in the billing system.

Then, there are all the other rules, policies and practices of the enterprise that guide what can or can't be done and how it should be done. These are missing rom pure APIs, schemas or documentations. Yet, they impact how, say AI agentic digital workers should operate.

In the approach that we discuss here, this can be captured in the meta data ingested during RTDC (knowledge graph and model of constraints) used by the meta-agent to build API, processes, logic and UI [9,21], as well as part of the process through which experts (SMEs) can refine the systems to guide its output till performances and output is 100% correct.

---

[4] i.e., way too many projections to model, and with the AI team often hard pressed to know what projection to use when. We have seen performant, and knowledgeable AI teams brought to their knees with such challenges able to deliver in more than 6 months less that 2% of their target agentic use cases to connect to an ERP system. With RTDC and application-aware AI this could have been delivered in weeks, as we have done with similar ERP and other enterprise applications, with less than a handful of "developer managers", mostly managing the application-aware agentic AI platform lifecycle during the RTDC process.

[5] Furthermore, with the current wave of AI-related often affect these (non-AI) senior experts who would have been able to help [10,*14-18,*20].

4. Enterprise Integration Challenges

While the discussion has been so far focused on Enterprise integration for AI, for ages, similar problems have plagued enterprises when it comes to integrate their heterogeneous systems, and to maintain these integrations across additions of new systems, application upgrades, new acquisitions, and new requirements. Even building a data lake, and other data infrastructures *([22,23,26] and references therein)*, across these applications, present similar challenges to build the connectors, and keep track of the entities in the data lake, let alone understand their relationships when coming from different applications possibly unrelated, or produced by different vendors [9-*14*,21].

Traditional Enterprise Application Integration (EAI) and Application Integration Architecture (AIA) teams are caught in an impossible situation. The very experts they need to explain these opaque schemas are walking out the door, retiring, or especially because confused executive think that they are no more needed now that we have AI [*14-18*]. Without that tribal knowledge, manual data mapping is just blind guesswork. To deploy AI successfully today, an enterprise needs an architecture that can autonomously extract and apply this tacit logic directly into the software.

5. RTDC: Moving from Design-Time to Runtime

We have addressed the limitations of traditional integration by shifting the focus from human-driven Design-Time assistance to autonomous Runtime discovery design and execution. Traditional integration relies on humans manually defining routing logic and mapping data fields. Real-Time Discovery and (Self) Coding (RTDC) engine functions entirely differently. It acts as a self-evolving software organism embedded inside the stack [9*,10,14*,21].

The RTDC engine operates across four integrated layers [9*,10,14*,21].

5.1 The Discovery Layer (Total Introspection)

Before an AI can act, it must understand the environment. The Discovery Layer continuously scans the network to identify all active APIs, both documented ones and hidden Shadow APIs. It connects to SQL databases, NoSQL stores, and legacy mainframes to infer the underlying data models autonomously. When direct backend access is restricted, it could use UI / RPA-based introspection to analyze UI logs and screen buffers, reverse-engineering the schema from the outside in, possibly with sniffing of the network traffic to DB [9*,10,14*,21][6]. It also ingests

[6] In [21], we discuss the limitations of just using these UI-only scanning or UI + network sniffing techniques instead of discovering APIs and schema from AI scanning of API, schema analysis and meta-data: it limits where agentic

unstructured policy documents, semantically linking them to technical assets (e.g., automatically binding a "DOB" database column to a "GDPR Data Retention" PDF) to build a live Enterprise Knowledge Graph.

*Figure 2: It The four layers of the Real-Time Discovery and (Self) Coding (RTDC) engine, detailing the flow from autonomous discovery to generative execution This figure visualizes the four integrated layers of the RTDC engine: the Discovery Layer, which continuously scans the network to understand the environment; the Constraint Layer, which acts as the "Business Laws of Physics" to enforce rules and prevent AI hallucinations; the Agentic Layer, which consists of a conversational meta-agents that orchestrates complex goals into a sequence of sub-tasks; and the Generation Layer, which dynamically synthesizes integration code and user interfaces [9,10,*21*].*

5.2 The Constraint Layer (Guardian of Integrity)

To prevent AI hallucinations [16,17] from causing damage in regulated environments, the Constraint Layer enforces deterministic rules. This layer acts as the Laws of Physics for the business of the enterprise. So, for example, before the AI can execute a generated SQL query or API call, the action is simulated against a strict rule set. If a proposed action violates a rule, like trying to post a transaction to a closed fiscal period, a good example of tribal knowledge, it is instantly blocked. This layer also enforces Inherited Identity, meaning the AI strictly adopts the access permissions of the specific user making the request, neutralizing prompt injection attacks.

---

solutions can replace processes: inner process become harder to implement with AI agents [*10*,21]. Also, [a19] discusses examples of such limited solutions that can be found in the market today.

5.3 The Agentic Layer (Orchestration)

High-level user requests are managed by a Meta-Agent architecture. The orchestrator breaks down complex goals into a sequence of executable sub-tasks, handing them off to specialized worker agents (e.g., a SQL worker or an API connector worker). A Trajectory Prediction Engine analyzes the workflow before execution to catch logical conflicts or dead ends, ensuring the process terminates safely without wasting compute resources.

It also enable switching the underlying LLM models without major impact, other than taking advantage of the possible better performances of one over the other, thereby ensuring that the platform is futureproof [9,21]. *Note added on April 11, 2026: RTDC and meta-agent is a better approach than MCP, as discussed in [22,23,26].*

5.4 The Generation Layer (Just-in-Time Realization)

This is the execution engine. When a worker agent encounters an unfamiliar API or legacy protocol, it dynamically synthesizes the exact integration code (Python, SQL, JavaScript) needed to interface with that system. This code runs inside highly secure, ephemeral WebAssembly sandboxes equipped with Runtime Application Self-Protection (RASP) to block unauthorized network movements.

Additionally, this layer utilizes Generative UI (GenUI) to build custom dashboards, charts, and forms on the fly based on the user's specific request, rather than relying on static, pre-built interfaces. These are true persistent mini-apps, or agentic AI agents, which can be interacted with, and persisted to be available next time or to other personas or users in the future. Connectors towards other applications or systems implementing integrations patterns are just a particular case of such a generated output.

At every step, the meta-agents, and agents explain their decisions. This can be used by admins or experts to coach and refine it if anything is to be corrected. This is how close to 100% accuracy is achievable [9*,10,*21].

6. Implementation of Zero-Effort Deployment and Integration

In the complex landscape of modern enterprise data, the primary obstacle to digital agility is the manual labor required to integrate and map disparate data sources.

Zenera [3], provides RTDC capabilities as part of Zenera platform [3], which is a full application-aware agentic AI platform, which supports conversational management of the whole lifecycle of the platform, including configurations of its agents [a3], and coding of applications, generation of logic/process/integrations and dynamic UI [*11,13*]. Details and definition of such a platform are defined in [10,21]. In this paper we focus on the RTDC function, including the ability to (self) code the APIs/connectors to the target resources.

These provide a Zero-Effort deployment framework. By utilizing universal connectors for both APIs and databases. AI automatically understands data structures, and collects required data autonomously without building manual data mapping.

This transforms traditional, i.e., not AI enabled, or poorly AI enabled, enterprise applications into fully functional, interactive agents in minutes.

6.1 Key Features

6.1.1. Universal Data Source Ingestion

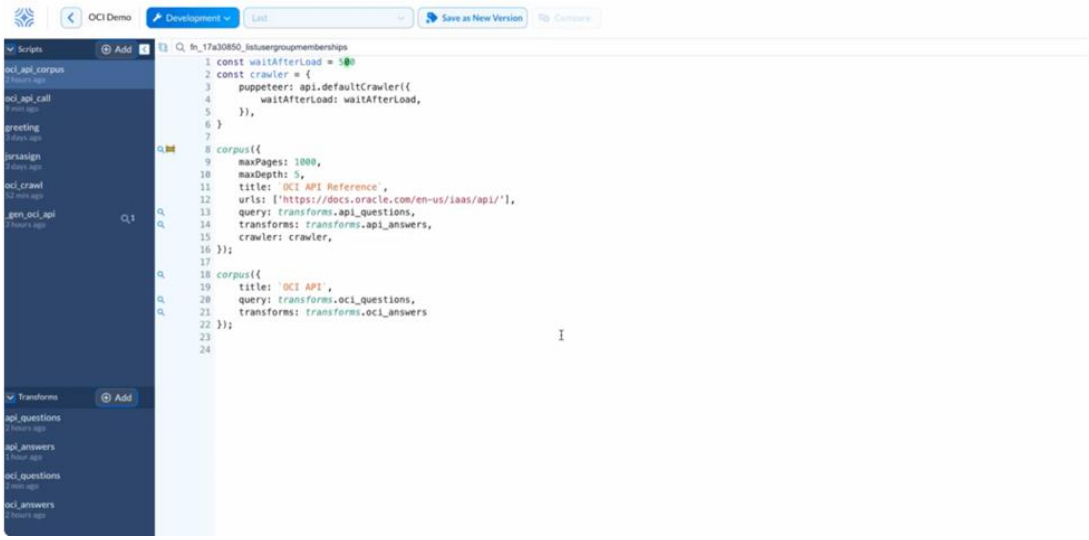

Figure 3: Screenshot of the ingestion of data (Zenera studio) [3].

It provides a unified connectivity layer for any enterprise application. Whether the data is hosted behind an API or within a database, the system utilizes universal connectors to establish a direct link. This enables the AI to understand the underlying data structure and collect required information autonomously, eliminating the need for traditional, manual integration projects.

6.1.2. Autonomous Schema Discovery & Mapping

RTDC removes the burden of manual endpoint or table mapping. The system's AI autonomously crawls data specifications, and database schemas, and related metadata to identify elements and their relationships. It dynamically generates the necessary logic to retrieve data on-the-fly, effectively translating natural language requests into precise data queries without developer intervention.

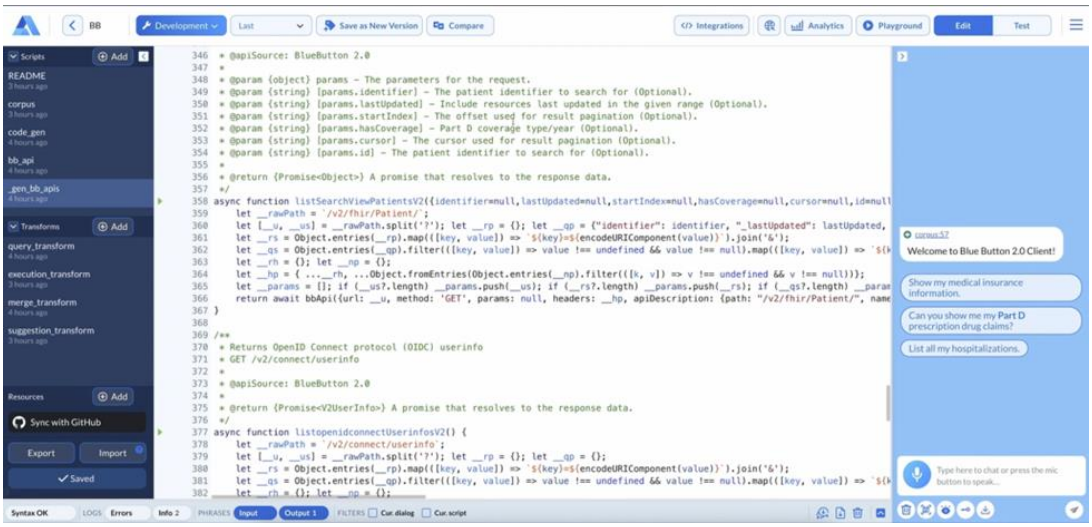

Figure 4: Screenshot oof the data crawling [3].

6.1.3. Auto-Generated Ontologies & Visual Graphs

Upon connecting to a data source, RTDC automatically builds a visual ontology and data graph. It identifies data types and relationships, creating a "model of constraints" that serves as the foundation for safe, explainable, and contextually aware execution across any connected enterprise application.

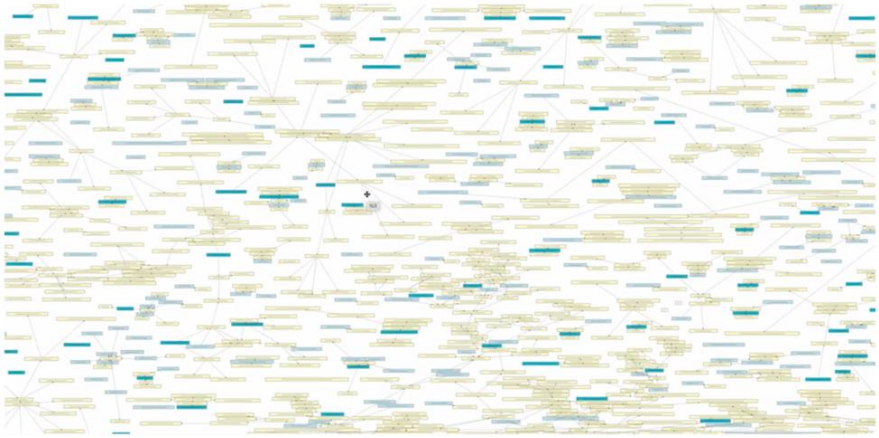

Figure 5: Example of visual ontology data graph built when connected to a data source.

6.1.4. Conversational Meta Agent Orchestration

An agentic interface lets users interact with complex systems using plain English. The Meta-Agent autonomously determines the sequence of calls required to fulfill a request, assembling the execution path in real-time without the need for pre-defined application logic or hard-coded scripts [*11,13*].

6.1.5. Dynamic UI Component Generation

To further reduce deployment effort, the application-aware AI platform automatically generates the user interface. Based on the retrieved data and user context, the system dynamically renders the most effective UI elements, e.g., sortable tables, paginated lists, or charts, and it optimizes the presentation for immediate decision-making.

Figure 6: Automatic and real time UI generation as logic is coded and added per a user intent: the application-aware AI platform generates what does not yet exist to interact with the user. This can then persisted for the future, and/or certain personas [3].

The best way to understand this concept of building UI that do not yet exist, is that when the process built on the application-aware platform needs to interact with the user, it generates the interface required. That interface can be persisted for future use.

6.1.6. Explainable AI via Reasoning Graphs

To maintain enterprise-grade trust, an application-aware AI platform provides full traceability through Reasoning Graphs [9,21]. These graphs document every step the AI takes. That is from data source selection to final transformation. This way the system ensures that all autonomous actions are transparent, auditable, and verifiable by human administrators.

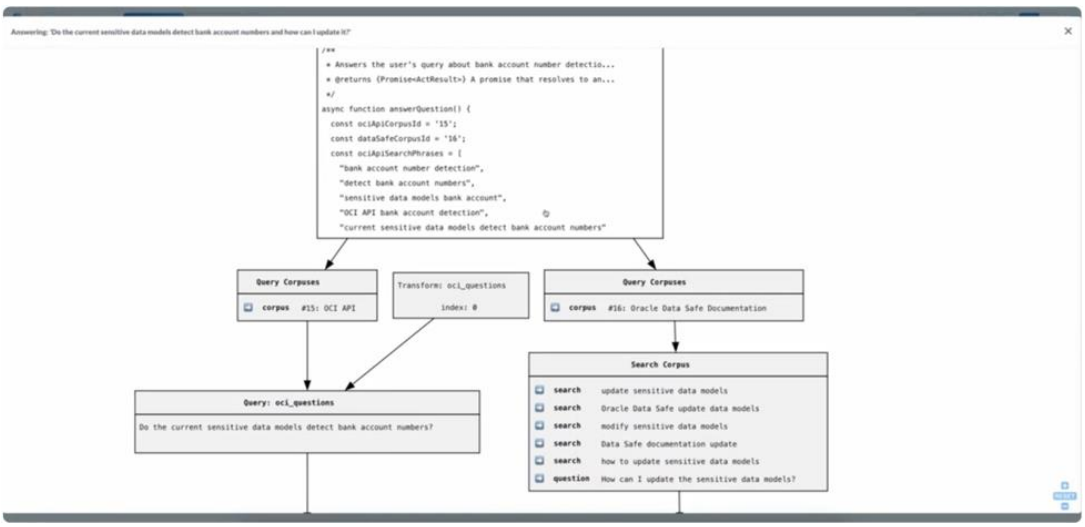

Figure 7: Example of reasoning graph as in [3].

6.1.7. Human-in-the-Loop Refinement: AI Coaching

An application-aware AI platform includes a continuous feedback loop where developers can review reasoning paths and fine-tune AI behavior. This capability ensures the system improves over time, allowing a single data steward to manage and validate complex autonomous pipelines with minimal effort.

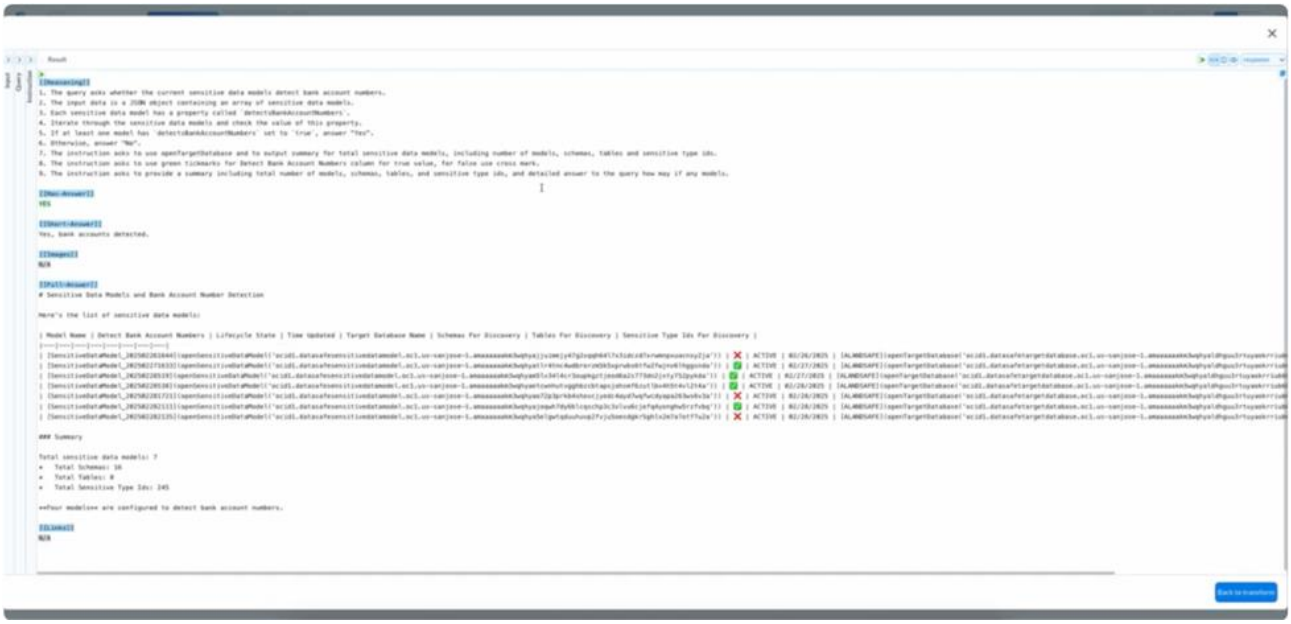

Figure 8: Interface where the reasoning graphs are details and can be corrected by humans [3].

6.1.8 Constrained vibe coding

The (self) coding of the APIs / connectors when required, and of the application logic, agents or dynamic UI, can be seen as vibe coding constrained by the model of constraints, then a full SDLC (Software development lifecycle) managed by the platform [*10,14*]. Doing this can ensure coachable close to 100% accurate connectors and applications/agents [*14*], therefore overcoming the challenge inherent to AI coding and vibe coding [*14-17*].

6.2. Core Benefits

As we can see, an application-aware AI platform provides key benefits and differentiators:

- Elimination of Manual Mapping: AI-driven understanding of data structures removes the need for brittle, manually maintained data maps and wrappers.
- Rapid Time-to-Value: Enterprises can deploy interactive agents on top of existing applications in minutes by simply connecting the data source and allowing the AI to ingest the schema.
- Universal Interoperability: A single platform handles any enterprise application, providing a consistent agentic experience across the entire corporate data landscape.

In [a7,a8,a12,a20,a21,a24], we argue that RTDC and application-aware AI is a better way to approach agentic AI for enterprises than most of the currently widely used approaches, especially MCP, which should be used only when MCP server is already implemented, and one is not concerned with the problems that MCP can introduce.

7. Eliminating the Data Lake Bottleneck

Data lake and related structures like context lake, bring in their own problems *as discussed in [10,22,23,26] and references therein*. Again RTDC and application-aware AI platforms are recommended instead.

Anyway, notwithstanding these considerations, if one want to rely on data lakes, the financial and operational benefits of quasi-zero cost integration become clear when looking at modern data lake architectures, often used by others to develop agentic AI interacting with enterprise applications *([22,23,26] and references therein)*. Companies use data lakes to store massive amounts of raw data. The promise is that you can just dump the data in without defining a schema upfront, and once built, it exposes the data, that, otherwise, you would have to extract from heterogeneous applications.

The reality is much harder. Making that raw data usable requires data engineers to build incredibly complex ETL (Extract, Transform, Load) pipelines to manually cleanse and map the data. Because disparate systems use entirely

different naming conventions, analysts can't get a clear picture. The manual labor required to build and maintain these pipelines destroys the cost advantages of the data lake, and up to 73% of stored data often goes entirely unused.

RTDC eliminates this ingestion bottleneck. When you want to add a new data source to the lake, you simply link it using RTDC generated universal connectors. The Discovery Layer autonomously crawls the raw data. Using semantic vector embeddings and the Enterprise Knowledge Graph, it infers the schema and harmonizes differing naming conventions automatically. It realizes that cust_rev_2025 in System A means the same thing as Annual_Revenue_25 in System B. When an analyst asks a question in plain English, the Agentic Layer synthesizes the complex SQL or Python logic needed to extract and join the data dynamically at runtime. By removing manual ETL pipeline construction, the marginal cost of adding new data sources drops to practically zero.

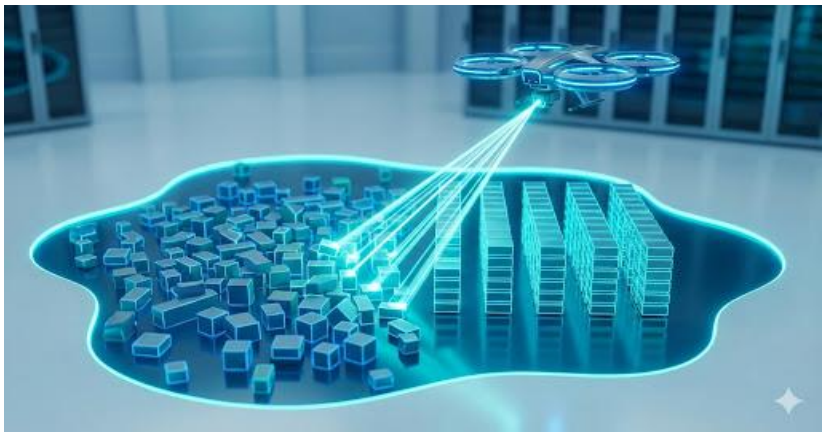

Figure 9: As depicted, RTDC eliminates the need for complex ETL pipelines in data lakes. The Discovery Layer autonomously crawls raw data, infers schemas, and harmonizes different naming conventions. This allows the Agentic Layer to synthesize the necessary logic to extract data (from heterogeneous sources) and join data dynamically at runtime, making the marginal cost of adding new data sources practically zero.

Even without arguing of data lakes are good approaches or not, we can immediately produce the connectors to support ETL and data ingestion and can update them whenever data sources change, or new appear. These are great benefits for traditional integrations, and to those who implement them, that they be the enterprise third party, consultant or professional service and other vendors.

8. Direct Integration and EAI Augmentation

RTDC doesn't force companies to rip out their existing IT infrastructure. The RTDC architecture offers two highly flexible integration patterns, which can be consider as AI-based new RTDC integration patterns. The previous section is a particular example.

8.1 Direct System-to-System RTDC Integration

For legacy endpoints that completely lack modern middleware, an RTDC platform can act as a direct bridge. Using meta-data or crawling discovery, or direct database connections, (and possibly UI scanning and Network sniffing as mentioned earlier), the engine synthesizes an ephemeral, point-to-point connector on the fly, and can then persist it, if suitable and needed. It retrieves the data, modifies it securely within the boundaries of Transactional Object Memory, and writes it directly to the target system. This allows businesses to instantly connect edge applications without paying for say heavy Enterprise Service Bus (ESB) platform, or other message brokers, with or without orchestrators.

8.2 RTDC Augmenting Existing EAI/AIA/Other Integration Patterns

For enterprises that have already invested heavily in centralized EAI message brokers [5], or Application Integration Architecture (AIA) frameworks governed by canonical data models [4,25], or other integration patterns and middleware [5], Zenera acts as an accelerator. The Discovery Layer ingests the existing WSDLs, OpenAPI specifications, and Canonical Data Models of the central hub. It learns the existing routing rules. For example, when a business unit needs to connect a new SaaS tool to the legacy ERP, the RTDC engine instantly synthesizes a connector that perfectly formats the data to match the legacy Canonical Data Model. It automates the last mile of integration, respecting all established enterprise governance protocols while cutting deployment timelines down to minutes.

Through the combination of real-time discovery, strict constraint modeling, and transactional object memory, RTDC approaches permanently resolves the deployment paradox. By shifting the heavy lifting of integration from human design-time labor to autonomous runtime execution, businesses bypass the stitching trap entirely. Enterprise intelligence is securely and permanently industrialized at zero effort and quasi-zero cost.

9. Conclusions

Adding intelligence to your enterprise shouldn't mean tearing down your current infrastructure or spending months hand-coding connections. By shifting the heavy lifting from manual engineering to an autonomous RTDC engine, application-aware AI platforms eliminate the traditional bottlenecks of software integration.

An application-ware AI platform does more than just rescue failing AI pilots; it fundamentally changes how your business systems talk to each other. Whether you need to build entirely new AI applications, untangle complex data lakes, or simply connect a modern SaaS product to a legacy mainframe, the RTDC engine generates the necessary connectors on the fly.

Ultimately, this allows your team to stop wasting time on manual maintenance and start deploying secure, highly accurate integrations instantly, requiring zero effort and operating at a quasi-zero cost.

In fact, the use of RTDC, along with application-aware AI, enable fast development and deployment of AI and agentic AI, and it presents a great solution for effortless integration, and a great alternative to practices of forward deployed engineers, who can be significantly replaced with RTDC.

An example of application-aware platform with RTDC, and such practices, can be found with Zenera [3,11,13].

———

REFERENCES

[1]: Jakub Szarmach, (2025), "STATE OF AI IN BUSINESS 2025", AI Governance Library, https://www.aigl.blog/state-of-ai-in-business-2025/, September 22, 2025.

[2]: Aditya Challapally et al., (2025), "The GenAI Divide. State of AI in Business 2025.MIT NANDA, https://mlq.ai/media/quarterly_decks/v0.1_State_of_AI_in_Business_2025_Report.pdf, July 2025.

[3]: Zenera, (2026), "The Enterprise AI Agent Factory. Your enterprise is unique. Your AI agents should be too. The Enterprise AI Problem, Solved", https://zenera.ai.

[4]: "Enterprise Integration Patterns", https://www.enterpriseintegrationpatterns.com/. Retrieved on February 16, 2026.

[5]: Wikipedia, "Application Integration Architecture", https://en.wikipedia.org/wiki/Application_Integration_Architecture. Retrieved on February 16, 2026.

[6]: Gartner, "Gartner Predicts Over 40% of Agentic AI Projects Will Be Canceled by End of 2027", June 25, 2025.

[7]: Stephane H. Maes, (2025), "The Gotchas of AI Coding and Vibe Coding. It's All About Support And Maintenance", https://doi.org/10.5281/zenodo.15343349, https://shmaes.wordpress.com/2025/04/28/the-gotchas-of-ai-coding-and-vibe-coding-its-all-about-support-and-maintenance/, April 28, 2025, (https://osf.io/kjz9t/download/).

[8]: Stephane H. Maes, (2025), "Ensuring the Maintainability and Supportability of "Vibe-Coded" Software Systems: A Framework for Bridging Intuition and Engineering Rigor", https://doi.org/10.5281/zenodo.15354102, https://shmaes.wordpress.com/2025/05/06/ensuring-the-maintainability-and-supportability-of-vibe-coded-software-systems-a-framework-for-bridging-intuition-and-engineering-rigor/, May 6, 2025, (https://osf.io/2nu8r/download/).

[9]: Stephane H. Maes, et al. (2026), US Patent Application, "AI System and Method of Meta-Agent and Application-Aware AI, Including Real-Time Discovery and Coding, for Deterministic Constraint Modeling, and Runtime Evolution of Software Applications", February, 2026.

[10]: Stephane H. Maes, (2026), "Agentic Smart ITIL, And The Disruption Of The Market Of Conventional Enterprise Applications", https://zenodo.org/doi/10.5281/zenodo.18870309, https://shmaes.wordpress.com/2026/03/04/agentic-smart-itil-and-the-disruption-of-the-market-of-conventional-enterprise-applications/, March 1, 2026. (https://osf.io/vz6jt/files/erw23).

[11]: Zenera, "Enterprise Analytics Reimagined, Build live reports and dashboards in minutes – no pipelines, no coding, https://zenera.ai/zeneraanalytics. Retrieved on March 5, 2026.

[12]: Stephane H. Maes, (2026), "Agentic AI, The Obsolescence of The Enterprise Application & Zenera, The Catalyst", Zenera, February 2026.

[13]: Zenera, (2026), “Zenera: From Conversation to Production in Minutes”, YouTube, https://zenera.ai/introvideo. Retrieved on March 1, 2026.

[14]: Stephane H. Maes, (2026), “Vibe-Coding and SDLC Constrained And Managed By An Application-Aware AI-Like Agentic Platform”, https://zenodo.org/doi/10.5281/zenodo.18972674, https://shmaes.wordpress.com/2026/03/11/vibe-coding-and-sdlc-constrained-and-managed-by-an-application-aware-ai-like-agentic-platform/, March 11, 2026. (https://osf.io/vz6jt/files/yv2xf).

[15]: Stephane H. Maes, (2025), “The Circle of Life for LLMs. Was the Reaction to DeepSeek Justified?”, https://zenodo.org/doi/10.5281/zenodo.14838733, https://shmaes.wordpress.com/2025/02/15/the-circle-of-life-for-llms-was-the-reaction-to-deepseek-justified/, February 5, 2025. (osf.io/j2vsz_v1, vixra:2503.0009v1).

[16]: Stephane H. Maes, (2024), “Fixing Reference Hallucinations of LLMs”, https://doi.org/10.5281/zenodo.14543939, https://shmaes.wordpress.com/2024/11/29/fixing-reference-hallucinations-of-llms/, November 29, 2024. (osf.io/u38w4/, viXra:2412.0149v1).

[17]: Stephane H. Maes, (2024), “The Trouble with GenAI: LLMs are still not any close to AGI. They will never be”, https://zenodo.org/doi/10.5281/zenodo.14567206, https://shmaes.wordpress.com/2024/12/26/the-trouble-with-genai-llms-are-still-not-any-close-to-agi-they-will-never-be/, December 25, 2024. (osf.io/qdaxm/, viXra:2501.0015v1).

[18]: Stephane H. Maes, (2026), “Evaluating the Efficacy of Artificial Intelligence in Software Engineering: A Post-February 2026 Analysis”, https://doi.org/10.5281/zenodo.19103818, https://shmaes.wordpress.com/2026/03/16/evaluating-the-efficacy-of-artificial-intelligence-in-software-engineering-a-post-february-2026-analysis/, March 13, 2026. (https://osf.io/vz6jt/files/gj4z2).

[19]: Ramu Sunkara, Stephane H. Maes, (2026), “Awareness Is the Real Differentiator in Enterprise AI”, LinkedIn post, March 25, 2026.

[20]: Stephane H. Maes, (2026), “Transforming Manufacturing Operations with Agentic ERP”, Zenera, March 27, 2026.

[21]: Stephane H. Maes, (2026), “The Era of Application-Aware AI”, https://zenodo.org/doi/10.5281/zenodo.19322700, https://shmaes.wordpress.com/2026/03/28/the-era-of-application-aware-ai/, February 11, 2026. (https://osf.io/yhwzr).

[22]: Stephane H. Maes, et al., (2026), “Avoid MCP for Enterprise Agentic AI”, to be made available at https://shmaes.wordpress.com/, April 3, 2026. (Embargoed – contact directly for now).

[23]: Stephane H. Maes, et al., (2026), “MCP is Limiting Agentic AI in Enterprises”, preprint to be made available at https://shmaes.wordpress.com/, April 6, 2026. (Embargoed – contact directly for now).

[24]: Stephane H. Maes, Ramu Sunkara, (2026), “One should prioritize investment in enterprise-suited AI agentic framework instead of next-gen LLMs”, LinkedIn post, April, 2026.

[25]: Stephane H. Maes, (2015), “Integrating operational and business support systems with a service delivery platform”, US patent 8966498. (The original AIA design).

[26]: Stephane H. Maes, Ramu Sunkara, (2026), “MCP is a problem for Enterprise AI”, LinkedIn post, April, 2026.