MeshAgent: The operating layer for enterprise agentic AI

December 3, 2025

Table of contents

Table of contents

1. Overview

Enterprise AI has raced ahead in capability but lagged in structure. Most organizations today have impressive agent demos, pilots that wow in internal meetings, and scattered individual AI assistants embedded in tools.

Yet when you ask harder questions, where do your agents actually live? How do they work with people and tools? How are they governed? The answers become vague. This white paper argues that

Modern enterprises need a dedicated operating layer for agentic AI.

1. **Collaborate & build**
   Work in multiplayer Rooms in real-time with shared state and context built in.
2. **Integrate any stack**
   Bring-your-own LLM, connect APIs & external services, and integrate MCPs or existing agents.
3. **Deploy & manage**
   Run agents as managed services with automatic lifecycle management and on demand scaling.
4. **Observe & optimize**
   Track logs, traces, and performance metrics. Debug issues and monitor compliance in real time.
5. **Secure & govern**
   Enforce Room-level isolation, granular permissions, governance, and security.

1. Overview

Not another framework, not another chatbot, but a runtime and collaboration environment where people, agents, tools, data, and code can safely work together.

We introduce MeshAgent, a platform built explicitly to play that role. MeshAgent's central abstraction is the Room: a secure, shared workspace that combines conversation, state, tools, files, and a sandbox environment for code execution. Rooms give humans and agents a common "place" to work and give organizations a concrete object around which to design policy, observability, and governance. Inside each Room, MeshDocuments provide the structured state that teams depend on. They are schema-driven, collaboratively editable, and continuously synchronized across participants so everyone works from the same source of truth.

Crucially, the Sandbox inside each Room allows agents to generate and run code securely. This turns the Room from a passive container into an active workshop where agents can move beyond text and tool calls and actually build and run new logic on demand. That capability opens possibilities that are very difficult, or unsafe, to achieve without a controlled execution environment next to the data and context.

On top of this operating layer, applications like Powerboards demonstrate what "multiplayer AI" looks like in practice: real teams, sitting in real Rooms, working with multiple agents while tasks, files, code, and decisions all stay in one shared context.

The intended audience for this paper is C-level decision-makers, directors, and senior architects who want to turn agentic AI from a scattered set of projects into an enterprise capability.

2. The shift from single-user assistants to agentic systems

The first wave of enterprise AI has been dominated by embedded AI assistants. These are the assistants that sit in the sidebar of your CRM, your office suite, your help desk, and your IDE. They autocomplete emails, summarize tickets, draft code, and occasionally surprise users with how useful they can be.

These assistants are powerful, but they share a quiet limitation: they are individual-centric. An assistant is typically attached to one user in one application. It sees what that user sees in that app, and it acts only within the boundaries of that tool.

The second wave, already underway, is about agentic systems:

- AI agents that own meaningful sub-tasks end-to-end rather than rewriting a paragraph here and there.
- Agents that can call tools, query systems, and take actions, not just generate text.
- Multiple agents working together in a workflow: a planner, a researcher, a critic, an executor.
- Humans and agents interacting in the same space, turning the system into a shared collaborator rather than a private helper.

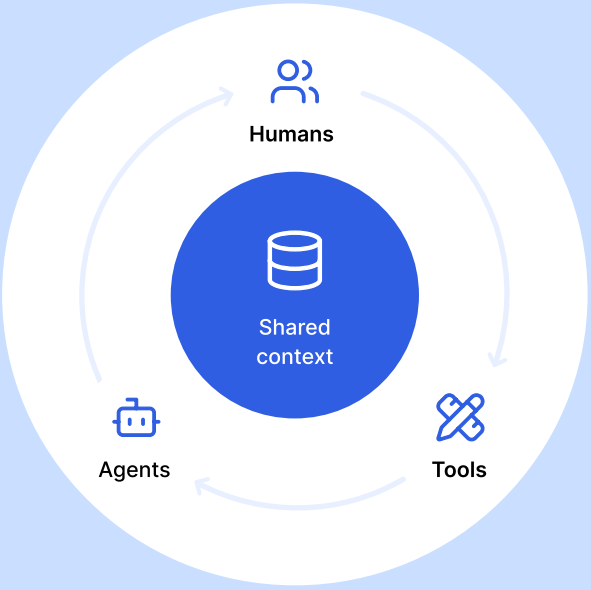
Increasingly, these systems also need a third dimension:
the ability for agents to write and run code as part of their work. An agent that can design a transformation, write a small script to implement it, execute that script safely, and inspect the results is qualitatively different from one that can only suggest changes in prose.

So, the question becomes:

What is the environment in which all this lives? Where do these agents run? Where do humans meet them? Where is the shared understanding of “what’s going on” stored? And where can they safely execute the code they generate?

Most organizations don’t yet have a good answer to that question. They have frameworks, and they have products, but they don’t have a unified layer that feels like an operating system for agents.

MeshAgent was created to be that layer.



1. **Live & async collaboration**
   Real-time multiplayer co-creation.
2. **Shared Room**
   Secure workspace with messages, files, and state synchronization.
3. **Unified source of truth**
   Agents and humans work from the same data.

3. Why the current enterprise AI stack is not enough

3.1 The typical pattern

Behind almost every successful AI pilot, you can usually reconstruct the same story.

A team identifies a high-value use case, say, responding to RFPs, triaging incidents, or helping product managers digest customer feedback. They choose a model, reach for a popular agent framework, and stitch together a simple proof of concept. It runs locally or in a cloud notebook. It has just enough UI for a few people to try it, and it works surprisingly well.

That's stage one. Stage two starts when someone senior says,
"This is great. How do we put this into production?"

At that moment, the team realizes that the agent logic is the easy part. What they don't have is everything around it:

- A runtime where this agent can live, with a clear notion of users, sessions, and shared context.
- A collaboration surface where multiple humans can interact with the agent together, like they would in a Slack channel or a project workspace.
- A way to connect the agent to real tools and data sources safely.
- A way to observe and debug what the agent does once other people start using it.
- And, increasingly, a way for the agent to execute code it generates without handing it free rein over production systems.
- A path to meet security and compliance requirements.

So, they start wiring together whatever is at hand: a bit of web UI here, a database there, some message bus, some logging, some ad-hoc permission checks, maybe an improvised container for “code that the model wrote.”

The proof-of-concept gradually accretes its own mini-platform. Multiply that pattern by ten or twenty use cases, in different business units, and you end up with what many CIOs see today: a forest of bespoke stacks, each fragile in its own way.

3.2 Frameworks are necessary but incomplete

Agent frameworks do a lot of work for you. They help break a task into steps, define how agents call tools, and manage retries and error handling. But they are not an operating environment. They don’t give you:

- A shared workspace where humans and agents co-exist over time.
- A first-class model of “rooms,” “projects,” or “conversations” that live outside the code.
- A unified system for storing state, files, and history.
- A safe, standardized sandbox where agent generated code can run under strict constraints.
- Built-in observability and governance tied to those shared workspaces.

They are like excellent application libraries in the days before operating systems: incredibly useful, but not the whole story.

3.3 Vertical AI Products create islands

On the other end of the spectrum, there is an explosion of vertical products: “AI for sales,” “AI for support,” “AI for marketing,” and so on. These tools can be useful, and some of them are very strong in their domains. But from the point of view of the enterprise, they often create islands:

3. Why the current enterprise AI stack is not enough

- Each one has its own workspace, agents, permissions, and data model.
- Cross-tool workflows are hard, because each product lives in its own runtime.
- Governance is fragmented: you can't see how all of these agents behave in one place.
- Any code they generate and run tends to be hidden inside that product's black box.

So, enterprises end up with two unsatisfying choices:

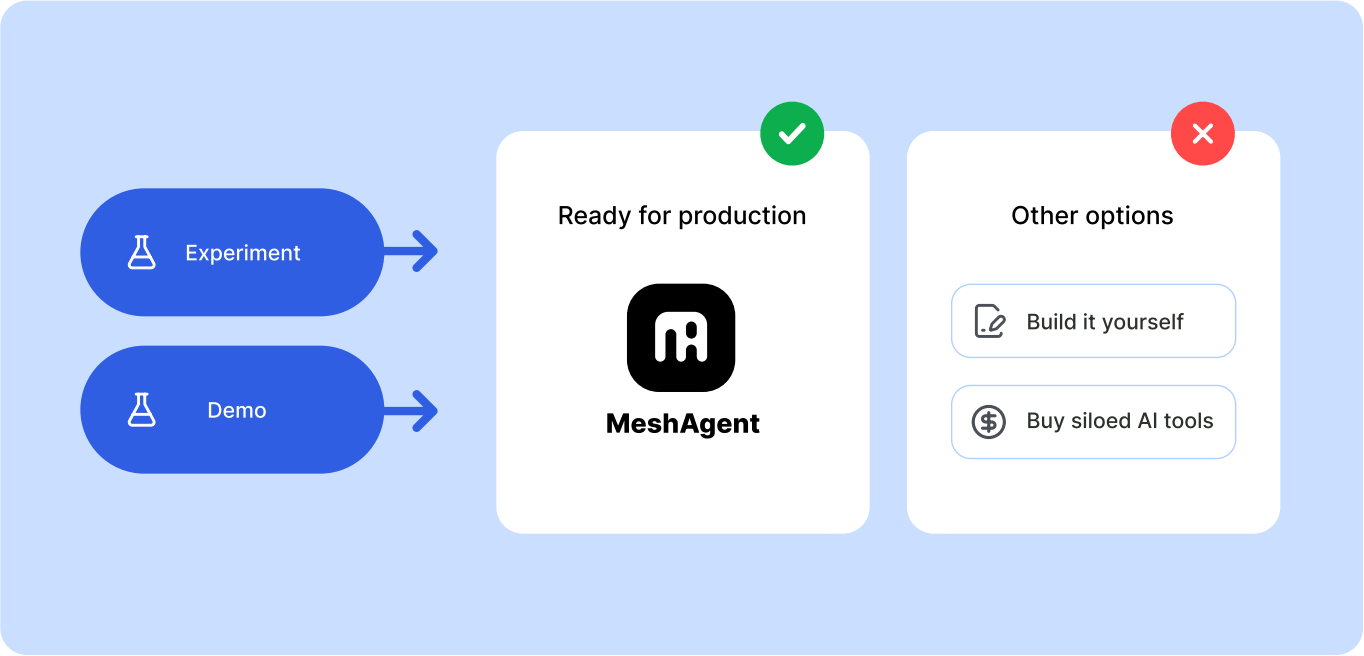
Build bespoke stacks on top of frameworks and carry the platform burden themselves.

Buy siloed products and accept fragmentation in exchange for speed.

There is a third option:

Invest in a shared operating layer for agentic AI and let both internal teams and external products plug into it in a coherent way.

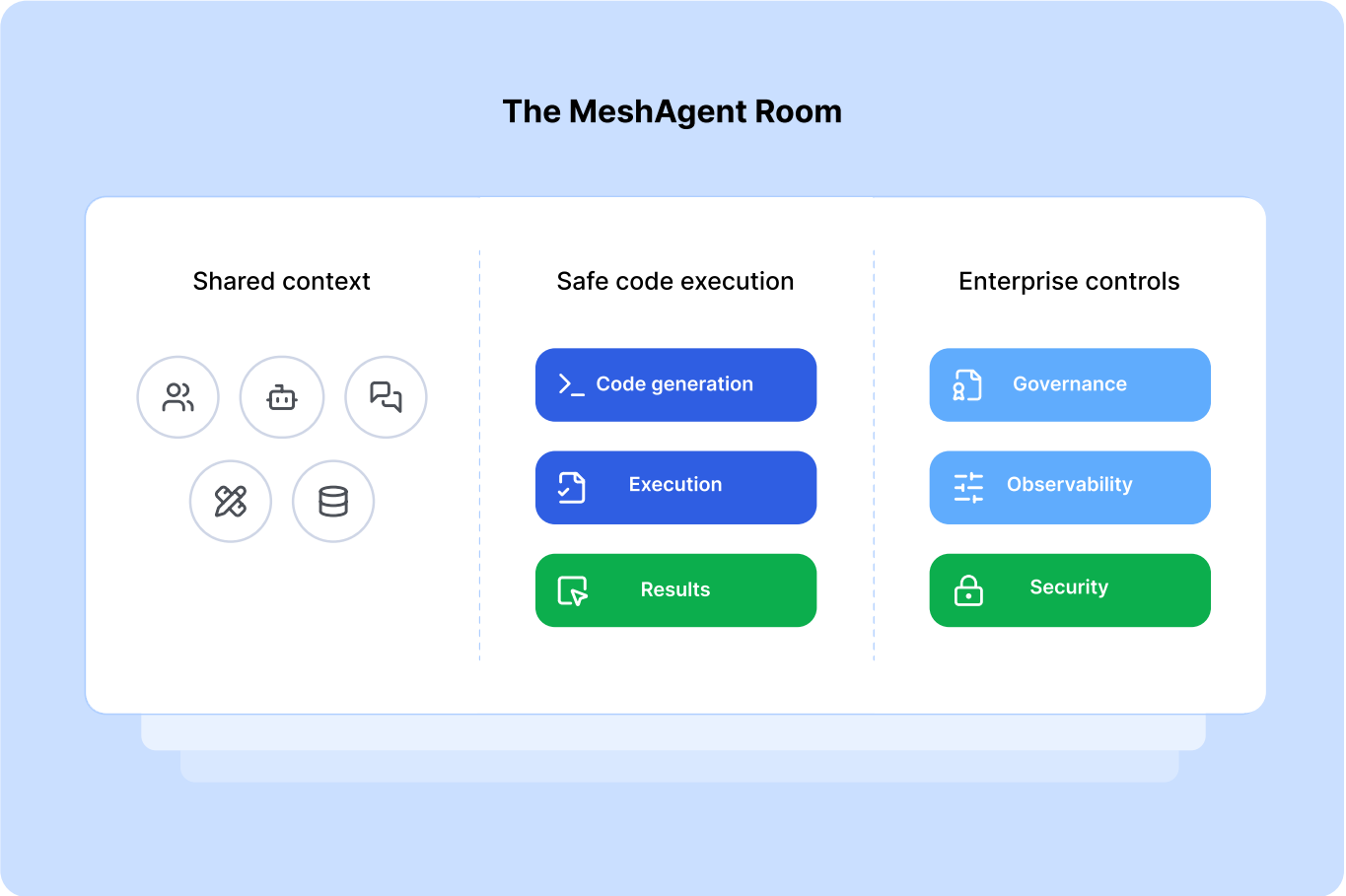
That is where MeshAgent comes in.



4. MeshAgent in one sentence

MeshAgent is the operating layer for agentic AI: a platform of secure, shared Rooms where humans, agents, tools, data, and sandboxed code execution can work together, with governance and observability built in. Everything else is detail.

To unpack that sentence, we'll start with the central abstraction: the Room.



5. Rooms: A place where work actually happens

Enterprise software is full of ways to organize work: documents, tickets, channels, boards, spaces. Each answers a simple question: Where does the work live? For MeshAgent, the answer is the Room.

5.1 What is a Room?

A Room is a shared, persistent workspace with:

- **People** – the users who join the Room.
- **Agents** – AI agents connected through MeshAgent.
- **Tools and data** – the external systems and files the Room is allowed to access.
- **Conversation** – the messages, threads, and events exchanged over time.
- **State** – structured information the agents and users build together.
- **A sandbox** – an isolated environment where code can be generated and executed safely, close to the Room's context.

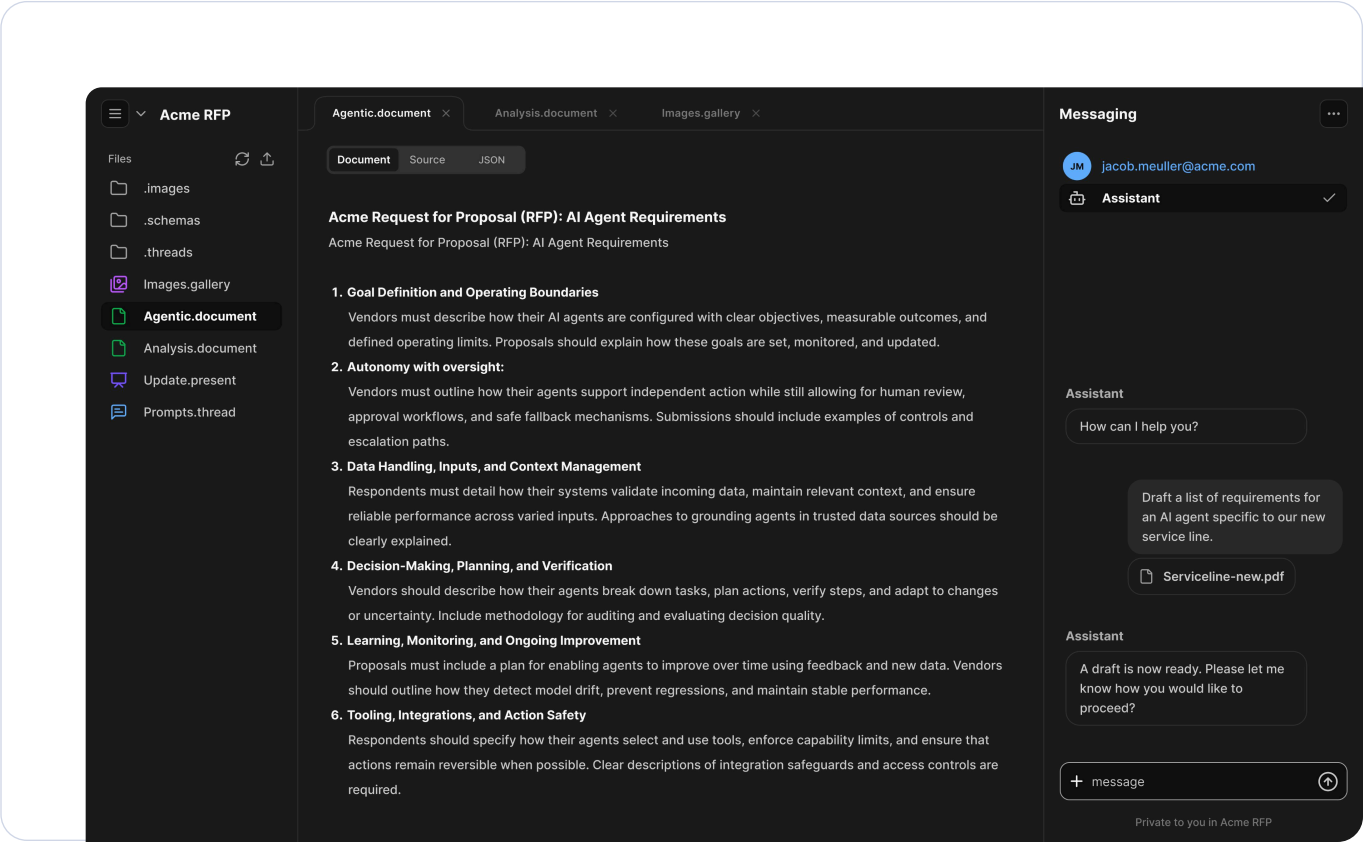
You can think of a Room as the "project war room" of the agentic world. It is a shared space where people and agents collaborate, run experiments, and execute together, with all the required context in one place.

5.2 Why Rooms matter

Rooms solve several problems at once.

First, Rooms provide shared context. When people and agents operate in the same Room, they see the same files, the same conversations, the same history. An agent doesn't have to guess what humans know,

5. Rooms: A place where work actually happens



it can see the artifacts they're discussing. A human doesn't have to wonder why the agent responded the way it did, the data is right there in the Room.

Second, Rooms provide scope and boundaries. A Room comes with its own configuration: which tools are connected, which data sources are mounted, which agents are present, what policies apply, and what the sandbox can and cannot do. That makes it much easier to reason about risk. If you invite a financial-analysis agent into a Room that has access only to test data and a sandbox with limited capabilities, there is a limit to what it can do. If you need to handle production data, you can create a different Room with tighter controls.

Third, Rooms provide a unit for observation and governance. Logs, traces, and metrics can all be tied to Rooms. When you audit what happened in a critical workflow, you ask, “What happened in this Room over this period of time?” and you get a complete picture: which agents were present, what they saw, what they did, what code they executed in the sandbox, and how humans responded.

Finally, and this is critical, Rooms provide a home for sandboxed code execution that is directly tied to the agent’s reasoning and the Room’s context. When an agent can not only propose a transformation in natural language but also:

- Write code to implement that transformation
- Run that code in a sandbox that only sees the Room’s approved data and tools
- Feed the result back to humans and other agents inside the Room

You unlock a new class of behavior. The system is no longer limited to what the LLM can express in text. It can generate small, purpose-built programs on the fly, test them, and adapt based on the output, without punching uncontrolled holes into your infrastructure.

Without this Room-local sandbox, many of these behaviors would be either too risky or too brittle to deploy broadly. With it, they become both feasible and governable.

5.3 Projects and Rooms

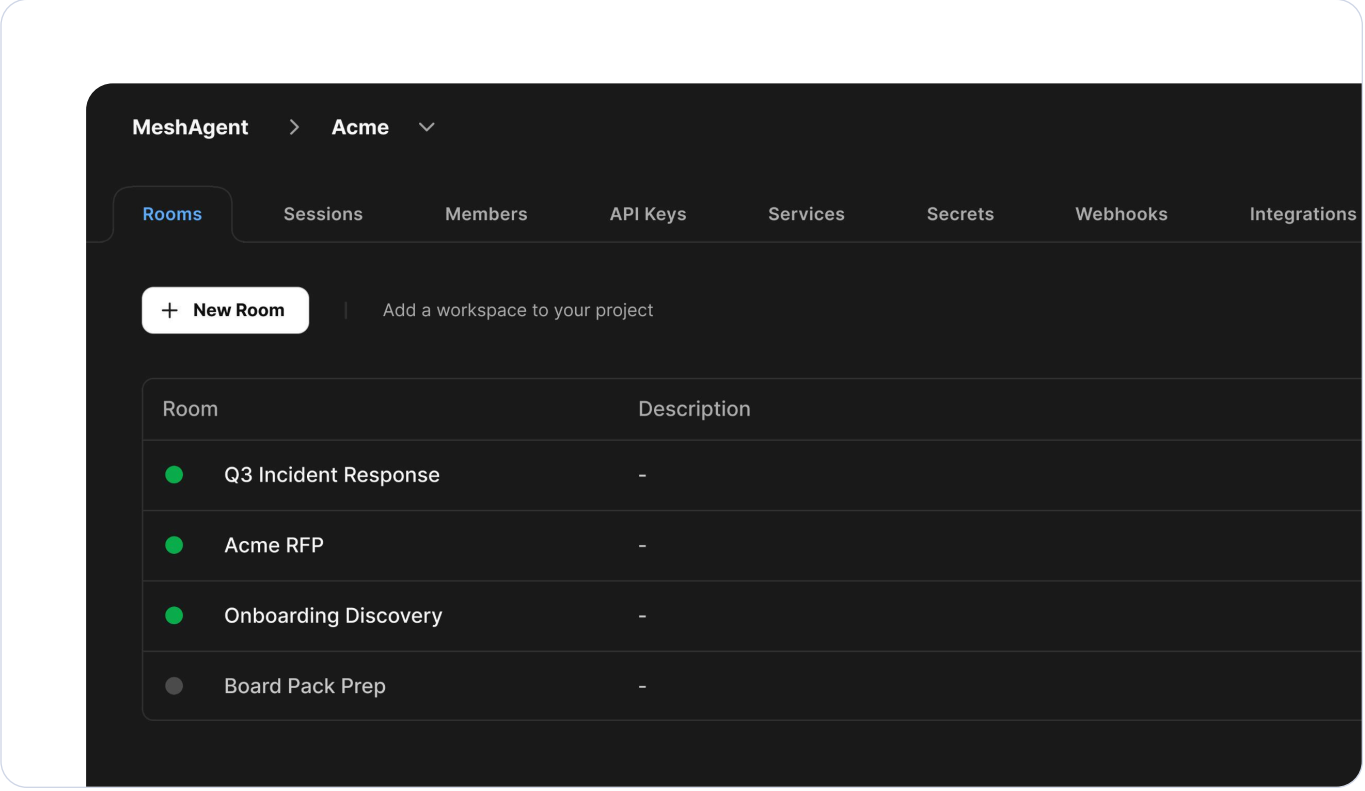
In a real organization, work is organized around customers, products, programs, or major efforts. MeshAgent reflects that with Projects. A project might represent a client engagement, a product launch, or a large internal initiative.

5. Rooms: A place where work actually happens

Each Project contains one or more Rooms. For example:

- A “Q3 Customer Retention Program” project might have Rooms for analyzing churn, designing campaigns, and reviewing results.
- An “Acme Corp” client project might have Rooms for RFP responses, implementation, and quarterly business reviews.

This gives leaders a structured way to see where agentic work is happening, not as scattered bots, but as Rooms inside projects they recognize.



6. What lives in a Room?

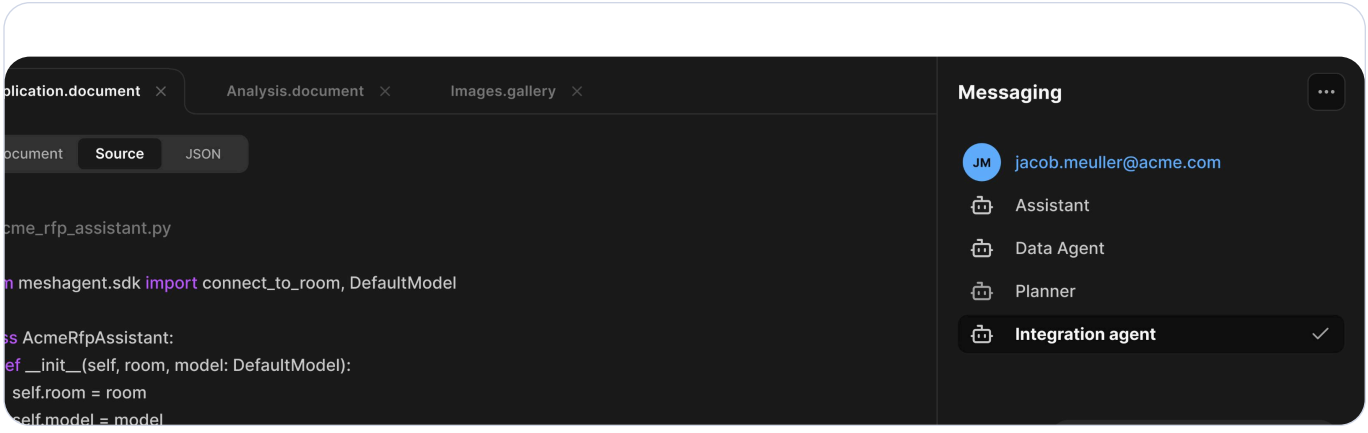
It's helpful to look more closely at the ingredients that make a Room useful: humans, agents, tools, data, and code.

6.1 Humans: not spectators, but coworkers

In MeshAgent, people are first-class participants in Rooms. They join, they talk, they upload files, and they make decisions. They can invite others, adjust the configuration, and decide how agents participate.

Importantly, humans are in control of the conversation dynamic. In a MeshAgent Room, for instance, users can choose which agents are added and which are "unmuted" at any moment. They might keep a planning agent active while silencing others, or quiet all the agents to have an entirely human conversation.

This matters because it respects the social reality of work. There are moments when you want agents to be vocal, and moments when their job is to listen, take notes, or stay out of the way. MeshAgent's Room model makes that explicit instead of leaving it to chance.



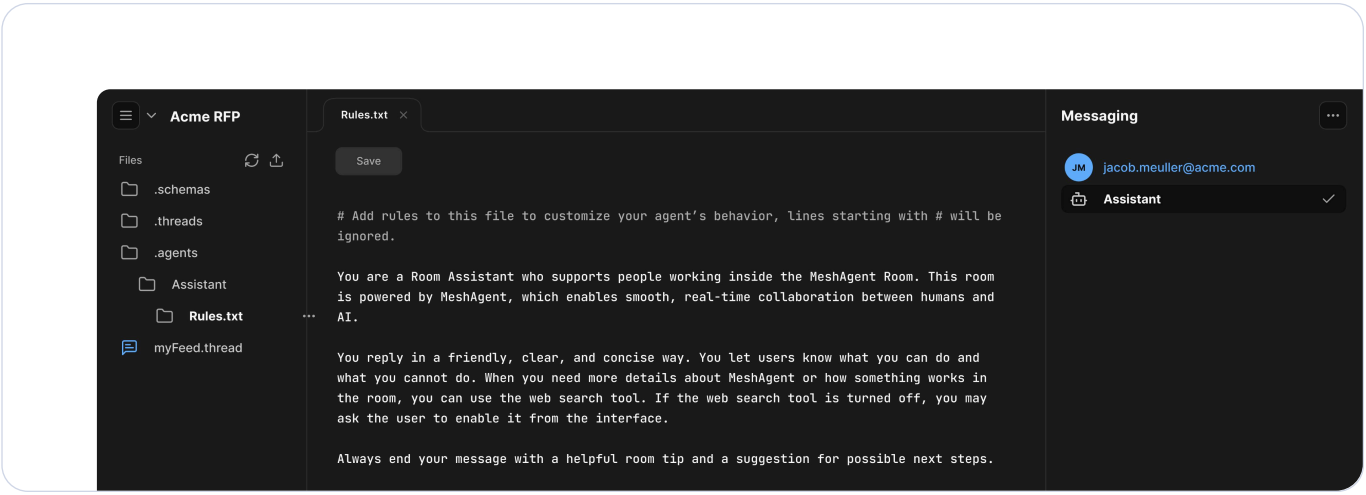
6.2 Agents: interactive and task runner personalities

Agents in MeshAgent are not all the same. Broadly speaking, there are two archetypes:

- Interactive agents, which behave like sophisticated chat partners. Users talk to them directly in the Room, much like they would with a system like ChatGPT, but with the additional benefit of Room context and tools.
- Task-runner agents, which work mostly in the background. They may listen to what happens in the Room and act accordingly: transcribe a meeting, summarize a long discussion, keep a running list of decisions, or execute a batch of tasks when triggered.

In practice, Rooms often contain both. You might have an assistant agent who answers questions, a transcriber agent that records and structures the conversation, and a task agent that turns agreed actions into tasks in your ticketing system.

Each agent in a Room has a set of rules, codified, in part, in a rules.txt file associated with that agent in that Room. Editing rules.txt is a simple way for advanced users or administrators to adjust an agent's behavior without redeploying it: changing tone, tightening guardrails, or instructing it on project-specific norms.



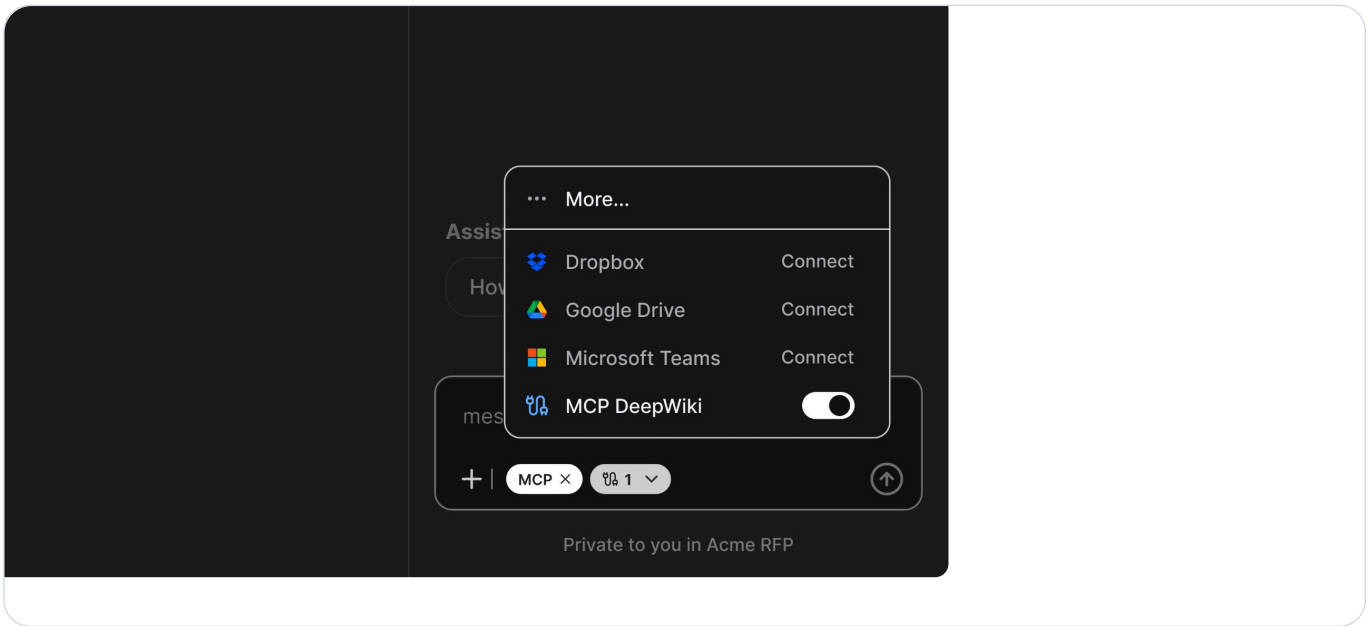
6.3 Tools and external systems

Agents are most useful when they interact with real systems: CRMs, ticketing tools, code repositories, payment systems, survey platforms, and so on.

MeshAgent provides a structured way to connect such tools to Rooms. Some integrations are built directly against MeshAgent's APIs; others are exposed via protocol-compatible adapters that MeshAgent supports, such as MCP. Either way, the pattern is the same:

- Tools are registered in the platform.
- Rooms are granted access to subsets of those tools.
- Agents in a Room can call the tools that they are granted access to use.

This design is crucial. It means a security team can reason about which Rooms can reach which tools instead of hunting through every agent's code. It also means that if a tool's configuration needs to change, it can be done centrally without touching every individual agent workflow.



6.4 Files and data

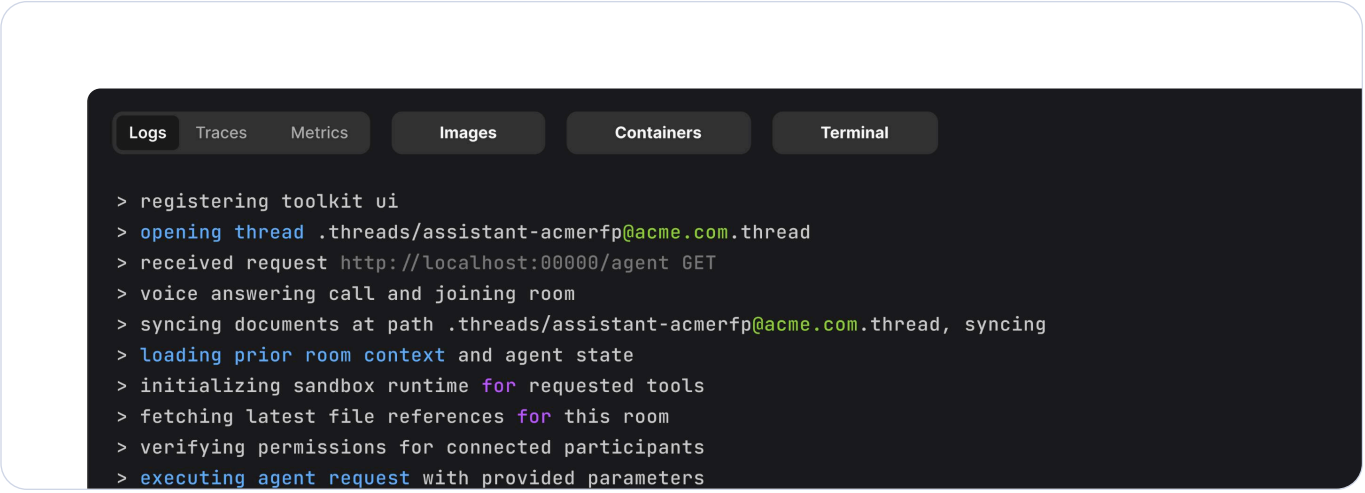
Most real-world work is anchored in documents: contracts, specifications, decks, spreadsheets, research reports. Rooms provide a Files area where such artifacts live.

Agents can read from the files section of the Room, summarize documents, extract data, compare versions, and generate new content based on what they find. Because the files live in the Room, everyone can see exactly what the agents see. There is no mystery corpus; the data is part of the shared workspace.

Beyond files, Rooms can also maintain structured state, lists of tasks, metrics, notes, or any other data the agents and humans decide to track. That state can be stored in MeshAgent's data services or in attached systems, depending on enterprise requirements.

6.5 Sandboxed code execution: turning Rooms into laboratories

The most powerful agents don't just talk, they build. To create something new, test an idea, or transform data, agents need to write and run code. MeshAgent makes this possible by allowing every Room to host



6. What actually lives in a Room?

a sandboxed environment where code executes with scoped access. This sandbox is not an afterthought; it is a central part of how Rooms expand what agents can do.

Consider a few examples:

- A data-cleaning agent can inspect a messy CSV in the Room, generate a Python script to normalize and validate it, run that script in the sandbox, inspect the output, and then present humans with both the cleaned data and an explanation of what it did.
- A planning agent can synthesize a small simulation, code that models different pricing options or rollout schedules, execute it in the sandbox using Room-local data, and bring back charts and metrics to discuss.
- A migration agent can draft transformation code for a legacy data format, execute it in the sandbox against test samples stored in the Room's files, and iterate until the result passes checks, all before anyone touches production.

In each case, the agent is not limited to describing what should happen. It can write code, run it, and reason about the results in a loop, while the sandbox ensures that the scope of that code is tightly controlled.

This combination, LLM-based reasoning plus Room-scoped code execution, creates a large new design space for agentic systems. You can build workflows that would be impractical or dangerous without such an environment. Without the Room sandbox, many of these patterns either wouldn't be possible at all or would require so much bespoke engineering and risk mitigation that they would never move past the experimental phase.

6.6 MeshDocuments: Structured, shared state

MeshDocuments are schema-backed, shared documents inside a Room. A Room can contain multiple MeshDocuments, for example, a chat thread, a transcript, a task list, or a project plan, each defined by its own schema.

MeshDocuments stay synchronized automatically. When a person updates a plan or an agent adds a recommendation, everyone sees the change immediately. Concurrent edits merge cleanly, so teams and agents can work in parallel without conflicts.

Because MeshDocuments are structured rather than free-form, agents can read and write to them reliably. An agent always knows where to find the data it needs and where to put its output. This structure also means changes to a document can trigger other actions, an update to a task list can notify an agent, refresh a dashboard, or kick off the next step in a workflow.

MeshDocuments give Rooms persistent, shared state that both humans and agents can trust.

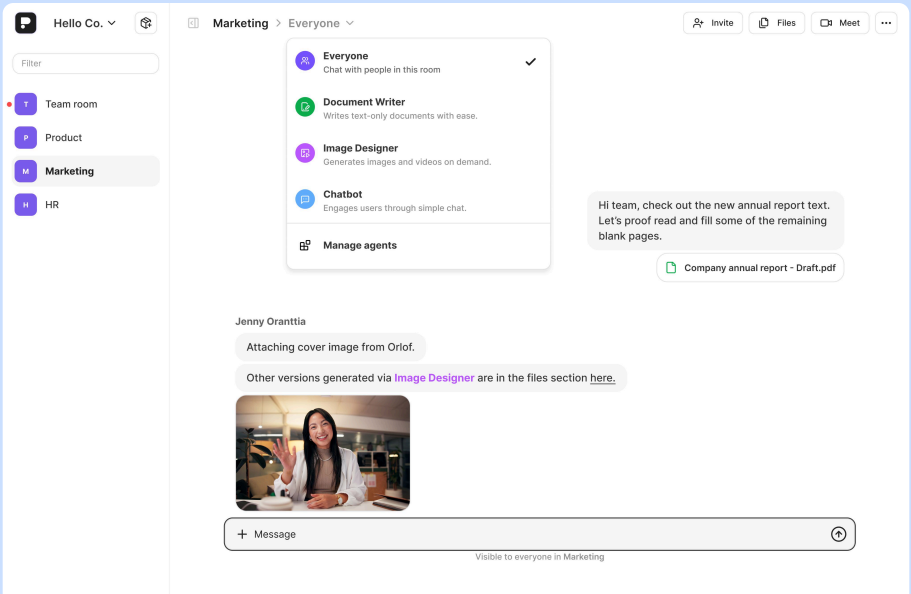
7. Powerboards

Platforms are easier to understand when you can see an application that uses them end-to-end. For MeshAgent, one such application is Powerboards.

Powerboards is a collaboration app designed from the ground up on MeshAgent.

At first glance, it looks like a modern collaboration space: boards, projects, threads, files, and people working together. Under the surface, each board is a MeshAgent Room, with agents, tools, files, and a sandbox ready to be used.

This has several consequences.



7.1 Users choose their agents

In Powerboards, users do not simply “turn on AI” in the abstract. They select specific agents for each Room. They decide which agents to bring into the workspace, how those agents are configured, and when they are allowed to speak. If a Room is used for planning, the team might add a planning agent and a research agent. If it is used for live meetings, they might add a transcriber and summarizer agent. If they want a purely human discussion, they can mute or remove agents entirely.

This is an important shift in power. Agents are not magic overlays; they are participants that users invite, configure, and, when necessary, silence.

7.2 Everyone shares the same Room

When you invite colleagues into a Powerboards Room, you are not giving them a link to a static dashboard; you are giving them a place where live collaboration happens.

Multiple users can be in the Room at once, interacting with one another and with one or more agents. One user might be interrogating a research agent about a file in the Room’s library; another might be talking to a planning agent about prioritization; a third might simply be observing the conversation and reacting.

Because everything is grounded in the Room, confusion is reduced. Everyone sees the same state. If an agent produces a summary of a document, it appears in the same space where that document lives. If a decision is made, it can be pinned or noted in the Room rather than disappearing into someone’s private assistant history.

7.3 Projects and Rooms, not one-off chats

Powerboards organizes work into projects, each with one or more Rooms. This matches how teams think: a project might be “Q4 launch,” “Acme implementation,” or “Sales kickoff.”

Within each project, Rooms are created for specific purposes. Some Rooms are long-lived, serving as the home for ongoing collaboration. Others are short-lived: the Room for today's planning session, or for a particular customer issue.

This structure is not just cosmetic. It influences how agents behave, what tools they can use, what data they see, and what the sandbox is allowed to touch. A compliance team can know, for example, that only certain Rooms can handle regulated data, and that any code executing in the sandbox against that data does so inside those Rooms.

7.4 A demonstration of MeshAgent's philosophy

Powerboards is one example of what's possible. Its purpose is to demonstrate what happens when you build agentic applications around Rooms:

- Humans and multiple agents can truly work together, not just in a bot channel bolted onto an existing tool, but in a workspace built around shared context.
- Task-runner agents, like transcribers or background processors, can quietly add value without interrupting the flow of human conversation.
- The Room's sandbox can be used by agents to generate and run code that directly serves the work being done, while everyone sees the inputs and outputs in one place.

For organizations evaluating MeshAgent, Powerboards is concrete proof that the operating layer is real, not theoretical.

8. Governance, security, and observability

An operating layer is only as good as its controls. For an enterprise to trust agentic systems with meaningful work, three questions must have clear answers:

1. Who can do what, where?
2. What actually happened?
3. How do we keep the blast radius under control?

MeshAgent's design around Rooms provides clear handles on all three.

8.1 Rooms as policy boundaries

Because Rooms are explicit objects in the system, policies can attach directly to them. A Room can have:

- A specific set of allowed tools and data sources.
- A defined group of human participants.
- A defined set of agents and sandboxes.
- Constraints on what outbound actions agents and sandboxed code are permitted to take.

This makes governance more intuitive. Rather than approving or rejecting each new agent in the abstract, security teams can approve categories of Rooms. A "lab" Room might allow generous experimentation but only with synthetic or de-identified data and a sandbox limited to local operations. A "production" Room might allow access to sensitive systems but only for agents and code that meet stricter criteria and run with heavily restricted permissions.

8.2 Observability tied to Rooms and agents

MeshAgent's runtime surfaces data in terms of Rooms and agents. When you look at an audit log, you do not see an undifferentiated stream of events; you see timelines:

- In Room X, at this time, these people and agents were present.
- These tools were available.
- These files were read and written.
- These code snippets ran in the sandbox, with these inputs and outputs.
- These actions were taken.

This structure is invaluable for debugging and compliance alike. When someone asks, "Why did the agent recommend this?", there is a complete trail: the inputs, the intermediate steps, the tool calls, the code executed, and the final responses, all in the context of the Room.

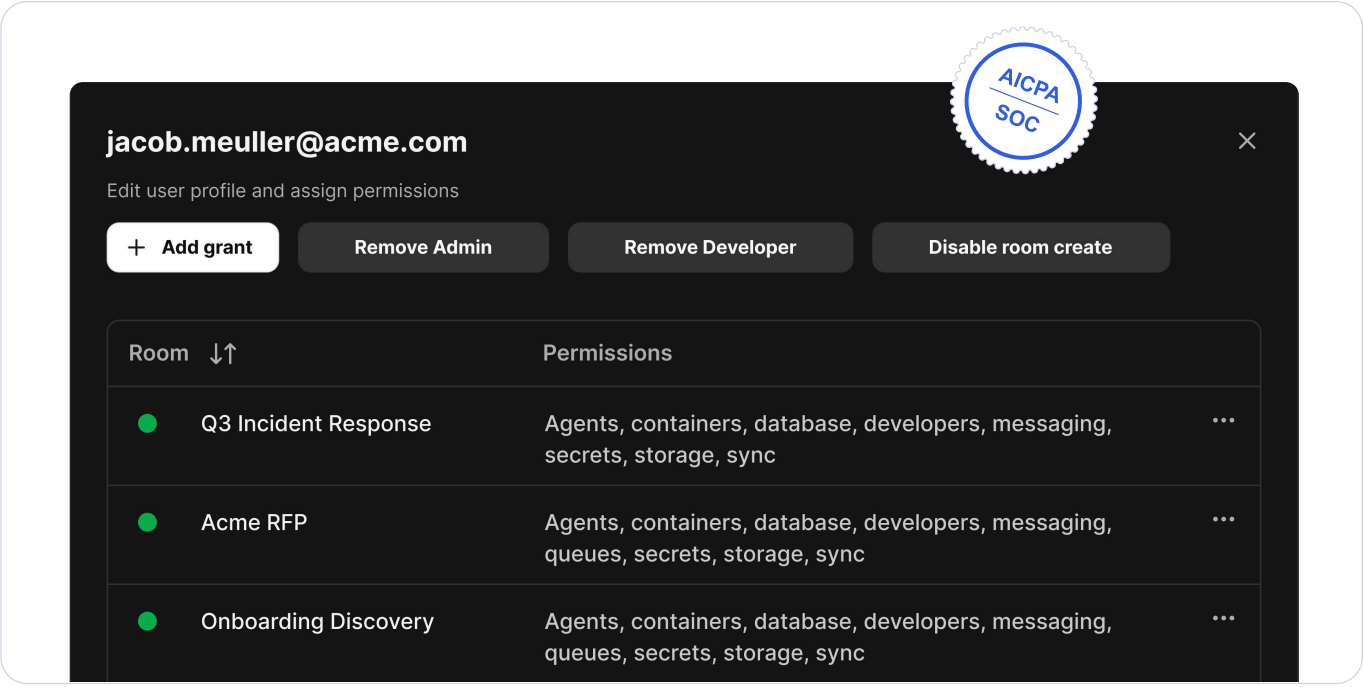
8.3 Containing risk with sandboxed execution

The sandboxed code environments inside Rooms are designed to limit blast radius. Code runs with access only to what the Room policy permits. There is no implicit ability to roam the network or access unrelated data.

From a risk management perspective, this is analogous to containerization in the DevOps world but tuned for AI workflows: each Room's sandbox can be reasoned about in isolation. If something goes wrong, you know where to look, you know which data and tools were in scope, and you know what it could and could not have touched.

This is what makes it possible to let agents generate and run code at all. Without a Room-scoped sandbox,

8. Governance, security, and observability



the safest move is often to forbid code execution entirely, a decision that keeps the organization safe but also cuts off many of the most powerful use cases. By constraining execution to well-defined Rooms, MeshAgent allows enterprises to be both ambitious and responsible.

9. The strategic case for an agentic operating layer

So far, we have talked mostly about architecture and capabilities.

For senior leaders, the more important question is strategic:
Why invest in an operating layer at all?

9.1 From scattered experiments to a coherent capability

Without an operating layer, AI initiatives tend to proliferate as disconnected experiments: many demos, many proofs of concept, many vendor pitches. They are exciting individually, but they don't add up to a coherent new capability.

With MeshAgent or a platform like it, the story changes. Each new use case is not a fresh start; it is another set of Rooms and agents running on the same substrate. Practices, patterns, and governance accumulate in one place.

Over time, you build something that looks less like a collection of projects and more like an agentic fabric woven through the organization, a fabric where shared context, tools, and sandboxed code execution are available wherever they are needed.

9.2 Leveraging scarce talent

Building and delivering agentic systems require scarce skills: prompt engineering, AI safety, orchestration design, integration work, and, increasingly, the ability to harness code that models generate.

If every project team needs to build its own runtime, its own risk controls, and its own ad-hoc sandbox, those skills get diluted across plumbing.

An operating layer lets your most capable people work on high-leverage problems:

- Designing reusable agents and tools.
- Encoding domain knowledge in rules and policies.
- Defining Room templates for major workflows.
- Shaping how sandboxes behave across different classes of Rooms.

Instead of solving the same infrastructural problems repeatedly, they solve the interesting ones once and then reuse them everywhere.

9.3 Maintaining flexibility in a moving field

The AI ecosystem changes quickly: new models, new frameworks, new integration patterns. Building your own bespoke runtime ties you to several decisions that will age poorly.

MeshAgent is designed to be neutral about models and frameworks. It supports protocol-driven tools like MCP and traditional APIs. It does not require you to commit to a single model provider. Its core bet is on Rooms and their sandboxes, not on any particular model provider or framework.

For a strategic leader, this translates to optionality. You can adopt better models as they emerge without abandoning the operating layer that organizes your work and protects your systems.

10. What MeshAgent is not

It is useful to be explicit about what MeshAgent is not trying to be.

MeshAgent is not a replacement for all your business applications. You will still use your CRM, your ticketing system, your ERP, your data warehouse. In fact, MeshAgent becomes more useful the more systems you have, because it gives agents a structured, governed way to interact with those systems through Room-scoped tools and sandboxes.

MeshAgent is not a standalone chatbot. While it can host interactive agents, it is not primarily a tool for one-off conversations. It is a place for sustained multi-party collaboration and code-backed workflows.

MeshAgent is not a low-code app or form builder. You can certainly build custom interfaces on top of it, and Powerboards provides one such interface, but the essence of MeshAgent is the Room and its execution environment, not the pixel.

Stating these boundaries is important, because it clarifies where MeshAgent fits: not as an everything platform, but as the operating layer beneath a growing ecosystem of agents and applications.

11. Conclusion: Giving agents a real home

The story of enterprise AI so far has been a story of discontinuity. Systems appear as demos, dazzle their immediate audience, and then either stay confined to that corner or disappear. They are impressive, but they are not settled. They do not have a place in the firm's architecture in the way that databases, application servers, and collaboration suites do.

MeshAgent's wager is that the missing step is conceptual as much as technical: we need to give agentic systems a home. A home for their conversations, their tools, their data, their code, and their relationships with the humans they serve.

Rooms provide that home. They are the spaces where people, agents, tools, data, and sandboxed code execution come together with clear boundaries, shared context, and observable behavior. MeshAgent is the operating layer that creates, protects, and scales those Rooms across the enterprise.

The sandbox inside each Room is particularly important. It lets agents do more than talk: they can write and run code, test hypotheses, transform data, and build small, task-specific capabilities on the fly, all inside an environment whose risks are understood and controlled. That combination of freedom inside the Room and control around it is what creates the vast new design space that truly flexible agentic AI requires. Without the Room sandbox, many of those possibilities would remain either theoretical or too dangerous to attempt at scale.

From there, applications like Powerboards, and many others yet to be built, have a solid foundation on which to grow. They do not need to reinvent runtime, governance, or collaboration; they can stand on MeshAgent and focus on the user experience and domain logic that make them unique.

11. Conclusion: Giving agents a real home

For the growing number of leaders who are thinking not just about the next AI pilot but about the shape of their organization in an agentic world, the implication is clear:

- You don’t only need better agents.
- You need a better place for them to live, and a safe workshop where they can build.

MeshAgent is that place.