ProductFuse by JIBE

IOT PLATFORM ENGINEERING · WHITEPAPER

The software layer your hardware team didn't plan for.

Why connected-product platforms fail at the connectivity layer, and what it takes to get it right.

ProductFuse by JIBE · Eindhoven, NL · productfuse.eu 2026

00 — Introduction

Every company building connected hardware eventually faces the same moment: the device works, the market exists, and the team is ready to scale. Then the platform work begins — and it is nothing like building the device.

The connectivity layer between a physical device and a production-grade digital service is where most connected-product projects break. Not because the engineers are wrong, but because the problem is consistently underestimated. Protocols don't behave uniformly. Legacy devices respond differently from spec. Multiple backends need to co-exist. Certification environments require auditability that was never designed in. And the operational tooling — monitoring, firmware management, alerting, reporting — turns out to be a platform in its own right.

This paper is about the two failure patterns we encounter most often: protocol fragmentation handled too late, and non-functional requirements treated as a version-two problem. Both are predictable. Both are preventable. And both leave the same fingerprints on a project in trouble — slipping timelines, rising maintenance cost, and a platform that can't scale without being rebuilt.

Who this paper is for

CTOs, Heads of Engineering, and Innovation Leads at companies building or operating connected hardware. If you are planning a platform build, evaluating a software partner, or trying to understand why a previous attempt hit a wall — this paper is written for you.

The evidence behind this paper comes from nine years of building and operating IoT platforms in production — across EV charging networks, smart energy systems, connected hospitality infrastructure, and device certification environments. We have made mistakes, shipped them, fixed them, and extracted the patterns. What follows is what we learned.

FAILURE PATTERN 01

Protocol fragmentation, handled too late.

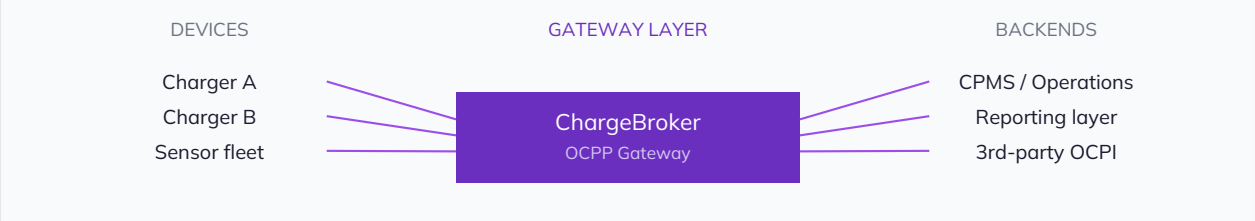
Most hardware companies discover their device ecosystem is more fragmented than expected only after they start building the platform. Three charger models from two manufacturers implement OCPP 1.6 differently. A legacy sensor fleet speaks a proprietary protocol. The CPMS the business already uses expects a direct WebSocket connection to every device. A new reporting aggregator needs a separate data feed from the same sessions.

The instinct is to solve each integration separately as it appears. A custom adapter here, a message-translation layer there. By the time the fourth integration is in place, the architecture has become a mesh of point-to-point connections — each one fragile, none of them governed, and all of them owned by a single engineer who is now the most dangerous dependency in the system.

> *"By the time the fourth integration is in place, the architecture has become a mesh of point-to-point connections — each one fragile, none of them governed."*

The right model is a gateway that owns the device connection and routes to everything else. One governed connection per device. Backends register against the gateway, not against individual devices. Adding a backend becomes a configuration change, not a re-integration. The device never sees conflicting commands from competing systems because no two systems talk to the device directly.

This is the orchestration model. It requires a conscious architectural decision early, before the integrations compound. Teams that make it late face a partial rebuild under delivery pressure. Teams that make it early treat every subsequent integration as configuration.



One governed connection per device — multiple backends by configuration.

FAILURE PATTERN 02

Non-functional requirements treated as version two.

Non-functional requirements — scalability, resilience, security, auditability, and performance under load — share a common fate in most platform projects: they are acknowledged in the architecture document and deferred to a later sprint. The reasoning is always the same: first prove the concept, then harden it. Version one ships. The hardening sprint never arrives.

The consequences are not visible until scale. A platform that handles fifty devices in a test environment reveals its architecture at five thousand. A data pipeline that works when sessions are counted in hundreds fails compliance audits when regulators require session-level traceability and the records were never structured for export. A firmware update flow that worked manually across a small fleet becomes inoperable when it needs to run concurrently across thousands of devices in unpredictable network conditions.

> *"A platform that handles fifty devices in a test environment reveals its architecture at five thousand."*

Scalability

Device count grows non-linearly. Gateway, data pipeline, and operations layer must be designed to scale horizontally before first production deployment.

Auditability

Regulatory and commercial use cases require complete, signed session records. Summaries and estimates do not satisfy auditors or compliance frameworks.

Resilience

Field device network conditions are unpredictable. Failover, replay, and reconnection logic must be designed in — not bolted on after the first outage.

Security

TLS, device authentication, security profiles, and role-based access are prerequisites for any enterprise or regulated deployment — not post-launch features.

The fix is not to build everything on day one. It is to make architectural decisions that keep the door open: data models structured for auditability from the start, gateway connections designed to handle failover, infrastructure provisioned to scale without re-architecture. The cost of getting this right early is small. The cost of retrofitting it at scale is measured in months.

01 — What the platform layer actually needs to do

Before evaluating any partner or technology, it helps to be explicit about what a production-grade IoT platform must actually deliver. The following is a capability map — not a services list, but a diagnostic. If any of these areas is absent or underspecified in an architecture, the platform has a gap.

01 Connectivity & protocol layer

Every device connects through a governed gateway that owns the OCPP, MQTT, or proprietary session and routes to registered backends. Translation, filtering, and version mediation happen here — not in every backend separately.

02 Device management

Remote configuration, firmware staging and deployment, health monitoring, diagnostics, and incident response — at fleet scale, across heterogeneous hardware, under unpredictable network conditions.

03 Cloud and edge architecture

The platform must run reliably across public cloud, private cloud, on-premise, and air-gapped environments. Decisions made for one deployment model must not prevent another.

04 Data pipelines and auditability

Session-level data captured at the gateway, structured for billing, reporting, compliance, and certification. Every record signed, timestamped, and exportable. Summaries are not a substitute.

05 Operational tooling

Multi-tenant customer management, per-device diagnostics, smart alerting with severity levels, CDR generation, and fleet-wide configuration. Not a dashboard — the control layer the business runs on.

06 Testing, certification & release

QA frameworks for device validation, structured release and acceptance processes, certification-ready workflows. First-time-right delivery requires quality built into the process, not inspected in at the end.

02 — What nine years in production taught us

ChargeBroker — the OCPP gateway at the core of the ProductFuse EV charging platform — has been running in production since 2017, across networks in seven countries, built by a team that contributed directly to the OCPP standard. What follows is not a case study. It is a distillation of what operating a platform at this scale, under real-world pressure, actually teaches.

1 The spec is not the implementation.

OCPP 1.6 describes a standard. Dozens of charger manufacturers implement it differently. The gateway must handle edge cases the spec never anticipated: chargers that send malformed messages, firmware that drops the WebSocket on update, devices that report incorrect meter values. Robustness at the gateway level is the only protection downstream systems have.

2 Multi-backend orchestration is the default state, not an edge case.

Almost every operator running a mature network needs to route session data to more than one backend: a CPMS for operations, a reporting layer for revenue, an OCPI endpoint for roaming. A platform designed for one backend is already legacy before it ships.

3 Auditability must be designed in at session level.

Several industries now face regulatory or commercial requirements where session-level records are the only acceptable evidence. Building a reporting layer on aggregated data after the fact is not a viable path. The data structure must support it from day one.

4 Operational tooling is a product, not a spreadsheet.

The team operating thousands of devices cannot work from raw logs and manual firmware pushes. Alerting with severity levels, automated recovery, fleet-wide configuration, and per-device diagnostics are the difference between a platform the business can run and one requiring constant engineering intervention.

03 — How to evaluate a platform engineering partner

The difference between a platform engineering partner and a software development supplier is not technical capability — it is operational accountability. Whether the partner has built something similar, run it in production, and carries the institutional knowledge of what breaks at scale. These six questions separate the two.

Q1 Do you operate what you build?

A partner who only delivers code hands off the hardest problems. Ask for a platform they built and currently operate — not as a managed-service abstraction, but as the engineering team responsible when something breaks at 2am.

Q2 Have you dealt with protocol fragmentation at scale?

Ask what protocols they have implemented in production, how they handle non-compliant device behaviour, and what their gateway architecture looks like. Generic answers indicate a team that has not yet encountered the problem.

Q3 How do you approach non-functional requirements?

Ask specifically: how is scalability designed in? What does the data model look like for auditability? How is resilience handled at the gateway? A partner who cannot give concrete answers has not built a platform that had to meet them under pressure.

Q4 What does your QA and release process look like?

First-time-right delivery in IoT requires structured device validation, regression testing against real hardware, and a release process the operations team trusts. Ask for the specifics — not the methodology, the actual steps.

Q5 What is your team's attrition rate?

IoT platform knowledge is institutional. High turnover loses the people who know why decisions were made and where the edge cases are. Team stability is a direct proxy for continuity of your platform.

Q6 How is a long-term engagement different from a project?

Connected-product platforms run, evolve, and need operational continuity for years. A partner without a model for recurring platform responsibility is structurally misaligned — regardless of how good the first delivery is.

04 — A note on ProductFuse

ProductFuse is the IoT platform engineering practice of JIBE, based in Eindhoven. We help companies that build or operate connected hardware turn their products into scalable, reliable, and commercially viable platforms — from architecture and integrations to cloud applications, device management, testing, and long-term operational support.

We are not a consultancy that designs a slide deck and leaves execution to others. We build and operate our own products — ChargeBroker for EV charging orchestration, Zyonara for smart energy management, GuestConnect for hospitality and commercial display infrastructure — and we apply the same engineering discipline to our customers' platforms. Our architecture choices, QA processes, and integration patterns are shaped by production pressure, not by theory.

Our team of 20–40 senior engineers operates from Eindhoven, Cluj, and Sofia, with well below-average attrition and continuity of platform knowledge that compounds over years of engagement. AI-assisted delivery accelerates our cycles without compromising the quality or predictability that enterprise-grade platforms require.

ChargeBroker
EV Charging Orchestration
In production since 2017 · 7 countries · contributed to the OCPP standard

Zyonara
Smart Energy Management
Real-time monitoring, energy optimization, and solar integration

GuestConnect
Hospitality & Displays
Philips PPDS ecosystem · device management at commercial scale

If the challenges described in this paper sound familiar, we are happy to start with a conversation about your specific platform situation — no generic proposal, a real discussion about what you are building and where it needs to go.

ProductFuse by JIBE
hello@productfuse.eu · productfuse.eu · Eindhoven, NL