Financial Systems Archetype Guidance Documents

Here is a draft overview you can use near the beginning of the document set, probably in an introductory section before the individual parts are described.

Motivation for the Document Structure

The document set is organised as a progression from conceptual foundations through platform-independent architecture, platform-specific realisation, and implementation guidance. This structure is intentional. It separates enduring architectural meaning from technology choices, deployment decisions, and executable artefacts. By doing so, the architecture can remain stable even as particular tools, middleware products, programming languages, or runtime environments change.

At the conceptual level, the documents define the problem space, the governing principles, and the architectural concerns the system must address. This includes policy-governed release, sovereignty, residency, distributed governance, data in motion, data at rest, metadata, aggregate views, evidence, and the separation of data, control, release, and cost-recovery concerns. The conceptual material explains why the architecture exists and what kinds of obligations it must satisfy before any specific technology is selected.

The Platform-Independent Model, or PIM, translates those concepts into reusable architectural structures. It defines logical nodes, planes, topics, data structures, interaction patterns, governance boundaries, and lifecycle responsibilities without committing to a particular implementation technology. The PIM is the main architectural contract. It allows different communities, jurisdictions, vendors, and implementation teams to understand the same system architecture in a common way.

The Platform-Specific Model, or PSM, maps the PIM onto selected technologies and standards. In the current work, this includes realising the architecture using DDS topics, QoS profiles, IDL structures, node roles, generated types, and runtime conventions. The PSM does not replace the PIM. It shows how the PIM can be implemented using a specific technical stack while preserving the architectural intent defined at the conceptual and platform-independent levels.

The implementation guidance then provides the practical instructions needed to build, test, package, run, and maintain a working demonstration or deployment. This includes repository layout, coding conventions, generated files, scripts, logging, lifecycle reporting, configuration, build procedures, and acceptance criteria. These materials are deliberately kept separate from the conceptual and PIM-level documents because they are more likely to change as the implementation matures.

This layered document structure supports disciplined evolution. Conceptual principles should change rarely. PIM structures should change only through architectural review. PSM mappings may change as technology choices evolve. Implementation guidance may change frequently as the team learns from coding, testing, deployment, and operational experience. Keeping these levels separate prevents implementation details from distorting the architecture and from burying architectural concepts inside code-level instructions.

The structure also supports multi-jurisdictional and multi-vendor participation. Different organisations may implement different nodes, use different internal technologies, or operate under different legal and policy regimes, while still conforming to the same conceptual and platform-independent

architecture. This is essential for a distributed financial systems architecture, in which no single participant owns the entire system, but all participants must contribute to its integrity, observability, and governability.

In short, the document progression from Conceptual to PIM to PSM to implementation guidance preserves a clear chain of reasoning: why the system exists, what architectural obligations it must satisfy, how those obligations are represented independently of technology, how they are realised using selected technologies, and how they are implemented in working software. This separation of concerns is what allows the architecture to remain understandable, auditable, portable, and durable over time.