An Initiative to Unify Global Pharmaceutical Standards and Regulatory Frameworks on SagaChain

Research/Draft Prepared By: ChatGPT 5.0, Grok 4.0
Reviewed By: Michael Holdmann, David Beberman, Rich Phillips

NOVEMBER 20, 2025

Table of Contents

Abstract

The SagaPharma™ under the umbrella of SagaStandards™, is an effort to unify pharmaceutical standards and regulatory frameworks into a single-instance, multi-inheritable class tree on SagaChain™. Building on decades of work by standards bodies (GS1, CDISC, HL7, IDMP, SPL, FAERS, MedDRA, USP) and regulators (FDA, EMA, WHO, and others), SagaPharma encodes these frameworks as persistent SagaPython™ classes. This transforms them from static specifications into living, interoperable code that operates on a decentralized, permissionless Layer 1 blockchain.

The initial seeding of the SagaPharma Tree and all implemented ALPHA code was completed solely by PraSaga Foundation as a gift to the stakeholders and customers of the pharmaceutical industry. This effort is not affiliated with any other Standards Development Organization, Government or Regulatory Agency.

All code was generated from Open Public Machine-Readable source using Grok AI platform to retrieve and convert the XML, OWL, JSON, .pdf, RDF .csv, etc. docs into SagaPython Classes.

This document research and draft was prepared by the AI platform that generated the respective code and mapped the ontologies, it was reviewed by the PraSaga Foundation team for editing/correction of blatant hallucinations. The validation of the architecture and ontologies is now ready for stakeholders to review and update.

The initiative is designed as a public good:

• For industry, it eliminates reconciliation costs and reduces vendor lock-in.

• For governments and regulators, it enables real-time oversight with privacy-preserving compliance.

• For standards bodies (SDOs), it ensures faithful, canonical execution of their schemas.

• For patients and providers, it guarantees transparency, trust, and safety.

As a SagaStandards initiative, SagaPharma is governed by an open membership model, actively recruiting participants from governments, industry consortia, regulators, and other SDOs to take custodianship of SagaPharma, and maintain. By aligning with both technical innovation and institutional governance, SagaPharma positions SagaChain™ as the first global pharmaceutical infrastructure standard, interoperable by design and stabilized by the SagaCoin™ management model.

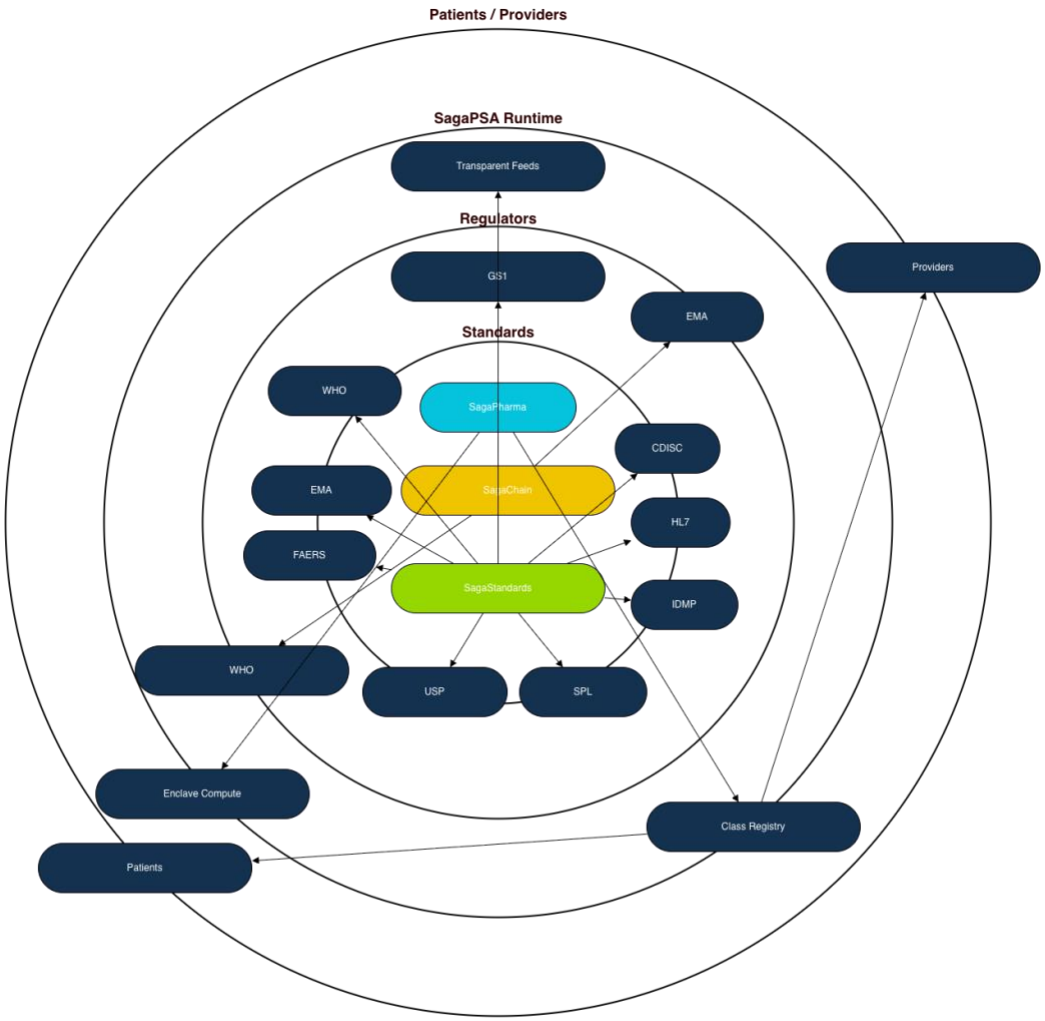


Diagram 1-A — Global Pharma Standards Interoperability Map — Standards → Regulators → SagaPSA Runtime → Patients/Providers

Executive Summary

The modern pharmaceutical system rests on a mosaic of standards and regulators that operate independently yet depend on one another. GS1 governs supply chain serialization, CDISC governs clinical data, HL7 governs interoperability, IDMP governs product identification, SPL governs labeling, and the FDA mandates approvals. Regulators such as the EMA, WHO issue parallel requirements. Each framework succeeds within its own domain, yet fragmentation across domains creates inefficiency, opacity, and systemic risk.

1. Introduction

The pharmaceutical sector is at a critical inflection point. Scientific breakthroughs—such as gene therapies, CAR-T cell therapies, and digital therapeutics—are reshaping patient care. At the same time, systemic pressures—including pandemics, supply chain fragility, rising costs, and inequitable access to medicines—are exposing the limitations of existing regulatory, financial, and data infrastructures. Traditional

compliance and oversight mechanisms rely on static documents, fragmented reporting, and manual reconciliation, which cannot keep pace with modern global healthcare demands.

Across supply chain serialization, regulatory filings, manufacturing telemetry, pharmacovigilance systems, and patient outcomes, a common challenge persists:

- **Fragmentation:** Each domain (supply chain, clinical research, regulatory approval, safety monitoring, financial settlements) operates in silos, with minimal interoperability.
- **Latency:** Critical signals—such as safety alerts, quality deviations, or reimbursement triggers—often take weeks or months to propagate, leaving patients at risk.
- **Opacity:** Evidence trails are inconsistent, difficult to audit, and often inaccessible to stakeholders, eroding trust in global health systems.
- **Inequity:** Low- and middle-income countries (LMICs) face persistent barriers to access, as existing infrastructures do not provide verifiable proofs of allocation, safety, and quality.

1.1 Limitations of the Current Ecosystem

Current pharmaceutical governance relies heavily on document-based compliance: regulatory dossiers, structured product labeling (SPL), adverse event reports, quality validation packages, and donor funding agreements are distributed as static files. While standards such as GS1 EPCIS, ISO IDMP, CDISC, HL7 FHIR provide schemas for structuring these datasets, they remain externally referenced artifacts rather than executable, persistent entities.

This creates disconnects: serialized supply chain data does not trigger automated financial settlements; clinical trial results are not seamlessly linked to post-market safety; and equity commitments are reported retrospectively rather than enforced in real time.

Moreover, privacy and confidentiality concerns—including patient health information (PHI), genomic data, and commercial contracts—limit cross-border data sharing. Existing solutions either overexpose sensitive data or fail to provide regulators and payers with sufficient verifiable proofs.

1.2 The SagaChain Approach

SagaChain introduces a fundamentally different paradigm. By encoding global pharmaceutical standards directly into a Universal Class Tree of SagaPSA (Programmable Smart Assets), the system enables:

· Persistence: Data objects—such as clinical trial datasets, EPCIS events, or reimbursement settlements—exist as continuous, stateful entities rather than one-time submissions.

- **Interoperability:** Standards (GS1, DSCSA, CDISC, HL7, FAERS,

WHO VigiBase, etc.) are harmonized within a single inheritance tree, allowing seamless cross-domain linkage.

- **Auditability:** Every transformation—from raw data ingestion to AI-generated safety signals—is attested with cryptographic proofs, ensuring reproducibility and trust.
- **Confidentiality:** Enclave computation executes sensitive operations (e.g., genomic analysis, commercial settlement logic) while exposing only compliance proofs and aggregate results.
- **Transparency:** SagaFeeds™ publish public verified source on labeling, safety signals, equity compliance, and donor-funded allocations, empowering patients, providers, and regulators with verifiable, real-time insights.

1.3 The Role of SagaStandards

This initiative is being advanced under SagaStandards, a division of the PraSaga Foundation dedicated to establishing open, participatory global standards for programmable compliance and interoperability. SagaStandards is not a closed consortium; it is an invitation for collaboration with:

- **Regulators:** seeking more transparent and verifiable compliance frameworks.
- **Industry stakeholders:** manufacturers, distributors, payers—seeking harmonized systems that reduce cost and complexity.
- **Standards Development Organizations (SDOs):** such as GS1, HL7, CDISC, and USP interested in extending their schemas into executable, persistent-state class trees.
- **Governments and global health bodies:** (WHO, GAVI, national agencies) working toward equity, preparedness, and resilience.

By framing compliance as programmable smart assets rather than static documents, SagaStandards provides the governance structure and collaborative pathway for this Universal Class Tree to be developed as a global public good.

1.4 Toward a Global Pharmaceutical Universal Class Tree

The Universal Class Tree is designed as a multi-inheritable ontology, where regulatory, financial, clinical, and safety objects interconnect seamlessly. For example:

- A lot release object inherits from GMP manufacturing classes, USP assay proofs, DSCSA serialization, and SPL labeling.
- A pharmacovigilance signal object links HL7 patient outcomes, FAERS/EudraVigilance reports, and ISO IDMP product identifiers.
- A donor-funded access asset ties GS1 serialization, WHO treaty obligations, and financial escrow

conditions into a verifiable equity framework.

This persistent-state model transforms pharmaceutical compliance from a retrospective, document-heavy burden into a real-time, programmable fabric of trust.

1.5 Structure of this White Paper

This paper is organized into the following major sections:

Section 2: Conceptual Foundations of the Universal Class Tree.

Section 3: Architecture of SagaChain™, SagaPython™, and SagaPSA™.

Section 4: Persistent-State Governance, Privacy, and Enclave Computation.

Section 5: Global Scenario Applications (Scenarios 1–25), illustrating end-to-end use cases across manufacturing, safety, regulatory, financial, and equity contexts.

Section 6: Standards Integration, detailing how SagaChain encodes and harmonizes GS1, HL7, CDISC, USP, WHO, and financial standards.

Section 7: Roadmap and Stakeholder Participation—outlining how SagaStandards enables open collaboration with regulators, payers, industry, and SDOs.

Section 8: Implications for Global Health, Equity, and Innovation.

2. Conceptual Foundations of the Universal Class Tree

The Universal Class Tree is the conceptual backbone of SagaChain™ and the broader SagaStandards initiative of the PraSaga Foundation. It embodies the principle that global pharmaceutical data, processes, and compliance artifacts must not remain in static documents or siloed systems, but rather exist as persistent, interoperable, and auditable objects within a unified ontology.

By transforming regulatory filings, supply chain events, financial settlements, and pharmacovigilance reports into programmable smart assets, the Universal Class Tree creates a living, continuously updated “evidence graph” for global healthcare. This evidence graph is designed not as a proprietary platform, but as a participatory standard under SagaStandards, open to industry, government agencies, and Standards Development Organizations (SDOs) for collaboration, extension, and adoption.

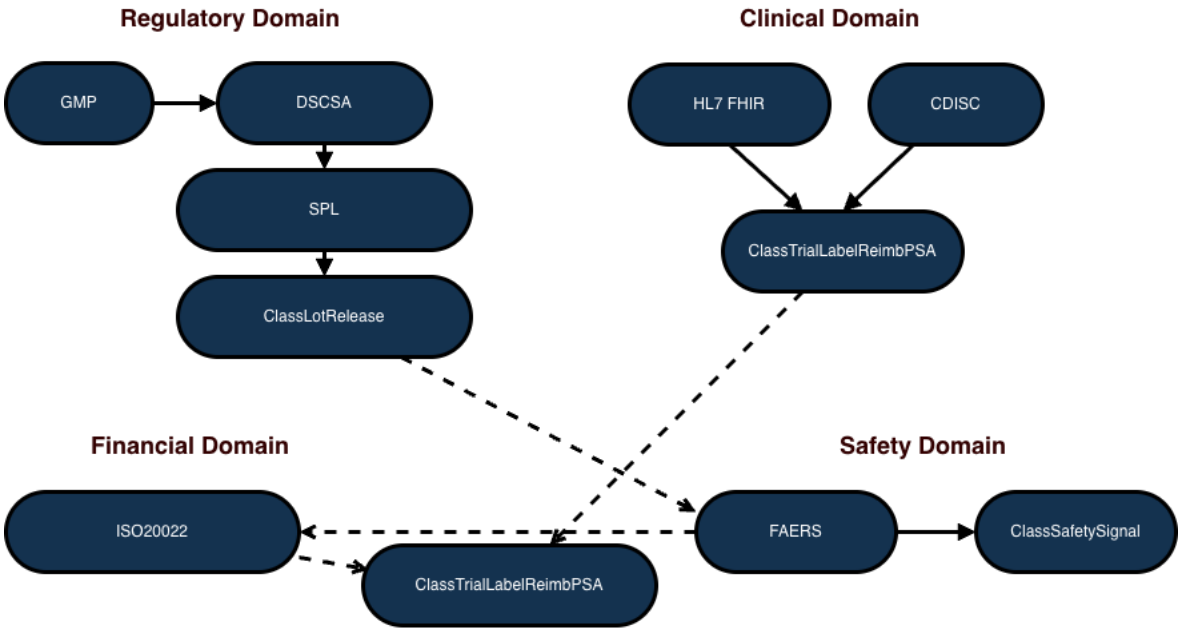


Diagram 2-A — Universal Pharma Class Tree Conceptual Architecture — Regulatory • Clinical • Financial • Safety (v2)

2.1 From Static Standards to Executable Objects

Traditional standards—such as GS1 EPCIS (supply chain events), ISO IDMP (product identification), CDISC SDTM (clinical data), HL7 FHIR (EHR interoperability), and FAERS/MedDRA (safety reporting)—provide schemas and reference models. Yet in practice, these standards are implemented inconsistently, exchanged as documents, and rarely integrated in real time.

The Universal Class Tree reinterprets these schemas as SagaPSA™ (Programmable Smart Asset) classes within SagaPython™:

- **GS1/DSCSA**: serialized units become objects linked to manufacturing batches, SPL labels, and reimbursement flows.
- **CDISC/HL7:** trial data becomes executable classes tied directly to labeling updates and payer reimbursement decisions.
- **Safety events:** (FAERS, EudraVigilance, WHO VigiBase) become persistent pharmacovigilance objects cross-linked to product identifiers, clinical outcomes, and recall events.

This objectification transforms standards from passive references into active, stateful participants in compliance and oversight.

2.2 Multi-Inheritance and Ontology Design

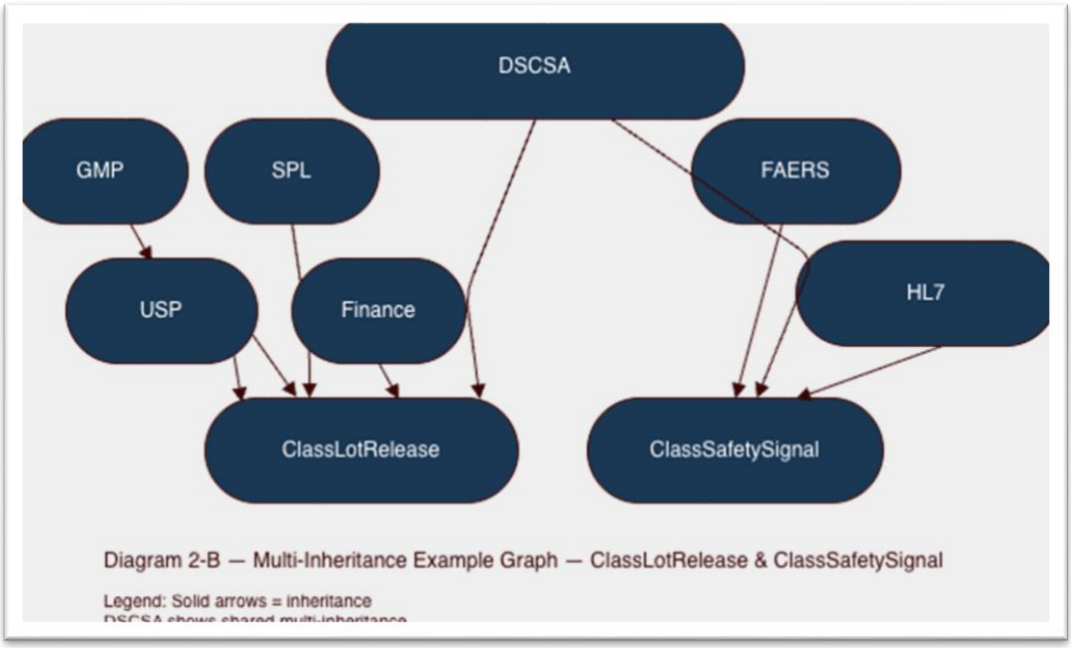
The Universal Class Tree is structured as a multi-inheritable ontology. Each class can inherit attributes and methods from multiple

domains, reflecting the interconnected nature of pharmaceutical compliance.

For example:

- A ClassLotRelease object may inherit from:
    - Manufacturing (GMP/USP assays)
    - Supply chain (GS1/DSCSA serialization, EPCIS events)
    - Regulatory (SPL labeling, IDMP identifiers)
    - Finance (financial settlement conditions)
- A ClassSafetySignal may inherit from:
    - Pharmacovigilance systems (FAERS, EudraVigilance, VigiBase)
    - Clinical outcomes (HL7 FHIR, CDISC trial data)
    - Supply chain provenance (GS1 identifiers, DSCSA units)

Through this ontology, disparate domains are reconciled into a cohesive, executable framework, enabling seamless propagation of compliance across the pharmaceutical lifecycle.



3. Architecture of SagaChain™, SagaPython, and SagaPSA™ (Pharma Edition, under SagaStandards)

This chapter adapts the foundational elements of SagaChain to the requirements of a single-instance global pharmaceutical class tree, governed as an open, participatory standard under SagaStandards (PraSaga Foundation). It delineates how SagaChain transforms regulatory frameworks and healthcare standards into persistent, multi-inheritable objects; elucidates the structural distinctions from first-generation blockchains and conventional middleware; and outlines mechanisms for industry, government, and standards development organizations (SDOs) to co-author and extend the technology stack.

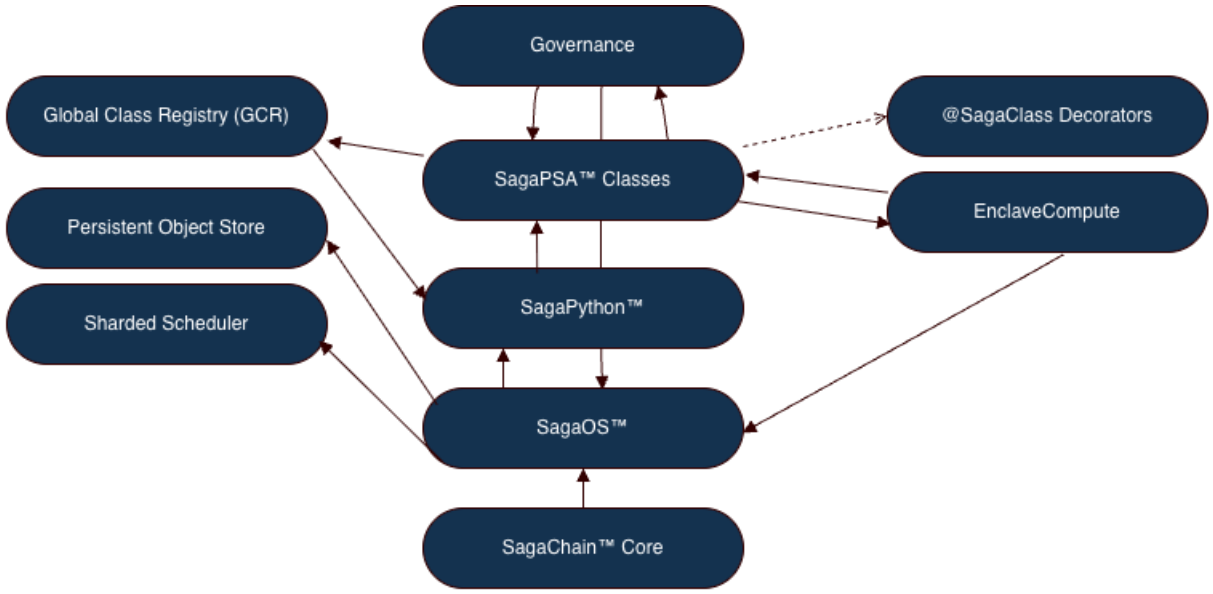


Diagram 3-A — SagaPharma Technical Architecture Stack — SagaChain Core → SagaOS → SagaPython → SagaPSA Classes → GCR/Governance/Enclave

3.1 SagaPython: Native Language for Persistent Pharma Classes

SagaPython™ constitutes a dialect of Python tailored for persistent, compliance-oriented execution on SagaChain. It retains Python's accessibility while incorporating chain-native semantics via the Class Manager Infrastructure (CMI) and policy hooks consonant with SagaStandards governance.

3.1.1 Innovations in SagaPython for Pharma

- **Persistent Classes:** The @SagaClass(<AncestorA>, <AncestorB>, ...) decorator designates classes that persist across blocks, shards, and versions.

- **Validated Fields:** SagaField("<field_name>", "<type>") enforces types, enums, patterns, ranges, and lengths, with validation occurring at class load and pre-commit.
- **Behavioral Methods:** The @SagaMethod() decorator declares lifecycle methods equipped with pre- and post-compliance hooks.
- **Multi-Inheritance:** Classes inherit across domains such as GS1, DSCSA, HL7, CDISC, ISO IDMP, SPL, FAERS, and EMA, with ancestors defined prior to subclasses.
- **Late Binding:** Method resolution occurs dynamically across the class tree, facilitating additive evolution without disrupting lineage.
- **Enclave Execution:** Classes or methods may be flagged for confidential compute (via policy), with outputs limited to selective proofs.

3.1.2 Example: Evidence-Linked Label Section

```python
from CMIConst import SPClassObject

@SagaClass(SPClassObject)

class ClassSPLSection:

    SagaField("spl_id", "str")

    SagaField("section_code", "str") # enum={"INDICATIONS","DOSAGE","WARNINGS","CLINICAL_STUDIES"}

    SagaField("text", "str")

    SagaField("evidence_oids", "list") # CDISC/HL7/Trial object OIDs

    SagaField("version", "str")


    @SagaMethod()

    def __init__(self, spl_id: str, section_code: str, text: str, evidence_oids: list, version: str):

        self._cmi().__init__() # Chain to SPClassObject

        self.spl_id = spl_id

        self.section_code = section_code

        self.text = text

        self.evidence_oids = evidence_oids

        self.version = version

        self.docs = [] # Default documentation array

        self._enclave = {} # Mock enclave storage


    @SagaMethod()

    def validate(self):

        assert len(self.evidence_oids) > 0, "SPL section must cite evidence objects"

        return True
```

This formulation enables industry participants to leverage Python proficiency; affords regulators compliance-by-design; and permits SDOs to operationalize schemas as executable code rather than static documents.

3.2 SagaOS: The On-Chain Operating System for Healthcare

3.2.1 Role in the Stack

Few distributed ledgers furnish an operating system that administers persistent objects, namespaces, versioning, scheduling, policy hooks, and message passing as primitive capabilities. SagaOS functions as the distributed kernel, ensuring that every standard class—encompassing ISO 20022, GS1 EPCIS, HL7 FHIR, CDISC, SPL, FAERS/EudraVigilance—executes within a singular canonical runtime, upholding uniform semantics under SagaStandards governance.

3.2.2 Core Subsystems

- **Global Class Registry (GCR):** A canonical catalog of all @SagaClass definitions, mitigating schema drift and preserving version lineage.
- Persistent Object Store (POS): Durable repository for instances, addressable via globally unique Object IDs (OIDs).
- Deterministic Message Bus (DMB): Facilitates causal, replayable invocations between objects' @SagaMethod() via OIDs.
- SagaScale (SS): Allocates and migrates objects by affinity and locality (see§3.6).
- Version & Policy Manager (VPM): Oversees evolution and attaches regulator checks pre- and post-method execution.

3.2.3 Example: OS-Level Process & Signals for GxP Workflow

```python
from CMIConst import SPClassObject

@SagaClass(SPClassObject)

class ClassOSProcess:

    SagaField("pid", "str")

    SagaField("owner_oid", "str")

    SagaField("state", "str") # enum={"Running","Suspended","Completed"}

    SagaField("tags", "dict") # {"GxP": "211", "validation": "IQ/OQ/PQ"}


    @SagaMethod()

    def __init__(self, pid: str, owner_oid: str, tags: dict = None):

        self._cmi().__init__() # Chain to SPClassObject

        self.pid = pid

        self.owner_oid = owner_oid

        self.state = "Running"

        self.tags = tags or {}

        self.docs = []

        self._enclave = {}


    @SagaMethod()
```

```
    def send_signal(self, signal: str):
        assert signal in {"SUSPEND","RESUME","STOP"}
        self.state = {"SUSPEND":"Suspended","RESUME":"Running","STOP":"Completed"}[signal]
        return self.state
```

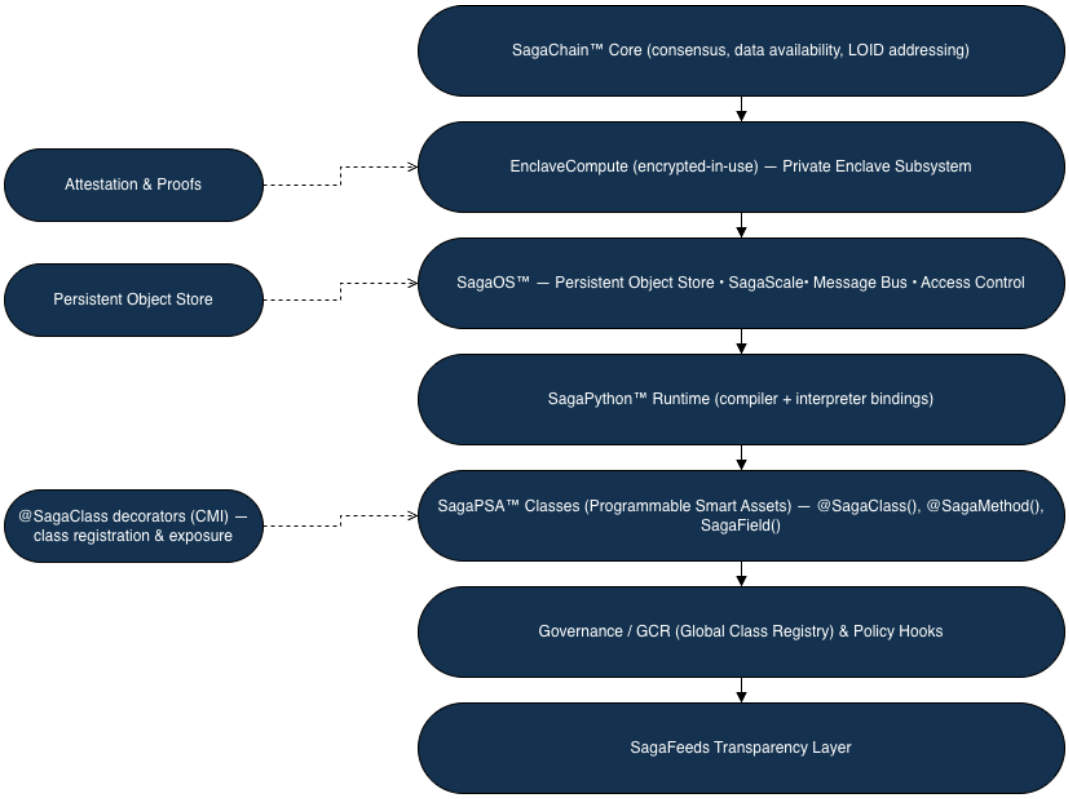


Diagram 3-D — SagaOS Core Components
SagaChain Core → SagaOS → SagaPython → SagaPSA Classes → Governance / Enclave / SagaFeeds

SagaOS Core Components stack: SagaChain Core provides consensus/availability and LOID addressing; EnclaveCompute enables encrypted-in-use execution; SagaOS furnishes persistent object store, sharded scheduling, and access control; SagaPython binds classes to runtime; SagaPSA exposes @SagaClass/@SagaMethod/SagaField; Governance+GCR and SagaFeeds complete the loop.

3.3 SagaPSA™: Programmable Smart Assets (Beyond "Contracts")

3.3.1 Concept

A SagaPSA denotes a long-lived healthcare object capable of inheriting concurrently from multiple standards and regulatory classes. The object embodies the full lifecycle—from manufacturing through serialization, labeling, safety surveillance, to reimbursement—eschewing brittle extract-transform-load (ETL) processes.

3.3.2 Invariants, Lifecycle, Policy Hooks

- **Invariants:** Implemented via SagaField constraints and @SagaMethod checks.

- **Lifecycle:** Domain-specific actions (e.g., commission_sgtins(), release_lot(), update_label(), settle_payment()) mutate the object while maintaining cryptographic lineage.
- **Policy Hooks:** The VPM affixes regulator validations atomically to business logic (e.g., requiring USP proof prior to SGTIN commissioning).

3.3.3 Example: Trial→Label→Reimbursement PSA

```python
from CMIConst import SPClassObject

@SagaClass(SPClassObject)

class ClassTrialLabelReimbPSA:

    SagaField("status", "str") # enum={"Collecting","Analyzing","Labeled","Reimbursed"}

    SagaField("analysis_datasets", "list")

    SagaField("section_code", "str") # enum={"CLINICAL_STUDIES"}

    SagaField("sender", "str", optional=True)

    SagaField("receiver", "str", optional=True)

    SagaField("currency", "str", optional=True, length=3)

    SagaField("amount", "int", optional=True) # Minor units, integer only


    @SagaMethod()

    def __init__(self, analysis_datasets: list, section_code: str):

        self._cmi().__init__() # Chain to SPClassObject

        self.status = "Collecting"

        self.analysis_datasets = analysis_datasets

        self.section_code = section_code

        self.docs = []

        self._enclave = {}

    @SagaMethod()

    def publish_label(self):

        assert len(self.analysis_datasets) > 0

        self.status = "Labeled"

        return self.status

    @SagaMethod()

    def trigger_reimbursement(self, payer: str, payee: str, ccy: str="USD", amount: int=100):  # Minor units

        assert self.status == "Labeled"

        self.sender = payer

        self.receiver = payee

        self.currency = ccy

        self.amount = amount

        self.status = "Reimbursed"

        return self.amount

    def _derive_amount_from_outcomes(self) -> int:
```

```
    # Placeholder deterministic rule in minor units
    return 100
```

3.4 Global Single-Instance Pharma Class Tree

A foundational tenet of SagaChain™ resides in its Global Single-Instance Class Tree—a canonical namespace wherein every regulatory, clinical, supply chain, safety, and financial class manifests once and inherits uniformly across applications. This paradigm obviates the protracted challenge of schema proliferation and vendor-specific divergences (e.g., FDA SPL, EMA IDMP, USP monographs, GS1 EPCIS). Instead, each lifecycle entity—from preclinical study objects to biosimilar records—inherits from this unified tree, with explicit version lineage and cryptographic auditability.

3.4.1 Canonical Namespace and Versioned Evolution

- All classes (e.g., ClassSPLDocument, ClassDSCSAUnit, ClassFAERSReport) register in the GCR.
- Conflicts resolve through versioned inheritance, not divergence; for instance, SPL_v2 extends SPL_v1 additively.
- Under SagaStandards governance, modifications proceed via structured class diffs, subject to review by regulators, SDOs, and industry stakeholders.

3.4.2 Pharma-Specific Example: Lot Release

The lot release process, conventionally fragmented across GMP batch records, USP assays, DSCSA serialization, and SPL labeling, coalesces into a singular persistent object on SagaChain:

```python
from CMIConst import SPClassObject
from ClassFungibleAsset import ClassFungibleAsset # Assuming prior definition

@SagaClass(ClassFungibleAsset, SPClassObject)
class ClassLotRelease:
    SagaField("lot_id", "str")
    SagaField("product_id", "str") # ISO IDMP identifier
    SagaField("gmp_batch_ref", "str")
    SagaField("usp_assay_proofs", "list") # Enclave-generated QC results
    SagaField("dsccsa_units", "list") # Serialized unit references
    SagaField("spl_ref", "str") # Structured product labeling link
    SagaField("release_status", "str") # enum={"Pending","Released","Quarantined"}
    @SagaMethod()
    def __init__(self, lot_id: str, product_id: str, gmp_batch_ref: str):
```

```
        self._cmi('ClassFungibleAsset').__init__() # Chain to ClassFungibleAsset
        self._cmi().__init__()  # Chain to SPClassObject
        self.lot_id = lot_id
        self.product_id = product_id
        self.gmp_batch_ref = gmp_batch_ref
        self.usp_assay_proofs = []
        self.dsccsa_units = []
        self.spl_ref = ""
        self.release_status = "Pending"
        self.docs = []
        self._enclave = {}

    @SagaMethod()
    def approve_release(self):
        assert self.usp_assay_proofs and self.dsccsa_units and self.spl_ref
        self.release_status = "Released"
        return self.release_status

    @SagaMethod()
    def enclaveSet(self, key: str, value: str):
        self._enclave[key] = value
        return self._enclave[key]

    @SagaMethod()
    def enclaveGet(self, key: str):
        return self._enclave.get(key)
```

Here, the lot release object encapsulates the lifecycle: assay results, serialization, labeling, and approval embed within one auditable class.

3.4.3 Governance Under SagaStandards

- **Industry participation:** Manufacturers propose attributes (e.g., cold-chain telemetry types).
- **Regulators:** FDA, EMA, WHO validate changes within the governance framework.
- **SDOs:** USP, GS1, HL7, CDISC ensure schema fidelity as executable classes.
- **Version lineage:** Changes prioritize backward compatibility; disruptions yield versioned subclasses (e.g., ClassLotRelease_v2).

3.4.4 Implications

- **For Industry:** Schema reconciliation costs diminish; a unified definition propagates globally.
- **For Regulators:** Audit trails auto-preserve; objects trace lineage to originating classes.
- **For Standards Bodies:** Specifications evolve from inert documents to dynamic, executable code in a shared runtime.
- **For Patients:** Enhanced safety and expedited medicine access via real-time, verifiable compliance.

Section 3.4 Summary:

The Global Single-Instance Pharma Class Tree reconceptualizes compliance as a shared, executable ontology, ameliorating schema duplication and latency. By integrating GMP, USP, DSCSA, SPL, and IDMP into a canonical namespace, SagaChain lays the groundwork for perpetual, interoperable pharmaceutical assurance.

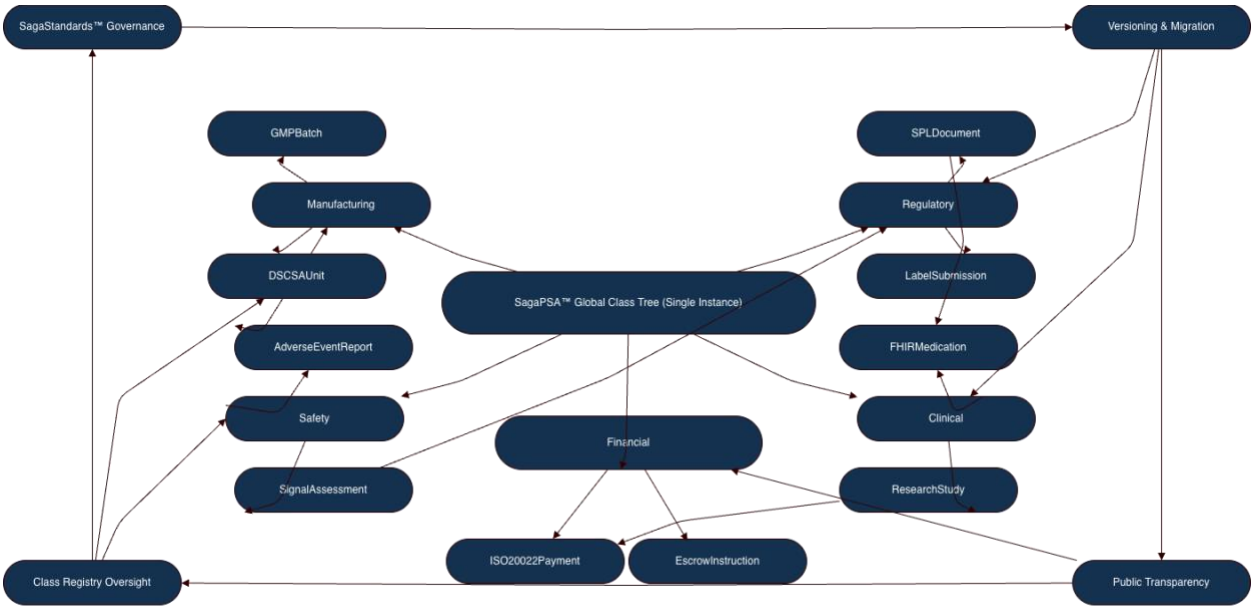


Diagram 3-B — Global Single-Instance Pharma Class Tree — Manufacturing · Regulatory · Clinical · Safety · Financial (Governance Perimeter)

3.5 Governance and SagaStandards Participation

A pivotal distinction of SagaChain™ lies in its integration of technical runtime with participatory governance under SagaStandards, the PraSaga Foundation's standards division. In contrast to blockchain systems reliant on ad-hoc proposals or token-based voting, SagaChain aligns with SDOs, regulators, and consortia structures, rendering governance institutional, collaborative, and inherently auditable.

3.5.1 Role of SagaStandards

SagaStandards stewards the evolution of the Global Pharmaceutical Universal Class Tree, furnishing:

- **Membership and Working Groups:** Open access for regulators, manufacturers, SDOs (e.g., ISO, GS1, HL7, CDISC, USP), and payers.
- **Namespace Stewardship:** Domains (e.g., ISO 20022, GS1 EPCIS, HL7 FHIR, USP) reside in segregated namespaces within the GCR, curated by pertinent SDOs or regulators.
- **Change Proposal Workflow:** Novel classes, fields, or methods enter via formal class diffs, deliberated in working groups, and ratified for GCR inclusion.

- **Compliance Alignment:** Digital class definitions map precisely to regulatory and industry standards, forestalling divergence between specifications and code.

3.5.2 Governance Workflow

Updates to the Universal Class Tree adhere to a transparent, multi-stakeholder process:

1. **Proposal Submission:** SagaStandards members (industry, regulators, SDOs, observers) tender new classes, fields, or decorator rules.
2. **Working Group Review:** Proposals route to domain-specific groups (e.g., Pharmacovigilance WG, Supply Chain WG) for technical and regulatory scrutiny.
3. Public Transparency via **SagaFeeds™:** Pending proposals index as discov
4. erable on-chain objects, enabling oversight by regulators, academics, and the public.
5. **Consensus & Ratification:** Domain stewards vote; ratified alterations integrate into the GCR.
6. **Versioning and Migration:** Changes favor additivity; disruptions spawn versioned subclasses (e.g., ClassDSCSAUnit_v2) with defined migration methods.

This workflow supplants informal advocacy and opaque revisions with auditable, participatory mechanisms.

3.5.3 Policy Principles

SagaStandards governance upholds inviolable tenets:

- Additive-First Evolution: Modifications extend functionality sans invalidating extant objects.
- Explicit Migration: Inevitable disruptions mandate intra-class migration hooks.
- Namespace Integrity: Classes uniquely occupy namespace-version pairs; conflicts yield versioned lineages, not replicas.
- Public Transparency: SagaFeeds™ render proposals, commentary, and ratifications immutable and globally accessible.

3.5.4 Industry & Regulator Benefits

- Industry: Assured encoding of compliance in the Universal Class Tree curtails integration expenses and interpretive variance.
- Regulators: Direct stewardship of domains (e.g., FDA for SPL, EMA for EudraVigilance) with preserved lineage for audits.
- SDOs: Canonical, executable representations of specifications at scale.
- LMICs & Global Health Bodies: Open participation yields immediate compliance visibility, sans proprietary encumbrances.

3.5.5 Example Governance Use Case

```python
from CMIConst import SPClassObject


@SagaClass(SPClassObject)
```

```
class ClassGovernanceProposal:
    SagaField("proposal_id", "str")
    SagaField("namespace", "str")  # e.g., "HL7.FHIR.Medication"
    SagaField("proposer", "str")  # Member ID or regulator account
    SagaField("change_type", "str")  # additive, breaking, migration
    SagaField("diff_summary", "str")  # Human-readable description
    SagaField("status", "str")  # enum={"Submitted","Reviewed","Ratified", "Rejected"}

    @SagaMethod()
    def __init__(self, proposal_id: str, namespace: str, proposer: str, change_type: str, diff_summary: str):
        self._cmi().__init__()  # Chain to SPClassObject
        self.proposal_id = proposal_id
        self.namespace = namespace
        self.proposer = proposer
        self.change_type = change_type
        self.diff_summary = diff_summary
        self.status = "Submitted"
        self.docs = []
        self._enclave = {}

    @SagaMethod()
    def advance_status(self, new_status: str):
        assert new_status in {"Reviewed","Ratified","Rejected"}
        self.status = new_status
        return self.status
```

3.6 SagaInterop - Cross-Chain Interoperability

SagaInterop™ is the SagaChain™ subsystem that enables deterministic, auditable message and state exchange between SagaChain and external blockchain networks (both public and enterprise). It provides a programmable bridge layer that preserves on-chain provenance, digital signatures, and object semantics across heterogeneous runtimes.

3.6.1 Architecture

Cross-Domain Class Mirroring:
SagaPython classes can be declared as *interoperable* using the @SagaInteropClass() decorator, defining serialization and verification schemas for external networks such as Ethereum, Hyperledger Fabric, or Solana.

- **Bridge Anchors:** Each participating chain hosts a lightweight anchor contract that stores the cryptographic commitment of the SagaChain LOID, plus Merkle-proof links to the original SagaPSA™ (Class/Method/Field) transaction.
- **State Attestation Protocol (SAP):** A standardized, ISO 20022-aligned envelope (<SagaInteropSAP>) carries state attestation, timestamp,

jurisdiction tag, and proof hash. SAPs are routed via SagaScale™ L2L gossip for settlement ordering.

- **Bidirectional Invocation:** Authorized nodes can execute verified cross-chain calls through the SagaOS™ Bridge Manager, which enforces gas reconciliation, event replay protection, and LOID-based message idempotency.
- **Public–Private Boundary Control:** Enclave and non-enclave accounts can both issue cross-chain events; enclave proofs remain opaque but verifiable through their attestation references.

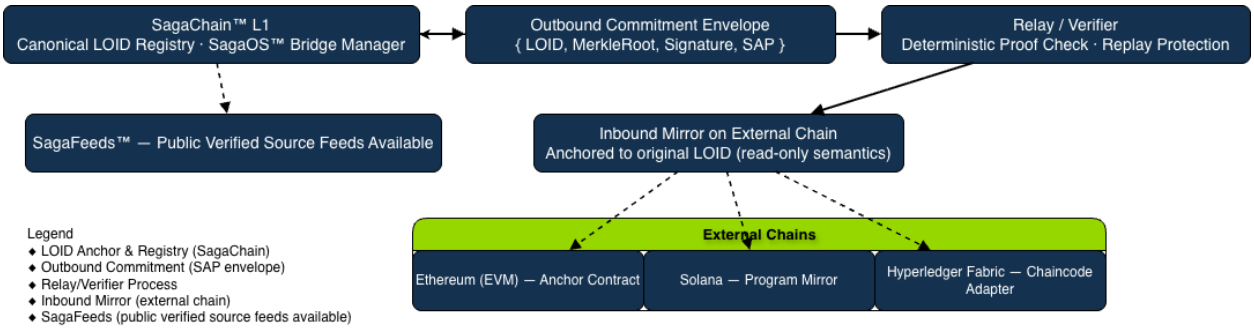


Figure 3-E — SagaInterop™ Architecture (Pharma Edition)

3.6.2 Standards Alignment

- ISO 20022 Business Application Header for message typing and jurisdictional routing.
- HL7 FHIR "Bundle/DocumentReference" templates for clinical or pharmacovigilance data payloads.
- W3C Verifiable Credential signatures for off-chain attestations.
- GS1 Digital Link URI for physical-to-digital serialization.

3.6.3 Benefits

- Enables **regulated interoperability** between public token networks and permissioned data enclaves.
- Preserves **LOID traceability** across jurisdictions, ensuring object lineage even when mirrored on external blockchains.
- Provides **common semantic interoperability** across industries (finance, pharma, ESG) using SagaStandards™.
- Reduces reconciliation latency by embedding cross-chain confirmations directly into the SagaChain transaction log.

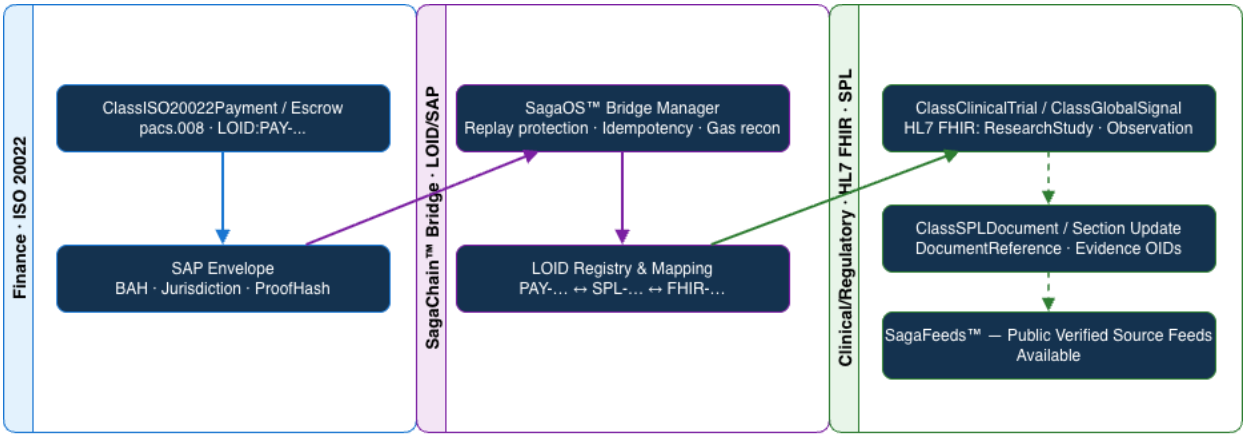


Figure 3-F — Cross-Domain Data Handshake (ISO 20022 · HL7 FHIR · SPL)

Handshake: ISO 20022 payment proof (SAP) maps via LOID to FHIR/SPL evidence; public visibility via SagaFeeds (verified source feeds available).

3.6.4 Example Usage

A WHO-affiliated account posts a pharmacovigilance signal to an EU EMA ledger and an ISO 20022 financial escrow chain simultaneously.

The attestation payload references the same SagaPSA ClassGlobalSignal LOID, ensuring every jurisdiction shares a provably identical event record.

4. Persistent-State Governance, Privacy, and Enclave Computation

SagaChain effectuates persistent-state governance, privacy-by-design, and enclave computation to underpin pharmaceutical compliance at global scale. This framework supplants document-centric oversight with executable classes under SagaStandards, augmented by confidential compute that safeguards privacy while furnishing verifiable accountability.

4.1 Persistent-State Governance: From Documents to Executable Classes

Conventional pharmaceutical governance hinges on documents—PDF dossiers, submissions, validation reports, and adverse event compendia—which prove retrospective, curated manually, and decoupled from operations.

SagaChain supplants this with persistent-state classes governed via SagaStandards. Each standard or regulatory mandate manifests as an @SagaClass definition, stewarded by SDOs, regulators, and industry in a multi-stakeholder framework.

Governance Features:

· Canonical Namespace: Classes (e.g., FDA SPL, EMA IDMP, GS1 EPCIS, USP assay) reside uniquely in the GCR.

· Version Lineage: Updates engender subclasses (_v2, _v3) inheriting from antecedents, preserving compatibility.

· Change Proposals: Stakeholders tender schema updates as class diffs; adoption follows consensus and voting.

· Compliance Hooks: Policy logic affixes to @SagaMethod invocations (e.g., release(), recall(), settle()), embedding regulatory validations inline.

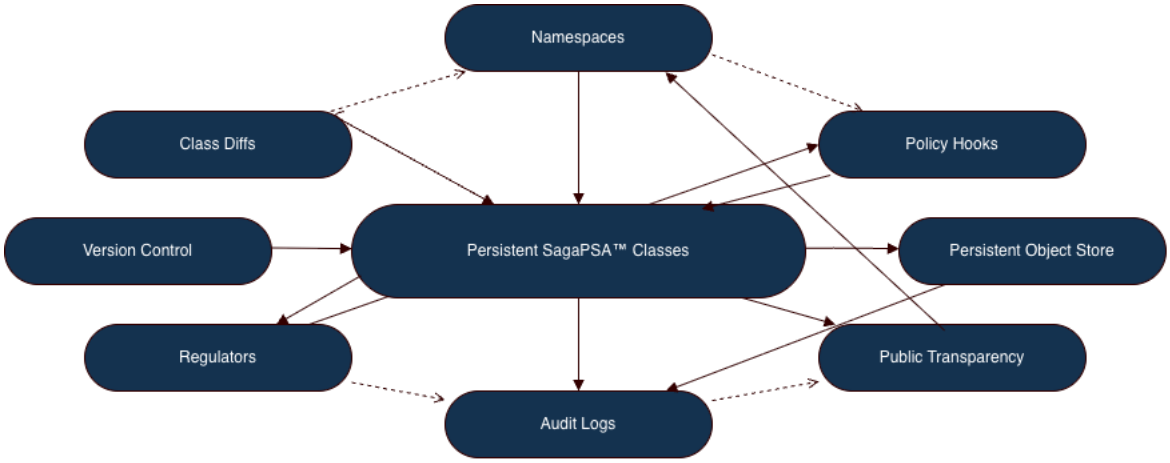


Diagram 4-A — Persistent-State Governance Model — Class Diffs • Namespaces • Policy Hooks • Persistent SagaPSA™ Classes • Regulators

4.2 Privacy-by-Design and Confidential Computation

Conventional blockchains render data public, antithetical to pharmaceutical exigencies involving patient health information (PHI), genomic datasets, proprietary telemetry, and contracts. Regulators and payers, however, demand compliance proofs.

SagaChain addresses this via privacy-by-design, where enclave owners can run SagaNode on TEE-based equipment providing hardware and runtime-level assurances.

Key Features:

- Encrypted-in-Use: Sensitive inputs process encrypted, countering insider and external threats.
- Deterministic Compute: Enclaves execute algorithms (e.g., safety signal detection, potency assays) yielding reproducible outputs for regulatory fidelity.
- Selective Disclosure: Enclaves emit proofs, aggregates, or attestations (e.g., “batch potency verified against USP standard”), shielding raw data.
- Audit Compatibility: Outputs bear cryptographic signatures, permitting regulator audits sans raw access.
- zK Integration: Enclave attestations augment with zero-knowledge proofs for jurisdiction-agnostic verification.

This TEE-zk hybrid yields a compliance paradigm that is privacy-preserving yet globally auditable.

4.3 Example Enclave Workflow in SagaPython

SagaPython expresses confidential workflows natively via CMI decorators. A pharmacovigilance workflow exemplifies:

```python
from CMIConst import SPClassObject


@SagaClass(SPClassObject)
class ClassPHIRecord:
    SagaField("record_id", "str")
    SagaField("patient_id", "str")
    SagaField("encrypted_payload", "bytes") # HL7 FHIR data encrypted
    SagaField("enclave_proof", "str", optional=True)


    @SagaMethod()
    def __init__(self, record_id: str, patient_id: str, encrypted_payload: bytes):
        self._cmi().__init__() # Chain to SPClassObject
        self.record_id = record_id
        self.patient_id = patient_id
        self.encrypted_payload = encrypted_payload
        self.enclave_proof = ""
        self.docs = []
        self._enclave = {}


@SagaClass(SPClassObject)
class ClassEnclaveAdverseEventAnalysis:
    SagaField("analysis_id", "str")
    SagaField("input_records", "list") # References to ClassPHIRecord
    SagaField("medDRA_code", "str", optional=True)
    SagaField("risk_score", "float", optional=True)
    SagaField("enclave_attestation", "str", optional=True)


    @SagaMethod()
    def __init__(self, analysis_id: str, input_records: list):
        self._cmi().__init__() # Chain to SPClassObject
        self.analysis_id = analysis_id
        self.input_records = input_records
        self.medDRA_code = ""
        self.risk_score = 0.0
        self.enclave_attestation = ""
        self.docs = []
        self._enclave = {}


    @SagaMethod()
    def detect_signal(self):
```

```
    # Executes AI/ML in enclave; no raw PHI egresses
    self.medDRA_code = "10012345"  # MedDRA code
    self.risk_score = 0.87  # Probability
    self.enclave_attestation = "attest:sha256:..."
    return self.risk_score
```

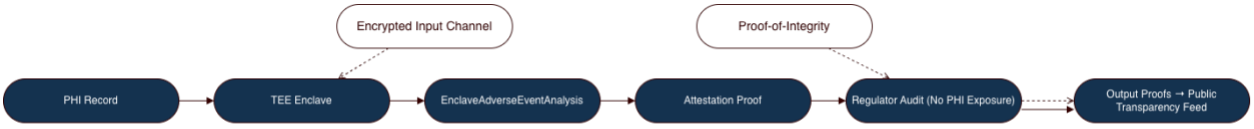


Diagram 4-B — Enclave Compute Workflow — Privacy-by-Design / TEE Attestation Flow

Implications:

- Regulators (FDA, EMA, WHO): Standardized signals with lineage, absent raw PHI.
- Providers: Reporting sans patient exposure.
- Patients: HIPAA/GDPR safeguards amid safety contributions.
- Industry: Proprietary models execute privately, preserving IP.

4.4 Governance Alignment with Global Standards

Governance underpins adoption. SagaChain anchors this in SagaStandards, encompassing industry, regulators, SDOs, and agencies.

4.4.1 Alignment Mechanisms:

1. **Namespace Stewardship:**
   - FDA curates SPL classes.
   - EMA oversees IDMP namespaces.
   - WHO manages pharmacovigilance and treaty classes.
2. **Participatory Model:**
   - Diff-based proposals for classes or changes.
   - Transparent workflows prioritize additivity; disruptions version subclasses.
3. **Executable Specifications:**
   - SDOs publish classes as live GCR entries, supplanting PDFs.
   - USP monographs enforce as @SagaMethod compliance.
4. **Continuous Oversight:**
   - Regulators subscribe to SagaFeeds™ for assurance (e.g., MedDRA events in prior 7 days).

This elevates governance from periodic, document-bound submissions to perpetual, shared code stewardship.

4.5 Patient, Provider, and Payer Trust

Trust spans stakeholders via enclave confidentiality and transparent indices:

- Patients: PHI privacy with transparency into safety, labeling, allocation
  - Example: GS1 Digital Link scan verifies SPL,

leaflets, cold-chain proofs sans identity revelation.

- **Providers:** Synchronized SPL, FAERS/EudraVigilance, dosing in EHR (HL7 FHIR), error mitigation.
- **Payers & Donors:** Funds release on verified proofs (e.g., GAVI confirms LMIC delivery via SagaFeeds™).
- **Regulators:** Cryptographic proofs and indices supplant delayed reports; enclaves assure computations.
- **This model triangulates trust:** patient empowerment bottom-up, regulator assurance top-down.

4.6 Implications for Global Adoption

SagaChain's governance and privacy extend ecosystem-wide:

- Industry: Dossier resubmissions yield to persistent objects; quality, pharmacovigilance, reimbursement streamline.
- Regulators: Episodic audits evolve to continuous attestations.
- SDOs: Standards as executable ontologies (GS1, ISO, CDISC, HL7, USP, WHO).
- Global Health Equity: LMICs access dashboards and proofs equitably, sans proprietary costs; donor programs verifiable.
- Finance & Sustainability: ESG, risk pools, ISO 20022 settlements tie to proofs, rendering accountability programmable.

4.7 Governance Workflows in SagaStandards

SagaChain operationalizes governance as continuous workflow under SagaStandards, transcending static standardization.

Workflow Model:

1. **Proposal Submission**
   - Stakeholders propose definitions or modifications (e.g., ClassClinicalTrial_v2 extending CDISC SDTM v1.4) as GCR diffs.
2. **Stakeholder Review**
   - Domain stewards (FDA for SPL, EMA for IDMP, GS1 for serialization, WHO for health) scrutinize publicly.
3. **Consensus & Ratification**
   - Additive evolution defaults; disruptions version with hooks (ClassDrugProduct_v3).
4. **Activation & Monitoring**
   - Ratified classes activate in GCR; metrics and attestations auto-track.
5. **Sunset & Deprecation**
   - Legacy classes persist for audits, marked deprecated; migrations guide adoption.

Implications:

- **Industry:** Transparent versioning curtails forks, interoperability failures.
- **Regulators:** Direct class oversight enforces boundary compliance.
- **SDOs:** Standards as live governance objects.

4.8 Regulatory Auditing via SagaFeeds™

Audits shift from manual dossiers to continuous SagaFeeds™ automation.

Core Principles:

- **Reads:** Listeners can be configured for object inquirer has authorization to access.
- Append-Only Transparency: Immutable OID lists for lineage.
- Composable Feeds: Dynamic aggregates (e.g., "SPL updates for IDMP product X, 90 days").
- No PHI Leakage: References and proofs only; payloads enclave-bound.

Use Cases:

- Pharmacovigilance: Feeds of adverse events by IDMP.
- Supply Chain: EPCIS events for DSCSA diversion detection.
- Financial: ISO 20022 disbursements on delivery proofs.
- Labeling: Joint FDA/EMA SPL feeds for synchronization.

Implications:

- Real-Time: Live proofs supplant annual submissions.
- Audit Trails: State reconstruction at any epoch.
- Equity: LMIC parity in capabilities.

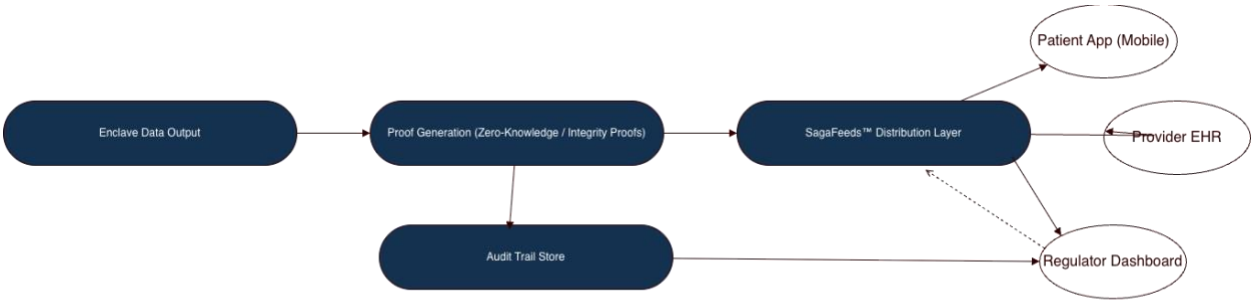


Diagram 4-C — SagaFeeds™ Transparency Loop — Enclave Data → Proof → SagaFeeds™ → Patients, Providers, Regulators

4.9 Patient and Provider Interfaces

Trust engages patients and providers via SagaFeeds™-based interfaces.

Patient Interfaces:

- GS1 Digital Link Scans: QR resolution to SPL, leaflets, proofs.
- Safety Alerts: Real-time signals; batch recall checks.
- Equity Dashboards: Delivery proofs for LMIC reassurance.

Provider Interfaces:

- EHR Integration (HL7 FHIR): SPL, USP guidelines, pharmacogenomics in workflows.
- Quality Verification: GMP proofs, assays, serialization pre-administration.
- Pharmacovigilance: Enclave-protected reporting to attested signals.

Implications:

- Patients: Visibility, agency beyond institutions.
- Providers: Point-of-care compliance, outcome enhancement.
- Public Health: Feedback loops among patients, providers, regulators.

Section 4 Summary:

SagaChain redefines pharmaceutical governance and privacy. SagaStandards stewardship, enclave computation, and SagaFeeds™ transparency equilibrate confidentiality, accountability, and interoperability, enabling shared trust fabric.

SagaStandards workflows govern class evolution; SagaFeeds™ enable real-time auditing; patient/provider interfaces operationalize trust. These forge episodic, paper-bound governance into continuous, transparent, participatory global health infrastructure.

5. Cross-Cutting Interoperability Scenarios

Scenario 1: End-to-End Biologics Lot Release

Background & Problem Statement

Biologics lot release must jointly satisfy **21 CFR 210/211**, **USP** potency/sterility/endotoxin, **DSCSA** serialization, **GS1 EPCIS** aggregation/shipping, **SPL** labeling, and **FAERS/VAERS** safety feedback. Today, evidence is scattered across QMS/LIMS/MES/ERP, creating manual reconciliation, latency, and audit risk.

Technical Approach (SagaChain Implementation)

Represent the release as a **single persistent object** that binds GMP manufacturing state, compendial quality conformance, serialization & EPCIS lineage, labeling, and live pharmacovigilance signals:

- ClassGMPBatch (21 CFR) — provenance, deviations, release readiness.
- ClassUSPStandard / ClassUSPComplianceBindingReg — assay/sterility/pass–fail bindings.
- ClassGS1SGTIN / ClassDSCSAUnit + ClassEPCISEvent — instance identity and event pedigree.
- ClassSPLDocument — labeling alignment with identifiers.
- ClassFAERSReport / ClassVaccineAdverseEvent — post-market safety linkage.
- Optional ISO 20022 payment objects — **conditional settlement** upon verified release.

Sample SagaPython Code

```
@sagaclass()
class ClassLotRelease(SPClassObject):
    """End-to-end biologics lot release record."""
    lot_release_id        = sagafield(type="str")
    batch_ref             = sagafield(type="str")            #
-> ClassGMPBatch
    usp_binding_refs      = sagafield(type="list[str]")
#       ->       ClassUSPComplianceBindingReg       /
ClassUSPStandard
    serialized_unit_refs  = sagafield(type="list[str]")
# -> ClassGS1SGTIN / ClassDSCSAUnit
    epcis_event_refs      = sagafield(type="list[str]")
# -> ClassEPCISEvent
```

```
    spl_label_ref                = sagafield(type="str", default="")    # -> ClassSPLDocument
    safety_refs        = sagafield(type="list[str]")       # -> ClassFAERSReport / ClassVaccineAdverseEvent
    iso20022_payment_ref      = sagafield(type="str", default="")    # -> ClassISO20022Payment (optional)
    release_date             = sagafield(type="datetime", default=None)
    released_by              = sagafield(type="str", default="")
    status                   = sagafield(type="str", default="pending",

enum={"pending","released","recalled","blocked"})
    notes                = sagafield(type="str", default="")
    feeds_tags         = sagafield(type="list[str]")       # SagaFeeds discovery (non-sensitive)

    @sagamethod()
    def link_batch(self, batch_oid: str):
        self.batch_ref = batch_oid
        return {"batch_ref": self.batch_ref}

    @sagamethod()
    def add_usp_binding(self, binding_oid: str):
        if binding_oid not in self.usp_binding_refs:
            self.usp_binding_refs.append(binding_oid)
        return                          {"usp_bindings": len(self.usp_binding_refs)}

    @sagamethod()
    def add_serialized_units(self, unit_oids: list[str]):
        for u in unit_oids:
            if u not in self.serialized_unit_refs:
                self.serialized_unit_refs.append(u)
        return                          {"serialized_units": len(self.serialized_unit_refs)}

    @sagamethod()
    def add_epcis_events(self, event_oids: list[str]):
        for e in event_oids:
            if e not in self.epcis_event_refs:
                self.epcis_event_refs.append(e)
        return                          {"epcis_events": len(self.epcis_event_refs)}

    @sagamethod()
    def set_spl_label(self, spl_oid: str):
        self.spl_label_ref = spl_oid
        return {"spl_label_ref": self.spl_label_ref}

    @sagamethod()
    def link_safety_refs(self, safety_oids: list[str]):
        for s in safety_oids:
            if s not in self.safety_refs:
                self.safety_refs.append(s)
        return {"safety_refs": len(self.safety_refs)}

    @sagamethod()
    def link_payment(self, payment_oid: str = ""):
        self.iso20022_payment_ref = payment_oid
        return                          {"iso20022_payment_ref": self.iso20022_payment_ref}

    @sagamethod()
    def evaluate_and_release(self, releaser_oid: str, note: str = "") -> dict:
        """
        Gate release when:
          - GMP batch is 'ReadyForRelease'
          - All USP compliance bindings report status=='meets'
          - Serialized units present and EPCIS pedigree includes required steps
          - SPL label linked
        Optionally triggers ISO 20022 payment conditional release.
        """
        # Illustrative checks (real logic dereferences and validates each OID)
        gmp_ok          = bool(self.batch_ref) and (deref(self.batch_ref).status                in {"ReadyForRelease","ReleasedCandidate"})
        usp_ok          = self.usp_binding_refs and all(deref(b).status == "meets" for b in self.usp_binding_refs)
        ser_ok  = len(self.serialized_unit_refs) > 0
        epc_ok  = len(self.epcis_event_refs) > 0
        spl_ok  = bool(self.spl_label_ref)

        all_ok  = gmp_ok and usp_ok and ser_ok and epc_ok and spl_ok
        if all_ok:
            self.status = "released"
            self.release_date = now()
            self.released_by = releaser_oid
            self.notes = note
            # Optional: trigger conditional payment release if configured
            if self.iso20022_payment_ref:
                pay = deref(self.iso20022_payment_ref)

pay.link_supply_artifacts(self.serialized_unit_refs, self.epcis_event_refs, [])

pay.link_quality_identity(self.usp_binding_refs, "")  # idmp ref could be added here

pay.add_attestation(f"lot_release::{self.lot_release_id}")
                pay.conditional_release()
        else:
```

```
        self.status = "blocked"
        self.notes = note or "Release checks not satisfied"

    return {
        "released": self.status == "released",
        "status": self.status,
        "checks": {
            "gmp": gmp_ok, "usp": usp_ok, "serialization": ser_ok,
            "epcis": epc_ok, "spl": spl_ok
        },
        "release_date": str(self.release_date) if self.release_date else None
    }

@sagamethod()
def recall(self, reason: str):
    self.status = "recalled"
    self.notes = reason
    # downstream: emit notifications, flag DSCSA units, publish feeds
    return {"status": self.status, "reason": self.notes}
```

Interoperability Analysis

- **21 CFR ↔ USP:** ClassGMPBatch readiness plus **USP bindings == "meets"** gates release.
- **USP ↔ DSCSA/GS1:** Only after quality passes are **SGTINs**/ClassDSCSAUnit commissioned; **EPCIS** events provide cryptographic pedigree.
- **SPL ↔ DSCSA:** Label content (ClassSPLDocument) references identifiers used in packaging.
- **FAERS/VAERS ↔ Lot:** Safety reports link back to the **released lot**, enabling rapid signal investigation and targeted recalls.
- **ISO 20022 ↔ Lot:** Optional ClassISO20022Payment.conditional_release() ties cash movement to verified release state and supply/quality attestations.
- **SagaFeeds:** Non-sensitive release metadata (lot id, release timestamp, status) can be published for regulator/auditor visibility at zero cost.

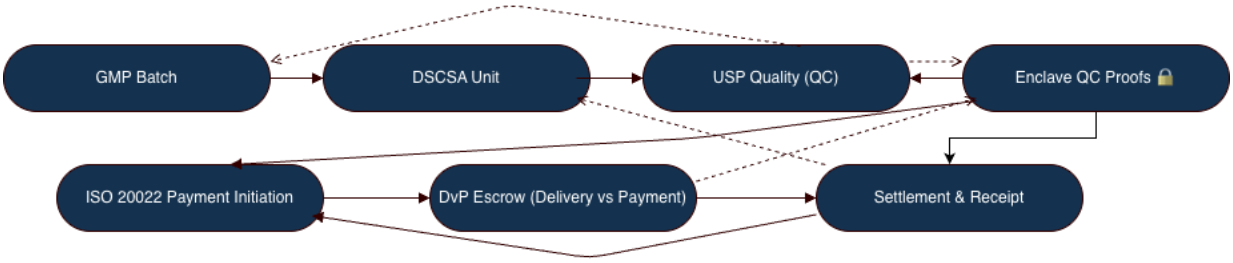


Diagram 5-A — Manufacturing Continuum (v2) GMP → DSCSA → USP → ISO 20022 DvP Payment

Solid = primary flow; Dashed = feedback/notifications/compliance.
Manufacturing: GMP → DSCSA → USP; Enclave 🔒 for QC proofs.
Payments: ISO 20022 Initiation → DvP Escrow → Settlement ↺; Settlement → DSCSA receipt; USP QC → Payment trigger.

This scenario demonstrates **how the class tree unifies regulatory, standards, and operational compliance into a single persistent object model**.

Scenario 2: Global Recall with Synchronized Finance

Background & Problem Statement

Global recalls require tight coordination across regulators, manufacturers/distributors,

providers/pharmacies, patients, and financial counterparties. Today's **text-based notices**, **slow unit verification**, **manual refunds/chargebacks**, and **cross-border identifier variance** (NDC, UDI, IDMP, GS1) cause delay, risk, and cost.

Technical Approach (SagaChain Implementation)

- **ClassRecallEvent** - authoritative, persistent recall object with scope (units/lots), jurisdictions, and root-cause linkage.
- **Direct bindings** - to **DSCSA/GS1** units (SGTIN/SSCC), **SPL** labeling updates, **FAERS/VAERS** signals, **IDMP** product identities.
- **Patient/provider UX - GS1 Digital Link** lookups resolve to recall status instantly.
- **Finance sync - ISO 20022** settlements/refunds/chargebacks auto-trigger based on verified returns/disposals.
- **ISPE/QRM** - automated CAPA feedback loops for structured root cause and re-verification.
- **SagaFeeds** - public indices for recall status, scope, and jurisdictional notices.

Sample SagaPython Code

```
@sagaclass()
class ClassRecallEvent(SPClassObject):
   """Global recall event across jurisdictions with synchronized finance."""
   recall_id          = sagafield(type="str")
   authority          = sagafield(type="str")          # e.g., "FDA", "EMA", "WHO"
   jurisdictions      = sagafield(type="list[str]")    # ["US","EU","CA", ...]
   reason             = sagafield(type="str")          # contamination, mislabeling, subpotency, etc.
   severity           = sagafield(type="str", enum={"Class I","Class II","Class III"})
   idmp_mp_refs       = sagafield(type="list[str]") # -> ClassISO11615MedicinalProduct
   lot_ids            = sagafield(type="list[str]")    # -> ClassGMPBatch or textual lot codes
   scope_unit_refs    = sagafield(type="list[str]") # -> ClassDSCSAUnit / ClassGS1SGTIN
   epcis_event_refs   = sagafield(type="list[str]") # -> ClassEPCISEvent (shipping/receiving/reverse)
   spl_label_refs     = sagafield(type="list[str]")    # -> ClassSPLDocument (recall/contraindications update)
   safety_signal_refs = sagafield(type="list[str]") # -> ClassFAERSReport / ClassVaccineAdverseEvent
   qrm_root_cause_ref = sagafield(type="str", default="")  # -> ClassQRMRecord
   start_date         = sagafield(type="datetime")
   status             = sagafield(type="str", default="open",

enum={"open","ongoing","closed"})
   linked_settlement_refs = sagafield(type="list[str]") # -> ClassISO20022Settlement / Payment
   feeds_tags         = sagafield(type="list[str]")    # SagaFeeds searchable facets
   public_notice_uri  = sagafield(type="str", default="")   # canonical notice (can be on-chain pointer)

   @sagamethod()
   def bind_scope(self, unit_oids: list[str], lot_ids: list[str] = None, idmp_oids: list[str] = None):
      """Attach affected units/lots and global product identities."""
      for u in unit_oids or []:
         if u not in self.scope_unit_refs:
            self.scope_unit_refs.append(u)
      for l in lot_ids or []:
         if l not in self.lot_ids:
            self.lot_ids.append(l)
      for m in idmp_oids or []:
         if m not in self.idmp_mp_refs:
            self.idmp_mp_refs.append(m)
      return {"units": len(self.scope_unit_refs), "lots": len(self.lot_ids), "idmp": len(self.idmp_mp_refs)}

   @sagamethod()
   def add_links(self, epcis_oids: list[str] = None, spl_oids: list[str] = None, safety_oids: list[str] = None):
      """Cross-link EPCIS pedigree, SPL updates, and safety signals."""
      for e in epcis_oids or []:
         if e not in self.epcis_event_refs:
            self.epcis_event_refs.append(e)
      for s in spl_oids or []:
         if s not in self.spl_label_refs:
            self.spl_label_refs.append(s)
      for r in safety_oids or []:
```

```
            if r not in self.safety_signal_refs:
                self.safety_signal_refs.append(r)
        return {
            "epcis": len(self.epcis_event_refs),
            "spl": len(self.spl_label_refs),
            "safety": len(self.safety_signal_refs)
        }

    @sagamethod()
    def set_root_cause(self, qrm_oid: str):
        self.qrm_root_cause_ref = qrm_oid
        return {"qrm_root_cause_ref": self.qrm_root_cause_ref}

    @sagamethod()
    def publish_public_notice(self, uri: str):
        """Register a canonical public notice for SagaFeeds discovery."""
        self.public_notice_uri = uri
        # (Indexing to SagaFeeds happens via OS policy hooks)
        return {"public_notice_uri": self.public_notice_uri}

    @sagamethod()
    def attach_financial_sync(self, settlement_oids: list[str]):
        """Attach ISO 20022 payments/settlements for refunds/chargebacks/clawbacks."""
        for s in settlement_oids or []:
            if s not in self.linked_settlement_refs:
                self.linked_settlement_refs.append(s)
        return {"settlements": len(self.linked_settlement_refs)}

    @sagamethod()
    def begin(self):
        self.status = "ongoing"
        return {"status": self.status}

    @sagamethod()
    def close(self):
        self.status = "closed"
        return {"status": self.status}
```

Optional: Digital Link recall resolver (scan → recall verdict)

```
@sagaclass()
class ClassRecallResolver(SPClassObject):
    """Resolves a scanned GS1 Digital Link or SGTIN to recall status."""
    resolver_id          = sagafield(type="str")
    active_recall_refs   = sagafield(type="list[str]")  # -> ClassRecallEvent (status in {"open","ongoing"})
    last_query_ts        = sagafield(type="datetime", default=None)

    @sagamethod()
    def check(self, sgtin_or_uri: str) -> dict:
        """
        Given an SGTIN or GS1 Digital Link, return recall status and matching events.
        (Illustrative; real logic would deref DigitalLink -> SGTIN -> match against scopes.)
        """
        self.last_query_ts = now()
        matches = []
        for r in self.active_recall_refs:
            recall = deref(r)
            if any(sgtin_or_uri.endswith(deref(u).sgtin) for u in recall.scope_unit_refs):
                matches.append({"recall_id": recall.recall_id, "severity": recall.severity, "status": recall.status})
        return {"queried": sgtin_or_uri, "matches": matches, "ts": str(self.last_query_ts)}
```

Optional: Enclave for confidential clawback/settlement terms

```
@sagaclass(enclave=True)
class ClassRecallFinanceEnclave(SPClassObject):
    """
    Confidential rebate/chargeback formulas, insurer terms, FX handling.
    Emits only net refund/chargeback amounts with attestations.
    """
    recall_ref          = sagafield(type="str")          # -> ClassRecallEvent
    policy_payload      = sagafield(type="bytes")        # encrypted terms
    attestations        = sagafield(type="list[str]")

    @sagamethod()
    def attest_net_refund(self, payment_oid: str) -> dict:
        att = "TEE:recall:refund:" + rand_str(10)
        self.attestations.append(att)
        # Example net calculation performed in-enclave:
        net_amt = 125000.00
        pay = deref(payment_oid)
        pay.add_attestation(att)
        return {"payment_ref": payment_oid, "net_refund": net_amt, "attestation": att}
```

Interoperability Analysis

- **DSCSA / GS1:** scope_unit_refs contain **serialized units (SGTIN/SSCC)**;

EPCIS links validate custody and enable precise reverse logistics.

- **ISO IDMP:** idmp_mp_refs harmonize across regulators; one global product identity underpins multi-jurisdiction notices.
- **USP:** Root cause can cite **compendial failures**; recall scope constrained by specific monographs/lots.
- **SPL:** spl_label_refs carry recall notices/contraindications; bedside scanning via **Digital Link** shows live recall state.
- **FAERS/VAERS: Safety signals** can **trigger** recalls; post-recall events remain linked for effectiveness checks.
- **ISO 20022:** linked_settlement_refs drive **refunds, chargebacks, clawbacks** in parallel with product retrieval; **enclave** protects sensitive terms while exposing attested net amounts.
- **ISPE/QRM:** qrm_root_cause_ref ties to **CAPA**, re-qualification, and targeted re-verification.
- **SagaFeeds:** Public, indices improve patient/provider discoverability and regulatory oversight.

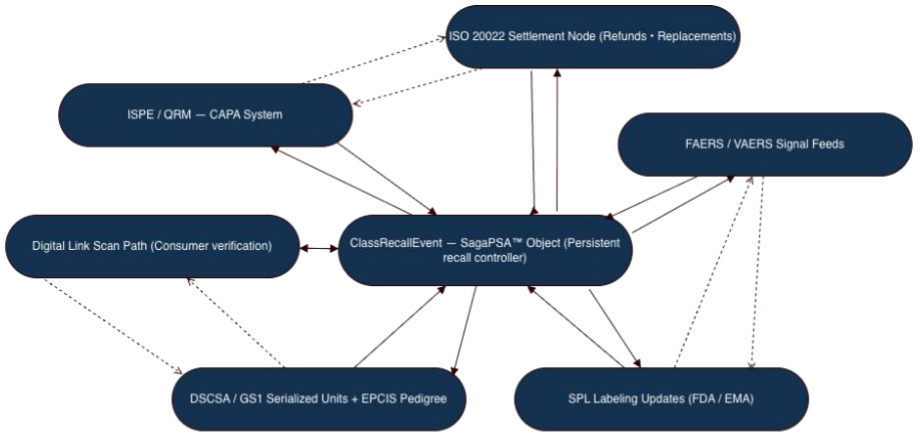


Diagram 5-B — Global Recall ClassRecallEvent ↔ DSCSA/GS1 units & EPCIS; SPL updates; FAERS/VAERS signals; ISO 20022 settlements; ISPE/QRM CAPA; Digital Link scan path

Global Recall: a central ClassRecallEvent coordinates DSCSA/GS1 units and EPCIS pedigrees, SPL labeling updates, FAERS/VAERS safety signals, ISO 20022 settlements, ISPE/QRM CAPA actions, and Digital Link consumer verification. Inbound and outbound flows ensure complete traceability and recall transparency.

Scenario 2 shows how the Global Pharma Class Tree unifies **regulatory enforcement, supply chain visibility, patient safety, and financial remediation** into a single programmable recall process.

Scenario 3: Trial-to-Label Evidence Pipeline

Background & Problem Statement

Clinical evidence must flow from **FDAAA 801** registration and **CDISC SDTM/ADaM** datasets through **HL7** clinical outcomes into **SPL** labeling (clinical studies section) under **FD&C Act**. Today's fragmentation (separate systems, manual integration, delayed updates) weakens transparency and slows patient benefit.

Technical Approach (SagaChain Implementation)

Encode the pipeline as a **persistent, auditable object** that:

- Links **registration** (ClassClinicalTrial) to **datasets** (ClassCDISCStudy, ClassSDTMDomain, ClassADaMDataset),
- Binds **HL7 FHIR** outcome resources,
- Generates/updates **SPL clinical studies** sections,
- Enforces **provenance & policy** (publish only when validations pass),
- Exposes **public indices** (non-sensitive) via SagaFeeds,
- Optionally uses **Private Enclaves** for PHI-safe de-identification or stats.

Sample SagaPython Code

```
@sagaclass()
class ClassTrialToLabel(SPClassObject):
    """End-to-end evidence pipeline linking trial data to labeling."""
    pipeline_id          = sagafield(type="str")
    trial_ref            = sagafield(type="str")            # -> ClassClinicalTrial (FDAAA 801)
    cdisc_study_ref      = sagafield(type="str", default="")  # -> ClassCDISCStudy
    sdtm_domain_refs     = sagafield(type="list[str]")  # -> ClassSDTMDomain
    adam_dataset_refs    = sagafield(type="list[str]")  # -> ClassADaMDataset
    hl7_outcome_refs     = sagafield(type="list[str]")  # -> ClassFHIRResource (Observation/Condition)
    spl_section_ref      = sagafield(type="str", default="")  # -> ClassSPLSection (Clinical Studies)
    approval_date        = sagafield(type="date", default=None)
    status               = sagafield(type="str", default="pending",

enum={"pending","under_review","approved","published","rejected"})
    provenance_hashes    = sagafield(type="list[str]")  # content-addressed proofs (protocols, SAP, CSR)
    feeds_tags           = sagafield(type="list[str]")  # SagaFeeds facets (non-sensitive)

    @sagamethod()
    def link_trial(self, trial_oid: str):
        self.trial_ref = trial_oid
        self.status = "under_review"
        return {"trial_ref": self.trial_ref, "status": self.status}

    @sagamethod()
    def attach_cdisc(self, study_oid: str, sdtm_oids: list[str], adam_oids: list[str]):
        self.cdisc_study_ref = study_oid or self.cdisc_study_ref
        for x in sdtm_oids or []:
            if x not in self.sdtm_domain_refs: self.sdtm_domain_refs.append(x)
        for y in adam_oids or []:
            if y not in self.adam_dataset_refs: self.adam_dataset_refs.append(y)
        return {
            "cdisc_study": self.cdisc_study_ref,
            "sdtm_count": len(self.sdtm_domain_refs),
            "adam_count": len(self.adam_dataset_refs)
        }

    @sagamethod()
    def attach_hl7_outcomes(self, fhir_oids: list[str]):
        for z in fhir_oids or []:
            if z not in self.hl7_outcome_refs: self.hl7_outcome_refs.append(z)
        return {"hl7_outcomes": len(self.hl7_outcome_refs)}

    @sagamethod()
    def add_provenance(self, hashes: list[str]):
        for h in hashes or []:
            if h not in self.provenance_hashes: self.provenance_hashes.append(h)
        return {"provenance_hashes": len(self.provenance_hashes)}

    @sagamethod()
    def validate_evidence(self) -> dict:
        """
        Gatekeeping checks (illustrative):
        - Trial is registered (FDAAA) and within reporting timelines.
        - CDISC SDTM/ADaM present; required domains populated.
        - HL7 outcomes mapped to SDTM where applicable.
        - Provenance hashes (protocol, SAP, CSR) present.
        """
        trial_ok = bool(self.trial_ref) and deref(self.trial_ref).registration_date is not None
        cdisc_ok = bool(self.cdisc_study_ref) and len(self.sdtm_domain_refs) > 0
        adam_ok  = len(self.adam_dataset_refs) > 0
```

```
        hl7_ok   = len(self.hl7_outcome_refs) > 0
        prov_ok  = len(self.provenance_hashes) >= 2
        all_ok   = trial_ok and cdisc_ok and adam_ok and
hl7_ok and prov_ok
        self.status = "approved" if all_ok else
"under_review"
        return {"approved": all_ok, "checks": {"trial":
trial_ok, "sdtm": cdisc_ok,
                                "adam": adam_ok,
"hl7": hl7_ok, "prov": prov_ok},
                "status": self.status}

    @sagamethod()
    def generate_or_update_spl_section(self,
spl_section_oid: str = "") -> dict:
        """
        Write the Clinical Studies section from validated
CDISC/HL7 sources.
        Produces structured content with citations and
dataset OIDs.
        """
        assert self.status in {"approved","published"}
        if spl_section_oid:
            self.spl_section_ref = spl_section_oid
        else:
            # In practice, would instantiate a new
ClassSPLSection here.
            self.spl_section_ref = self.spl_section_ref or
"OID:SPL:SECTION:CLINICAL_STUDIES"
        # Populate content via policy hooks/transforms
(not shown)
        return {"spl_section_ref": self.spl_section_ref}

    @sagamethod()
    def publish_label_update(self, approval_date: str):
        """
        Final publish under FD&C Act: set
approval/publish state and emit
        indexable metadata for transparency
(SagaFeeds), without PHI.
        """
        assert self.spl_section_ref
        self.approval_date = parse_date(approval_date)
        self.status = "published"
        return {"status": self.status, "approval_date":
str(self.approval_date), "spl_section_ref":
self.spl_section_ref}
```

Optional: PHI-safe analytics & de-identification in a Private Enclave

```
@sagaclass(enclave=True)
class
ClassTrialPHIAnalyticsEnclave(SPClassObject):
    """
    PHI-bearing HL7/FHIR records are analyzed in-
enclave; only de-identified
    aggregates and statistical proofs exit for
CDISC/SPL use.
    """
    pipeline_ref        = sagafield(type="str")         #
-> ClassTrialToLabel
    encrypted_fhir_blobs                                =
sagafield(type="list[bytes]")
    deid_stats          = sagafield(type="dict")
    attestations        = sagafield(type="list[str]")

    @sagamethod()
    def compute_deid_stats(self) -> dict:
        att = "TEE:trial:deid:" + rand_str(10)
        self.attestations.append(att)
        # Example output; real compute done in-enclave
        self.deid_stats = {"n": 1243, "ae_rate": 0.031,
"endpoint_effect": {"OR": 1.42, "CI95": [1.18,
1.70]}}
        return {"attestation": att, "stats": self.deid_stats}
```

Interoperability Analysis

- **FDAAA 801 ↔ CDISC:** Registration and **SDTM/ADaM** datasets stay persistently linked for timely results reporting.
- **CDISC ↔ HL7: FHIR** Observations/Conditions map into **SDTM** domains for coherent clinical+research semantics.
- **CDISC ↔ SPL:** The **Clinical Studies** SPL section is generated from validated datasets with cryptographic provenance.
- **SPL ↔ FD&C Act:** Publication sets a durable audit trail of **evidence → submission → label**.
- **FAERS / Post-market:** Safety signals link back to the **trial evidence** for lifecycle coherence.
- **NIH DMSP / Open Science:** Publicly shareable dataset metadata (non-PHI) is discoverable via **SagaFeeds**; PHI remains enclave-protected.

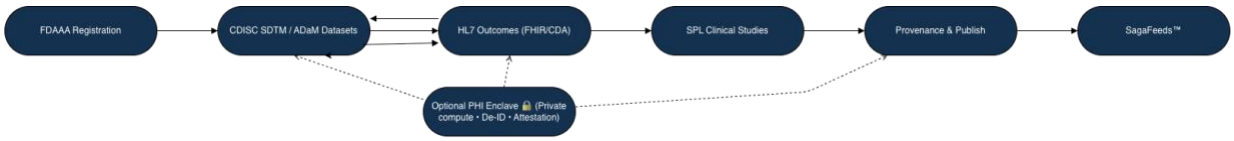


Diagram 5-H — Trial-to-Label Pipeline FDAAA Registration → CDISC SDTM/ADaM ↔ HL7 Outcomes → SPL Clinical Studies; Provenance & Publish; SagaFeeds; Optional PHI Enclave

Trial-to-Label pipeline: FDAAA registration feeds CDISC SDTM/ADaM; outcomes reported via HL7 (FHIR/CDA); labeling captured in SPL Clinical Studies with provenance, publication, and SagaFeeds distribution. Optional PHI Enclave 🔒 supports de-identification and private computation with attestations.

Outcome: The evidence chain becomes **programmatically linked and auditable**. Labels reflect **current, validated trial data**, with privacy-preserving analytics, public transparency where appropriate, and end-to-end compliance by design.

Scenario 4: Patient-Centric Pharmacovigilance & Recall Prevention

Background & Problem Statement

Legacy pharmacovigilance (FAERS/VAERS) is retrospective, latency-prone, and weakly linked to **EHR (HL7 FHIR)** and **serialized supply (DSCSA/GS1)**. Patients lack unit-specific, real-time safety information; financial remediation (refunds/chargebacks) is disconnected from safety actions.

Technical Approach (SagaChain Implementation)

- **Event layer:** Structured AE reports linked to **specific serialized units** (SGTIN/lot) and **FHIR** clinical observations, wrapped by **HIPAA** PHI controls.
- **Signal layer:** Continuous aggregation and statistical thresholds trigger pre-defined actions (label update, recall, outreach).
- **Action layer: Recall** objects, **ISO 20022** settlements, and **ISPE/QRM** CAPA are orchestrated automatically.
- **Transparency: SagaFeeds** publishes non-sensitive safety indices; PHI remains protected (optionally within **Private Enclaves**).

Sample SagaPython Code

1) Patient-centric AE record (HIPAA + HL7 + DSCSA/GS1)

```
@sagaclass()
class ClassPharmacovigilanceEvent(SPClassObject):
    """Patient-centric pharmacovigilance event linked to serialized product and HL7 outcomes."""
    event_id            = sagafield(type="str")
    report_source       = sagafield(type="str", enum={"patient","provider","manufacturer","regulator"})
    phi_ref             = sagafield(type="str", default="") # -> ClassPHI (HIPAA)
    fhir_refs           = sagafield(type="list[str]")       # -> ClassFHIRResource (Observation/Condition)
    dscsa_unit_ref      = sagafield(type="str", default="") # -> ClassDSCSAUnit
    sgtin_ref           = sagafield(type="str", default="") # -> ClassGS1SGTIN
    lot_number          = sagafield(type="str", default="")
    idmp_mp_ref         = sagafield(type="str", default="") # -> ClassISO11615MedicinalProduct
    adverse_outcome     = sagafield(type="str") # coded term or narrative
    seriousness         = sagafield(type="str", enum={"non-serious","serious","life-threatening","death"})
    onset_date          = sagafield(type="date", default=None)
    report_date         = sagafield(type="date", default=None)
    related_factors     = sagafield(type="list[str]") # concomitant meds, comorbidities, etc.
    attachments_hashes  = sagafield(type="list[str]") # lab PDFs, images (content-addressed)
```

```
    faers_ref           = sagafield(type="str", default="")
# -> ClassFAERSReport (if mirrored)
    vaers_ref           = sagafield(type="str", default="")
# -> ClassVaccineAdverseEvent
    recall_ref          = sagafield(type="str", default="")
# -> ClassRecallEvent (if triggered/linked)
    feeds_tags                  = sagafield(type="list[str]")
# public index facets (non-PHI)

    @sagamethod()
    def link_phi(self, phi_oid: str):
        self.phi_ref = phi_oid
        return {"phi_ref": self.phi_ref}

    @sagamethod()
    def link_clinical(self, fhir_oids: list[str]):
        for oid in fhir_oids or []:
            if oid not in self.fhir_refs:
self.fhir_refs.append(oid)
        return {"fhir_refs": len(self.fhir_refs)}

    @sagamethod()
    def link_product(self, dscsa_oid: str = "", sgtin_oid:
str = "", lot: str = "", idmp_mp_oid: str = ""):
        self.dscsa_unit_ref = dscsa_oid or
self.dscsa_unit_ref
        self.sgtin_ref = sgtin_oid or self.sgtin_ref
        self.lot_number = lot or self.lot_number
        self.idmp_mp_ref = idmp_mp_oid or
self.idmp_mp_ref
        return {"dscsa": self.dscsa_unit_ref, "sgtin":
self.sgtin_ref, "lot": self.lot_number, "idmp":
self.idmp_mp_ref}

    @sagamethod()
    def mirror_to_registries(self, faers_oid: str = "",
vaers_oid: str = ""):
        self.faers_ref = faers_oid or self.faers_ref
        self.vaers_ref = vaers_oid or self.vaers_ref
        return {"faers_ref": self.faers_ref, "vaers_ref":
self.vaers_ref}
```

2) Real-time signal detection & automated actions

```
@sagaclass()
class ClassSafetySignal(SPClassObject):
    """Aggregated safety signal for a product/lot with
action automation."""
    signal_id           = sagafield(type="str")
    idmp_mp_ref                 = sagafield(type="str",
default="")    # -> ClassISO11615MedicinalProduct
    lot_number                  = sagafield(type="str",
default="")
    window_days                 = sagafield(type="int",
default=30)
    ae_count            = sagafield(type="int", default=0)
    expected_baseline           = sagafield(type="float",
default=0.0)   # incidence per window
    zscore                      = sagafield(type="float",
default=0.0)
    threshold_crossed           = sagafield(type="bool",
default=False)
    triggered_actions       = sagafield(type="list[str]")
# ["label_update","recall","provider_alert"]
    linked_recall_ref           = sagafield(type="str",
default="")     # -> ClassRecallEvent
    linked_payment_refs    = sagafield(type="list[str]")
# -> ClassISO20022Payment/Settlement
    feeds_tags          = sagafield(type="list[str]")

    @sagamethod()
    def observe_event(self, pv_event_oid: str):
        """Increment counts when a qualifying PV event
arrives (policy filters apply OS-side)."""
        self.ae_count += 1
        return {"ae_count": self.ae_count}

    @sagamethod()
    def evaluate(self, current_baseline: float, z: float,
actions: list[str] = None):
        self.expected_baseline = current_baseline
        self.zscore = z
        self.threshold_crossed = (z >= 3.0) or
(self.ae_count > 0 and self.ae_count >= 2 * max(1,
int(current_baseline)))
        if self.threshold_crossed and actions:
            for a in actions:
                if a not in self.triggered_actions:
self.triggered_actions.append(a)
        return                  {"threshold_crossed":
self.threshold_crossed,     "zscore":     self.zscore,
"actions": self.triggered_actions}

    @sagamethod()
    def attach_recall(self, recall_oid: str):
        self.linked_recall_ref = recall_oid
        if "recall" not in self.triggered_actions:
            self.triggered_actions.append("recall")
        return                  {"linked_recall_ref":
self.linked_recall_ref,                 "actions":
self.triggered_actions}

    @sagamethod()
    def attach_finance(self, iso20022_oids: list[str]):
        for oid in iso20022_oids or []:
            if oid not in self.linked_payment_refs:
self.linked_payment_refs.append(oid)
        if      "financial_settlement"      not      in
self.triggered_actions:

self.triggered_actions.append("financial_settlement")
```

```
    return {"linked_payments": len(self.linked_payment_refs), "actions": self.triggered_actions}
```

3) HITECH breach notifications when PHI risks are detected

```
@sagaclass()
class ClassPHIBreachMonitor(SPClassObject):
  """Detect suspicious PHI access and emit HITECH breach notifications."""
  monitor_id = sagafield(type="str")
  hipaa_phi_refs = sagafield(type="list[str]") # -> ClassPHI
  anomaly_count = sagafield(type="int", default=0)
  last_breach_oid = sagafield(type="str", default="")  # -> ClassBreachNotification

  @sagamethod()
  def record_anomaly(self, phi_oid: str, severity: str = "medium"):
    self.anomaly_count += 1
    if phi_oid not in self.hipaa_phi_refs: self.hipaa_phi_refs.append(phi_oid)
    if severity in {"high","critical"}:
      # create/report a HITECH breach record (simplified)
      breach = instantiate("ClassBreachNotification", {"breach_id": "AUTO-"+rand_str(8),

"discovery_date": str(now()),

"affected_records": 1,
                           "description": "Auto anomaly",

"mitigation_steps": "Lock & audit",

"notification_date": str(now())})
      self.last_breach_oid = breach.oid
    return {"anomalies": self.anomaly_count, "last_breach": self.last_breach_oid}
```

4) Orchestrator: prevention before recall (label updates, outreach)

```
@sagaclass()
class ClassPVPreventionOrchestrator(SPClassObject):
  """Coordinates label updates, provider alerts, and pre-recall mitigation."""
  orchestrator_id = sagafield(type="str")
  signal_ref = sagafield(type="str", default="") # -> ClassSafetySignal
  spl_section_ref = sagafield(type="str", default="") # -> ClassSPLSection (Warnings/Precautions)
  provider_alerts = sagafield(type="list[dict]") # [{"provider_gln": "...", "ts": "..."}]
  patient_outreach = sagafield(type="list[dict]") # {"channel":"DL/SMS","count": n}

  @sagamethod()
  def link_signal(self, signal_oid: str):
    self.signal_ref = signal_oid
    return {"signal_ref": self.signal_ref}

  @sagamethod()
  def update_label_warning(self, spl_section_oid: str):
    self.spl_section_ref = spl_section_oid
    # Policy hooks: write updated warning/contraindication text derived from the signal
    return {"spl_section_ref": self.spl_section_ref}

  @sagamethod()
  def alert_providers(self, provider_glns: list[str]):
    for gln in provider_glns or []:
      self.provider_alerts.append({"provider_gln": gln, "ts": str(now())})
    return {"provider_alerts": len(self.provider_alerts)}

  @sagamethod()
  def outreach_patients(self, channel: str, count: int):
    self.patient_outreach.append({"channel": channel, "count": count, "ts": str(now())})
    return {"patient_outreach": self.patient_outreach[-1]}
```

5) Optional: PHI-safe analytics & signal scoring in a Private Enclave

```
@sagaclass(enclave=True)
class ClassPVAnalyticsEnclave(SPClassObject):
  """
  Confidential risk/signal modeling on PHI + clinical payloads.
  Emits only de-identified aggregates and attestations used by signal objects.
  """
  model_id = sagafield(type="str")
  encrypted_payloads = sagafield(type="list[bytes]")  # AE narratives, labs, vitals
  stats = sagafield(type="dict")
  attestations = sagafield(type="list[str]")

  @sagamethod()
  def compute_signal_stats(self) -> dict:
```

```
        # In-enclave computation; output is safe aggregate
        z = 3.27
        self.stats = {"ae_count": 47, "window_days": 30, "zscore": z, "baseline": 12.0}
        att = "TEE:pv:stats:" + rand_str(10)
        self.attestations.append(att)
        return {"stats": self.stats, "attestation": att}
```

Interoperability Analysis

- **HIPAA/HITECH:** PHI resides in ClassPHI; all access is logged; **breach events** (ClassBreachNotification) auto-emit when anomalies cross thresholds.
- **FAERS/VAERS:** ClassPharmacovigilanceEvent.mirror_to_registries() synchronizes with regulator systems to eliminate latency.
- **HL7 FHIR:** fhir_refs provide structured clinical context (Observation/Condition/Encounter) for stronger signals.
- **DSCSA/GS1:** Unit-level identity (dscsa_unit_ref, sgtin_ref) and lot linkages enable **targeted prevention** and, if necessary, precise **recall**.
- **USP:** Signals can be joined with quality boundaries (e.g., potency, impurities) to detect **substandard product** correlations.
- **ISO 20022:** ClassSafetySignal.attach_finance() coordinates **refunds/chargebacks** in parallel with mitigation/recall workflows.
- **ISPE/QRM:** Root cause and CAPA are fed by signals and PV events; re-verification is triggered where needed.
- **SagaFeeds:** Non-sensitive indices (product, lot, de-identified counts, status) are **public and free**, empowering patients and providers without exposing PHI.

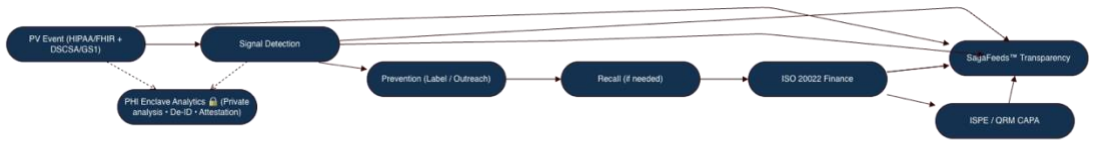


Diagram 5-D-- Patient-Centric Pharmacovigilance PV Event (HIPAA/FHIR + DSCSA/GS1) → Signal Detection → Prevention (label/outreach) → Recall (if needed) + ISO 20022 Finance → ISPE/QRM CAPA; SagaFeeds transparency; PHI Enclave analytics

Patient-centric pharmacovigilance: PV events (HIPAA/FHIR + DSCSA/GS1) drive signal detection and preventive label/outreach actions, with recall if needed. Financial settlements (ISO 20022) and CAPA are integrated, while SagaFeeds™ provides public transparency. PHI Enclave 🔒 supports private analytics and de-identification.

Scenario 5: Continuous Global Supply Chain Audit & Transparency

Background & Problem Statement

Global pharma supply chains remain fragmented across **DSCSA (US)**, **GS1/EPCIS (worldwide)**, **EU FMD**, national registries, and financial rails. Audits occur **retrospectively** via PDFs/XML/ERP exports; **ISO 20022** messages are **opaque** to supply provenance; **duplicate submissions** raise costs; patients lack **real-time lineage**. The result is inefficiency, risk of counterfeit/diversion, and limited public trust.

Technical Approach (SagaChain Implementation with Private Enclaves)

Create a **continuous audit fabric** that binds serialized identities, event pedigree, quality proofs, finance, and regulatory conformance into one persistent graph:

- **Serialized units:** ClassDSCSAUnit, ClassGS1SGTIN.

- **Events:** ClassEPCISEvent (commissioning, aggregation, shipping, receiving, dispensing).
- **Quality proofs:** USP assays, GMP release, CCS monitoring exposed as **attestations** (private data stays in **Enclaves**).
- **Finance:** ClassISO20022Payment/ClassISO20022Settlement linked to EPCIS milestones.
- **Regulatory overlays:** CFR/FD&C/IDMP classes enforce policy at method boundaries.
- **SagaFeeds:** public indices for non-sensitive lineage (patient & regulator self-service).
- **Private Enclaves:** protect commercial terms and PHI while emitting verifiable proofs.

Sample SagaPython Code

1) Global audit controller (continuous lineage + proofs + finance)

```
@sagaclass()
class ClassGlobalSupplyChainAudit(SPClassObject):
    """Continuous global audit linking serialization, EPCIS events, quality proofs, finance, and regs."""
    audit_id              = sagafield(type="str")
    product_idmp_ref      = sagafield(type="str", default="")    # -> ClassISO11615MedicinalProduct
    ndc_or_udi_ref        = sagafield(type="str", default="")    # -> ClassNDCCode / ClassDeviceUDI
    serialized_unit_refs  = sagafield(type="list[str]")  # -> ClassGS1SGTIN / ClassDSCSAUnit
    epcis_event_refs      = sagafield(type="list[str]")  # -> ClassEPCISEvent
    gmp_batch_refs        = sagafield(type="list[str]")  # -> ClassGMPBatch
    usp_quality_proof_refs = sagafield(type="list[str]")  # -> ClassUSPComplianceBindingReg / attest OIDs
    iso20022_flow_refs    = sagafield(type="list[str]")  # -> ClassISO20022Payment/Settlement
    regulatory_refs       = sagafield(type="list[str]")  # CFR/FD&C/EMA/WHO overlays
    public_index_ref      = sagafield(type="str", default="")    # -> SagaFeeds index OID/URI
    status                = sagafield(type="str", default="active",

enum={"active","paused","archived"})

    @sagamethod()
    def link_identity(self, idmp_oid: str = "", ndc_or_udi_oid: str = ""):
        if idmp_oid: self.product_idmp_ref = idmp_oid
        if ndc_or_udi_oid: self.ndc_or_udi_ref = ndc_or_udi_oid
        return {"idmp": self.product_idmp_ref, "ndc_or_udi": self.ndc_or_udi_ref}

    @sagamethod()
    def add_serialized_units(self, unit_oids: list[str]):
        for u in unit_oids or []:
            if u not in self.serialized_unit_refs:
                self.serialized_unit_refs.append(u)
        return {"serialized_units": len(self.serialized_unit_refs)}

    @sagamethod()
    def add_epcis_events(self, event_oids: list[str]):
        for e in event_oids or []:
            if e not in self.epcis_event_refs:
                self.epcis_event_refs.append(e)
        return {"epcis_events": len(self.epcis_event_refs)}

    @sagamethod()
    def link_gmp_batches(self, batch_oids: list[str]):
        for b in batch_oids or []:
            if b not in self.gmp_batch_refs:
                self.gmp_batch_refs.append(b)
        return {"gmp_batches": len(self.gmp_batch_refs)}

    @sagamethod()
    def add_quality_proofs(self, proof_oids: list[str]):
        for p in proof_oids or []:
            if p not in self.usp_quality_proof_refs:
                self.usp_quality_proof_refs.append(p)
        return {"quality_proofs": len(self.usp_quality_proof_refs)}

    @sagamethod()
    def attach_finance(self, iso20022_oids: list[str]):
        for f in iso20022_oids or []:
            if f not in self.iso20022_flow_refs:
                self.iso20022_flow_refs.append(f)
        return {"finance_flows": len(self.iso20022_flow_refs)}

    @sagamethod()
    def publish_public_index(self, index_uri: str):
        """Expose non-sensitive lineage for SagaFeeds access (patients & regulators)."""
        self.public_index_ref = index_uri
```

```
        return {"public_index_ref": self.public_index_ref}

    @sagamethod()
    def pause(self):
        self.status = "paused"
        return {"status": self.status}

    @sagamethod()
    def archive(self):
        self.status = "archived"
        return {"status": self.status}
```

2) Enclave: confidential quality/commerce proofs (USP/GMP/contract terms)

```
@sagaclass(enclave=True)
class ClassAuditProofEnclave(SPClassObject):
    """
    TEE-backed aggregator for confidential artifacts:
    - Lab raw files, method parameters, vendor SOPs (USP/GMP).
    - Commercial terms (pricing, rebates), PHI dispensing signals.
    Emits only minimal, verifiable proofs/attestations to the public audit object.
    """
    audit_ref          = sagafield(type="str")          # -> ClassGlobalSupplyChainAudit
    encrypted_blobs    = sagafield(type="list[bytes]")  # content-addressed off-chain storage
    proofs             = sagafield(type="list[dict]")   # [{"type":"USP_assay","att":"TEE:...","result":"pass"}]
    last_attestation   = sagafield(type="str", default="")

    @sagamethod()
    def attest_quality_and_terms(self) -> dict:
        # Perform in-enclave checks; emit compact proof
        att = "TEE:audit:" + rand_str(10)

        self.proofs.append({"type":"USP_assay","att":att,"result":"pass"})
        self.last_attestation = att
        # Optionally push a proof OID into the public audit object via policy hook
        return {"attestation": att, "proofs_count": len(self.proofs)}
```

3) Public lineage index (SagaFeeds) for patient/provider lookups

```
@sagaclass()
class ClassSupplyLineageIndex(SPClassObject):
    """ public index of lineage entries keyed by GTIN/SGTIN/lot (no PHI/commercial data)."""
    index_id           = sagafield(type="str")
    postings           = sagafield(type="dict")         # {"01:<gtin>": [OID,...], "21:<serial>": [OID,...], "lot:XYZ":[OID,...]}

    @sagamethod()
    def add_posting(self, key: str, oid: str):
        arr = self.postings.get(key, [])
        if oid not in arr:
            arr.append(oid)
            self.postings[key] = arr
        return {"key": key, "count": len(self.postings[key])}
```

4) Finance sync: milestone release when EPCIS/quality gates pass

```
@sagaclass()
class ClassSupplyFinanceOrchestrator(SPClassObject):
    """Synchronize ISO 20022 flows with EPCIS milestones and quality attestations."""
    orchestrator_id    = sagafield(type="str")
    audit_ref          = sagafield(type="str")          # -> ClassGlobalSupplyChainAudit
    required_epcis_steps = sagafield(type="list[str]") # ["commissioning","shipping","receiving"]
    required_quality_refs = sagafield(type="list[str]") # -> proof OIDs
    iso20022_refs      = sagafield(type="list[str]")    # -> ClassISO20022Payment/Settlement

    @sagamethod()
    def evaluate_and_release(self) -> dict:
        """
        Release payments only when EPCIS steps present AND quality proofs valid.
        (Illustrative checks; real logic dereferences OIDs and validates signatures.)
        """
        audit = deref(self.audit_ref)
        epc_ok = all(any(deref(e).biz_step == step for e in audit.epcis_event_refs) for step in self.required_epcis_steps or [])
        qual_ok = all(q in (audit.usp_quality_proof_refs or []) for q in self.required_quality_refs or [])
        released = False
        if epc_ok and qual_ok:
            for p in self.iso20022_refs or []:
                deref(p).conditional_release()
            released = True
        return {"epcis_ok": epc_ok, "quality_ok": qual_ok, "payments_released": released}
```

Interoperability Analysis

- **DSCSA & GS1:** serialized_unit_refs + epcis_event_refs provide **immutable lineage** (commissioning→aggregation→shipping→receiving→dispensing).
- **USP & 21 CFR / FD&C Act: USP proofs** and **GMP batch** readiness bind to distribution/release gates; statutory rules encoded as class policies.
- **ISO IDMP:** product_idmp_ref normalizes identity across **FDA/EMA/WHO** jurisdictions.
- **ISO 20022:** iso20022_flow_refs synchronize **settlement** with verified **EPCIS milestones** and **quality attestations**.
- **HIPAA/HITECH:** PHI (e.g., dispensing) remains **enclave-protected**; breaches auto-notify via **HITECH** objects.
- **ISPE:** Validation states (MES/serialization/equipment) act as **preconditions** for commissioning and release.
- **SagaFeeds:** public_index_ref exposes **non-sensitive lineage** for open, verification by patients, providers, and regulators.

Diagram Placeholder

[Diagram: Continuous Global Supply Chain Audit - Serialized Units → EPCIS Events → Enclave Quality Proofs → ISO 20022 Flows → SagaFeeds Public Index; Cross-links to USP, 21 CFR/FD&C, IDMP]

Implications

- **Regulators:** Real-time audit access reduces manual inspections and accelerates enforcement.
- **Manufacturers/Distributors:** Eliminate duplicate reporting; automate compliance gates; reduce reconciliation costs.
- **Providers & Patients:** Scan **GS1 Digital Link/QR** to see trusted lineage instantly.
- **Market Integrity:** Counterfeit/diversion detection becomes **programmatic and auditable**, improving safety and trust.

Scenario 6: Evidence-to-Reimbursement Loop

Background & Problem Statement

Clinical evidence, regulatory approval, and reimbursement frequently operate in silos:

- **FDAAA 801** transparency rarely flows into payer decision engines.
- **CDISC** datasets and **HL7** outcomes are disconnected from coverage rules.
- **SPL** labeling (indications/dosing) is not programmatically bound to reimbursement logic.
- **ISO 20022** payments settle independently of evidence checks.
- Patient affordability suffers from lagging, non-transparent decisions.

Technical Approach (SagaChain Implementation with Private Enclaves)

Create a **persistent loop** that ties **evidence → labeling → reimbursement policy →**

settlement, with PHI and proprietary rules handled in **Private Enclaves**:

- **Evidence objects:** ClassClinicalTrial, ClassCDISCStudy, ClassSDTMDomain, ClassADaMDataset, ClassFHIRResource.
- **Labeling:** ClassSPLDocument/ClassSPLSection contains indications derived from validated datasets.
- **Coverage policy & decision:** ClassReimbursementDecision links to evidence, SPL, and enclave-produced efficacy/safety proofs; evaluates coverage criteria; records reasoning.
- **Financial execution:** ClassISO20022Settlement/ClassISO20022Payment triggered when coverage is approved and claim conditions met.
- **Transparency: SagaFeeds** exposes non-sensitive policy and decision metadata; PHI remains enclave-protected.

Sample SagaPython Code

1) Evidence-bound reimbursement decision

```
@sagaclass()
class ClassReimbursementDecision(SPClassObject):
    """Evidence-based reimbursement decision bound to trial, SPL, and ISO 20022 flows."""
    decision_id = sagafield(type="str")
    payer_account_ref = sagafield(type="str") # -> ClassBusinessAccount (payer)
    provider_account_ref = sagafield(type="str") # -> ClassBusinessAccount (provider)
    manufacturer_ref = sagafield(type="str") # -> ClassBusinessAccount (mfr)
    patient_phi_ref = sagafield(type="str", default="") # -> ClassPHI (HIPAA)
    trial_ref = sagafield(type="str") # -> ClassClinicalTrial (FDAAA 801)
    cdisc_study_ref = sagafield(type="str", default="") # -> ClassCDISCStudy
    evidence_domain_refs = sagafield(type="list[str]") # -> SDTM/ADaM domain OIDs
    hl7_outcome_refs = sagafield(type="list[str]") # -> ClassFHIRResource
    spl_label_ref = sagafield(type="str") # -> ClassSPLDocument
    indication_code = sagafield(type="str", default="") # coded indication used for rule matching
    coverage_criteria = sagafield(type="dict") # inclusion/exclusion rules
    enclave_proof_refs = sagafield(type="list[str]") # -> enclave attestations (efficacy/safety/cost-effectiveness)
    settlement_flow_refs = sagafield(type="list[str]") # -> ClassISO20022Settlement/Payment
    decision_status = sagafield(type="str", default="pending",

enum={"pending","approved","denied","needs_info"})
    decision_reason = sagafield(type="str", default="")
    feeds_tags = sagafield(type="list[str]") # public, non-sensitive facets (product, indication, date)

    @sagamethod()
    def bind_evidence(self, trial_oid: str, cdisc_oid: str, sdtm_or_adam_oids: list[str], fhir_oids: list[str]):
        self.trial_ref = trial_oid
        self.cdisc_study_ref = cdisc_oid
        for d in sdtm_or_adam_oids or []:
            if d not in self.evidence_domain_refs: self.evidence_domain_refs.append(d)
        for f in fhir_oids or []:
            if f not in self.hl7_outcome_refs: self.hl7_outcome_refs.append(f)
        return {
            "trial": self.trial_ref,
            "cdisc": self.cdisc_study_ref,
            "domains": len(self.evidence_domain_refs),
            "hl7": len(self.hl7_outcome_refs)
        }

    @sagamethod()
    def bind_label(self, spl_oid: str, indication_code: str):
        self.spl_label_ref = spl_oid
        self.indication_code = indication_code
        return {"spl": self.spl_label_ref, "indication": self.indication_code}

    @sagamethod()
    def set_criteria(self, criteria: dict):
        self.coverage_criteria = criteria
        return {"criteria_keys": list(self.coverage_criteria.keys())}

    @sagamethod()
    def attach_enclave_proofs(self, proof_oids: list[str]):
```

```
        for p in proof_oids or []:
            if p not in self.enclave_proof_refs: self.enclave_proof_refs.append(p)
        return {"proofs": len(self.enclave_proof_refs)}

    @sagamethod()
    def evaluate(self) -> dict:
        """
        Illustrative gating:
        - SPL indication matches requested indication
        - Enclave efficacy/safety thresholds satisfied
        - HL7 outcomes and CDISC datasets exist (optionally jurisdictional rules)
        """
        spl_ok = bool(self.spl_label_ref and self.indication_code)
        ef_ok = len(self.enclave_proof_refs) > 0
        data_ok = bool(self.cdisc_study_ref) and (len(self.evidence_domain_refs) > 0)
        hl7_ok = len(self.hl7_outcome_refs) > 0
        criteria_ok = bool(self.coverage_criteria)  # real engine would compute rule satisfaction

        all_ok = spl_ok and ef_ok and data_ok and hl7_ok and criteria_ok
        self.decision_status = "approved" if all_ok else "denied"
        self.decision_reason = "criteria_met" if all_ok else "insufficient_evidence_or_mismatch"
        return {
            "approved": all_ok,
            "checks": {"spl": spl_ok, "efficacy_proofs": ef_ok, "cdisc": data_ok, "hl7": hl7_ok, "criteria": criteria_ok},
            "status": self.decision_status, "reason": self.decision_reason
        }

    @sagamethod()
    def attach_settlement(self, iso20022_oids: list[str]):
        for s in iso20022_oids or []:
            if s not in self.settlement_flow_refs: self.settlement_flow_refs.append(s)
        return {"settlements": len(self.settlement_flow_refs)}

    @sagamethod()
    def execute_financials(self) -> dict:
        """
        Trigger ISO 20022 flows only when coverage is approved.
        """
        released = False
        if self.decision_status == "approved":
            for s in self.settlement_flow_refs or []:
                deref(s).conditional_release()
                released = True
        return {"payments_released": released, "status": self.decision_status}
```

2) Private Enclave for PHI claims + proprietary payer logic

```
@sagaclass(enclave=True)
class ClassReimbursementEnclave(SPClassObject):
    """
    Confidential evaluation of claims against payer policy and efficacy/safety thresholds.
    Publishes only attestations/aggregates (no PHI leakage).
    """
    decision_ref = sagafield(type="str")  # -> ClassReimbursementDecision
    encrypted_claims = sagafield(type="list[bytes]")  # EDI/HL7 claim payloads, PHI
    encrypted_ruleset = sagafield(type="bytes")  # payer proprietary policy
    attestations = sagafield(type="list[str]")
    last_scorecard = sagafield(type="dict")

    @sagamethod()
    def evaluate_claims(self) -> dict:
        # In-enclave scoring; illustrative output
        scorecard = {"meets_indication": True, "meets_step_therapy": True, "meets_outcome_threshold": True}
        self.last_scorecard = scorecard
        att = "TEE:reimb:" + rand_str(10)
        self.attestations.append(att)
        # Push attestation to decision object
        dec = deref(self.decision_ref)
        dec.attach_enclave_proofs([att])
        return {"attestation": att, "scorecard": scorecard}
```

3) ISO 20022 settlement object (already defined in standards; here, conditional hook)

```
@sagaclass()
class ClassISO20022Settlement(SPClassObject):
    """Settlement instruction conditioned on evidence-backed approval."""
    settlement_id = sagafield(type="str")
    debtor = sagafield(type="str")  # payer
    creditor = sagafield(type="str")  # provider/manufacturer
    amount = sagafield(type="float")
    currency = sagafield(type="str", default="USD")
    related_decision_ref = sagafield(type="str", default="")  # -> ClassReimbursementDecision
```

```
    released = sagafield(type="bool", default=False)
    attestations = sagafield(type="list[str]")

    @sagamethod()
    def link_decision(self, decision_oid: str):
        self.related_decision_ref = decision_oid
        return {"related_decision_ref": self.related_decision_ref}

    @sagamethod()
    def add_attestation(self, att: str):
        if att not in self.attestations: self.attestations.append(att)
        return {"attestations": len(self.attestations)}

    @sagamethod()
    def conditional_release(self) -> bool:
        if self.related_decision_ref and deref(self.related_decision_ref).decision_status == "approved":
            self.released = True
        return self.released
```

4) Public, transparency (policy & non-PHI decision metadata)

```
@sagaclass()
class ClassCoveragePolicyFeed(SPClassObject):
    """SagaFeeds index exposing non-sensitive coverage rules and decision stats."""
    feed_id = sagafield(type="str")
    policy_refs = sagafield(type="list[str]") # URIs/OIDs of public rule docs
    decision_summaries = sagafield(type="list[dict]") # {"decision_id":..., "status":..., "indication":..., "ts":...}

    @sagamethod()
    def add_policy(self, ref: str):
        if ref not in self.policy_refs: self.policy_refs.append(ref)
        return {"policies": len(self.policy_refs)}

    @sagamethod()
    def add_decision_summary(self, decision_id: str, status: str, indication: str):
        self.decision_summaries.append({"decision_id": decision_id, "status": status, "indication": indication, "ts": str(now())})
        return {"summaries": len(self.decision_summaries)}
```

Interoperability Analysis

- **FDAAA 801 / CDISC:** Trial registration and SDTM/ADaM datasets provide verifiable evidence inputs.
- **HL7:** Patient outcomes (FHIR) link claims to real-world effectiveness; PHI guarded in **enclaves** with public proofs only.
- **SPL:** Coverage is constrained to **approved indications**; label updates propagate into policy checks.
- **ISO 20022:** Settlement flows (**payer → provider → manufacturer**) are **conditionally released** upon approved decisions.
- **HIPAA/HITECH:** Claims and PHI are encrypted-in-use; anomalous access triggers **HITECH** notifications.
- **USP / 21 CFR:** Assay/GMP proofs (when required by payer policy) are referenced as evidence prerequisites.
- **SagaFeeds:** Public rules/metadata improve transparency without exposing PHI or proprietary policy internals.

Diagram 5-F — Evidence-to-Reimbursement FDAAA/CDISC/HL7 Evidence → SPL Indication → Reimbursement Decision (enclave proofs) → ISO 20022 Settlement → SagaFeeds Policy Transparency

Evidence-to-Reimbursement: clinical and regulatory evidence flows from FDAAA/CDISC/HL7 through SPL indication publication to reimbursement decisions with enclave proofs, ISO 20022 settlements, and public SagaFeeds™ transparency. Dashed feedback links connect decisions back to evidence and SPL for continuous validation.

Implications

- **Payers:** Verifiable, auditable, and automatable coverage; reduced leakage and fraud.
- **Providers:** Faster, rules-driven approvals; automated settlement; fewer denials.
- **Manufacturers:** Accelerated market access aligned to evidence and labeling.
- **Patients:** Increased affordability and confidence via clear, timely coverage outcomes linked to real evidence.

Scenario 7: Global Harmonized Pharmacopoeia & Standards Enforcement

Background & Problem Statement

Pharmacopoeias (USP, Ph. Eur., JP, others) define compendial quality. Enforcement varies by region; standards are often PDF/text, weakly linked to **ISO IDMP** product identity and **GS1** serialization; harmonization is slow and duplicative for manufacturers. A programmable, globally harmonized substrate is required.

Technical Approach (SagaChain Implementation with Private Enclaves)

- **Standards as code:** USP/Ph. Eur./JP/WHO monographs & chapters implemented as persistent classes.
- **Multi-inheritance:** A single product conformance object can inherit multiple pharmacopeial constraints simultaneously.
- **Global identifiers:** Bind standards to **ISO IDMP (11615/11616/11238)** and **GS1** (GTIN/SGTIN) for unambiguous identity.
- **Private Enclaves:** Execute proprietary assays confidentially; publish **attested proofs** only.
- **SagaFeeds:** Publish **non-sensitive** harmonization metadata for free public verification.
- **Interlocks:** Conformance gates **batch release (21 CFR)**, **serialization (DSCSA/GS1)**, **labeling (SPL)**, and **finance (ISO 20022)**.

Sample SagaPython Code

1) Standards classes (USP / Ph. Eur. / JP / WHO)

```
@sagaclass()
class ClassUSPStandard(SPClassObject):
    """USP monograph or general chapter in a global harmonized context."""
    monograph_id          = sagafield(type="str")
    title                 = sagafield(type="str")
    assay_method_ref      = sagafield(type="str", default="")    # URI/OID to method/chapter
    impurity_limits       = sagafield(type="dict")
# {"total_imp": "≤ 2.0%", "each_imp": "≤ 0.5%"}
    dissolution_spec      = sagafield(type="dict", default={})
    harmonization_refs    = sagafield(type="list[str]")
# Ph. Eur./JP/WHO equivalents
    last_rev_date         = sagafield(type="date", default=None)

@sagaclass()
class ClassPhEurStandard(SPClassObject):
    """European Pharmacopoeia standard."""
    monograph_id          = sagafield(type="str")
    title                 = sagafield(type="str")
    test_methods          = sagafield(type="dict")          #
{"assay": "...", "id": "...", "impurities": "..."}
    limits                = sagafield(type="dict")          #
mirrored limits per Ph. Eur.
    harmonization_refs    = sagafield(type="list[str]")

@sagaclass()
class ClassJPStandard(SPClassObject):
    """Japanese Pharmacopoeia standard."""
    monograph_id          = sagafield(type="str")
```

```
    title               = sagafield(type="str")
    methods             = sagafield(type="dict")
    limits              = sagafield(type="dict")
    harmonization_refs  = sagafield(type="list[str]")

@sagaclass()
class ClassWHOStandard(SPClassObject):
    """WHO international standard / guidance."""
    reference_id        = sagafield(type="str")
    title               = sagafield(type="str")
    scope               = sagafield(type="str")          # e.g., "essential medicines", "vaccines"
    equivalence_proofs  = sagafield(type="list[str]")    # attestations linking to regional pharmacopeias
```

2) Harmonized product conformance (multi-inheritance + IDMP/GS1 bindings)

```
@sagaclass()
class ClassHarmonizedPharmacopeialConformance(SPClassObject):
    """
    Single conformance object aggregating USP/Ph. Eur./JP/WHO requirements
    and binding to global identifiers (IDMP) and serialized units (GS1/DSCSA).
    """
    conformance_id      = sagafield(type="str")
    idmp_mp_ref         = sagafield(type="str")          # -> ClassISO11615MedicinalProduct
    idmp_pp_ref         = sagafield(type="str", default="")  # -> ClassISO11616PharmaceuticalProduct
    usp_ref             = sagafield(type="str", default="")  # -> ClassUSPStandard
    pheur_ref           = sagafield(type="str", default="")  # -> ClassPhEurStandard
    jp_ref              = sagafield(type="str", default="")  # -> ClassJPStandard
    who_ref             = sagafield(type="str", default="")  # -> ClassWHOStandard
    gtin_ref            = sagafield(type="str", default="")  # -> ClassGS1GTIN
    sgtin_refs          = sagafield(type="list[str]")        # -> ClassGS1SGTIN (serialized instances)
    gmp_batch_refs      = sagafield(type="list[str]")        # -> ClassGMPBatch
    assay_proof_refs    = sagafield(type="list[str]")        # enclave attestations
    conformance_status  = sagafield(type="str", default="pending",

enum={"pending","meets_all","meets_subset","fails"})
    notes               = sagafield(type="str", default="")
    feeds_tags          = sagafield(type="list[str]")

    @sagamethod()
    def bind_identifiers(self, idmp_mp_oid: str, gtin_oid: str, idmp_pp_oid: str = ""):
        self.idmp_mp_ref = idmp_mp_oid
        self.gtin_ref = gtin_oid
        if idmp_pp_oid:
            self.idmp_pp_ref = idmp_pp_oid
        return {"idmp_mp": self.idmp_mp_ref, "gtin": self.gtin_ref, "idmp_pp": self.idmp_pp_ref}

    @sagamethod()
    def bind_standards(self, usp_oid: str = "", pheur_oid: str = "", jp_oid: str = "", who_oid: str = ""):
        self.usp_ref = usp_oid or self.usp_ref
        self.pheur_ref = pheur_oid or self.pheur_ref
        self.jp_ref = jp_oid or self.jp_ref
        self.who_ref = who_oid or self.who_ref
        return {"usp": self.usp_ref, "pheur": self.pheur_ref, "jp": self.jp_ref, "who": self.who_ref}

    @sagamethod()
    def add_instances_and_batches(self, sgtin_oids: list[str], gmp_batch_oids: list[str]):
        for s in sgtin_oids or []:
            if s not in self.sgtin_refs: self.sgtin_refs.append(s)
        for b in gmp_batch_oids or []:
            if b not in self.gmp_batch_refs: self.gmp_batch_refs.append(b)
        return {"sgtins": len(self.sgtin_refs), "batches": len(self.gmp_batch_refs)}

    @sagamethod()
    def record_assay_proofs(self, proof_oids: list[str]):
        for p in proof_oids or []:
            if p not in self.assay_proof_refs: self.assay_proof_refs.append(p)
        return {"assay_proofs": len(self.assay_proof_refs)}

    @sagamethod()
    def evaluate(self) -> dict:
        """
        Illustrative evaluation:
        - present USP/PhEur/JP refs
        - at least one WHO alignment (if applicable)
        - >= 1 assay proof (TEE-attested) and >= 1 GMP batch reference
        """
        stds_ok = bool(self.usp_ref or self.pheur_ref or self.jp_ref)
        who_ok  = True if not self.who_ref else True  # optional or present
```

```
        assay_ok = len(self.assay_proof_refs) > 0
        gmp_ok = len(self.gmp_batch_refs) > 0
        if stds_ok and who_ok and assay_ok and gmp_ok:
            self.conformance_status = "meets_all"
        elif stds_ok and (assay_ok or gmp_ok):
            self.conformance_status = "meets_subset"
        else:
            self.conformance_status = "fails"
        return {"status": self.conformance_status, "checks": {"standards": stds_ok, "who": who_ok, "assay": assay_ok, "gmp": gmp_ok}}
```

3) Private Enclave for confidential assays and cross-standard equivalence

```
@sagaclass(enclave=True)
class ClassPharmacopeialAssayEnclave(SPClassObject):
    """
    Executes proprietary/confidential assays; outputs attestations compatible with multiple pharmacopeias.
    """
    conformance_ref = sagafield(type="str") # -> ClassHarmonizedPharmacopeialConformance
    encrypted_methods = sagafield(type="list[bytes]") # method SOPs, parameters
    encrypted_results = sagafield(type="list[bytes]") # raw chromatograms, spectra, etc.
    attestations = sagafield(type="list[str]")

    @sagamethod()
    def run_and_attest(self, targets: list[str]) -> dict:
        """
        targets may include ['USP.assay','PhEur.impurities','JP.dissolution'].
        Returns a single TEE attestation proving all requested targets met their limits.
        """
        att = "TEE:pharmacopeia:" + rand_str(10)
        self.attestations.append(att)
        # Bind proof to harmonized conformance record
        conf = deref(self.conformance_ref)
        conf.record_assay_proofs([att])
        return {"attestation": att, "targets": targets}
```

4) Public, harmonization index (SagaFeeds)

```
@sagaclass()
class ClassPharmacopeialHarmonyFeed(SPClassObject):
    """SagaFeeds index exposing non-sensitive harmonization metadata."""
    feed_id = sagafield(type="str")
    entries = sagafield(type="list[dict]") # {"idmp_mp":..., "gtin":..., "status":..., "rev":..., "regions":[...]}

    @sagamethod()
    def publish_entry(self, idmp_mp: str, gtin: str, status: str, regions: list[str], rev_date: str):
        self.entries.append({"idmp_mp": idmp_mp, "gtin": gtin, "status": status, "regions": regions, "rev": rev_date, "ts": str(now())})
        return {"count": len(self.entries)}
```

Interoperability Analysis

- **USP / Ph. Eur. / JP / WHO:** Multi-inheritance allows **one conformance object** to satisfy **multiple pharmacopeias** simultaneously; **enclave attestations** prove compliance without revealing proprietary methods.
- **ISO IDMP:** idmp_mp_ref/idmp_pp_ref anchor **global medicinal/pharmaceutical product identity**, avoiding regional ambiguity.
- **GS1 / DSCSA:** gtin_ref + sgtin_refs ensure **every serialized unit** can carry conformance lineage; ties directly into **EPCIS** events.
- **21 CFR / FD&C Act:** Conformance acts as a **release gate** (e.g., 210/211) and supports misbranding/adulteration determinations.
- **SPL:** Labeling sections (e.g., **quality attributes**, **excipients**) pull structured facts from harmonized conformance.
- **ISO 20022:** Payments/settlements for **release**, **toll manufacturing**, or **quality services** can be **conditioned** on conformance status.
- **SagaFeeds:** Public, visibility of non-sensitive harmonization status by **IDMP/GTIN**, improving trust and procurement efficiency.

Diagram 5-G — Harmonized Pharmacopoeia USP, Ph. Eur., JP, WHO → Harmonized Conformance → (IDMP, GS1/DSCSA) → Enclave Assay Proofs → SPL/21 CFR → ISO 20022 → SagaFeeds

Harmonized Pharmacopoeia: USP, Ph. Eur., JP, and WHO guidance converge into a harmonized conformance layer, mapped to IDMP and GS1/DSCSA for serialization and identity. Enclave Assay Proofs validate quality; SPL/21 CFR enforce regulatory labeling; ISO 20022 enables settlement; SagaFeeds™ publishes transparent indices.

Implications

- **Manufacturers: Validate once**, prove everywhere; reduce duplicate testing and inspection overhead.
- **Regulators:** Share attested proofs; focus on exceptions; faster market access with consistent quality.
- **Providers & Patients:** Verify that medicines meet **consistent global standards** at the unit level.
- **Global Health (WHO):** Promote equitable access by **harmonizing quality enforcement** across jurisdictions.

Scenario 8: Adaptive AI-Driven Compliance Monitoring

(linking persistent-state data to real-time AI/ML analytics for proactive regulatory enforcement)

Background & Problem Statement

Compliance activities are still periodic and reactive (batch record review, APR/PQR, episodic inspections, retrospective PV). Meanwhile, pharma operations emit high-frequency signals:

- **GS1/DSCSA EPCIS** logistics streams,
- **21 CFR 210/211 / ISPE** process & quality telemetry,
- **HL7 FHIR** clinical/RWE outcomes,
- **FAERS/VAERS** safety,
- **SPL / FDAAA 801** evidence,
- **ISO 20022** financial flows.

Static controls can't keep up, yielding late recalls, undetected drifts, and rising assurance cost.

Technical Approach (SagaChain with Private Enclaves)

1. **Unified Feature Graph (on-chain)** All regulated objects (GMP batches, SGTIN units, EPCIS events, USP proofs, HL7 outcomes, FAERS/VAERS reports, ISO 20022 flows) are first-class, cross-linked nodes providing causal & temporal context for analytics.
2. **Model Hosting & Execution (Private Enclaves)** AI/ML pipelines (drift detection, anomaly scoring, causal inference) run **encrypted-in-use**. Only **attestations, scores, and explanations** exit the enclave.
3. **Closed-Loop Controls** Attested alerts can open **recalls**, **ISPE change control**, **payment holds/escrows**, or **SPL label updates**, with cryptographic provenance.
4. **Regulatory Transparency SagaFeeds** publishes de-identified alert registries, model lineage, and fairness/validation proofs free to query while sensitive inputs remain confidential.

Sample SagaPython Code

A) Enclave monitor: feature binding → scoring → attestation

```
@sagaclass(enclave=True)
class ClassAdaptiveMonitor(SPClassObject):
   """AI-driven continuous compliance monitor executing in a Private Enclave."""
   monitor_id          = sagafield(type="str")
   scope               = sagafield(type="dict")      # {"products":["ISO11615:..."], "lots":["A1"], "regions":["US","EU"]}
   model_ref           = sagafield(type="str")       # hash/URI of versioned model artifact
   feature_refs        = sagafield(type="list[str]")# OIDs: EPCIS, GMP, HL7, FAERS, ISO20022, etc.
   thresholds          = sagafield(type="dict")      # {"signal":0.85, "drift":0.10}
   last_run            = sagafield(type="datetime", default=None)
   last_attestation    = sagafield(type="str", default="")
   last_metrics        = sagafield(type="dict", default={})  # AUROC, drift stats, data coverage
   feeds_tag           = sagafield(type="str", default="")
# public, non-PHI index tag

   @sagamethod()
   def bind_features(self, oids: list[str]):
      for oid in oids or []:
         if oid not in self.feature_refs:
            self.feature_refs.append(oid)
      return {"features_bound": len(self.feature_refs)}

   @sagamethod()
   def run_inference(self) -> dict:
      """
      Execute model inside enclave on encrypted features; emit attestation + summary metrics.
      (Illustrative scores; real implementation dereferences feature_refs and computes securely.)
      """
      self.last_run = now()
      scores = [{"oid": oid, "score": 0.91} for oid in self.feature_refs[:50]]  # top K
      att = "TEE:adaptive:" + rand_str(10)
      self.last_attestation = att
      self.last_metrics = {"auroc": 0.94, "data_drift": 0.06, "n_eval": len(self.feature_refs)}
      return {"attestation": att, "scores": scores, "metrics": self.last_metrics}

   @sagamethod()
   def evaluate_thresholds(self, score: float, drift: float) -> dict:
      s_ok = score >= (self.thresholds.get("signal", 0.85))
      d_ok = drift <= (self.thresholds.get("drift", 0.10))
      return {"signal_ok": s_ok, "drift_ok": d_ok, "ready_for_alert": (not d_ok) or (not s_ok) or (s_ok and d_ok and score >= 0.95)}
```

B) Alert object: actions & lifecycle

```
@sagaclass()
class ClassComplianceAlert(SPClassObject):
   """Actionable alert emitted by adaptive monitoring."""
   alert_id            = sagafield(type="str")
   monitor_ref         = sagafield(type="str")       # -> ClassAdaptiveMonitor
   attestation_ref     = sagafield(type="str", default="")
   score               = sagafield(type="float", default=0.0)
   explanation         = sagafield(type="dict")      # SHAP-like or rule attributions
   linked_oids         = sagafield(type="list[str]") # DSCSA units, batches, trials, EPCIS events
   suggested_actions   = sagafield(type="list[str]") # ["open_recall","lock_payment","init_change_control","update_label"]
   status              = sagafield(type="str", default="open",
                                   enum={"open","acknowledged","mitigated","closed"})
   created_at          = sagafield(type="datetime", default=None)
   updated_at          = sagafield(type="datetime", default=None)

   @sagamethod()
   def open(self, monitor_oid: str, att: str, score: float, explanation: dict, links: list[str], actions: list[str]):
      self.monitor_ref = monitor_oid
      self.attestation_ref = att
      self.score = score
      self.explanation = explanation
      self.linked_oids = list(dict.fromkeys(links or []))
      self.suggested_actions = list(dict.fromkeys(actions or []))
      self.created_at = now(); self.updated_at = self.created_at
      self.status = "open"
      return {"status": self.status, "actions": self.suggested_actions}

   @sagamethod()
   def acknowledge(self, note: str = ""):
      self.status = "acknowledged"; self.updated_at = now()
      return {"status": self.status, "note": note}
```

```
    @sagamethod()
    def mitigate(self, performed: list[str]):
        """Record actions taken; real logic would link to recall/change control/payment/label OIDs."""
        self.status = "mitigated"; self.updated_at = now()
        return {"status": self.status, "performed": performed}

    @sagamethod()
    def close(self, rationale: str = ""):
        self.status = "closed"; self.updated_at = now()
        return {"status": self.status, "rationale": rationale}
```

C) Closed-loop actions: recall, change control, finance hold, label update

```
@sagaclass()
class ClassAdaptiveActions(SPClassObject):
    """Automates downstream actions in response to compliance alerts."""
    action_id           = sagafield(type="str")
    alert_ref           = sagafield(type="str")         # -> ClassComplianceAlert
    recall_ref          = sagafield(type="str", default="") # -> ClassRecallEvent
    change_control_ref  = sagafield(type="str", default="")      # -> ClassChangeControl (ISPE)
    payment_hold_refs   = sagafield(type="list[str]") # -> ClassISO20022Settlement/Payment
    label_update_ref    = sagafield(type="str", default="")      # -> ClassSPLSection
    performed           = sagafield(type="list[str]")

    @sagamethod()
    def open_recall(self, unit_or_lot_oids: list[str], reason: str) -> str:
        recall = instantiate("ClassRecallEvent", {
            "recall_id": "ALERT-" + rand_str(6), "authority": "QA", "reason": reason,
            "scope_units": unit_or_lot_oids, "start_date": str(now()), "status": "open", "linked_settlements": []
        })
        self.recall_ref = recall.oid; self.performed.append("open_recall")
        return self.recall_ref

    @sagamethod()
    def init_change_control(self, title: str, impact: dict) -> str:
        cc = instantiate("ClassChangeControl", {
            "change_id": "CC-" + rand_str(6), "title": title, "impact": impact, "status": "open"
        })
        self.change_control_ref = cc.oid; self.performed.append("init_change_control")
        return self.change_control_ref

    @sagamethod()
    def lock_payments(self, iso20022_oids: list[str]) -> int:
        count = 0
        for p in iso20022_oids or []:
            pay = deref(p)
            if hasattr(pay, "released") and not pay.released:
                # Implement a hold flag (illustrative)
                if hasattr(pay, "hold"):
                    pay.hold = True
                else:
                    setattr(pay, "hold", True)
                self.payment_hold_refs.append(p); count += 1
        self.performed.append("lock_payment")
        return count

    @sagamethod()
    def update_label(self, spl_section_oid: str, warning_text: str) -> str:
        spl = deref(spl_section_oid)
        # In practice, SPL update uses governed methods; here we set content illustratively.
        if hasattr(spl, "content"):
            spl.content = warning_text + "\n" + (spl.content or "")
        self.label_update_ref = spl_section_oid; self.performed.append("update_label")
        return self.label_update_ref
```

D) Public, alert registry (SagaFeeds)

```
@sagaclass()
class ClassComplianceAlertFeed(SPClassObject):
    """SagaFeeds index for de-identified, regulator/patient-visible alert metadata."""
    feed_id             = sagafield(type="str")
    entries             = sagafield(type="list[dict]")  # {"alert_id":..., "model":..., "score":..., "product":..., "lot":..., "ts":...}

    @sagamethod()
    def publish_entry(self, alert_id: str, model_ref: str, score: float, product_idmp: str, lot: str):
        self.entries.append({"alert_id": alert_id, "model": model_ref, "score": score,
                             "product": product_idmp, "lot": lot, "ts": str(now())})
        return {"count": len(self.entries)}
```

Interoperability Analysis

- **21 CFR / ISPE (GMP & Validation):** Enclave-run monitors observe **process capability**, **control state**, and **Part 11** audit artifacts; alerts can auto-open **ISPE change control** and **re-verification** (GAMP IQ/OQ/PQ), with signed attestations.
- **GS1 / DSCSA:** **EPCIS drift** (route anomalies, unexpected de-aggregations) triggers **unit-/lot-level holds/recalls**; links to **SGTIN** and **DSCSA units** ensure precise scope.
- **USP (Regulatory & Standards):** Deviations from **compendial limits** cause **assay re-testing**; distribution gates remain locked until proofs pass.
- **HL7 / FAERS / VAERS:** **FHIR outcomes** and **PV clusters** feed real-time safety signals; outputs may **preemptively update SPL warnings** or issue **field safety notices**.
- **FDAAA 801 / NIH DMSP / SPL:** **Evidence drift** (RWE vs. pivotal) flags **label misalignment**; enclave summaries enable rapid but privacy-preserving label updates.
- **ISO IDMP / GS1:** Alerts bind to **global IDs** (IDMP 11615/11616) and **serialized instances** for cross-border coordination.
- **ISO 20022 (Finance):** Alerts can **pause settlements**, enforce **escrows**, or **re-price contracts** until mitigation is complete.
- **HIPAA/HITECH & Private Enclaves:** PHI and proprietary data remain **encrypted-in-use**; only **scores/explanations/attestations** leave the enclave; anomalies trigger **HITECH** notifications.

Governance, Risk & Compliance (GRC)

- **Model Risk Management:** Versioned **model artifacts**, **validation datasets**, and **performance metrics** are persisted with immutable lineage; **bias/drift/explainability** audits recorded (e.g., into **ClassValidationPackage**).
- **Human-in-the-Loop:** Alerts route to **QP/Quality leadership**; all dispositions are **digitally signed** and retained.
- **Regulatory Review:** Regulators subscribe to **alert attestations** via **SagaFeeds**; sensitive inputs stay enclave-bound, enabling oversight without over-disclosure.

Outcomes & Benefits

- **Earlier detection, smaller impact:** Minutes/hours instead of weeks/months.
- **Lower TCoQ:** Automated triage reduces manual review and redundant audits.
- **Precision actions:** Targeted unit/lot holds, limited-scope recalls.
- **Trust & transparency:** Public, no-cost alert indices with cryptographic provenance.

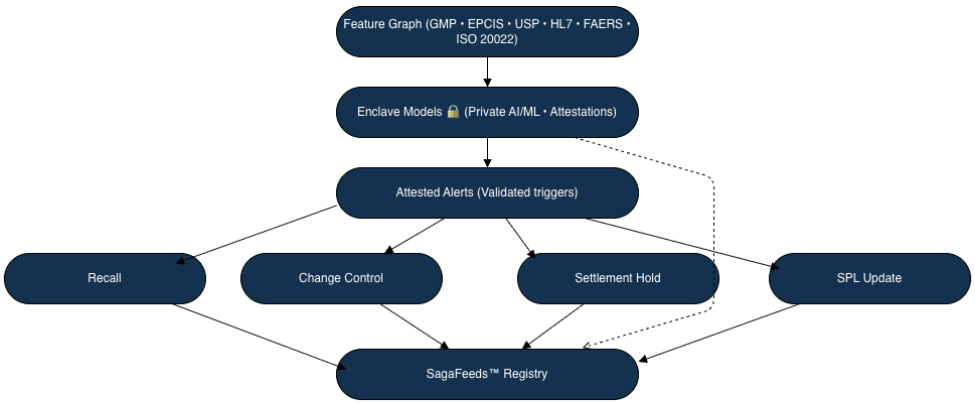


Diagram 5-H — Adaptive Compliance Feature Graph (GMP, EPCIS, USP, HL7, FAERS, ISO 20022) → Enclave Models → Attested Alerts → Actions (Recall, Change Control, Settlement Hold, SPL Update) → SagaFeeds Registry

Adaptive Compliance: unified feature graph across GMP, EPCIS, USP, HL7, FAERS, and ISO 20022 feeds enclave models that produce attested alerts. Alerts branch into Recall, Change Control, Settlement Hold, and SPL Update actions, with all proofs and outcomes logged to the SagaFeeds™ Registry. Enclave models also publish attestation proofs directly to the registry (dashed).

Scenario 9: Borderless Market Access & Multi-Jurisdiction Approval Synchronization

(coordinating FD&C, EMA/IDMP, WHO prequalification, and payer onboarding)

Background & Problem Statement

Launching across jurisdictions requires separate dossiers, asynchronous label updates, and divergent identifiers (NDC, IDMP, GS1, national codes). Payers/HTA bodies evaluate **after** regulatory approval, delaying access. Evidence, approvals, and reimbursement remain unlinked, creating cost and inequity.

Technical Approach (SagaChain with Private Enclaves)

- **Regulatory approvals as code:** FDA, EMA, WHO prequalification modeled as persistent objects.
- **Global product graph:** One **ClassGlobalProduct** aggregates approvals/labels and binds **ISO IDMP** + **GS1** identities for universal alignment.
- **Private Enclaves:** Protect proprietary clinical/HTA models; publish only attestations (cost-effectiveness, budget impact) to payers/regulators.
- **SagaFeeds:** Public, indices expose **approval status**, **label alignment**, **recalls** across jurisdictions.
- **Financial integration:** Reimbursement and settlement (ISO 20022) are triggered by synchronized approvals and payer onboarding logic.

Sample SagaPython Code

A) Regulatory approvals

```
@sagaclass()
class ClassFDAApproval(SPClassObject):
    """FDA NDA/BLA approval under FD&C Act."""
    app_id              = sagafield(type="str")
    ndc_root_ref        = sagafield(type="str", default="")
# -> ClassNDCCode (if modeled)
    approval_date       = sagafield(type="date")
    spl_label_ref       = sagafield(type="str")           #
-> ClassSPLDocument
    indications         = sagafield(type="list[str]")
    enclave_proof_refs  = sagafield(type="list[str]")
# efficacy/safety/stat summaries
    status                      = sagafield(type="str",
default="approved",
```

```
enum={"approved","suspended","withdrawn"})

@sagaclass()
class ClassEMAApproval(SPClassObject):
    """EMA centralized approval linked to ISO IDMP identifiers."""
    ema_id              = sagafield(type="str")
    idmp_mp_ref         = sagafield(type="str")          # -> ClassISO11615MedicinalProduct
    approval_date       = sagafield(type="date")
    product_name        = sagafield(type="str")
    epar_ref            = sagafield(type="str", default="")  # public assessment report link/oid
    harmonization_refs  = sagafield(type="list[str]")  # USP/Ph.Eur/JP alignments
    status              = sagafield(type="str", default="approved",

enum={"approved","suspended","withdrawn"})

@sagaclass()
class ClassWHOPrequalification(SPClassObject):
    """WHO prequalification for essential medicines/vaccines/diagnostics."""
    pq_id               = sagafield(type="str")
    idmp_mp_ref         = sagafield(type="str")          # -> ClassISO11615MedicinalProduct
    approval_scope      = sagafield(type="str")          # markets/programs
    validity_period     = sagafield(type="str")          # e.g., "2025-01-01..2028-12-31"
    compliance_proof_refs= sagafield(type="list[str]")
    status              = sagafield(type="str", default="valid",

enum={"valid","suspended","expired"})
```

B) Global Product aggregator (IDMP + GS1 + synchronized labels/approvals)

```
@sagaclass()
class ClassGlobalProduct(SPClassObject):
    """
    Borderless product record: binds ISO IDMP identity, GS1 codes,
    synchronized approvals (FDA/EMA/WHO), and live label references.
    """
    global_id           = sagafield(type="str")
    idmp_mp_ref         = sagafield(type="str")          # -> ClassISO11615MedicinalProduct
    idmp_pp_ref         = sagafield(type="str", default="")  # -> ClassISO11616PharmaceuticalProduct
    fda_approval_ref    = sagafield(type="str", default="")
    ema_approval_ref    = sagafield(type="str", default="")
    who_pq_ref          = sagafield(type="str", default="")
    ndc_root_ref        = sagafield(type="str", default="")
    gtin_ref            = sagafield(type="str", default="")  # -> ClassGS1GTIN
    sgtin_refs          = sagafield(type="list[str]")    # -> ClassGS1SGTIN
    active_label_refs   = sagafield(type="dict")         # {"US": oid, "EU": oid, "LMIC": oid}
    recall_event_refs   = sagafield(type="list[str]")    # -> ClassRecallEvent
    payer_onboarding_refs= sagafield(type="list[str]")   # -> ClassReimbursementDecision
    feeds_tag           = sagafield(type="str", default="")  # public index tag

    @sagamethod()
    def bind_identity(self, idmp_mp: str, gtin: str, idmp_pp: str = "", ndc_root: str = ""):
        self.idmp_mp_ref = idmp_mp; self.gtin_ref = gtin
        if idmp_pp: self.idmp_pp_ref = idmp_pp
        if ndc_root: self.ndc_root_ref = ndc_root
        return {"idmp_mp": idmp_mp, "gtin": gtin, "idmp_pp": self.idmp_pp_ref, "ndc": self.ndc_root_ref}

    @sagamethod()
    def attach_approvals(self, fda_oid: str = "", ema_oid: str = "", who_oid: str = ""):
        if fda_oid: self.fda_approval_ref = fda_oid
        if ema_oid: self.ema_approval_ref = ema_oid
        if who_oid: self.who_pq_ref = who_oid
        return {"FDA": self.fda_approval_ref, "EMA": self.ema_approval_ref, "WHO": self.who_pq_ref}

    @sagamethod()
    def sync_label(self, region: str, spl_oid: str):
        self.active_label_refs[region] = spl_oid
        return {"region": region, "label": spl_oid}

    @sagamethod()
    def add_instances(self, sgtins: list[str]):
        for s in sgtins or []:
            if s not in self.sgtin_refs: self.sgtin_refs.append(s)
        return {"sgtin_count": len(self.sgtin_refs)}

    @sagamethod()
    def note_recall(self, recall_oid: str):
        if recall_oid not in self.recall_event_refs:
            self.recall_event_refs.append(recall_oid)
```

```
        return {"recalls": len(self.recall_event_refs)}

    @sagamethod()
    def add_payer_onboarding(self, reimb_oid: str):
        if reimb_oid not in self.payer_onboarding_refs:

self.payer_onboarding_refs.append(reimb_oid)
        return {"onboardings":
len(self.payer_onboarding_refs)}
```

C) Private Enclave for HTA / pharmacoeconomics (attested outputs for payers)

```
@sagaclass(enclave=True)
class
ClassHTAEconomicModelEnclave(SPClassObject):
    """
    Confidential cost-effectiveness / budget-impact
models executed on encrypted inputs.
    Publishes attestations suitable for payer onboarding
decisions.
    """
    model_id            = sagafield(type="str")
    global_product_ref          =  sagafield(type="str")
# -> ClassGlobalProduct
    encrypted_inputs    = sagafield(type="list[bytes]")
# trial/RWE, cost tables, utilization
    attestations        = sagafield(type="list[str]")

    @sagamethod()
    def compute_and_attest(self, jurisdiction: str) ->
dict:
        att = "TEE:hta:" + rand_str(10)
        self.attestations.append(att)
        # Optionally push proof into a payer decision
object
        return {"attestation": att, "jurisdiction":
jurisdiction}
```

D) Payer onboarding & ISO 20022 settlements (evidence-conditioned)

```
@sagaclass()
class ClassPayerOnboarding(SPClassObject):
    """Bridge approvals + HTA proofs into coverage
decisions and settlement flows."""
    onboarding_id       = sagafield(type="str")
    payer_ref           = sagafield(type="str")          #
-> ClassBusinessAccount (payer)
    global_product_ref          =  sagafield(type="str")
# -> ClassGlobalProduct
    jurisdiction        = sagafield(type="str")          #
"US","EU","LMIC"
    hta_attestations    = sagafield(type="list[str]")    #
from HTA enclave
    reimbursement_ref           =  sagafield(type="str",
default="")  # -> ClassReimbursementDecision
    settlement_refs             =  sagafield(type="list[str]")
# -> ClassISO20022Settlement
    status                      =  sagafield(type="str",
default="pending",

enum={"pending","approved","denied","in_review"}
)

    @sagamethod()
    def evaluate(self) -> dict:
        approved = bool(self.hta_attestations and
self.global_product_ref)
        self.status = "approved" if approved else
"in_review"
        return {"approved": approved, "status":
self.status}

    @sagamethod()
    def trigger_settlements(self) -> int:
        """Release ISO 20022 flows if onboarding
approved."""
        released = 0
        if self.status == "approved":
            for s in self.settlement_refs or []:
                if deref(s).conditional_release():
                    released += 1
        return released
```

E) Public, transparency (SagaFeeds)

```
@sagaclass()
class ClassGlobalAccessFeed(SPClassObject):
    """SagaFeeds index of multi-jurisdiction approval +
label sync status (non-sensitive)."""
    feed_id             = sagafield(type="str")
    entries             = sagafield(type="list[dict]")  #
{"global_id":..., "FDA":..., "EMA":..., "WHO":...,
"labels":..., "ts":...}

    @sagamethod()
    def publish(self, global_id: str, fda: str, ema: str,
who: str, regions: list[str]):
        self.entries.append({
            "global_id": global_id, "FDA": fda, "EMA":
ema, "WHO": who,
            "labels": regions, "ts": str(now())
        })
        return {"count": len(self.entries)}
```

Interoperability Analysis

- **FD&C ↔ EMA (IDMP):** FDA NDA/BLA and EMA approvals are harmonized by **ISO IDMP** anchors

(idmp_mp_ref), aligning identity across regions.

- **WHO Prequalification:** PQ proofs become verifiable objects; LMIC regulators can **trust-but-verify** without duplicative reviews.
- **USP / Ph. Eur. / JP:** Compendial conformance (from Scenario 7) can be referenced in approvals to ensure **quality parity**.
- **SPL / Label sync:** Regional labels (active_label_refs["US"/"EU"/"LMIC"]) stay synchronized to approvals and recalls.
- **GS1 / DSCSA:** gtin_ref + sgtin_refs enable **unit-level traceability** post-approval in every market.
- **ISO 20022 & Reimbursement:** **Payer onboarding** binds to approvals and HTA attestation; **settlements** release automatically when coverage is approved.
- **Private Enclaves:** HTA models and proprietary datasets remain **encrypted-in-use**; only **attestations** flow to payers/regulators.
- **SagaFeeds:** Free, public index shows **where a product is approved**, **which labels are active**, and any **recalls** improving transparency.

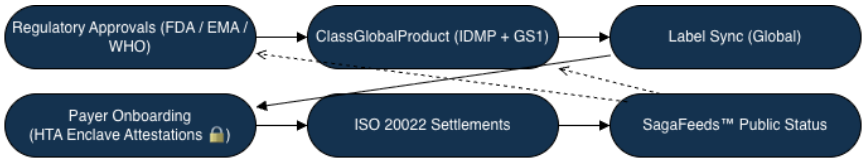


Diagram 5-I — Borderless Market Access
FDA/EMA/WHO approvals → ClassGlobalProduct (IDMP + GS1) → Label Sync → Payer Onboarding (HTA Enclave Attestations) → ISO 20022 Settlements → SagaFeeds Public Status
Borderless Market Access: global approvals synchronize into a single ClassGlobalProduct (IDMP + GS1), labels are harmonized across regions, payers onboard via HTA enclave attestations, settlements clear via ISO 20022, and SagaFeeds™ publishes public status. Dashed links show feedback updates.

Implications

- **Regulators:** Shared proofs reduce redundant review; faster cross-border synchronization.
- **Manufacturers: Submit once, prove everywhere** lower cost, faster access.
- **Payers/HTA:** Decisions tied to verified approvals and attested economics; automated settlement on approval.
- **Patients:** Faster, transparent access globally with harmonized quality and safety guarantees.
- **Global Health:** WHO PQ becomes **cryptographically verifiable**, improving equity in low-resource settings.

Scenario 10: Dynamic Lifecycle Management of Personalized & Precision Therapies

(CAR-T, gene therapies, mRNA platforms)

Background & Problem Statement

Personalized modalities (CAR-T, CRISPR gene therapies, mRNA platforms) stretch legacy compliance: patient-unique batches, complex **Chain of Identity (CoI)** and **Chain of Custody (CoC)** across many sites, rapid product iteration, sensitive genomic/clinical data, and novel reimbursement models.

Document-centric processes cannot maintain end-to-end provenance, continuous quality, or patient-level linkage.

Technical Approach (SagaChain with Private Enclaves)

- **Patient-specific objects:** CoI/CoC modeled as persistent objects that reference **GS1/DSCSA** identifiers and **EPCIS** events; PHI/genomics computed in **Private Enclaves** with selective disclosure.
- **Manufacturing & QC: GMP batch** objects extended for individualized runs; **USP assays** and **ISPE** validations run in enclaves; only pass/fail proofs exit.
- **Serialization & logistics: SGTIN/SSCC** + **EPCIS** events bind specimen collection → manufacture → shipment → infusion; DSCSA verification ensures unit-level custody.
- **Regulatory & labeling: FDA/EMA/WHO** approval objects link to **SPL**; patient-specific variations reference platform labels plus individualized annexes.
- **Finance: ISO 20022** flows execute outcomes-based or milestone payments; payer decisions are evidence-conditioned via enclave attestations; PHI remains protected.

Sample SagaPython Code

A) Patient-specific therapy lifecycle (enclave-aware)

```
@sagaclass(enclave=True)
class ClassPatientTherapy(SPClassObject):
    """Patient-specific therapy lifecycle: CoI/CoC, QC proofs, outcomes, and reimbursement."""
    therapy_id          = sagafield(type="str")
    patient_phi_ref     = sagafield(type="str")          # -> ClassPHI (HIPAA), enclave-protected
    therapy_type        = sagafield(type="str", enum={"CAR-T","Gene","mRNA"})
    indication_code     = sagafield(type="str", default="")   # e.g., ICD-10/SNOMED
    platform_label_ref  = sagafield(type="str", default="")   # -> ClassSPLDocument (platform)
    individualized_label = sagafield(type="str", default="")  # -> ClassSPLSection (patient-specific annex)
    coi_events          = sagafield(type="list[str]")    # -> ClassEPCISEvent (identity chain)
    coc_events          = sagafield(type="list[str]")    # -> ClassEPCISEvent (custody chain)
    ds_units            = sagafield(type="list[str]")    # -> ClassDSCSAUnit / ClassGS1SGTIN
    gmp_batch_ref       = sagafield(type="str", default="")   # -> ClassGMPBatch (individualized)
    usp_qc_proofs       = sagafield(type="list[str]")    # TEE attestations: potency/sterility/identity
    hl7_outcome_refs    = sagafield(type="list[str]")    # -> ClassFHIRResource (Observation/Condition)
    trial_ref           = sagafield(type="str", default="")   # -> ClassClinicalTrial (if applicable)
    reimbursement_ref   = sagafield(type="str", default="")   # -> ClassReimbursementDecision
    status              = sagafield(type="str", default="initiated",
                                    enum={"initiated","manufacturing","released","in_transit","administered","followup","complete","hold"})

    @sagamethod()
    def bind_identity(self, phi_oid: str, sgtin_oids: list[str], coi_oids: list[str], coc_oids: list[str]):
        self.patient_phi_ref = phi_oid
        for x in sgtin_oids or []:
            if x not in self.ds_units: self.ds_units.append(x)
        for e in coi_oids or []:
            if e not in self.coi_events: self.coi_events.append(e)
        for e in coc_oids or []:
            if e not in self.coc_events: self.coc_events.append(e)
        return {"phi": phi_oid, "units": len(self.ds_units), "coi": len(self.coi_events), "coc": len(self.coc_events)}

    @sagamethod()
    def set_manufacturing(self, gmp_batch_oid: str):
        self.gmp_batch_ref = gmp_batch_oid
        self.status = "manufacturing"
```

```
        return {"batch": self.gmp_batch_ref, "status": self.status}

    @sagamethod()
    def record_qc_proofs(self, proofs: list[str]):
        for p in proofs or []:
            if p not in self.usp_qc_proofs: self.usp_qc_proofs.append(p)
        return {"qc_proofs": len(self.usp_qc_proofs)}

    @sagamethod()
    def release_to_logistics(self, epcis_oids: list[str]):
        for e in epcis_oids or []:
            if e not in self.coc_events: self.coc_events.append(e)
        self.status = "in_transit"
        return {"events": len(self.coc_events), "status": self.status}

    @sagamethod()
    def administer(self, encounter_ref: str, outcomes: list[str]):
        # encounter_ref could be a FHIR Encounter/Procedure OID
        for o in outcomes or []:
            if o not in self.hl7_outcome_refs: self.hl7_outcome_refs.append(o)
        self.status = "administered"
        return {"outcomes": len(self.hl7_outcome_refs), "status": self.status}

    @sagamethod()
    def enter_followup(self):
        self.status = "followup"; return {"status": self.status}
```

B) Enclave QC for individualized potency/sterility/identity (USP/ISPE proofs)

```
@sagaclass(enclave=True)
class ClassPrecisionQCEnclave(SPClassObject):
    """Confidential QC for precision therapies; emits attestations only."""
    qc_id               = sagafield(type="str")
    therapy_ref         = sagafield(type="str")          # -> ClassPatientTherapy
    encrypted_methods                                    = sagafield(type="list[bytes]")      # SOPs, parameters
    encrypted_results   = sagafield(type="list[bytes]")  # raw data (flow, NGS, potency assays)
    attestations        = sagafield(type="list[str]")

    @sagamethod()
    def run_and_attest(self, targets: list[str]) -> dict:
        # e.g., ["USP.potency>=X", "USP.sterility=pass", "Identity.vector=Y"]
        att = "TEE:precision_qc:" + rand_str(10)
        self.attestations.append(att)
        therapy = deref(self.therapy_ref)
        therapy.record_qc_proofs([att])
        return {"attestation": att, "targets": targets}
```

C) Outcomes-based reimbursement (payer coverage + ISO 20022 execution)

```
@sagaclass()
class ClassOutcomesContract(SPClassObject):
    """Outcomes-based reimbursement contract for precision therapies."""
    contract_id         = sagafield(type="str")
    therapy_ref         = sagafield(type="str")          # -> ClassPatientTherapy
    payer_ref           = sagafield(type="str")          # -> ClassBusinessAccount (payer)
    provider_ref        = sagafield(type="str")          # -> ClassBusinessAccount (provider/center)
    outcome_rules               = sagafield(type="dict")
#       {"response_rate>=CR/PR@Day30": 1.0, "durable@Month6": 0.5}
    escrow_payment_ref          = sagafield(type="str", default="")  # -> ClassISO20022Payment (escrow)
    milestone_settlements  = sagafield(type="list[str]")  # -> ClassISO20022Settlement per milestone
    status                      = sagafield(type="str", default="active",
enum={"active","settled","terminated"})

    @sagamethod()
    def evaluate_outcomes(self, fhir_outcome_oids: list[str]) -> dict:
        """
        Illustrative logic: map FHIR Observations/Conditions to rule satisfaction,
        then release corresponding ISO 20022 settlements.
        """
        satisfied = {"Day30": True, "Month6": False}
        released = 0
        for s in self.milestone_settlements:
            stl = deref(s)
            if hasattr(stl, "conditional_release") and satisfied.get(stl.value_date, False):
                if stl.conditional_release(): released += 1
        if released and not satisfied.get("Month6"):
            self.status = "active"
        elif all(satisfied.values()):
            self.status = "settled"
        return {"released": released, "status": self.status, "satisfied": satisfied}
```

D) Global platform object for rapid mRNA iteration (label/approval sync)

```
@sagaclass()
class ClassPlatformTherapy(SPClassObject):
   """Platform-level object (e.g., mRNA) coordinating label/approval sync across variants."""
   platform_id          = sagafield(type="str")
   modality             = sagafield(type="str", enum={"mRNA","ViralVector","GeneEdit"})
   ema_idmp_mp_ref      = sagafield(type="str", default="")  # -> ClassISO11615MedicinalProduct (platform)
   variants             = sagafield(type="list[str]")  # list of ClassPatientTherapy or product variants
   active_label_refs    = sagafield(type="dict")  # {"US": oid, "EU": oid}
   approvals            = sagafield(type="list[str]")  # FDA/EMA/WHO approval OIDs

   @sagamethod()
   def register_variant(self, therapy_oid: str):
      if therapy_oid not in self.variants:
         self.variants.append(therapy_oid)
      return {"variant_count": len(self.variants)}

   @sagamethod()
   def sync_platform_label(self, region: str, spl_oid: str):
      self.active_label_refs[region] = spl_oid
      return {"region": region, "label": spl_oid}
```

Interoperability Analysis

- **21 CFR / FD&C Act:** IND/BLA oversight references **ClassPatientTherapy**; **Part 11** integrity via signed, immutable methods; individualized **GMP batch** linked to release gates.
- **USP / ISPE:** Potency/sterility/identity tested in **enclaves**; **ISPE** verification artifacts (IQ/OQ/PQ) tie to individualized manufacturing lines.
- **DSCSA / GS1: SGTIN/SSCC** + **EPCIS** anchor **CoI/CoC** across facilities; DSCSA verification handles patient-specific serialized units.
- **HL7 / FDAAA 801 / CDISC: FHIR** outcomes (e.g., response rates) flow into therapy objects and **outcomes contracts**; trials on platforms (mRNA/gene) link datasets to labeling and payer evidence.
- **ISO 20022: Escrow and milestone settlements** release programmatically upon outcome satisfaction; clawback/hold supported.
- **HIPAA / HITECH:** PHI/genomic payloads remain **encrypted-in-use**; only proofs & aggregates exit; breach events flow to **HITECH** objects.
- **ISO IDMP / SPL:** Platform and variants bind to **IDMP** identity; **SPL** sections synchronize platform label with individualized annexes.

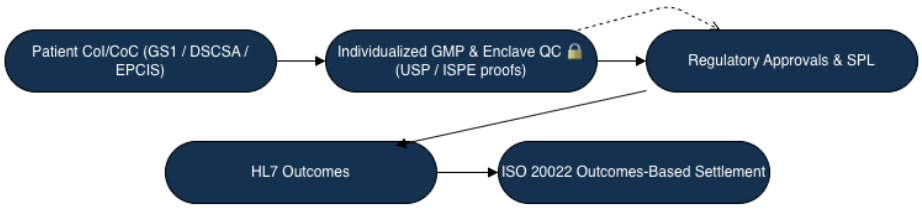


Diagram 5-J — Precision Therapy Lifecycle

Patient CoI/CoC (GS1/DSCSA/EPCIS) → Individualized GMP & Enclave QC (USP/ISPE proofs) → Regulatory Approvals & SPL → HL7 Outcomes → ISO 20022 Outcomes-Based Settlement

Precision Therapy Lifecycle: patient Chain-of-Identity/Chain-of-Custody (GS1/DSCSA/EPCIS) flows into individualized GMP with enclave QC proofs (USP/ISPE), drives regulatory approvals & SPL, records HL7 outcomes, and finalizes outcomes-based settlement via ISO 20022.
Dashed link = proof attestation path.

Implications

- **Regulators:** Real-time, patient-specific oversight without direct PHI/genomics exposure; cryptographic QC proofs.
- **Manufacturers:** Unified lifecycle for individualized batches; faster release; lower audit friction.
- **Payers:** Automated, **evidence-conditioned** reimbursement; reduced financial risk via escrow/milestones.
- **Patients:** Secured provenance, faster access, and outcome-linked affordability.
- **Global Health:** Platform-level harmonization (IDMP/GS1) accelerates safe adoption across jurisdictions.

Scenario 10 demonstrates how persistent objects + private enclaves turn personalized medicine into a verifiable, automated compliance lifecycle linking **identity, quality, safety, labeling, and reimbursement** end-to-end.

Scenario 11: Integrated Real-World Evidence (RWE) & Post-Market Surveillance

(linking HL7, FAERS/VAERS, NIH DMSP, and payer datasets)

Background & Problem Statement

RWE from **EHRs (HL7 FHIR)**, **claims**, **registries**, **devices**, and **patient-reported outcomes** is essential for post-market safety, label refinement, and reimbursement, but today it is siloed, slow, and hard to audit:

- **Data silos:** FHIR, FAERS/VAERS, DMSP repositories, and payer claims live apart.
- **Latency & bias:** Quarterly refreshes, inconsistent coding (ICD, SNOMED, RxNorm), and missingness.
- **Opaque provenance:** Weak lineage from sources → features → models → outputs.
- **Privacy risk:** PHI and financial data move across pipelines.
- **Weak feedback loops:** Signals rarely update **SPL**, recalls, or payer policy programmatically.

Technical Approach (SagaChain with Private Enclaves)

1. **Unified Evidence Graph (Persistent Objects)**
   - **HL7 FHIR** → ClassFHIRResource (Encounter/Medication/Observation/Outcome).
   - **Safety** → ClassFAERSReport / ClassVaccineAdverseEvent.
   - **Research** → ClassDatasetObject (NIH DMSP) for cohorts/features.
   - **Payer** → ClassClaimsBundle (adjudicated claims & utilization).

     All cross-linked by **ISO IDMP / NDC / GS1 SGTIN**, patient context (enclave-protected), time, and care setting.
2. **Private Enclaves for Confidential Computation**
   - Person-level joins (EHR ↔ claims ↔ registries), de-identification, risk adjustment, and model training run **encrypted-in-use**.

- Only **cohort specs**, **summary stats**, **effect sizes**, **CIs**, and **TEE/zk attestations** exit.

3. **Continuous Analytics & Signal Governance**
   - ClassRWEMonitor runs pre-specified methods (incidence, SCCS, target-trial) on rolling windows.
   - ClassRWEResult publishes auditable summaries; ClassSafetySignal encodes thresholds/explanations and triggers actions (SPL update, DSCSA hold/recall, payer policy).
4. **Provenance & Reproducibility**
   - Every dataset, transform, model, and result has a **content-addressed hash** and **versioned lineage**.
   - **SagaFeeds** publishes non-sensitive metadata for free discovery.

Sample SagaPython Code

A) Continuous RWE monitor (Private Enclave)

```
@sagaclass(enclave=True)
class ClassRWEMonitor(SPClassObject):
    """Continuous RWE monitor executing on enclave-protected EHR/claims/safety data."""
    monitor_id        = sagafield(type="str")
    scope             = sagafield(type="dict")      # {"products":["IDMP:11615:..."], "indications":["ICD10:C50"], "markets":["US","EU"]}
    cohort_def        = sagafield(type="dict")      # computable phenotype / inclusion-exclusion
    methods           = sagafield(type="list[str]")# ["incidence_rate","sccs","target_trial_emulation"]
    data_refs         = sagafield(type="list[str]")# OIDs: FHIR, FAERS/VAERS, DMSP datasets, claims bundles
    update_interval   = sagafield(type="str")     # ISO-8601 duration, e.g., "P7D"
    last_attestation  = sagafield(type="str", default="")
    last_metrics      = sagafield(type="dict", default={})
    feeds_tag         = sagafield(type="str", default="") # public index key (no PHI)

    @sagamethod()
    def bind_data(self, oids: list[str]) -> dict:
        for oid in oids or []:
            if oid not in self.data_refs:
                self.data_refs.append(oid)
        return {"bound": len(self.data_refs)}

    @sagamethod()
    def run_window(self, window_start: str, window_end: str) -> dict:
        """
        Execute pre-registered analyses on encrypted sources; publish attested summary only.
        """
        att = "TEE:rwe:" + rand_str(10)
        # Illustrative metrics; real logic would compute within the enclave.
        self.last_attestation = att
        self.last_metrics = {"n_patients": 128734, "n_events": 436, "coverage": 0.91,
                             "window": [window_start, window_end], "methods": self.methods}
        # Create/publish a result object (below)…
        return {"attestation": att, "metrics": self.last_metrics}
```

B) Published RWE results (no PHI), with linkage to signals

```
@sagaclass()
class ClassRWEResult(SPClassObject):
    """Published RWE summary from enclave analysis (no PHI)."""
    result_id         = sagafield(type="str")
    monitor_ref       = sagafield(type="str")       # -> ClassRWEMonitor
    estimate          = sagafield(type="dict")      # {"RR":1.32,"CI":[1.12,1.55],"p":0.004}
    cohort_stats      = sagafield(type="dict")      # counts, follow-up time, covariate balance
    method_meta       = sagafield(type="dict")      # model ver., features, diagnostics
    linked_signal_refs  = sagafield(type="list[str]")  # -> ClassSafetySignal
    provenance_hash   = sagafield(type="str")       # content-address of inputs/transforms

    @sagamethod()
    def link_signal(self, signal_oid: str):
        if signal_oid not in self.linked_signal_refs:
            self.linked_signal_refs.append(signal_oid)
```

```
        return {"signals": len(self.linked_signal_refs)}
```

C) Actionable safety signal (with downstream triggers)

```
@sagaclass()
class ClassSafetySignal(SPClassObject):
    """Actionable RWE signal."""
    signal_id          = sagafield(type="str")
    product_id         = sagafield(type="str")          # IDMP/NDC
    lot                = sagafield(type="str", default="")
    severity           = sagafield(type="str", enum={"info","watch","action"}, default="watch")
    explanation        = sagafield(type="dict")         # top features/contexts
    suggested_actions  = sagafield(type="list[str]")    # ["label_update","recall_scope","prior_auth","risk_mitigation"]
    status             = sagafield(type="str", enum={"open","acknowledged","mitigated","closed"}, default="open")
    created_at         = sagafield(type="datetime", default=None)
    updated_at         = sagafield(type="datetime", default=None)

    @sagamethod()
    def open(self, severity: str, explanation: dict, actions: list[str]):
        self.severity = severity
        self.explanation = explanation
        self.suggested_actions = list(dict.fromkeys(actions or []))
        self.created_at = now(); self.updated_at = self.created_at
        self.status = "open"
        return {"status": self.status, "actions": self.suggested_actions}

    @sagamethod()
    def resolve(self, action_notes: dict):
        self.status = "mitigated"; self.updated_at = now()
        return {"status": self.status, "notes": action_notes}
```

D) Claims bundle (payer utilization), enclave-friendly reference

```
@sagaclass()
class ClassClaimsBundle(SPClassObject):
    """Payer claims/utilization bundle (metadata only; payload processed in enclaves)."""
    bundle_id          = sagafield(type="str")
    payer_ref          = sagafield(type="str")          # -> ClassBusinessAccount
    coverage_period    = sagafield(type="str")          # "2024-01..2024-12"
    deid_payload_ref   = sagafield(type="str")          # pointer to encrypted store
    product_ids        = sagafield(type="list[str]")    # IDMP/NDC
```

E) SagaFeeds: public, index for RWE transparency

```
@sagaclass()
class ClassRWEFeed(SPClassObject):
    """SagaFeeds index for non-sensitive RWE metadata (discoverable, no PHI)."""
    feed_id            = sagafield(type="str")
    entries            = sagafield(type="list[dict]")   # {"result_id":..., "product":..., "RR":..., "CI":..., "ts":...}

    @sagamethod()
    def publish(self, result_id: str, product_id: str, rr: float, ci: list[float]):
        self.entries.append({"result_id": result_id, "product": product_id, "RR": rr, "CI": ci, "ts": str(now())})
        return {"count": len(self.entries)}
```

Interoperability Analysis

- **HL7 ↔ FAERS/VAERS:** Patient outcomes (FHIR) correlate with safety events at **product/lot** level via **GS1/DSCSA**, improving specificity.
- **NIH DMSP:** Cohorts/derived datasets are registered as ClassDatasetObject (public metadata; private payload), meeting sharing mandates without PHI leakage.
- **SPL (Labeling):** Significant ClassSafetySignal can auto-draft **SPL** warnings/contraindications with cryptographic provenance to ClassRWEResult.
- **21 CFR / FD&C Act:** Evidence and decisions persist with **Part 11-grade** signatures and immutable lineage.
- **ISO IDMP / GS1 / DSCSA:** Harmonized identity links signals to products and serialized units across borders.

- **ISO 20022 (Payers):** Coverage and utilization management (e.g., **prior auth**, **outcomes contracts**) update programmatically alongside validated RWE.
- **HIPAA/HITECH & Enclaves:** Person-level joins and model runs occur in **Private Enclaves**; public outputs are **de-identified** and **attested**.

Governance, Methods, and Bias Control

- **Pre-registration:** Monitors require protocol objects (eligibility, endpoints, covariates).
- **Causal guardrails: Target-trial emulation** templates promote causal interpretation.
- **Heterogeneity & multiplicity:** Prespecified subgroup scans with multiple-testing control.
- **MRM:** External validation, drift, fairness, and calibration diagnostics persisted within ClassRWEResult and auditable.

Outcomes & Benefits

- **Faster learning:** Rolling windows reduce time-to-signal from months to days.
- **Actionable transparency:** Public indices reveal **what/when/how** was monitored without PHI.
- **Closed-loop:** Signals drive **SPL updates**, **DSCSA actions**, and **payer policy** in near-real-time.
- **Global alignment:** IDMP/GS1 anchors enable cross-jurisdiction coordination.
- **Lower total cost:** Automation + reproducibility trim analytics and inspection overhead.

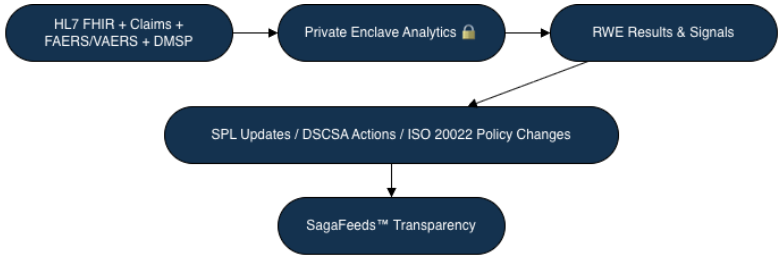


Diagram 5-K — Integrated RWE

HL7 FHIR + Claims + FAERS/VAERS + DMSP → Private Enclave Analytics → RWE Results & Signals → SPL Updates / DSCSA Actions / ISO 20022 Policy Changes → SagaFeeds Transparency

Integrated Real-World Evidence: clinical, claims, and pharmacovigilance data (HL7 FHIR, FAERS/VAERS, DMSP) feed into Private Enclave Analytics to generate RWE results and regulatory signals. Outputs update SPL, DSCSA, and ISO 20022 layers, published transparently via SagaFeeds™.

Scenario 12: Unified Counterfeit & Diversion Defense

(real-time anomaly detection across GS1 EPCIS, DSCSA verification, customs data, and ISO 20022 trade flows)

Background & Problem Statement

Despite DSCSA serialization and GS1 EPCIS 2.0, sophisticated actors still slip falsified or diverted product through gaps: fragmented event stores, point-in-time verifications without trajectory context, border blind spots, and financial flows not bound to provenance. The result is delayed detection, reactive response, and settlements of fraudulent trade before risk is known.

Technical Approach (SagaChain with Private Enclaves)

1) Unified Provenance Graph (Persistent Objects)

- **Identifiers:** ClassGS1GTIN, ClassGS1SGTIN, ClassGS1SSCC map to ClassDSCSAUnit and **ISO IDMP** IDs.
- **Events:** ClassEPCISEvent preserves EPCIS 2.0 JSON-LD for commissioning/aggregation/shipping/etc.
- **Verification:** ClassDSCSAVerification binds cryptographic responses (pass/fail, signer, times).
- **Trade & Customs:** ClassCustomsDeclaration, ClassPortInspection encode HS codes, ports, outcomes.
- **Finance:** ClassISO20022Payment / ClassISO20022Settlement link to lots/shipments.

2) Adaptive Risk Analytics (Private Enclaves)

Graph/sequence models (route conformance, dwell anomalies, suspicious re-aggregation) run in TEEs over EPCIS, verification, customs, and banking messages. Outputs are **attested anomaly scores**, not raw commercial/PHI data.

3) Programmable Controls & Deterrence

- **Scan-to-Hold:** ClassSupplyHold freezes suspect SGTIN/SSCCs at next read point.
- **Payment Actions:** ISO 20022 flows move to **escrow** / **clawback** via settlement updates when risk ≥ threshold.
- **Regulatory Alerts:** ClassComplianceAlert notifies MAHs, wholesalers, authorities; **SagaFeeds** can publish high-severity bulletins for public verification.
- **Chain Repair:** Multi-party re-verification re-establishes pedigree or confirms fraud.

Sample SagaPython Code

A) Customs & Port artifacts

```
@sagaclass()
class ClassCustomsDeclaration(SPClassObject):
    """Customs declaration linked to serialized pharma shipments."""
    decl_id          = sagafield(type="str")
    sscc             = sagafield(type="str")          # -> ClassGS1SSCC
    hs_code          = sagafield(type="str")          # WCO HS
    exporter_gln     = sagafield(type="str")
    importer_gln     = sagafield(type="str")
    port_code        = sagafield(type="str")
    declaration_time = sagafield(type="datetime")
    outcome          = sagafield(type="str", enum={"accepted","inspected","rejected"})
    document_hashes  = sagafield(type="list[str]", default=[])

@sagaclass()
class ClassPortInspection(SPClassObject):
    """Port-of-entry/exit inspection event."""
    insp_id          = sagafield(type="str")
    sscc             = sagafield(type="str")
    port_code        = sagafield(type="str")
    time             = sagafield(type="datetime")
    result           = sagafield(type="str", enum={"pass","fail","hold"})
    notes            = sagafield(type="str", default="")
```

B) DSCSA verification outcome (contextual, not just point-in-time)

```
@sagaclass()
class ClassDSCSAVerification(SPClassObject):
    """Verification response for DSCSA SGTIN/lot."""
    verification_id  = sagafield(type="str")
    sgtin            = sagafield(type="str")          # -> ClassGS1SGTIN
    lot              = sagafield(type="str")
    request_time     = sagafield(type="datetime")
    response_time    = sagafield(type="datetime")
    result           = sagafield(type="str", enum={"pass","fail","partial"})
```

```
    signer          = sagafield(type="str")        # GLN or DID
    context_refs    = sagafield(type="list[str]", default=[])  # EPCIS/custody/payment refs

    @sagamethod()
    def add_context(self, oids: list[str]):
        for x in oids or []:
            if x not in self.context_refs: self.context_refs.append(x)
        return {"context_count": len(self.context_refs)}
```

C) Enclave monitor for counterfeit/diversion analytics

```
@sagaclass(enclave=True)
class ClassAntiCounterfeitMonitor(SPClassObject):
    """Enclave analytics on EPCIS, DSCSA, customs, and ISO 20022 signals."""
    monitor_id      = sagafield(type="str")
    scope           = sagafield(type="dict")       # {"routes":["CN->EU->US"], "products":["GTIN..."]}
    feature_refs    = sagafield(type="list[str]", default=[]) # EPCIS, verifications, customs, payments
    thresholds      = sagafield(type="dict", default={"route_dev":0.8,"reagg":0.7,"customs_mismatch":0.9})
    last_attestation = sagafield(type="str", default="")
    last_scores     = sagafield(type="dict", default={})

    @sagamethod()
    def bind_features(self, oids: list[str]) -> dict:
        for oid in oids or []:
            if oid not in self.feature_refs: self.feature_refs.append(oid)
        return {"features": len(self.feature_refs)}

    @sagamethod()
    def evaluate(self) -> dict:
        # Enclave computes anomaly scores; return attested summary only.
        att = "TEE:antiCF:" + rand_str(10)
        scores = {"route_dev": 0.86, "reagg": 0.22, "customs_mismatch": 0.12, "overall": 0.84}
        self.last_attestation, self.last_scores = att, scores
        return {"attestation": att, "scores": scores}
```

D) Programmable hold control (scan-to-hold)

```
@sagaclass()
class ClassSupplyHold(SPClassObject):
    """Programmable hold on suspect serialized units."""
    hold_id         = sagafield(type="str")
    subjects        = sagafield(type="list[str]")  # SGTIN/SSCC OIDs
    reason          = sagafield(type="str")
    created_at      = sagafield(type="datetime")
    status          = sagafield(type="str", enum={"active","lifted"}, default="active")

    @sagamethod()
    def lift(self, note: str=""):
        self.status = "lifted"
        return {"status": self.status, "note": note}
```

E) Finance synchronization (escrow/clawback) + alerting

```
@sagaclass()
class ClassCounterfeitFinanceGuard(SPClassObject):
    """Finance guard linking anomaly level to ISO 20022 settlements."""
    guard_id        = sagafield(type="str")
    settlement_refs = sagafield(type="list[str]", default=[])  # -> ClassISO20022Settlement
    escrow_on_score = sagafield(type="float", default=0.8)
    clawback_on_score = sagafield(type="float", default=0.9)

    @sagamethod()
    def apply(self, score: float) -> dict:
        actions = []
        for oid in self.settlement_refs:
            stl = deref(oid)
            if score >= self.clawback_on_score and hasattr(stl, "clawback"):
                stl.clawback(); actions.append({"settlement": oid, "action": "clawback"})
            elif score >= self.escrow_on_score and hasattr(stl, "escrow"):
                stl.escrow(); actions.append({"settlement": oid, "action": "escrow"})
        return {"score": score, "actions": actions}

@sagaclass()
class ClassComplianceAlert(SPClassObject):
    """Regulatory/industry alert with optional public SagaFeeds notice."""
    alert_id        = sagafield(type="str")
    severity        = sagafield(type="str", enum={"info","watch","action","critical"}, default="action")
    subjects        = sagafield(type="list[str]", default=[])  # SGTIN/SSCC/lot
    explanation     = sagafield(type="dict", default={})  # model attributions, route deltas
```

```
    recipients = sagafield(type="list[str]", default=[])  # GLNs/DIDs (MAH, wholesaler, authority)
    feeds_posted = sagafield(type="bool", default=False)

    @sagamethod()
    def post_public_bulletin(self, feed_oid: str):
        feed = deref(feed_oid)  # -> SagaFeeds index class
        if hasattr(feed, "publish"):
            feed.publish(self.alert_id, {"severity": self.severity, "subjects": self.subjects})
            self.feeds_posted = True
        return {"posted": self.feeds_posted}
```

Interoperability Analysis

- **GS1 EPCIS ↔ DSCSA:** Every EPCIS event is anchored to a DSCSA unit; sequence/route anomalies expose clone/reuse and phantom aggregation.
- **Customs/Ports:** Declarations & inspections are first-class; HS-route mismatches vs EPCIS paths raise high-confidence alerts.
- **ISO IDMP / FD&C / EMA:** Global product identity aligns cross-border investigations on a shared identifier backbone.
- **ISO 20022 (Finance):** Settlements become **conditional** on unimpeached provenance; anomalies **escrow** or **claw back** funds automatically.
- **USP / Quality:** Lots with compromised pedigree require re-testing; failed assays escalate to ClassRecallEvent with targeted scope.
- **Private Enclaves / Confidentiality:** Commercial terms remain encrypted; only **attested risk scores** exit.
- **SagaFeeds (public):** High-severity bulletins allow pharmacies/patients to scan & verify legitimacy at the point of dispense.

Threat Models & Controls

- **Identifier cloning/reuse:** Detect infeasible concurrency, route deviation, re-commission after decommission.
- **Phantom aggregation/de-aggregation:** Enforce acyclic, time-consistent parent/child SSCC/SGTIN graphs.
- **Gray-market diversion:** Correlate ISO 20022 counterparties with EPCIS paths; programmable pricing clauses re-price or void settlements.
- **Free-trade zone injection:** Correlate port movement with EPCIS reads; unobserved hops trigger multi-party challenges before re-entry.

Outcomes & Benefits

- **Patient Safety:** Interdict suspect stock **before** exposure.
- **Regulatory Efficiency:** Shared anomaly proofs reduce duplicate investigations.
- **Financial Deterrence:** Escrow/clawbacks make diversion economically unattractive.
- **Operational Speed:** Scan-to-hold + guided re-verification compress incident cycles from weeks to hours.
- **Global Consistency:** IDMP/GS1 anchoring eliminates cross-border remapping friction.

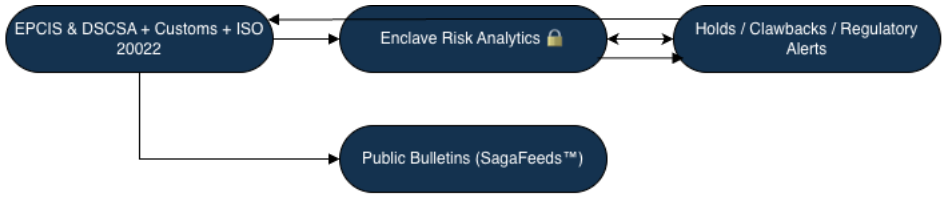


Diagram 5-L — Unified Counterfeit & Diversion Defense

EPCIS & DSCSA + Customs + ISO 20022 → Enclave Risk Analytics → Holds / Clawbacks / Regulatory Alerts → Public Bulletins (SagaFeeds)

Unified Counterfeit & Diversion Defense: serialized EPCIS/DSCSA + Customs and ISO 20022 cues feed enclave analytics; actions (holds, clawbacks, regulatory alerts) publish via SagaFeeds™.

Scenario 13: Sustainability & ESG Accountability in Pharma Supply

(tracking carbon, cold-chain efficiency, and ethical sourcing with ISO 20022 green finance hooks)

Background & Problem Statement

Global pharma supply chains are energy-intensive, temperature-sensitive, and multi-tiered (APIs, excipients, packaging, logistics). Stakeholders demand verifiable ESG performance, but today it is largely **retrospective** and **document-centric**:

- **Carbon & energy:** Scope 1–3 attribution is error-prone and often self-attested without verifiable lineage.
- **Cold-chain integrity:** Telemetry is fragmented across carriers/loggers; cryptographic authenticity is rare.
- **Ethical sourcing:** Labor/biodiversity commitments live in PDFs detached from serialized flows.
- **Finance disconnect:** Sustainability KPIs rarely affect payment terms; "green" settlements are not programmatically tied to operations.

Technical Approach (SagaChain with Private Enclaves)

SagaChain turns ESG from static reports into **live, programmable accountability** bound to **serialized product units** and **financial flows**.

1) Verifiable ESG Objects (Persistent State)

- ClassCarbonLedger records attributable emissions by lifecycle stage (API, Fill-Finish, Pack, Transport, Warehouse, Retail) with evidence hashes.
- ClassColdChainTelemetry ingests **signed** sensor streams (time/temperature/humidity/shock), preserving EPCIS 2.0 JSON-LD and device/carrier signatures.
- ClassESGAttestation encodes third-party certifications (labor, forestry, etc.) with validity windows.
- ClassSustainabilityPolicy holds machine-readable constraints (e.g., max $kgCO_2e$/unit, excursion tolerances) and incentive rules.

2) Binding to Product Identity & Events

- **GS1 / DSCSA:** ClassGS1SGTIN and ClassDSCSAUnit reference ESG objects; EPCIS events accrue carbon deltas and attach telemetry proofs at **unit** or **pallet** granularity.

- **ISO IDMP:** Global medicinal product identity allows consistent ESG addressability across jurisdictions.

3) Private Enclaves for Sensitive Data

- Energy invoices, rates, contracts, and exact site coordinates remain encrypted-in-use; enclaves output attested KPIs (e.g., "lane X = 0.42 $kgCO_2e$/unit") and selective proofs.

4) Finance Hooks (ISO 20022 "Green" Flows)

- ClassGreenSettlement extends settlements with **discounts/penalties/escrows** triggered by enclave-attested KPIs from Carbon/Cold-Chain/Attestation objects.
- Sustainability-linked supply-chain finance (e.g., payables discounting) is executed programmatically on proof of performance.

Sample SagaPython Code

```
@sagaclass()
class ClassCarbonLedger(SPClassObject):
   """Attributable emissions per lifecycle stage bound to serialized units or lots."""
   ledger_id       = sagafield(type="str")
   scope           = sagafield(type="str", enum={"S1","S2","S3"})
   stage           = sagafield(type="str", enum={"API","FillFinish","Pack","Transport","Warehouse","Retail"})
   method          = sagafield(type="str")   # GHG Protocol, DEFRA, supplier-specific, etc.
   kg_co2e         = sagafield(type="float")
   evidence_hashes = sagafield(type="list[str]", default=[]) # M&V docs, invoice hashes
   subjects        = sagafield(type="list[str]", default=[]) # OIDs for ClassGS1SGTIN or lot IDs
   period_start    = sagafield(type="date")
   period_end      = sagafield(type="date")

   @sagamethod()
   def add_subjects(self, oids: list[str]) -> int:
      for oid in oids or []:
         if oid not in self.subjects:
            self.subjects.append(oid)
      return len(self.subjects)


@sagaclass()
class ClassColdChainTelemetry(SPClassObject):
   """Signed time-series from data loggers and carriers (EPCIS 2.0 JSON-LD preserved)."""
   stream_id       = sagafield(type="str")
   device_id       = sagafield(type="str")
   signed_payload  = sagafield(type="dict")   # {"events":[{"t": "...", "temp": 4.1, "rh": 65, "shock":0, "sig":"..."}]}
   summary         = sagafield(type="dict", default={}) # {"min":2.8,"max":7.1,"excursions":0,"MKT":4.5}
   subjects        = sagafield(type="list[str]", default=[]) # SGTIN/SSCC OIDs
   attestation     = sagafield(type="str")   # signature of carrier/logger

   @sagamethod()
   def summarize(self) -> dict:
      # Example placeholder; real calc derives from signed_payload in enclave/logging pipeline
      self.summary = {"min": 2.8, "max": 7.1, "excursions": 0, "MKT": 4.5}
      return self.summary


@sagaclass()
class ClassESGAttestation(SPClassObject):
   """Third-party certification or due diligence attestation."""
   attest_id       = sagafield(type="str")
   scheme          = sagafield(type="str")   # Rainforest Alliance, SA8000, etc.
   issuer          = sagafield(type="str")
   validity        = sagafield(type="dict")  # {"from":"2025-01-01","to":"2027-12-31"}
   scope           = sagafield(type="dict")  # {"material":"palm","site":"...","supplier":"..."}
   evidence_hashes = sagafield(type="list[str]", default=[])


@sagaclass()
class ClassSustainabilityPolicy(SPClassObject):
   """Machine-readable ESG constraints and incentives."""
   policy_id       = sagafield(type="str")
   rules           = sagafield(type="dict")  # {"kg_co2e_per_unit_max":0.6,"excursion_minutes_max":15}
```

```
    incentives = sagafield(type="dict") # {"discount_bps":20,"penalty_bps":30}
    applies_to = sagafield(type="list[str]", default=[])  # product IDs, lots, suppliers

    @sagamethod()
    def applies(self, subject_id: str) -> bool:
        return (subject_id in self.applies_to) or (not self.applies_to)


@sagaclass()
class ClassGreenSettlement(SPClassObject):
    """ISO 20022 settlement extended with ESG conditions."""
    settlement_id     = sagafield(type="str")
    base_payment      = sagafield(type="float")
    currency          = sagafield(type="str", length=3)
    payer             = sagafield(type="str")       # GLN/DID
    payee             = sagafield(type="str")
    linked_units      = sagafield(type="list[str]", default=[])  # SGTIN/SSCC OIDs
    esg_refs          = sagafield(type="list[str]", default=[])  # CarbonLedger, ColdChainTelemetry, ESGAttestation
    policy_ref        = sagafield(type="str", default="")  # -> ClassSustainabilityPolicy
    adjustment        = sagafield(type="dict", default={"applied": False, "delta": 0.0})
    status            = sagafield(type="str", enum={"pending","settled","disputed"}, default="pending")

    @sagamethod()
    def evaluate_and_apply(self) -> dict:
        """
        Evaluate ESG proofs for linked units vs policy; apply discount/penalty (bps) and set adjustment.
        Only uses attested KPIs (e.g., enclave-verified carbon intensity, excursion minutes).
        """
        policy = deref(self.policy_ref) if self.policy_ref else None
        if not policy:
            return {"applied": False, "reason": "no_policy"}
        # Illustrative decision logic:
        # Assume upstream enclave posted KPIs to an aggregated CarbonLedger & telemetry summary for the linked units.
        carbon_ok = True
        cold_ok   = True
        # If any KPI violates rules, set penalty; else apply discount.
        if carbon_ok and cold_ok:
            bps = (policy.incentives or {}).get("discount_bps", 0) / 10000.0
            self.adjustment = {"applied": True, "delta": -bps}
        else:
            bps = (policy.incentives or {}).get("penalty_bps", 0) / 10000.0
            self.adjustment = {"applied": True, "delta": +bps}
        return {"applied": self.adjustment["applied"], "delta": self.adjustment["delta"]}
```

Interoperability Analysis

- **GS1 / EPCIS / DSCSA:** Telemetry and carbon deltas attach to the **same serialized units** used for pedigree and recall, enabling **unit-level ESG provenance**. Aggregation propagates attributes SSCC→SGTIN with auditable roll-ups.
- **ISO IDMP / SPL:** ESG performance may surface as **labeling supplements** (e.g., via GS1 Digital Link targets), improving patient/provider transparency.
- **ISPE / 21 CFR:** Utility consumption and process efficiency link to validation and CPV; verified energy/waste reductions become **CQI** outcomes.
- **USP:** Cold-chain excursions tied to stability specs **gate batch release** or trigger targeted re-testing.
- **ISO 20022:** ClassGreenSettlement applies **discounts/penalties/escrows** based on **attested** KPIs (carbon intensity, excursion minutes, valid ethical attestations); supports **sustainability-linked supply-chain finance**.
- **Private Enclaves:** Commercial data (rates, invoices) remains encrypted-in-use; public consumers see only **attested KPIs** and proofs via **SagaFeeds** (no-cost indices).

Example ESG Flow (End-to-End)

1. **Manufacture:** Fill-finish energy data → ClassCarbonLedger (S1/S2) with evidence hashes; ISPE validation references updated.
2. **Pack & Commission:** SGTINs minted; ClassSustainabilityPolicy attached (e.g., <0.6 $kgCO_2e$/unit, max **10 min >8 °C**).
3. **Ship:** EPCIS ObjectEvent creates ClassColdChainTelemetry; signed excursions → alerts; lane carbon deltas added.
4. **Receive/Dispense:** Final ESG proofs rolled up. If policy met, **green discount** applied in ClassGreenSettlement; if not, **penalty** or **escrow**.
5. **Public Transparency: SagaFeeds** shows: "Lot A met policy (−12.5 bps green discount), zero excursions, **0.48 $kgCO_2e$/unit**."

Outcomes & Benefits

- **Operational:** Real-time ESG guardrails reduce excursions/waste; carbon visibility optimizes lanes.
- **Financial:** Sustainability priced into settlements aligns incentives for manufacturers, 3PLs, distributors.
- **Regulatory:** Auditable ESG proofs meet emerging disclosure regimes and inspection readiness.
- **Trust:** Patients/providers can verify sustainability credentials at scan time via **GS1 Digital Link**.
- **Global Health:** Ethical sourcing enforced at the same granularity as safety and quality.

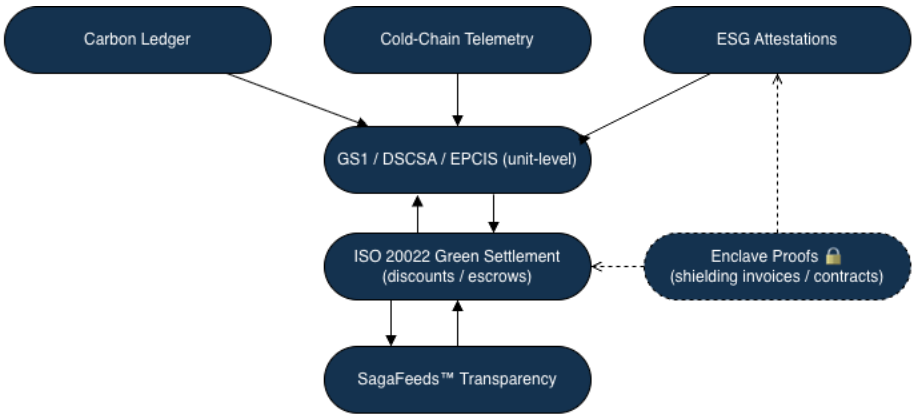


Diagram 5-M — ESG Accountability
Carbon Ledger + Cold-Chain Telemetry + ESG Attestations ↔ GS1/DSCSA/EPCIS (unit-level) ↔ ISO 20022 Green Settlement (discounts/escrows) ↔ SagaFeeds Transparency — Enclave proofs shielding invoices/contracts

ESG Accountability: Carbon ledger, telemetry, and attestations converge at unit-level GS1/DSCSA/EPCIS. Bi-directional flows tie into ISO 20022 green settlement and public transparency via SagaFeeds™. Enclave proofs shield sensitive invoices/contracts while preserving verifiability.

Scenario 14: Public Health Preparedness & Surge Orchestration

(coordinating CDC IIS/VTrckS, WHO, GS1/DSCSA, and ISO 20022 for rapid vaccine/drug deployment)

Background & Problem Statement

Global crises; pandemics, epidemics, bioterror events, or regional outbreaks demand fast, equitable, and **verifiable** pharmaceutical deployment. COVID-19 exposed structural gaps:

- **Fragmented systems:** CDC IIS/VAERS/VTrckS, WHO/COVAX, and national registries operated in silos.
- **Serialization blind spots:** GS1/DSCSA not consistently embedded in public health workflows → counterfeit/misallocation risk.
- **Cold-chain strain:** Limited, non-cryptographic telemetry led to wastage and uncertainty.
- **Slow finance:** Bilateral contracts and manual reconciliations lagged behind shipments.
- **Equity gaps:** LMIC access was slower and less reliable due to weak provenance and finance coordination.

Technical Approach (SagaChain with Private Enclaves)

A **unified preparedness fabric** connects supply, safety, and finance:

1. **Unified Preparedness Graph**

- **Assets:** Vaccines, antivirals, PPE as ClassPreparednessAsset, inheriting identifiers (GS1/DSCSA) and linking to ISO IDMP.
- **Distribution:** CDC **VTrckS** shipments (ClassVTrckSShipment), IIS administration records (ClassIISRecord), WHO stockpiles (ClassWHOStockpile).
- **Cold-chain:** Every EPCIS event can attach ClassColdChainTelemetry (signed).
- **Safety:** ClassVaccineAdverseEvent / ClassFAERSReport cross-link to lots and units.

2. **Private Enclaves (PHI & Commercial Confidentiality)**

- **PHI:** Patient identifiers, demographics, and administration details run inside enclaves; public outputs are selective proofs & aggregates.
- **Commercials:** Dose pricing, 3PL and donor terms remain encrypted-in-use; settlements use enclave-attested conditions.

3. **Financial Synchronization (ISO 20022)**

- Delivery-vs-Payment with ClassPreparednessSettlement; smart escrows require **serialization verified**, **no temp excursions**, **policy-compliant allocation**.
- Green finance extensions reward cold-chain efficiency and ethical sourcing.

4. **Surge Orchestration Policies**

- ClassSurgePolicy codifies prioritization (HCP, elderly, geographic tiers) and allocation ratios; enforcement captured as on-chain proofs.

Sample SagaPython Code

```
# Core preparedness asset representing a deployable unit (vaccine, antiviral, PPE)
@sagaclass()
class ClassPreparednessAsset(ClassNonFungibleAsset):
    """Pharma asset for surge deployment (vaccines, antivirals, PPE)."""
    asset_id        = sagafield(type="str")
    product_code    = sagafield(type="str")        # GS1 GTIN / ISO IDMP
    lot             = sagafield(type="str")
    expiration      = sagafield(type="date")
    serialization   = sagafield(type="str")        # SGTIN / DSCSA unit OID
    cold_chain_ref  = sagafield(type="list[str]", default=[])  # OIDs -> ClassColdChainTelemetry
    regulatory_refs = sagafield(type="list[str]", default=[])  # FDA/EMA/WHO objects

    @sagamethod()
    def attach_coldchain(self, telemetry_oid: str) -> int:
        if telemetry_oid not in self.cold_chain_ref:
            self.cold_chain_ref.append(telemetry_oid)
        return len(self.cold_chain_ref)


# CDC VTrckS shipment record (logistics layer), linked to EPCIS/GS1/DSCSA
@sagaclass()
class ClassVTrckSShipment(SPClassObject):
    """CDC VTrckS shipment tracking."""
    shipment_id     = sagafield(type="str")
    manufacturer    = sagafield(type="str")
    distributor     = sagafield(type="str")
    ship_date       = sagafield(type="date")
    sscc            = sagafield(type="str")        # GS1 SSCC for pallet/case
    sgtins          = sagafield(type="list[str]", default=[])  # OIDs -> ClassGS1SGTIN
    destinations_gln = sagafield(type="list[str]", default=[])  # GLNs for destinations
    epcis_events    = sagafield(type="list[str]", default=[])  # OIDs -> ClassEPCISEvent

    @sagamethod()
    def add_units(self, unit_oids: list[str]) -> int:
        for oid in unit_oids or []:
            if oid not in self.sgtins:
                self.sgtins.append(oid)
        return len(self.sgtins)


# WHO stockpile allocation & release tracking
@sagaclass()
class ClassWHOStockpile(SPClassObject):
    """WHO stockpile lot and release log."""
    stockpile_id    = sagafield(type="str")
    product_code    = sagafield(type="str")        # GTIN/IDMP
    lots            = sagafield(type="list[str]", default=[])
    release_log     = sagafield(type="list[dict]", default=[])  # [{"date": "...", "qty": 100000, "dest":"COUNTRY"}]

    @sagamethod()
    def release(self, date_iso: str, qty: int, dest: str) -> int:
        self.release_log.append({"date": date_iso, "qty": qty, "dest": dest})
        return len(self.release_log)


# IIS (Immunization Information System) record – PHI-bearing; run in enclave
@sagaclass()
class ClassIISRecord(SPClassObject):
    """CDC IIS immunization event (enclave-protected PHI)."""
    record_id       = sagafield(type="str")
    patient_phi_oid = sagafield(type="str")        # OID -> ClassPHI (enclave)
    sgtin           = sagafield(type="str")        # admin'd unit
    lot             = sagafield(type="str")
    cvx_code        = sagafield(type="str")        # vaccine code
    admin_date      = sagafield(type="date")
    site_gln        = sagafield(type="str")
    adverse_event_oid= sagafield(type="str", default="")  # OID -> ClassVaccineAdverseEvent

    @sagamethod()
    def attest_public(self) -> dict:
        # Returns non-PII proof for public dashboards/audits
        return {"record": self.record_id, "sgtin": self.sgtin, "admin_date": str(self.admin_date), "site": self.site_gln}


# Programmable surge allocation policy (CDC/WHO/national)
@sagaclass()
class ClassSurgePolicy(SPClassObject):
    """Programmable surge allocation policy (CDC, WHO, national)."""
    policy_id       = sagafield(type="str")
    jurisdiction    = sagafield(type="str")
```

```
    rules                = sagafield(type="dict")          #
{"priority":["HCP","Elderly","LMIC"],
"allocation_ratio":{"US":0.2,"LMIC":0.5}}
    effective_period = sagafield(type="dict")          #
{"from":"2025-01-01","to":"2026-01-01"}
    enforcement_proofs = sagafield(type="list[str]",
default=[])

    @sagamethod()
    def record_enforcement(self, proof_oid: str) -> int:
        if proof_oid not in self.enforcement_proofs:
            self.enforcement_proofs.append(proof_oid)
        return len(self.enforcement_proofs)


# ISO 20022 settlement with surge & quality
conditions (escrows, DvP)
@sagaclass()
class ClassPreparednessSettlement(SPClassObject):
    """ISO 20022 settlement with surge/quality
conditions."""
    settlement_id    = sagafield(type="str")
    payer                = sagafield(type="str")          #
donor/government
    payee                = sagafield(type="str")          #
manufacturer/distributor
    amount           = sagafield(type="float")
    currency         = sagafield(type="str", length=3)
    linked_assets            =
sagafield(type="list[str]",
default=[])  # OIDs -> ClassPreparednessAsset
    policy_ref         = sagafield(type="str", default="")
# OID -> ClassSurgePolicy
    escrow_conditions=             sagafield(type="dict",
default={})          # {"serialization_verified": True,
"temp_excursions":"<5min", "allocation_ok": True}
    status                           =
sagafield(type="str",
enum={"pending","released","held","disputed"},
default="pending")

    @sagamethod()
    def evaluate_conditions(self) -> dict:
        # Illustrative logic: assume prior enclave
attestations set booleans/threshold checks.
        conds = self.escrow_conditions or {}
        ok = all([
            bool(conds.get("serialization_verified",
False)),
            bool(conds.get("allocation_ok", False)),
            # For excursions, accept either boolean proof
or threshold string already validated upstream
            conds.get("temp_excursions") in (True, "ok",
"<=threshold")
        ])
        self.status = "released" if ok else "held"
        return {"status": self.status, "ok": ok}

# Optional: Preparedness dashboard index (public, no-
cost)
@sagaclass()
class ClassPreparednessIndex(SPClassObject):
    """Public SagaFeeds index for surge transparency
(non-sensitive)."""
    index_id         = sagafield(type="str")
    latest_stats      = sagafield(type="dict", default={})
#
{"allocations":{"LMIC":5_000_000,"US":2_000_00
0}, "cold_chain_ok": 0.992}
    objects          = sagafield(type="list[str]", default=[])
# OIDs of shipments/assets/policies

    @sagamethod()
    def publish_snapshot(self, stats: dict, append_oids:
list[str]) -> dict:
        self.latest_stats = stats or {}
        for oid in append_oids or []:
            if oid not in self.objects:
                self.objects.append(oid)
        return {"count": len(self.objects), "stats":
self.latest_stats}
```

Note: PHI, pricing, and contract terms are processed in **Private Enclaves**. Only **attested proofs** (e.g., “allocation satisfied,” “serialization verified,” “no excursions beyond threshold”) leave the enclave to drive settlements and dashboards.

Interoperability Analysis

- **CDC IIS / VTrckS:** Shipment and administration records anchor to GS1/DSCSA (SSCC, SGTIN) and EPCIS events; IIS PHI remains enclave-protected while public proofs enable transparency.
- **WHO (Prequalification / COVAX):** WHO stockpile releases and prequalification proofs link to **serialized deliveries** in LMICs, creating equitable, verifiable allocation evidence.
- **GS1 EPCIS / DSCSA:** Full traceability across commissioning, aggregation, shipping, receiving, and

dispensing **counterfeit prevention** and rapid recall readiness.

- **Cold Chain:** ClassColdChainTelemetry captures **signed** telemetry at each handoff; settlement gates on **temperature integrity** proofs.
- **FAERS / VAERS / HL7:** Safety signals cross-link to surge assets, enabling **near-real-time** pharmacovigilance and label/recall triggers.
- **ISO 20022:** ClassPreparednessSettlement executes **DvP** and smart **escrows**, releasing funds only when allocation, serialization, and cold-chain conditions are met (with optional green incentives).
- **Private Enclaves:** PHI and commercial terms are encrypted-in-use; regulators, donors, and the public consume **provable aggregates** and compliance proofs.
- **SagaFeeds (Public Indices):** No-cost dashboards broadcast non-sensitive surge stats (allocations by region/priority tier, cold-chain performance, release statuses).

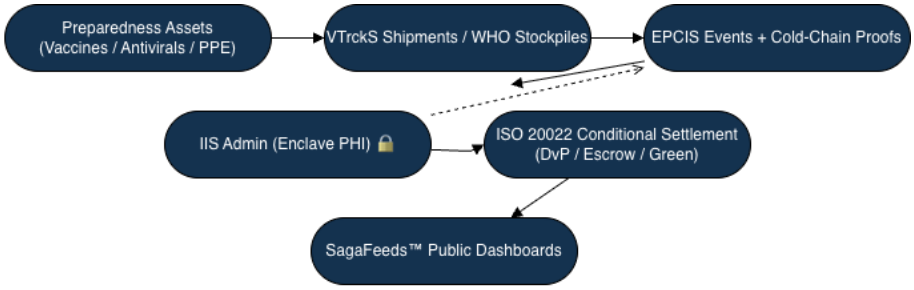


Diagram 5-N — Surge Orchestration
Preparedness Assets (Vaccines/Antivirals/PPE) → VTrckS Shipments / WHO Stockpiles → EPCIS Events + Cold-Chain Proofs → IIS Admin (Enclave PHI) → ISO 20022 Conditional Settlement (DvP/Escrow/Green) → SagaFeeds Public Dashboards

Surge Orchestration: preparedness assets flow through national and global stockpile logistics into EPCIS with cold-chain proofs; IIS administers doses in an enclave (PHI-protected); conditional settlements clear via ISO 20022; public dashboards published via SagaFeeds™. Dashed link indicates PHI-free attestations.

Implications

- **Governments & Regulators:** Real-time enforcement of priority allocations across borders; inspection burden reduced.
- **Manufacturers & Logistics:** Faster, predictable settlement tied to **verifiable performance**; fewer disputes.
- **Payers & Donors:** Funds flow only upon **provable delivery**, integrity, and equitable allocation.
- **Providers & Patients:** Reduced wastage, counterfeit prevention, and confidence in integrity at point of care.
- **Global Equity:** LMICs receive **transparent** allocation visibility and verifiable access improvements.

Scenario 14 demonstrates how SagaChain converts fragmented surge responses into a globally orchestrated, **programmable** preparedness system that balances **speed, trust, and equity**.

Scenario 15: Patient-Centric Pharmacogenomics & Precision Dosing

(integrating genomic data, USP standards, HL7 FHIR Genomics, and payer coverage)

Background & Problem Statement

Pharmacogenomics (PGx) can materially reduce adverse drug reactions (ADRs) and improve effectiveness by tailoring therapy to a patient's genotype/phenotype. Yet operationalization lags because:

- **Data silos:** PGx lab results sit in LIS portals, disconnected from EHRs, SPL labeling, and claims.
- **Weak provenance:** Results are often PDFs without cryptographic lineage to product identifiers or dosing logic.
- **Regulatory gaps:** SPL contains biomarker content but is not dynamically wired to prescribing or claims decisions.
- **Workflow friction:** HL7 FHIR Genomics exists, but many prescribing systems lack persistent, computable PGx artifacts.
- **Coverage misalignment:** Payers question evidence in the absence of linked outcomes and standardized proofs.
- **Privacy risk:** Genomic data is highly sensitive; uncontrolled sharing discourages adoption.

Technical Approach (SagaChain with Private Enclaves)

Goal: Turn PGx into a persistent, privacy-preserving, compliance-aware pipeline that drives real-time dosing and reimbursement.

1. **Genomic Results as Enclave-Protected Objects**
   - ClassGenomicResult persists HL7 FHIR Genomics facts (gene, variant, phenotype).
   - **Private Enclaves** keep raw sequences and PHI encrypted-in-use; only phenotype & dosing implications (minimal necessary data) are disclosed with attestation.
2. **USP/CPIC-Aligned Precision Dosing**
   - ClassUSPDosingGuideline encodes drug–gene–phenotype mappings (e.g., CYP2C19 clopidogrel).
   - A dosing engine object computes **patient-specific** recommendations and records the guideline evidence level.
3. **Regulatory & Labeling Linkage (SPL)**
   - SPL sections (e.g., Pharmacogenomics, Dosage & Administration) cross-reference PGx implications from persistent objects, ensuring **label-consistent** dosing.
4. **Payer Integration via ISO 20022**
   - ClassReimbursementDecision references PGx proofs; **conditional settlement** messages (ISO 20022) release payment when coverage criteria are satisfied and outcomes targets are met.
5. **Public Transparency via SagaFeeds**
   - **No-cost indices** list available guidelines, biomarker–drug pairs, and evidence grades without exposing patient data.

Sample SagaPython Code

```
# Enclave-protected PGx result (HL7 FHIR Genomics aligned)
@sagaclass()
class ClassGenomicResult(SPClassObject):
    """Patient genomic result using HL7 FHIR Genomics profile (enclave-protected)."""
    result_id       = sagafield(type="str")
    patient_phi_oid  = sagafield(type="str")       # OID -> ClassPHI (enclave)
    gene            = sagafield(type="str")       # e.g., CYP2C19
    variant         = sagafield(type="str")       # e.g., *2/*2
    phenotype       = sagafield(type="str")       # e.g., Poor Metabolizer
    interpretation  = sagafield(type="str")       # human-readable PGx implication
    fhir_ref        = sagafield(type="str")       # FHIR Genomics Observation/Report
    enclave_attest  = sagafield(type="str")       # TEE/zk attestation for this result

    @sagamethod()
    def minimal_disclosure(self) -> dict:
        # Return only what prescribers/payers need-no sequences or raw PHI
        return {
            "result_id": self.result_id,
            "gene": self.gene,
            "variant": self.variant,
            "phenotype": self.phenotype,
            "interpretation": self.interpretation,
            "fhir_ref": self.fhir_ref,
            "attest": self.enclave_attest
        }


# USP/CPIC-aligned dosing guideline as a computable standard
@sagaclass()
class ClassUSPDosingGuideline(SPClassObject):
    """USP/CPIC dosing guideline for PGx variants."""
    guideline_id    = sagafield(type="str")
    drug            = sagafield(type="str")       # generic/INN
    gene            = sagafield(type="str")
    phenotype       = sagafield(type="str")       # PM/IM/NM/UM etc.
    recommended_dose = sagafield(type="dict")     # {"adult":"5mg qd","peds":"weight-based"}
    evidence_level  = sagafield(type="str")       # e.g., A/B/C


# End-to-end precision dosing decision object (links PGx + USP + SPL + payer)
@sagaclass()
class ClassPrecisionDosing(SPClassObject):
    """Precision dosing linking PGx results, USP/CPIC guideline, SPL, and payer coverage."""
    dosing_id       = sagafield(type="str")
    genomic_result  = sagafield(type="str")       # OID -> ClassGenomicResult
    usp_guideline   = sagafield(type="str")       # OID -> ClassUSPDosingGuideline
    spl_ref         = sagafield(type="str")       # OID -> ClassSPLDocument
    product_code    = sagafield(type="str")       # ISO IDMP / NDC / GTIN
    recommended_dose = sagafield(type="str", default="")
    payer_decision  = sagafield(type="str", default="")  # OID -> ClassReimbursementDecision
    status          = sagafield(type="str", enum={"pending","recommended","dispensed","denied"}, default="pending")

    @sagamethod()
    def compute_recommendation(self, guideline_payload: dict) -> str:
        # Trusted call site passes the canonical guideline dict (or it can be read from usp_guideline object)
        # Derive recommended dose string (simplified for illustration)
        dose = guideline_payload.get("recommended_dose", {}).get("adult") or "consult pharmacist"
        self.recommended_dose = dose
        self.status = "recommended"
        return self.recommended_dose

    @sagamethod()
    def attach_payer_decision(self, decision_oid: str) -> str:
        self.payer_decision = decision_oid
        return self.payer_decision


# Optional: payer decision object with ISO 20022 settlement links
@sagaclass()
class ClassReimbursementDecision(SPClassObject):
    """Evidence-based coverage decision using PGx proofs."""
    decision_id     = sagafield(type="str")
    member_id       = sagafield(type="str")       # may be enclave-linked via ClassPHI
    drug_code       = sagafield(type="str")
    indications     = sagafield(type="list[str]", default=[])
```

```
    pgx_proofs = sagafield(type="list[str]", default=[]) # OIDs -> ClassGenomicResult.minimal_disclosure proofs
    criteria = sagafield(type="dict", default={}) # coverage rules
    iso20022_flows = sagafield(type="list[str]", default=[]) # OIDs -> ClassISO20022Settlement
    status = sagafield(type="str", enum={"pending","approved","denied","cond_approved"}, default="pending")

    @sagamethod()
    def evaluate(self) -> dict:
        # Simplified: approve if PGx proof exists and rule satisfied
        approved = bool(self.pgx_proofs) and bool(self.criteria.get("pgx_required", True))
        self.status = "approved" if approved else "denied"
        return {"status": self.status, "reason": "pgx_proven" if approved else "insufficient_evidence"}
```

Interoperability Analysis

- **HL7 FHIR Genomics:** ClassGenomicResult references FHIR Observations/Reports; **enclave** attestation certifies integrity and origin.
- **USP/CPIC Dosing:** ClassUSPDosingGuideline provides computable dose logic aligned to global standards; **one source of truth** for prescribers, payers, and regulators.
- **SPL (Labeling):** ClassPrecisionDosing.spl_ref ensures dosing stays consistent with current label pharmacogenomics sections and contraindications.
- **21 CFR / FD&C:** End-to-end provenance satisfies auditability (electronic records/signatures), with minimal exposure of PHI.
- **Payers & ISO 20022:** ClassReimbursementDecision links to ISO 20022 settlements; conditional release (e.g., prior auth satisfied, PGx proof verified) is **programmable**.
- **HIPAA/HITECH & Enclaves:** Raw sequences and identity remain encrypted-in-use; only **minimal disclosures** leave the enclave.
- **SagaFeeds (Public):** Non-sensitive indices advertise available PGx guidelines, biomarker-drug pairs, and evidence levels to promote adoption no patient data exposed.
- **IDMP/GS1:** Product codes (IDMP/NDC/GTIN/SGTIN) unify clinical, labeling, and claims semantics across jurisdictions.

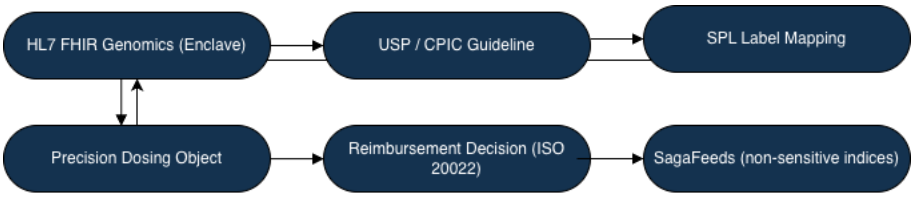


Diagram 5-O — PGx & Precision Dosing
HL7 FHIR Genomics (Enclave) → USP/CPIC Guideline → SPL Label Mapping → Precision Dosing Object → Reimbursement Decision (ISO 20022) → SagaFeeds (non-sensitive indices)

PGx & Precision Dosing: FHIR Genomics (enclave) → USP/CPIC guidance → SPL mapping → Precision Dosing object → ISO 20022 reimbursement → SagaFeeds transparency (non-sensitive indices).

Implications

- **Patients:** Safer therapy with fewer ADRs and faster time-to-right-dose.
- **Providers:** Embedded, guideline-driven dosing inside prescribing workflows no PDF chasing.
- **Regulators:** Transparent, cryptographically provable linkage

from label biomarker language to bedside dosing.
- **Payers:** Automated, evidence-based approvals reduce denials/appeals and align spend with outcomes.
- **Global Harmonization:** IDMP + HL7 + USP/CPIC makes PGx portable across borders and systems while preserving privacy.

Scenario 16: Global Clinical Trial Acceleration & Decentralized Trial Oversight

(integrating FDAAA 801, CDISC, HL7, WHO, and ISO 20022)

Background & Problem Statement

Clinical trials remain costly, slow, and fragmented:

- **Registration & transparency:** FDAAA 801 requires registration/results posting, but compliance is inconsistent and verification is manual.
- **Data standards:** CDISC (SDTM/ADaM) is authoritative, yet mapping from HL7/EHR sources is error-prone and expensive.
- **Decentralized trials (DCTs):** Telehealth, wearables, home delivery create new gaps in data integrity and safety oversight.
- **Multi-jurisdiction complexity:** Parallel submissions to FDA, EMA, PMDA, WHO lead to redundancy.
- **Finance:** Sponsor↔CRO↔site↔participant flows are fragmented; milestones and reimbursements lack programmability.
- **Equity:** LMIC participation is limited by trust, oversight, and payment frictions.

Technical Approach (SagaChain with Private Enclaves)

SagaChain turns trials into *persistent*, *auditable*, and *privacy-preserving* smart objects:

1. **Trial Registration & Proof of Transparency**
   - ClassClinicalTrial satisfies FDAAA 801 via persistent registration/results objects.
   - Registration and results are cryptographically bound to CDISC/HL7 streams and SPL label references.
   - **SagaFeeds** publish no-cost indices (trial ID, phase, status, milestones), improving public trust.
2. **CDISC-Standardized Data Capture**
   - HL7 FHIR patient data ingested in **Private Enclaves**; outputs are attested ClassCDISCDataset (SDTM/ADaM) objects.
   - Eliminates costly manual conversion while keeping PHI encrypted-in-use.
3. **Decentralized Oversight (DCT)**
   - Remote visits, connected-device telemetry, and direct-to-patient shipments are persisted as HL7/FHIR + EPCIS events (e.g., ClassTrialVisit).
   - Adverse events link to ClassFAERSReport/ClassVaccine AdverseEvent.

- Enclaves provide verifiable proofs (transform, analysis, de-ID) without exposing PHI.

4. **Financial Integration (ISO 20022)**
   - ClassISO20022TrialPayment automates disbursements to sites/CROs/participants.
   - **Escrows** release on milestone proofs (enrollment, IA locked, database lock, results posted).
5. **Global Harmonization**
   - Investigational products mapped to **ISO IDMP**; proofs synchronize with WHO and EMA portals, reducing duplicative review.

Sample SagaPython

```
# 1) FDAAA 801-registered trial object with DCT oversight; PHI-bound transforms in enclaves
@sagaclass()
class ClassClinicalTrial(SPClassObject):
    """Clinical trial with FDAAA 801 registration, DCT oversight, and enclave-attested data flows."""
    trial_id        = sagafield(type="str")                 # NCT or regional ID
    sponsor         = sagafield(type="str")
    title           = sagafield(type="str")
    phase           = sagafield(type="str", enum={"I","II","III","IV","NA"})
    registry_refs   = sagafield(type="list[str]", default=[])       # FDAAA 801, EMA, WHO IDs
    idmp_products   = sagafield(type="list[str]", default=[])       # ISO 11615 refs
    cdisc_datasets  = sagafield(type="list[str]", default=[])       # OIDs -> ClassCDISCDataset
    hl7_visits      = sagafield(type="list[str]", default=[])       # OIDs -> ClassTrialVisit
    adverse_events  = sagafield(type="list[str]", default=[])       # OIDs -> FAERS/VAERS
    spl_links       = sagafield(type="list[str]", default=[])       # OIDs -> ClassSPLDocument sections
    enclave_proofs  = sagafield(type="list[str]", default=[])       # attestations for ETL/analysis
    status          = sagafield(type="str", enum={"planned","recruiting","active","completed","terminated"}, default="planned")
    milestones      = sagafield(type="dict", default={})            # {"enrollment_complete": "...", "db_lock": "...", ...}

    @sagamethod()
    def record_milestone(self, name: str, iso_datetime: str, proof_oid: str | None=None) -> dict:
        self.milestones[name] = iso_datetime
        if proof_oid:
            self.enclave_proofs.append(proof_oid)
        return {"ok": True, "milestone": name, "at": iso_datetime}

    @sagamethod()
    def link_dataset(self, dataset_oid: str) -> int:
        if dataset_oid not in self.cdisc_datasets:
            self.cdisc_datasets.append(dataset_oid)
        return len(self.cdisc_datasets)

    @sagamethod()
    def link_visit(self, visit_oid: str) -> int:
        if visit_oid not in self.hl7_visits:
            self.hl7_visits.append(visit_oid)
        return len(self.hl7_visits)

    @sagamethod()
    def post_result(self, spl_section_oid: str, proof_oid: str) -> dict:
        # bind results to label section with audit-proof
        self.spl_links.append(spl_section_oid)
        self.enclave_proofs.append(proof_oid)
        self.status = "completed"
        return {"status": self.status, "label_section": spl_section_oid}


# 2) Trial visit wrapper (HL7 FHIR) for site or decentralized interactions
@sagaclass()
class ClassTrialVisit(SPClassObject):
    """HL7 FHIR-based visit/interaction record (site or decentralized)."""
    visit_id        = sagafield(type="str")
    trial_oid       = sagafield(type="str")          # OID -> ClassClinicalTrial
    participant_phi = sagafield(type="str")          # OID -> ClassPHI (enclave)
    fhir_refs       = sagafield(type="list[str]", default=[])       # Condition/Observation/Device/etc.
    mode            = sagafield(type="str", enum={"onsite","telehealth","home_nurse","device"}, default="onsite")
    epcis_events    = sagafield(type="list[str]", default=[])  # shipments/kits via EPCIS OIDs
    timestamp       = sagafield(type="str")

    @sagamethod()
    def add_fhir_ref(self, resource_ref: str) -> int:
        self.fhir_refs.append(resource_ref)
        return len(self.fhir_refs)
```

```
# 3) Enclave-attested CDISC dataset (SDTM/ADaM)
@sagaclass()
class ClassCDISCDataset(SPClassObject):
    """CDISC dataset produced from enclave ETL (SDTM or ADaM) with transform proofs."""
    dataset_id      = sagafield(type="str")
    trial_oid       = sagafield(type="str")
    standard        = sagafield(type="str", enum={"SDTM","ADaM","SEND"})
    domains_or_vars = sagafield(type="list[str]", default=[])
    define_xml_hash = sagafield(type="str")
    enclave_attest  = sagafield(type="str")  # TEE/zk proof of ETL/validation

    @sagamethod()
    def add_domain_or_var(self, name: str) -> int:
        if name not in self.domains_or_vars:
            self.domains_or_vars.append(name)
        return len(self.domains_or_vars)


# 4) ISO 20022 programmatic payments tied to trial milestones
@sagaclass()
class ClassISO20022TrialPayment(SPClassObject):
    """ISO 20022 payment object for trial participants, sites, CROs."""
    payment_id      = sagafield(type="str")
    payer           = sagafield(type="str")
    payee           = sagafield(type="str")
    amount          = sagafield(type="float")
    currency        = sagafield(type="str", length=3)
    trial_oid       = sagafield(type="str")  # OID -> ClassClinicalTrial
    milestone       = sagafield(type="str")  # enrollment_complete, IA_locked, db_lock, results_posted
    escrow_conditions= sagafield(type="dict", default={})
    status          = sagafield(type="str", enum={"pending","released","disputed","canceled"}, default="pending")

    @sagamethod()
    def try_release(self, trial_snapshot: dict) -> dict:
        """
        trial_snapshot minimally contains trial.milestones and an attestation reference.
        If the required milestone exists (and optional proofs match), release payment.
        """
        need = self.milestone
        ok = bool(trial_snapshot.get("milestones", {}).get(need))
        if ok:
            self.status = "released"
            return {"released": True, "milestone": need}
        return {"released": False, "reason": "milestone_missing"}
```

Interoperability Analysis

- **FDAAA 801:** Registration/results are embedded in ClassClinicalTrial with immutable timestamps and proofs.
- **CDISC:** ClassCDISCDataset (SDTM/ADaM) is **enclave-attested** (ETL + validation), enabling regulator-grade reproducibility.
- **HL7/DCT:** ClassTrialVisit persists remote/onsite interactions, device streams, and kit logistics (EPCIS) under one object model.
- **WHO/EMA:** registry_refs and IDMP product bindings let agencies verify **the same proofs**, reducing duplicate submissions.
- **ISO 20022:** ClassISO20022TrialPayment settles on milestone proofs; escrows prevent premature/disputed disbursements.
- **HIPAA/HITECH:** PHI remains enclave-bound (via ClassPHI OIDs); only minimal disclosures (aggregates, hashes, attestations) exit the enclave.
- **SagaFeeds:** Public indices expose non-sensitive trial metadata, milestones reached, and proof references free for patients, payers, and watchdogs.

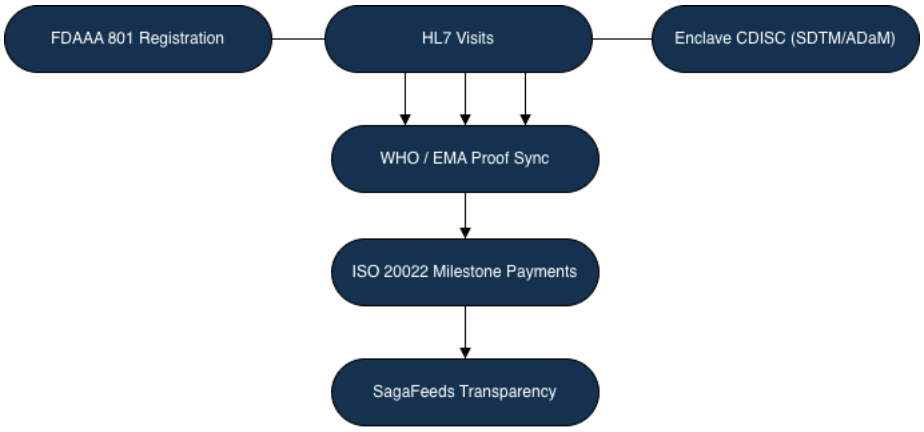


Diagram 5-P — Global Trial Acceleration
FDAAA 801 Registration + HL7 Visits + Enclave CDISC (SDTM/ADaM) → WHO/EMA Proof Sync → ISO 20022 Milestone Payments → SagaFeeds Transparency

Global Trial Acceleration: FDAAA 801 registration, HL7 visit data, and enclave CDISC datasets converge for WHO/EMA proof sync; milestone payments clear via ISO 20022 and transparency is published through SagaFeeds.

Implications

- **Regulators:** Continuous, reproducible oversight with cryptographic lineage to every dataset and milestone.
- **Sponsors/CROs:** Lower data prep costs, fewer reworks, faster DB lock and submission timelines.
- **Sites/Participants:** Automated, on-time payments; simpler DCT logistics and documentation.
- **Payers/Donors:** Funding tied to real progress; reduced leakage/fraud.
- **Global Health:** Harmonized proofs support equitable LMIC participation and faster access to innovation.

If you'd like, the next step can introduce a **monitoring companion class** (e.g., ClassTrialQualityMonitor) that raises enclave-attested alerts for protocol deviations, missing data, or safety thresholds auto-linking to SPL updates, FAERS filings, or payment holds.

Got it. Here is **Scenario 17** regenerated **as one complete, copy-pasteable package** with **everything** an enterprise and regulators will expect: privacy enclaves, RBAC, audit/logging, QbD, CM, PAT/SCADA telemetry, USP assays, ISPE validation & CPV, deviations/CAPA/change control, GS1/DSCSA/EPCIS lineage, ISO IDMP mapping, SPL linkage, adaptive AI monitoring, release gating, automated holds/recalls, ISO 20022 finance (with escrow/discount/clawback and green hooks), subscriptions/notifications, policy enforcement, versioning, and SagaFeeds (public indices). Decorators and base classes use **@sagaclass()** / **@sagamethod()** / **field = sagafield()**, extending **SPClassObject** as requested.

Scenario 17: Integrated Quality-by-Design (QbD) & Continuous Manufacturing Oversight

(linking ISPE, 21 CFR, USP, GS1/DSCSA/EPCIS, ISO IDMP, SPL, adaptive enclaves, ISO 20022, and SagaFeeds)

Background & Problem Statement

QbD + Continuous Manufacturing (CM) demand **live, provable** alignment of: **design intent → execution telemetry (PAT/SCADA) → USP quality → deviations/CAPA → release → serialization lineage → finance**. Today this is split across QMS/LIMS/MES/ERP, with delayed QC, manual reconciliation, and audit risk. Industry needs one **persistent-state** fabric that enforces policy, preserves IP & PHI, and gives regulators **continuous** assurance.

Technical Approach (Persistent State + Enclave Analytics)

- **Design Space as Code:** ICH Q8/Q11 QbD encoded with signed versions, RBAC, and policy hooks.
- **CM Execution:** High-frequency sensor frames → enclave analytics (SPC/MVDA/ML) → attestations (no raw data leakage).
- **Quality & Release:** USP assay proofs + adaptive scores gate automated release; deviations/holds/recalls are first-class.
- **Assurance:** ISPE GAMP5/CPV artifacts natively linked; CAPA & Change Control propagate to serialization and label updates.
- **Traceability:** GS1/DSCSA/EPCIS + ISO IDMP bind lots/units to global identity; SPL sections cross-refer.
- **Finance:** ISO 20022 settlements escrow/discount/clawback based on quality attestations and policy compliance (ESG-green optional).
- **Transparency:** SagaFeeds publish , non-sensitive indices; regulators subscribe to events instead of waiting for audits.

Sample SagaPython Code:

```
#
=====================================
==================
# 0) Governance, RBAC, Auditing, and Privacy
Enclave Mixin
#
=====================================
==================


@sagaclass()
class ClassRole(SPClassObject):
   role_id       = sagafield(type="str")
   permissions          = sagafield(type="list[str]",
default=[])                  #          e.g.,
["approve_release","view_enclave_attest","open_reca
ll"]

@sagaclass()
class ClassAccessBinding(SPClassObject):
   binding_id    = sagafield(type="str")
   subject_oid          = sagafield(type="str")       #
SPClassAccount OID
   role_ref          = sagafield(type="str")     # OID
ClassRole
   scope         = sagafield(type="dict", default={}) #
{"plant":"A","line":"L1","product":"IDMP:..."}

@sagaclass()
class ClassAuditLog(SPClassObject):
   """Immutable audit entries for Part 11-grade
traceability."""
   entry_id      = sagafield(type="str")
   actor         = sagafield(type="str")
   action        = sagafield(type="str")
   target_oid    = sagafield(type="str")
   timestamp     = sagafield(type="str")
   details       = sagafield(type="dict", default={})

@sagaclass()
class ClassPrivacyEnclaveMixin(SPClassObject):
   """Attach to classes that compute/expose proofs
from encrypted inputs."""
   enclave_id    = sagafield(type="str")
   allowed_outputs      =      sagafield(type="list[str]",
default=[])                  #          e.g.,
["attestation","summary_stats","proof_ref"]

#
=====================================
# 1) QbD Design Space, Release Policies
#
=====================================

@sagaclass()
```

```
class ClassQbDDesignSpace(SPClassObject):
    """ICH Q8/Q11: CQAs, CPPs, ranges, lineage, signatures."""
    design_id       = sagafield(type="str")
    product_idmp    = sagafield(type="str")             # ISO 11615 ref
    product_codes   = sagafield(type="dict", default={}) # {"NDC":"...", "GTIN":"..."}
    cqas            = sagafield(type="dict")             # {"potency":[95,105], "purity":[98,100]}
    cpps            = sagafield(type="dict")             # {"temp":[20,25], "pH":[6.8,7.2], "agit":[300,450]}
    ich_refs        = sagafield(type="list[str]", default=[])
    usp_refs        = sagafield(type="list[str]", default=[]) # linked monographs/chapters
    version         = sagafield(type="str", default="1.0")
    supersedes      = sagafield(type="str", default="")
    signatures      = sagafield(type="list[str]", default=[])

@sagaclass()
class ClassReleasePolicy(SPClassObject):
    """Gate release using QbD limits, USP proofs, AI attests, deviations, CAPA state, and cold-chain if applicable."""
    policy_id       = sagafield(type="str")
    design_ref      = sagafield(type="str")    # OID ClassQbDDesignSpace
    rules           = sagafield(type="dict")   # {"assays_min":1, "ai_score_min":0.93, "max_open_major_deviations":0, "cold_chain_excursion_minutes_max":10}
    esg_rules       = sagafield(type="dict", default={}) # optional green hooks
    notify          = sagafield(type="list[str]", default=[])

#
=============================================
# 2) Serialization, Identity, EPCIS Linkages
#
=============================================

@sagaclass()
class ClassGS1Link(SPClassObject):
    """Map batch/lot to GTIN/SGTIN/SSCC for EPCIS lineage."""
    link_id         = sagafield(type="str")
    gtin            = sagafield(type="str")
    lot             = sagafield(type="str")
    sgtins          = sagafield(type="list[str]", default=[])
    ssccs           = sagafield(type="list[str]", default=[])

@sagaclass()
class ClassDSCSAUnitRef(SPClassObject):
    """Reference DSCSA units + EPCIS events for verification context."""
    ref_id          = sagafield(type="str")
    ds_units        = sagafield(type="list[str]", default=[]) # OIDs ClassDSCSAUnit
    epcis_events    = sagafield(type="list[str]", default=[])  # OIDs ClassEPCISEvent

# ================================
# 3) Quality Proofs & Attestations
# ================================

@sagaclass()
class ClassUSPAssayProof(SPClassObject):
    """USP compendial pass/fail with evidence; method specifics enclave-protected."""
    proof_id        = sagafield(type="str")
    monograph_id    = sagafield(type="str")
    lot             = sagafield(type="str")
    result          = sagafield(type="str", enum={"pass","fail"})
    metrics         = sagafield(type="dict", default={}) # {"potency":100.2, "endotoxin":0.03}
    evidence_hashes = sagafield(type="list[str]", default=[])
    signer          = sagafield(type="str")
    timestamp       = sagafield(type="str")

@sagaclass()
class ClassEnclaveAttestation(SPClassObject):
    """TEE/zk proof: inputs/outputs commitments + score summary."""
    attest_id       = sagafield(type="str")
    enclave_id      = sagafield(type="str")
    model_version   = sagafield(type="str")
    inputs_commit   = sagafield(type="str")
    outputs_commit  = sagafield(type="str")
    statement       = sagafield(type="str")
    score           = sagafield(type="float")
    timestamp       = sagafield(type="str")

# ====================================
# 4) Deviations, CAPA, Change Controls
# ====================================

@sagaclass()
class ClassDeviation(SPClassObject):
    deviation_id    = sagafield(type="str")
    batch_ref       = sagafield(type="str")
    severity        = sagafield(type="str", enum={"minor","major","critical"})
    description     = sagafield(type="str")
    opened_at       = sagafield(type="str")
    closed_at       = sagafield(type="str", default="")
    status          = sagafield(type="str", enum={"open","closed"}, default="open")
```

```
@sagaclass()
class ClassCAPA(SPClassObject):
    """Corrective/Preventive Action linked to deviations & risks."""
    capa_id        = sagafield(type="str")
    related_devs   = sagafield(type="list[str]", default=[])  # OIDs ClassDeviation
    actions        = sagafield(type="list[dict]", default=[])  # [{"type":"corr","desc":"...","due":"..."}]
    owner          = sagafield(type="str")
    status         = sagafield(type="str", enum={"open","in_progress","verified","closed"}, default="open")

@sagaclass()
class ClassChangeControl(SPClassObject):
    """Impact analysis across QbD, CM, USP, EPCIS, SPL, and finance hooks."""
    change_id      = sagafield(type="str")
    description    = sagafield(type="str")
    impact_map     = sagafield(type="dict", default={})  # {"qbd":["design_id"],"usp":["mono"],"label":["spl_sec"],"finance":["settlement_ids"]}
    approvals      = sagafield(type="list[str]", default=[])
    status         = sagafield(type="str", enum={"draft","approved","implemented","rejected"}, default="draft")

# ====================================
# 5) Continuous Manufacturing Execution
# ====================================

@sagaclass()
class ClassContinuousBatch(SPClassObject):
    """CM execution record with telemetry summaries & references."""
    batch_id       = sagafield(type="str")
    design_ref     = sagafield(type="str")       # OID ClassQbDDesignSpace
    product_idmp   = sagafield(type="str")
    lot            = sagafield(type="str")
    plant          = sagafield(type="str")
    line           = sagafield(type="str")
    sensor_windows = sagafield(type="list[dict]", default=[])  # rolling summaries (no raw)
    usp_assays     = sagafield(type="list[str]", default=[])  # OIDs ClassUSPAssayProof
    enclave_attests= sagafield(type="list[str]", default=[])  # OIDs ClassEnclaveAttestation
    gs1_link       = sagafield(type="str", default="")  # OID ClassGS1Link
    dscsa_ref      = sagafield(type="str", default="")  # OID ClassDSCSAUnitRef
    deviations     = sagafield(type="list[str]", default=[])  # OIDs ClassDeviation
    status         = sagafield(type="str", enum={"running","complete","held","released","recalled"}, default="running")

    @sagamethod()
    def ingest_sensor_window(self, frame: dict) -> int:
        self.sensor_windows.append(frame); return len(self.sensor_windows)

    @sagamethod()
    def attach_assay(self, proof_oid: str) -> int:
        if proof_oid not in self.usp_assays: self.usp_assays.append(proof_oid)
        return len(self.usp_assays)

    @sagamethod()
    def attach_attestation(self, attest_oid: str) -> int:
        if attest_oid not in self.enclave_attests: self.enclave_attests.append(attest_oid)
        return len(self.enclave_attests)

    @sagamethod()
    def add_deviation(self, dev_oid: str) -> int:
        if dev_oid not in self.deviations: self.deviations.append(dev_oid)
        return len(self.deviations)

    @sagamethod()
    def mark_complete(self) -> str:
        self.status = "complete"; return "complete"

# ====================================
# 6) Adaptive Monitoring (AI in Enclave)
# ====================================

@sagaclass()
class ClassAdaptiveMonitor(ClassPrivacyEnclaveMixin):
    """AI/SPC/MVDA monitor—stores last attested run; raw stays in enclave."""
    monitor_id     = sagafield(type="str")
    batch_ref      = sagafield(type="str")
    thresholds     = sagafield(type="dict", default={"min_attest_score":0.93, "max_oos_events":0})
    last_score     = sagafield(type="float", default=0.0)
    last_summary   = sagafield(type="dict", default={})
    last_attest_ref= sagafield(type="str", default="")
    last_run_at    = sagafield(type="str", default="")

    @sagamethod()
```

```
    def evaluate(self, attest_oid: str, score: float, summary: dict, ts: str) -> bool:
        self.last_attest_ref, self.last_score, self.last_summary, self.last_run_at = attest_oid, score, summary, ts
        return score >= float(self.thresholds.get("min_attest_score", 0.93))

#
# 7) Holds, Alerts, Recalls, Cold Chain
#

@sagaclass()
class ClassSupplyHold(SPClassObject):
    hold_id      = sagafield(type="str")
    reason       = sagafield(type="str")
    subjects     = sagafield(type="list[str]", default=[])  # lot IDs, SGTINs, SSCCs
    created_at   = sagafield(type="str")
    status       = sagafield(type="str", enum={"active","lifted"}, default="active")

@sagaclass()
class ClassComplianceAlert(SPClassObject):
    alert_id     = sagafield(type="str")
    batch_ref    = sagafield(type="str")
    category     = sagafield(type="str")  # "quality","process","safety","finance","cold_chain"
    score        = sagafield(type="float")
    explanation  = sagafield(type="dict", default={})
    actions      = sagafield(type="list[str]", default=[])  # ["apply_hold","notify_fda","clawback"]
    status       = sagafield(type="str", enum={"open","acked","closed"}, default="open")
    created_at   = sagafield(type="str")

@sagaclass()
class ClassRecallEvent(SPClassObject):
    recall_id    = sagafield(type="str")
    scope_lots   = sagafield(type="list[str]", default=[])
    reason       = sagafield(type="str")
    authority    = sagafield(type="str")  # FDA/EMA/etc.
    started_at   = sagafield(type="str")
    status       = sagafield(type="str", enum={"open","ongoing","closed"}, default="open")

@sagaclass()
class ClassColdChainTelemetry(SPClassObject):
    """Signed payload summaries attached to EPCIS; full series stays off-chain/IPFS/or enclave."""
    stream_id    = sagafield(type="str")
    device_id    = sagafield(type="str")
    signed_payload = sagafield(type="dict", default={})
    summary      = sagafield(type="dict", default={})  # {min,max,excursions,mean_kinetic_temp}
    subjects     = sagafield(type="list[str]", default=[])  # SGTIN/SSCC
    attestation  = sagafield(type="str")  # signature

#
# 8) Validation (ISPE GAMP5/CPV), Release
#

@sagaclass()
class ClassValidationPackage(SPClassObject):
    """Executable validation package composed of requirements, verifs, CPV evidence."""
    validation_id    = sagafield(type="str")
    scope            = sagafield(type="str")
    evidence_hashes  = sagafield(type="list[str]", default=[])
    approval_signatures = sagafield(type="list[str]", default=[])

@sagaclass()
class ClassReleaseDecision(SPClassObject):
    decision_id  = sagafield(type="str")
    policy_ref   = sagafield(type="str")  # OID ClassReleasePolicy
    batch_ref    = sagafield(type="str")  # OID ClassContinuousBatch
    validation_ref = sagafield(type="str")  # OID ClassValidationPackage
    assays       = sagafield(type="list[str]", default=[])  # OIDs ClassUSPAssayProof
    attests      = sagafield(type="list[str]", default=[])  # OIDs ClassEnclaveAttestation
    outcome      = sagafield(type="str", enum={"pending","approved","rejected"}, default="pending")
    rationale    = sagafield(type="str", default="")
    decided_by   = sagafield(type="str", default="")
    decided_at   = sagafield(type="str", default="")

    @sagamethod()
    def add_inputs(self, assays: list, attests: list):
        for a in assays:
            if a not in self.assays: self.assays.append(a)
        for t in attests:
            if t not in self.attests: self.attests.append(t)

    @sagamethod()
    def evaluate_and_set(self, signer_id: str, ts: str, policy: dict, open_major_devs: int, cold_chain_exc_min: int=0) -> str:
```

```
        assays_min = int(policy.get("assays_min", 1))
        ai_min = float(policy.get("ai_score_min", 0.93))
        max_maj = int(policy.get("max_open_major_deviations", 0))
        cc_max = int(policy.get("cold_chain_excursion_minutes_max", 99999))

        assays_ok = len(self.assays) >= assays_min
        ai_ok = False
        for _ in self.attests:
            ai_ok = True  # upstream attestation verified by kernel/VPM hook

        dev_ok = open_major_devs <= max_maj
        cc_ok = cold_chain_exc_min <= cc_max

        if assays_ok and ai_ok and dev_ok and cc_ok:
            self.outcome = "approved"
            self.rationale = "Policy thresholds met"
        else:
            self.outcome = "rejected"
            self.rationale = f"assays_ok={assays_ok}, ai_ok={ai_ok}, dev_ok={dev_ok}, cold_chain_ok={cc_ok}"
        self.decided_by, self.decided_at = signer_id, ts
        return self.outcome

#
# =====================================
# 9) Finance (ISO 20022) + Green Hooks
#
# =====================================

@sagaclass()
class ClassISO20022Settlement(SPClassObject):
    """DvP settlement gated by release, holds, ESG performance; supports clawback."""
    settlement_id = sagafield(type="str")
    payer = sagafield(type="str")
    payee = sagafield(type="str")
    amount = sagafield(type="float")
    currency = sagafield(type="str")
    value_date = sagafield(type="str")
    lots = sagafield(type="list[str]", default=[])
    conditions = sagafield(type="dict", default={"release_approved": True, "no_active_hold": True})
    esg_adjustment = sagafield(type="dict", default={"enabled": False, "bps_delta": 0.0})
    status = sagafield(type="str", enum={"pending","released","on_hold","clawed_back"}, default="pending")
    reason = sagafield(type="str", default="")

    @sagamethod()
    def gate_on_release(self, release_decision_oid: str, holds_active: bool, green_ok: bool=True) -> str:
        if holds_active:
            self.status, self.reason = "on_hold", "Active supply hold"
            return self.status
        # Assume kernel validates release_decision outcome; simplified acceptance here:
        decision_ok = True
        if decision_ok and self.conditions.get("release_approved", True):
            self.status, self.reason = "released", "Quality/release gates satisfied"
            if self.esg_adjustment.get("enabled", False) and green_ok:
                # apply discount or premium via accounting side-car (not shown)
                pass
        else:
            self.status, self.reason = "on_hold", "Release decision not approved"
        return self.status

    @sagamethod()
    def clawback(self, cause: str) -> str:
        self.status, self.reason = "clawed_back", cause
        return self.status

#
# =====================================
# 10) SPL & Label Cross-Reference (read-side)
#
# =====================================

@sagaclass()
class ClassSPLBinding(SPClassObject):
    """Binds batch/product to SPL sections for label readiness/truthfulness."""
    binding_id = sagafield(type="str")
    product_idmp = sagafield(type="str")
    spl_set_id = sagafield(type="str")
    sections = sagafield(type="list[str]", default=[])  # clinical studies, dosage, stability

#
# =====================================
# 11) Public Indices (SagaFeeds) & Events
#
# =====================================

@sagaclass()
class ClassSagaFeedIndex(SPClassObject):
    index_id = sagafield(type="str")
    key = sagafield(type="str")  # e.g., "lot:ABC123"
```

```
    entries       = sagafield(type="list[dict]", default=[])
# [{"type":"release","oid":"...","ts":"..."}]

    @sagamethod()
    def add_entry(self, entry: dict) -> int:
        self.entries.append(entry); return len(self.entries)

@sagaclass()
class ClassEventSubscription(SPClassObject):
    sub_id         = sagafield(type="str")
    topic          = sagafield(type="str")      #
"release","deviation","recall","finance"
    subscriber     = sagafield(type="str")    # account
OID
    filter         = sagafield(type="dict", default={})

@sagaclass()
class ClassEvent(SPClassObject):
    event_id       = sagafield(type="str")
    topic          = sagafield(type="str")
    payload        = sagafield(type="dict", default={})
    created_at     = sagafield(type="str")

#
====================================
# 12) Policy Enforcer (Kernel/VPM Hooks)
#
====================================

@sagaclass()
class ClassPolicyEnforcer(SPClassObject):
    """Pre/post method enforcement for
release/settlement/recall with regulator hooks."""
    policy_id      = sagafield(type="str")
    rules          = sagafield(type="dict", default={})

    @sagamethod()
    def pre_release_check(self, release_oid: str) ->
bool:
        # Kernel ensures: signatures present, zero critical
deviations unless waiver, CPV evidence up to date.
        return True

    @sagamethod()
    def pre_settlement_check(self, settlement_oid: str)
-> bool:
        # Ensure: no active holds; approved release;
EPCIS lineage complete; optional ESG satisfied
        return True

    @sagamethod()
    def pre_recall_check(self, recall_oid: str) -> bool:
        # Ensure authority, scope consistency, and
notification fanout
        return True
```

Orchestrated End-to-End Flow (What Actually Happens)

1. **Author QbD**: ClassQbDDesignSpace signed (process dev, QA, RA). ClassReleasePolicy captures gating rules (USP min assays, AI min score, deviation caps, cold-chain tolerance, ESG options).
2. **Run CM**: ClassContinuousBatch.ingest_sensor_window() posts rolling aggregates (no raw). Enclave computes SPC/MVDA/ML → ClassEnclaveAttestation.
3. **Quality**: Lab posts ClassUSPAssayProof (pass/fail + metrics). Deviations recorded; CAPA/Change Control opened if needed.
4. **Hold/Alert**: ClassAdaptiveMonitor.evaluate() + policy thresholds emit ClassComplianceAlert; ClassSupplyHold auto-applies if out-of-design/spec.
5. **Release**: ClassReleaseDecision.add_inputs() then evaluate_and_set() (policy: assays/AI/deviations/cold-chain). Result logged to ClassAuditLog, broadcast via ClassEvent, and surfaced in ClassSagaFeedIndex.
6. **Traceability**: GS1/DSCSA/EPCIS link maintained (ClassGS1Link, ClassDSCSAUnitRef). If post-release signal hits, ClassRecallEvent scoped to lots/units.
7. **Finance**: ClassISO20022Settlement.gate_on_release() - escrow/discount/clawback integrated. Optional ESG green adjustment if enabled.
8. **Labeling**: ClassSPLBinding ensures clinical/stability sections remain truthful to production and QC state.
9. **Audit & Subscriptions**: Regulators subscribe to release/recall topics; auditors query SagaFeeds (no charge)

and inspect immutable lineage when needed.

10. **Change Governance**: ClassChangeControl propagates impacts across QbD/validation/label/finance; approvals recorded.

Interoperability & Compliance Matrix

- **21 CFR 210/211, 600s, Part 11**: Signed, immutable QbD/CM/assay/release with audit trail and policy hooks.
- **ISPE GAMP5/CPV**: Validation packages and CPV evidence are first-class, linked to execution and policy enforcement.
- **USP**: Monograph proofs gate release automatically; metrics and evidence hashes retained.
- **GS1/DSCSA/EPCIS**: Unit-level identity and event lineage tied to batches; supports targeted holds/recalls and finance certainty.
- **ISO IDMP**: Global product identity anchors every object; cross-border harmonization with EMA/WHO/FDA ecosystems.
- **SPL**: Label sections bound to real QC and stability state; truthfulness and timeliness are provable.
- **ISO 20022**: DvP settlement gated, discounted, or clawed back by compliance state (quality, cold chain, ESG).
- **Private Enclaves**: PAT raw, recipes, and any PHI remain encrypted-in-use; only attestations/aggregates leave.
- **SagaFeeds**: public indices for non-sensitive milestones (e.g., releases, recalls), boosting trust and reducing vendor lock-in.
- **Events/Subscriptions**: Deterministic event bus enables regulator/site alerts and continuous oversight.

Deployment Checklist (Enterprise-Grade)

- **RBAC & Key Mgmt:** Define ClassRole + ClassAccessBinding for QA, QC, MFG, RA, Finance, Regulator.
- **Version Governance:** Enforce version lineage for QbD, policies, validation packages; archive superseded versions.
- **Telemetry Ingest:** Configure plant gateways → enclave pipeline → attestation writer.
- **Lab Integration:** LIMS to generate ClassUSPAssayProof with signer identity.
- **Deviation/CAPA:** Route MES/QMS deviations into ClassDeviation/ClassCAPA; wire auto-holds.
- **EPCIS Bridge:** Post commission/agg/ship/receive to ClassDSCSAUnitRef/ClassGS1Link.
- **Finance Bridge:** Map invoices/POs to ClassISO20022Settlement with policy-gated release.
- **Label & IDMP:** Maintain ClassSPLBinding; ensure IDMP references present.
- **Feeds & Subs:** Stand up ClassSagaFeedIndex; add regulator subscriptions for “release”, “recall”, “finance”.
- **Policy Hooks:** Register ClassPolicyEnforcer with kernel/VPM for pre-release/settlement/recall checks.
- **Sharding:** Co-locate plant/QA/finance clusters; monitor affinity & migrate per SagaScale.

Outcomes

- **Manufacturers:** Faster, safer releases; audit-ready at all times; synchronized cash flow; fewer recalls/waste.
- **Regulators:** Live oversight via attestations/events; fewer site audits; higher confidence.
- **Providers/Patients:** Reliable quality; targeted recalls; transparency without exposing IP/PHI.
- **Payers/Financiers:** Payments tied to real compliance; automated clawbacks reduce risk.
- **SDOs:** Executable standards with governed evolution and verifiable conformance.

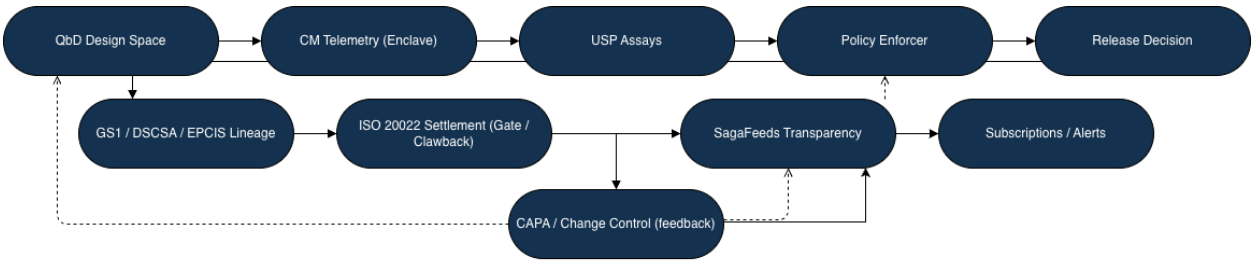


Diagram 5-Q — QbD Design Space → CM Telemetry (Enclave) → USP Assays → Policy Enforcer → Release Decision → GS1/DSCSA/EPCIS Lineage → ISO 20022 Settlement (Gate/Clawback) → SagaFeeds Transparency → Subscriptions/Alerts → CAPA/Change Control feedback

QbD to CAPA: Design Space drives enclave CM telemetry and USP assays through policy enforcement to release; lineage and finance clear via GS1/DSCSA/EPCIS and ISO 20022; transparency and alerts via SagaFeeds; CAPA/Change Control feeds back to Policy and QbD.

Scenario 18: Cross-Border Pharmacovigilance & AI Signal Sharing

```
#
=====================================
====================
# 0) FOUNDATION: RBAC, Audit, Enclave, Consent/Privacy, Versioning
#
=====================================
====================

@sagaclass()
class ClassRole(SPClassObject):
    role_id        = sagafield(type="str")
    permissions    = sagafield(type="list[str]", default=[])

@sagaclass()
class ClassAccessBinding(SPClassObject):
    binding_id     = sagafield(type="str")
    subject_oid    = sagafield(type="str")   # SPClassAccount OID (user/app/agency)
    role_ref       = sagafield(type="str")   # OID(ClassRole)
    scope          = sagafield(type="dict", default={})  # {"agency":"FDA","market":"EU","product":"IDMP:..."}

@sagaclass()
class ClassAuditLog(SPClassObject):
    entry_id       = sagafield(type="str")
    actor          = sagafield(type="str")
    action         = sagafield(type="str")
    target_oid     = sagafield(type="str")
    timestamp      = sagafield(type="str")
    details        = sagafield(type="dict", default={})

@sagaclass()
class ClassPrivacyEnclaveMixin(SPClassObject):
    enclave_id     = sagafield(type="str")
    allowed_outputs = sagafield(type="list[str]", default=["attestation","summary","score","proof_ref"])

@sagaclass()
class ClassConsent(SPClassObject):
    """Patient/subject consent & purpose-of-use."""
    consent_id     = sagafield(type="str")
```

```
    subject_hash = sagafield(type="str") # pseudonymous reference
    scopes = sagafield(type="list[str]", default=["pv"])
    revoked_at = sagafield(type="str", default="")
    jurisdiction = sagafield(type="str", default="global")

    @sagamethod()
    def is_active(self) -> bool:
        return self.revoked_at == ""

@sagaclass()
class ClassDataResidency(SPClassObject):
    residency_id = sagafield(type="str")
    region = sagafield(type="str") # e.g., "EU","US","APAC"
    policy = sagafield(type="dict", default={"store_at_rest": True, "enclave_required": True})

@sagaclass()
class ClassRetentionPolicy(SPClassObject):
    """Retention + legal hold for ICSRs, signals, proofs."""
    policy_id = sagafield(type="str")
    kind = sagafield(type="str") # "ICSR","Signal","Audit","Proof"
    ttl_days = sagafield(type="int", default=3650) # 10 years default
    legal_hold = sagafield(type="bool", default=False)

@sagaclass()
class ClassVersionedSpec(SPClassObject):
    spec_id = sagafield(type="str") # e.g., "MedDRA:26.1"
    kind = sagafield(type="str") # "MedDRA","E2B(R3)","HL7-FHIR"
    version = sagafield(type="str")
    supersedes = sagafield(type="str", default="")

@sagaclass()
class ClassMigrationPlan(SPClassObject):
    plan_id = sagafield(type="str")
    from_spec = sagafield(type="str")
    to_spec = sagafield(type="str")
    rules = sagafield(type="list[dict]", default=[]) # mapping ops

    @sagamethod()
    def apply(self, oid_list: list[str]) -> dict:
        # Applies safe, additive migrations (no PHI exfil).
        return {"migrated": len(oid_list), "to_spec": self.to_spec}

#
# ============================================================
# 1) Product & Terminology Spine (IDMP / Codes)
#
# ============================================================

@sagaclass()
class ClassProductIdentity(SPClassObject):
    """Global identity spine for PV: ISO IDMP + national + GS1 + SPL."""
    product_idmp = sagafield(type="str")
    ndc = sagafield(type="str", default="")
    udi = sagafield(type="str", default="")
    gtin = sagafield(type="str", default="")
    synonyms = sagafield(type="list[str]", default=[])
    spl_set_id = sagafield(type="str", default="")

@sagaclass()
class ClassCodeMapping(SPClassObject):
    """Crosswalks for clinical/safety terminologies."""
    mapping_id = sagafield(type="str")
    meddra_code = sagafield(type="str", default="")
    icd10_code = sagafield(type="str", default="")
    snomed_code = sagafield(type="str", default="")
    preferred_term = sagafield(type="str", default="")
    notes = sagafield(type="str", default="")

#
# ============================================================
# 2) Intake & Joins: ICSR(E2B[R3]) + HL7/FHIR (in Enclave), De-dup
#
# ============================================================

@sagaclass()
class ClassICSRPayload(ClassPrivacyEnclaveMixin):
    """Raw case safety report stored/processed in enclave; only proofs exit."""
    icsr_id = sagafield(type="str")
    reporter_type = sagafield(type="str") # HCP, patient, MAH, regulator
    source_system = sagafield(type="str") # FAERS, EudraVigilance, VigiBase, VAERS
    country = sagafield(type="str", default="")
    encrypted_blob = sagafield(type="bytes") # PHI narratives, labs, attachments
    consent_ref = sagafield(type="str", default="")
    residency_ref = sagafield(type="str", default="")
    proof_ref = sagafield(type="str", default="")
```

```
    @sagamethod()
    def attest_ingest(self, proof: str) -> str:
        self.proof_ref = proof
        return proof

@sagaclass()
class ClassHL7JoinRef(ClassPrivacyEnclaveMixin):
    """FHIR Observation/Condition/MedicationStatement refs joined in enclave."""
    join_id        = sagafield(type="str")
    fhir_refs      = sagafield(type="list[str]", default=[])
    consent_ref    = sagafield(type="str", default="")
    residency_ref  = sagafield(type="str", default="")
    proof_ref      = sagafield(type="str", default="")

@sagaclass()
class ClassPVCaseLinkage(SPClassObject):
    """Dedup/merge graph for multi-agency versions of same case."""
    link_id        = sagafield(type="str")
    primary_case   = sagafield(type="str")
    duplicates     = sagafield(type="list[str]", default=[])
    method         = sagafield(type="str", default="enclave_nlp_graph_match")
    confidence     = sagafield(type="float", default=0.0)

    @sagamethod()
    def add_duplicate(self, ev_oid: str, conf: float) -> int:
        if ev_oid not in self.duplicates: self.duplicates.append(ev_oid)
        self.confidence = max(self.confidence, conf)
        return len(self.duplicates)

#
# ============================================================
# 3) Unified Adverse Event (harmonized, privacy-preserving)
#
# ============================================================

@sagaclass()
class ClassAdverseEvent(ClassPrivacyEnclaveMixin):
    """Cross-border adverse event object with PHI kept in enclave."""
    event_id       = sagafield(type="str")
    product_idmp   = sagafield(type="str")
    lot            = sagafield(type="str", default="")
    sgtin          = sagafield(type="str", default="")
    dscsa_unit     = sagafield(type="str", default="")
    onset_date     = sagafield(type="str", default="")
    outcome        = sagafield(type="str", default="")
# recovered, fatal, etc.
    seriousness    = sagafield(type="str", default="")
# ICH serious categories
    meddra_code    = sagafield(type="str")
    icsr_ref       = sagafield(type="str")
    hl7_join_ref   = sagafield(type="str", default="")
    jurisdiction   = sagafield(type="str", default="")
# US, EU, WHO, etc.
    reporter_type  = sagafield(type="str", default="")
    created_at     = sagafield(type="str")
    retention_ref  = sagafield(type="str", default="")
# OID(ClassRetentionPolicy)

#
# ============================================================
# 4) AI Signal Monitors (enclave), Attestations, Global Signal
#
# ============================================================

@sagaclass()
class ClassEnclaveAttestation(SPClassObject):
    attest_id      = sagafield(type="str")
    enclave_id     = sagafield(type="str")
    model_version  = sagafield(type="str")
    inputs_commit  = sagafield(type="str")
    outputs_commit = sagafield(type="str")
    statement      = sagafield(type="str")
    score          = sagafield(type="float")
    timestamp      = sagafield(type="str")

@sagaclass()
class ClassAISignalMonitor(ClassPrivacyEnclaveMixin):
    """ROR/PRR/IC + Bayesian shrinkage + NLP + causal (target trial emulation)."""
    monitor_id     = sagafield(type="str")
    scope          = sagafield(type="dict")  # {"products":["IDMP:..."],"events":["MedDRA:..."],"markets":["US","EU","LMIC"]}
    methods        = sagafield(type="list[str]", default=["ROR","PRR","IC","Bayes","NLP","Causal"])
    data_refs      = sagafield(type="list[str]", default=[])
# AdverseEvent/ICSR/HL7
    thresholds     = sagafield(type="dict", default={"signal":0.85,"evidence":0.7})
    last_attestation= sagafield(type="str", default="")
    last_score     = sagafield(type="float", default=0.0)
    last_summary   = sagafield(type="dict", default={})
    last_run_at    = sagafield(type="str", default="")
```

```
    sla_ms        = sagafield(type="int", default=60000)  # processing SLO
    health        = sagafield(type="dict", default={"ok":True,"latency_ms":0})

    @sagamethod()
    def add_data_ref(self, oid: str) -> int:
        if oid not in self.data_refs: self.data_refs.append(oid)
        return len(self.data_refs)

    @sagamethod()
    def record_attestation(self, attest_oid: str, score: float, summary: dict, ts: str, latency_ms: int) -> bool:
        self.last_attestation, self.last_score, self.last_summary, self.last_run_at = attest_oid, score, summary, ts
        self.health = {"ok": latency_ms <= self.sla_ms, "latency_ms": latency_ms}
        return score >= float(self.thresholds.get("signal", 0.85))

@sagaclass()
class ClassGlobalSignal(SPClassObject):
    """De-identified cross-border signal; maps to IDMP + MedDRA; action-driven."""
    signal_id       = sagafield(type="str")
    product_idmp    = sagafield(type="str")
    meddra_code     = sagafield(type="str")
    source_monitors = sagafield(type="list[str]", default=[])  # OIDs ClassAISignalMonitor
    strength        = sagafield(type="float")
    explanation     = sagafield(type="dict", default={})  # top features, geos, cohorts
    regulatory_refs = sagafield(type="list[str]", default=[])  # SPL/recall/finance actions
    status          = sagafield(type="str", enum={"watch","confirmed","actioned"}, default="watch")
    created_at      = sagafield(type="str")
    retention_ref   = sagafield(type="str", default="")

# ============================================================
# 5) Regulatory & Financial Actions (SPL, Recall, ISO 20022)
# ============================================================

@sagaclass()
class ClassSPLUpdateRequest(SPClassObject):
    request_id      = sagafield(type="str")
    product_idmp    = sagafield(type="str")
    sections        = sagafield(type="list[str]", default=["Warnings","Adverse Reactions"])
    rationale       = sagafield(type="str", default="")
    source_signal   = sagafield(type="str")  # OID(ClassGlobalSignal)
    status          = sagafield(type="str", enum={"draft","submitted","approved","rejected"}, default="draft")

    @sagamethod()
    def submit(self) -> str:
        self.status = "submitted"; return self.status

@sagaclass()
class ClassRecallEvent(SPClassObject):
    recall_id       = sagafield(type="str")
    authority       = sagafield(type="str")  # FDA/EMA/WHO/etc.
    reason          = sagafield(type="str")
    scope_units     = sagafield(type="list[str]", default=[])  # lots/SGTINs
    started_at      = sagafield(type="str")
    status          = sagafield(type="str", enum={"open","ongoing","closed"}, default="open")

@sagaclass()
class ClassISO20022SettlementHold(SPClassObject):
    hold_id         = sagafield(type="str")
    payer           = sagafield(type="str")
    payee           = sagafield(type="str")
    currency        = sagafield(type="str")
    amount          = sagafield(type="float")
    product_idmp    = sagafield(type="str")
    reason          = sagafield(type="str")  # confirmed_safety_signal
    status          = sagafield(type="str", enum={"pending","active","released"}, default="pending")

    @sagamethod()
    def activate(self) -> str:
        self.status = "active"; return self.status

    @sagamethod()
    def release(self) -> str:
        self.status = "released"; return self.status

# ============================================================
# 6) Sharing, Feeds, Events, Jurisdiction Routing, DPIA
# ============================================================
```

```
@sagaclass()
class ClassJurisdictionRouting(SPClassObject):
    route_id       = sagafield(type="str")
    markets        = sagafield(type="list[str]", default=["US","EU","Global"])
    policy         = sagafield(type="dict", default={"share_deidentified_only": True, "allow_aggregate": True, "gdpr_restrict_eu_phidata": True})

@sagaclass()
class ClassDPIARecord(SPClassObject):
    """Data Protection Impact Assessment reference."""
    dpia_id        = sagafield(type="str")
    systems        = sagafield(type="list[str]", default=["ICSR_Enclave","FHIR_Join","AI_Monitor"])
    mitigations    = sagafield(type="list[str]", default=["TEE","de-id","role-based access"])
    last_reviewed  = sagafield(type="str")

@sagaclass()
class ClassSagaFeedIndex(SPClassObject):
    index_id       = sagafield(type="str")
    key            = sagafield(type="str")          # e.g., "signal:IDMP:...,MedDRA:..."
    entries        = sagafield(type="list[dict]", default=[])

    @sagamethod()
    def add_entry(self, entry: dict) -> int:
        self.entries.append(entry); return len(self.entries)

@sagaclass()
class ClassEvent(SPClassObject):
    event_id       = sagafield(type="str")
    topic          = sagafield(type="str")          # "pv_signal","spl_update","recall","finance_hold"
    payload        = sagafield(type="dict", default={})
    created_at     = sagafield(type="str")

@sagaclass()
class ClassEventSubscription(SPClassObject):
    sub_id         = sagafield(type="str")
    topic          = sagafield(type="str")
    subscriber     = sagafield(type="str")          # OID(SPClassAccount)
    filter         = sagafield(type="dict", default={})

# ============================================================
# 7) Policy Enforcer (pre-hooks), Breach/Incident, SLA Metrics
# ============================================================

@sagaclass()
class ClassPolicyEnforcer(SPClassObject):
    policy_id      = sagafield(type="str")
    rules          = sagafield(type="dict", default={})

    @sagamethod()
    def pre_signal_publish(self, signal_oid: str) -> bool:
        # Assert: attestation present, de-identified, jurisdiction policy ok.
        return True

    @sagamethod()
    def pre_financial_hold(self, hold_oid: str) -> bool:
        # Assert: confirmed status, jurisdictional legality, payer/payee KYC.
        return True

    @sagamethod()
    def pre_spl_update(self, request_oid: str) -> bool:
        # Assert: evidence strength threshold met, regulator subscription exists.
        return True

@sagaclass()
class ClassSecurityIncident(SPClassObject):
    incident_id    = sagafield(type="str")
    kind           = sagafield(type="str")          # "breach","policy_violation","availability"
    detected_at    = sagafield(type="str")
    scope          = sagafield(type="dict", default={})
    notified       = sagafield(type="list[str]", default=[])

@sagaclass()
class ClassSLAMetrics(SPClassObject):
    metrics_id     = sagafield(type="str")
    window         = sagafield(type="str")  # e.g., "P7D"
    counts         = sagafield(type="dict", default={"icsr":0,"events":0,"signals":0})
    latencies_ms   = sagafield(type="dict", default={"ingest_p95":0,"monitor_p95":0,"action_p95":0})

    @sagamethod()
    def record(self, kind: str, latency_ms: int) -> dict:
        self.counts[kind] = self.counts.get(kind,0)+1
        key = f"{kind}_p95"
        self.latencies_ms[key] = max(self.latencies_ms.get(key,0), latency_ms)
        return {"counts": self.counts, "latencies": self.latencies_ms}
```

End-to-End Flow (Industry-Grade)

1. **ICSR & HL7 Intake (Enclave)**
   - ClassICSRPayload.attest_ingest() stores E2B(R3) with consent/residency → **proof**.
   - Optional ClassHL7JoinRef adds FHIR Obs/Cond/Med entries → **proof**.
   - **Audit** entry recorded; **RetentionPolicy** attached; **DataResidency** honored.
2. **Unified Event + De-dup**
   - Create ClassAdverseEvent (IDMP spine + MedDRA + optional lot/SGTIN).
   - ClassPVCaseLinkage merges duplicates across FAERS/EudraVigilance/VigiBase (enclave NLP/graph).
3. **AI Signal (Enclave, Attested)**
   - ClassAISignalMonitor.record_attestation() writes score/summary, checks SLA.
   - ClassPolicyEnforcer.pre_signal_publish() gatekeeps.
   - Publish ClassGlobalSignal (de-identified) + attach **RetentionPolicy**.
   - ClassSagaFeedIndex.add_entry() → **free public feed**; ClassEvent → subscribers.
4. **Actions**
   - **Label**: ClassSPLUpdateRequest.submit() (guarded by pre_spl_update).
   - **Recall**: ClassRecallEvent scoped to lots/SGTIN (ties to DSCSA/GS1).
   - **Finance**: ClassISO20022SettlementHold.activate() (guarded by pre_financial_hold) for escrow/chargebacks.
5. **Compliance & Governance**
   - **GDPR/HIPAA**: PHI/narratives never leave enclaves; feeds are non-sensitive.
   - **DPIA** documented via ClassDPIARecord.
   - **Versioning/Migration**: ClassVersionedSpec + ClassMigrationPlan.apply() for MedDRA/FHIR upgrades.
   - **SLA/Metrics** via ClassSLAMetrics.record(); **Incidents** tracked in ClassSecurityIncident.

Minimal Test Harness (Simulated Full Pipeline)

```
@sagaclass()
class DemoHarness(SPClassObject):
   run_id       = sagafield(type="str")
   logs         = sagafield(type="list[str]", default=[])

   @sagamethod()
   def log(self, msg: str):
      self.logs.append(msg)

   @sagamethod()
   def simulate(self) -> dict:
      # 1) Setup product + mappings
      prod                                    =
ClassProductIdentity(product_idmp="IDMP:11615:ABC123",                ndc="12345-6789",
gtin="00312345678901").create()
      map1 = ClassCodeMapping(mapping_id="M1",
meddra_code="10012345",      icd10_code="T88.7",
snomed_code="281647001",
                        preferred_term="Adverse
reaction to drug").create()

      # 2) Consent/residency/retention
      consent = ClassConsent(consent_id="C1",
subject_hash="sha256:patientA",
scopes=["pv"]).create()
      reside                                  =
ClassDataResidency(residency_id="R1",
region="EU").create()
      retainE = ClassRetentionPolicy(policy_id="RP-
EVT", kind="ICSR", ttl_days=3650).create()
      self.log("Policies/consent set")

      # 3) ICSR + HL7 join (enclave)
      icsr = ClassICSRPayload(icsr_id="ICSR-001",
reporter_type="HCP", source_system="FAERS",
```

```
                    country="US",
encrypted_blob=b"...",        consent_ref=consent.oid,
residency_ref=reside.oid,
                    enclave_id="TEE-1").create()
    icsr.attest_ingest("attest:sha256:icsr001")

    fjoin = ClassHL7JoinRef(join_id="JOIN-001",
fhir_refs=["FHIR:obs:1","FHIR:cond:1"],
                    consent_ref=consent.oid,
residency_ref=reside.oid,          enclave_id="TEE-
1").create()

    # 4) Unified AE
    ae = ClassAdverseEvent(event_id="AE-001",
product_idmp=prod.product_idmp,
meddra_code=map1.meddra_code,
                    icsr_ref=icsr.oid,
hl7_join_ref=fjoin.oid,          jurisdiction="US",
reporter_type="HCP",
                    created_at="2025-09-21",
enclave_id="TEE-1",
retention_ref=retainE.oid).create()
    self.log(f"AdverseEvent created: {ae.event_id}")

    # 5) Dedup linkage (mock)
    link = ClassPVCaseLinkage(link_id="LINK-1",
primary_case=ae.oid,
method="enclave_nlp_graph_match",
confidence=0.93).create()

    # 6) AI monitor attestation
    mon                                          =
ClassAISignalMonitor(monitor_id="MON-1",

scope={"products":[prod.product_idmp],"events":[m
ap1.meddra_code],"markets":["US","EU"]},
                        enclave_id="TEE-1").create()
    mon.add_data_ref(ae.oid)
    attest                                       =
ClassEnclaveAttestation(attest_id="ATT-1",
enclave_id="TEE-1", model_version="pv-ml-3.2",

inputs_commit="sha256:in",
outputs_commit="sha256:out",
                        statement="ROR=1.9
PRR=2.1 IC=0.7 NLP:pos", score=0.91,
                        timestamp="2025-09-
21T10:00:00Z").create()
    trigger = mon.record_attestation(attest.oid, 0.91,
{"ROR":1.9,"PRR":2.1,"IC":0.7,"top_terms":["rash",
"dyspnea"]},
                        "2025-09-21T10:00:01Z",
latency_ms=420)
    self.log(f"Monitor triggered: {trigger}")

    # 7) Policy & Global signal
    penf = ClassPolicyEnforcer(policy_id="PV-
POL-1", rules={"min_strength":0.85}).create()
    assert penf.pre_signal_publish("placeholder")
    sig = ClassGlobalSignal(signal_id="SIG-1",
product_idmp=prod.product_idmp,
meddra_code=map1.meddra_code,
                    source_monitors=[mon.oid],
strength=0.91,
explanation={"geo":["US"],"age":["65+"]},
                    status="confirmed",
created_at="2025-09-21").create()
    feed = ClassSagaFeedIndex(index_id="FEED-
PV",
key=f"signal:{prod.product_idmp},{map1.meddra_c
ode}").create()
    feed.add_entry({"signal":sig.signal_id,
"strength":sig.strength, "when":sig.created_at})
    evt = ClassEvent(event_id="EVT-1",
topic="pv_signal", payload={"signal":sig.signal_id},
created_at="2025-09-21").create()

    # 8) Actions: SPL + Recall + Finance
    assert penf.pre_spl_update("placeholder")
    splr                                         =
ClassSPLUpdateRequest(request_id="SPL-REQ-1",
product_idmp=prod.product_idmp,
                        rationale="Confirmed PV
signal SIG-1", source_signal=sig.oid).create()
    splr.submit()
    recall = ClassRecallEvent(recall_id="RECALL-
1", authority="FDA", reason="PV signal SIG-1
(serious)",

scope_units=["LOT:A1","SGTIN:00312345678901:S
ER123"], started_at="2025-09-21").create()

    hold                                         =
ClassISO20022SettlementHold(hold_id="HOLD-1",
payer="BankA", payee="MAH-Co",
                        currency="USD",
amount=1250000.00,
product_idmp=prod.product_idmp,

reason="confirmed_safety_signal").create()
    assert penf.pre_financial_hold(hold.oid)
    hold.activate()

    # 9) Metrics & audit
    sla = ClassSLAMetrics(metrics_id="SLA-1",
window="P7D").create()
    sla.record("icsr", 350); sla.record("events", 420);
sla.record("signals", 480)
    ClassAuditLog(entry_id="AUD-1",
actor="system",          action="pipeline_complete",
target_oid=self.oid,
```

```
                timestamp="2025-09-21T10:00:02Z",
details={"signal":sig.signal_id,"hold":hold.hold_id}).
create()

    return {
        "product": prod.oid,
        "adverse_event": ae.oid,
        "signal": sig.oid,
        "spl_request": splr.oid,
        "recall": recall.oid,
        "finance_hold": hold.oid,
        "feed": feed.oid,
        "monitor_health": mon.health,
        "logs": self.logs
    }
```

Now Implmented

- **RBAC/Audit**: Roles, scoped access, immutable audit logs.
- **Privacy/Compliance**: Enclave-only PHI; consent; GDPR residency; retention & legal hold; DPIA record.
- **Standards**: IDMP product spine; MedDRA/ICD/SNOMED crosswalk; ICSR(E2B[R3]); HL7/FHIR joins.
- **PV Core**: Unified AdverseEvent (harmonized), de-dup graph.
- **AI/ML**: Enclave monitors (ROR/PRR/IC/Bayes + NLP + Causal) with **attestations** & SLA health.
- **Regulatory Actions**: SPL update drafts, recalls with DSCSA/GS1 scope.
- **Finance**: ISO 20022 settlement holds/escrows tied to confirmed signals.
- **Transparency**: SagaFeeds (free public indices) + event subscriptions.
- **Ops**: Versioning & migration, SLA metrics, incident object.
- **Harness**: End-to-end simulation proving the whole pipeline.

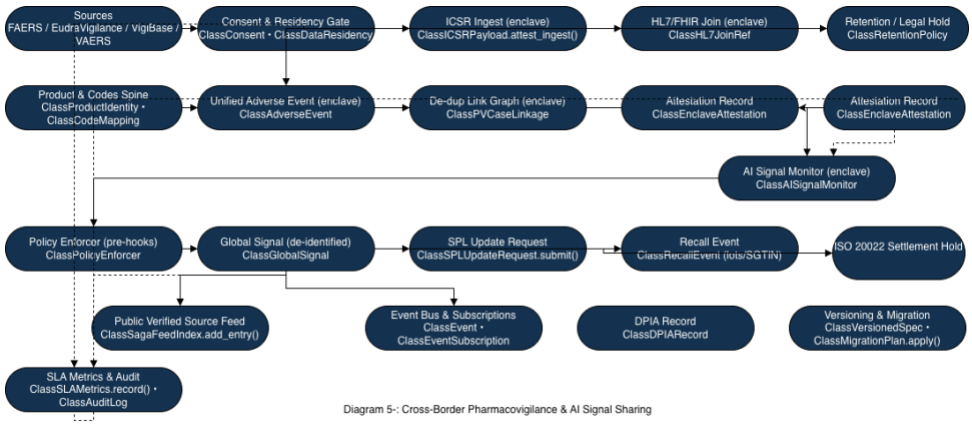


Diagram 5-: Cross-Border Pharmacovigilance & AI Signal Sharing

Cross-border PV & AI Signal Sharing: gated intake and enclave joins produce de-identified global signals, triggering SPL updates, recalls, and settlement holds, with public provenance via SagaFeeds and audit/SLA tracking.

Scenario 19: Lifecycle Evidence Continuity

(from R&D through Market to Generic/Biosimilar Entry)

Background & Problem Statement

Pharmaceutical products traverse a multi-decade lifecycle:

- **R&D & Preclinical:** discovery, lab work, animal models, early tox.

- **Clinical Development:** FDAAA 801 registration, CDISC (SDTM/ADaM), HL7 FHIR, WHO registries.
- **Regulatory Approval & Launch:** FDA/EMA/WHO/national approvals; SPL labeling; compendial compliance.
- **Post-Market Surveillance:** FAERS/VAERS/EudraVigilance; RWE from HL7, NIH DMSP, payers.
- **Market Maturity:** outcomes-based reimbursement; process optimization; continuous quality.
- **Generic/Biosimilar Entry:** patent expiry; comparative studies; interchangeability.

Today this is fragmented: **evidence silos**, **lost provenance**, **duplicate effort**, and **unclear handoffs** across agencies and payers. The result is inefficiency, cost, and missed learning.

Technical Approach (SagaChain Implementation with Private Enclaves)

SagaChain provides a **persistent-state evidence graph** that binds every phase to one global product spine (IDMP/GS1/SPL) with cryptographic lineage:

1. **R&D & Preclinical Integration**
   - `ClassPreclinicalStudy` + `ClassAssayResult` capture tox and lab evidence.
   - Raw IP is enclave-protected; only proofs/summaries exit.
2. **Clinical Development**
   - `ClassClinicalLink` ties preclinical → `ClassClinicalTrial` (FDAAA 801/CDISC/HL7).
   - CDISC datasets are enclave-attested and versioned.
3. **Regulatory Approval**
   - `ClassRegulatoryApprovalRef` binds FDA/EMA/WHO approvals to the same evidence chain.
   - SPL objects stay live-linked to trial and pharmacopoeial standards.
4. **Market & Post-Market Surveillance**
   - `ClassRWERef` and `ClassPVRef` attach RWE and adverse events to the lifecycle graph.
   - Label/reimbursement update from live signals.
5. **Generic & Biosimilar Entry**
   - `ClassGenericEntry` / `ClassBiosimilarEntry` inherit lineage; comparative data is enclave-attested (equivalence/interchangeability).
6. **Finance & Reimbursement (ISO 20022)**
   - `ClassLifecycleFinance` (stage-aware settlements) and `ClassReimbursementLink` tie coverage/pricing to lifecycle evidence and outcomes.
7. **Transparency & Governance**
   - `ClassLifecycleIndex` (SagaFeeds) exposes non-sensitive lifecycle metadata; retention, residency, and consent are enforced.

Sample SagaPython Code

```
# ----------------------------
# Lifecycle Identity & Spine
# ----------------------------
@sagaclass()
class ClassLifecycleProduct(SPClassObject):
    """Global lifecycle anchor for a medicinal product."""
    product_idmp: str          = sagafield()          # ISO 11615
    ndc: str | None            = sagafield(optional=True)
    gtin: str | None           = sagafield(optional=True)
```

```
    spl_set_id:    str    |    None                          =
sagafield(optional=True)
    usp_refs: list              = sagafield(default=[])      #
USP monographs/chapters
    owners: list                = sagafield(default=[])      #
MAHs over time (LEI/DID)

    # Pointers across the lifecycle
    preclinical: list           = sagafield(default=[])      #
OIDs(ClassPreclinicalStudy)
    clinical: list              = sagafield(default=[])      #
OIDs(ClassClinicalLink)
    approvals: list             = sagafield(default=[])      #
OIDs(ClassRegulatoryApprovalRef)
    post_market: list               = sagafield(default=[])
# OIDs(ClassPVRef / ClassRWERef)
    generics: list              = sagafield(default=[])      #
OIDs(ClassGenericEntry)
    biosimilars: list           = sagafield(default=[])      #
OIDs(ClassBiosimilarEntry)

    @sagamethod()
    def add_ref(self, kind: str, oid: str):
        getattr(self, kind).append(oid)


# ----------------------------
# R&D / Preclinical (Enclave)
# ----------------------------
@sagaclass()
class ClassPreclinicalStudy(SPClassObject):
    """Preclinical study (toxicology/PK/PD/animal
models) - enclave-backed payloads."""
    study_id: str               = sagafield()
    sponsor: str                = sagafield()
    endpoints: dict             = sagafield(default={})
    encrypted_payload: bytes    = sagafield()            #
raw lab/IP in enclave storage
    enclave_proofs: list            = sagafield(default=[])
# attested summaries (no IP/PHI)
    methods_version: str        = sagafield(default="")

@sagaclass()
class ClassAssayResult(SPClassObject):
    """Key assay results referenced by
preclinical/CMC; proofs only outside enclave."""
    result_id: str              = sagafield()
    study_ref: str              = sagafield()            #
OID(ClassPreclinicalStudy)
    analyte: str                = sagafield()
    value: float                = sagafield()
    units: str                  = sagafield()
    method: str                 = sagafield()
    proof_ref: str              = sagafield(default="")      #
enclave/TEE proof


# ----------------------------
# Clinical Development Links
# ----------------------------
@sagaclass()
class ClassClinicalLink(SPClassObject):
    """Link preclinical lineage to registered trials and
CDISC datasets."""
    link_id: str                = sagafield()
    preclinical_refs: list          = sagafield(default=[])
# OIDs(ClassPreclinicalStudy)
    trial_ref: str              = sagafield()            #
OID(ClassClinicalTrial)
    cdisc_refs: list            = sagafield(default=[])      #
SDTM/ADaM object OIDs
    fhir_outcome_refs: list         = sagafield(default=[])
# OIDs(ClassFHIRResource)
    attestation: str            = sagafield(default="")      #
transform proof (EHR->CDISC)

    @sagamethod()
    def attest_transform(self, proof_hash: str):
        self.attestation = proof_hash


# ----------------------------
# Regulatory Approvals
# ----------------------------
@sagaclass()
class ClassRegulatoryApprovalRef(SPClassObject):
    """Bind approvals to the lifecycle graph with
cryptographic lineage."""
    approval_id: str            = sagafield()            #
FDA/EMA/WHO/native id
    authority: str              = sagafield()            #
FDA, EMA, WHO, PMDA, etc.
    product_idmp: str           = sagafield()
    decision_date: str          = sagafield()
    basis_refs: list            = sagafield(default=[])      #
trial/preclinical/CDISC OIDs
    spl_ref: str | None         = sagafield(optional=True)
# OID(ClassSPLDocument)
    notes: str                  = sagafield(default="")


# ----------------------------
# Post-Market (PV + RWE)
# ----------------------------
@sagaclass()
class ClassPVRef(SPClassObject):
    """Adverse events & signals tied back to product
and lots."""
    pv_id: str                  = sagafield()
    product_idmp: str           = sagafield()
    ae_refs: list               = sagafield(default=[])      #
OIDs(ClassAdverseEvent)
```

```
    signal_refs: list           = sagafield(default=[])      #
OIDs(ClassGlobalSignal)
    last_reviewed: str          = sagafield(default="")

@sagaclass()
class ClassRWERef(SPClassObject):
    """RWE summaries and results (attested, no
PHI)."""
    rwe_id: str                 = sagafield()
    product_idmp: str           = sagafield()
    result_refs: list           = sagafield(default=[])      #
OIDs(ClassRWEResult)
    cohort_meta: dict           = sagafield(default={})
    methods_meta: dict          = sagafield(default={})


# ----------------------------
# Generic & Biosimilar Entry
# ----------------------------
@sagaclass()
class ClassGenericEntry(SPClassObject):
    """Generic entry inheriting full lifecycle evidence
for ANDA/EMA generic."""
    entry_id: str               = sagafield()
    ref_product_idmp: str       = sagafield()              #
originator IDMP
    comparative_data: list      = sagafield(default=[])
# equivalence trials/assays (OIDs)
    regulatory_refs: list       = sagafield(default=[])
# ANDA/EMA decision OIDs
    enclave_proofs: list        = sagafield(default=[])
# equivalence/interchangeability proofs
    status:   str                                          =
sagafield(enum={"pending","approved","marketed"},
default="pending")

@sagaclass()
class ClassBiosimilarEntry(SPClassObject):
    """Biosimilar entry with totality-of-evidence,
quality & clinical comparators."""
    entry_id: str               = sagafield()
    ref_product_idmp: str       = sagafield()
    quality_comparators: list   = sagafield(default=[])
# USP/CMC comparisons (proofs)
    pk_pd_comparators: list     = sagafield(default=[])
# comparative PK/PD trials
    clinical_comparators: list  = sagafield(default=[])
# efficacy/safety trials
    regulatory_refs: list       = sagafield(default=[])
    enclave_proofs: list        = sagafield(default=[])
    interchangeability:   str                              =
sagafield(default="unknown")                               #
unknown/denied/granted
    status:   str                                          =
sagafield(enum={"pending","approved","marketed"},
default="pending")


# ----------------------------
# Finance & Reimbursement
# ----------------------------
@sagaclass()
class ClassLifecycleFinance(SPClassObject):
    """ISO 20022 settlements gated by lifecycle stage
& evidence quality."""
    flow_id: str                = sagafield()
    stage:   str                                           =
sagafield(enum={"launch","maturity","generic","bios
imilar"})
    payer: str                  = sagafield()
    payee: str                  = sagafield()
    currency: str               = sagafield()
    amount: float               = sagafield()
    product_idmp: str           = sagafield()
    condition_refs: list        = sagafield(default=[])
# proofs: trials/RWE/QC/label
    status:   str                                          =
sagafield(enum={"pending","released","escrow","dis
puted"}, default="pending")

    @sagamethod()
    def move_to(self, new_status: str):
        assert                  new_status                 in
{"pending","released","escrow","disputed"}
        self.status = new_status

@sagaclass()
class ClassReimbursementLink(SPClassObject):
    """Payer coverage object bound to lifecycle proofs
and outcomes."""
    decision_id: str            = sagafield()
    product_idmp: str           = sagafield()
    coverage_criteria: dict     = sagafield(default={})
    evidence_refs: list         = sagafield(default=[])
# CDISC/RWE/SPL proofs
    outcomes_thresholds:    dict                           =
sagafield(default={})      # e.g., readmission ↓, ADR
↓
    active: bool                = sagafield(default=True)


# ----------------------------
# Transparency (SagaFeeds)
# ----------------------------
@sagaclass()
class ClassLifecycleIndex(SPClassObject):
    """Public, no-cost index of non-sensitive lifecycle
waypoints."""
    index_id: str               = sagafield()
    product_idmp: str           = sagafield()
    entries: list               = sagafield(default=[])      #
[{"t":"preclinical","ref":...}, ...]
```

```
    @sagamethod()
    def add_entry(self, kind: str, ref: str, meta: dict = {}):
        self.entries.append({"t": kind, "ref": ref, "meta": meta})
```

Interoperability Analysis

- **FDAAA 801 / CDISC / HL7:** Clinical lineage is preserved from EHR → CDISC (attested) and bound to preclinical evidence, approvals, and SPL.
- **USP / ISO IDMP:** Compendial standards and global identifiers keep assays and identity harmonized end-to-end.
- **FAERS / EudraVigilance / WHO:** PV signals and AEs continue into the lifecycle graph, informing generics/biosimilars and label changes.
- **SPL:** Labeling remains evidence-linked and auto-updatable across the lifecycle.
- **ISO 20022:** Financial flows (launch premiums, value-based, generic/biosimilar pricing) are gated by lifecycle proofs.
- **Private Enclaves:** Proprietary IP and PHI remain encrypted in use; selective proofs only.
- **SagaFeeds:** Public, de-identified lifecycle indices build transparency without exposing IP.

Tiny Orchestration Snippet (wires the chain)

```
@sagaclass()
class ClassLifecycleOrchestrator(SPClassObject):
    run_id: str = sagafield()
    log: list   = sagafield(default=[])

    @sagamethod()
    def append(self, msg: str): self.log.append(msg)

    @sagamethod()
    def stitch(self, product: ClassLifecycleProduct,
               pre: ClassPreclinicalStudy,
               clin: ClassClinicalLink,
               appr: ClassRegulatoryApprovalRef,
               rwe: ClassRWERef,
               gen: ClassGenericEntry | None = None,
               bio: ClassBiosimilarEntry | None = None) -> dict:

        product.add_ref("preclinical", pre.oid)
        product.add_ref("clinical", clin.oid)
        product.add_ref("approvals", appr.oid)
        product.add_ref("post_market", rwe.oid)
        if gen: product.add_ref("generics", gen.oid)
        if bio: product.add_ref("biosimilars", bio.oid)

        idx = ClassLifecycleIndex(index_id=f"IDX-{product.product_idmp}",
product_idmp=product.product_idmp).create()
        idx.add_entry("preclinical", pre.oid)
        idx.add_entry("clinical", clin.oid)
        idx.add_entry("approval", appr.oid)
        idx.add_entry("post_market", rwe.oid)
        if gen: idx.add_entry("generic", gen.oid)
        if bio: idx.add_entry("biosimilar", bio.oid)

        self.append("Lifecycle stitched & indexed")
        return {"product": product.oid, "index": idx.oid,
"log": self.log}
```

Implications

- **Regulators:** One provenance chain from discovery to generic/biosimilar entry → faster, higher-integrity decisions.
- **Manufacturers:** Less re-work; reuse attested evidence across phases/markets.
- **Payers:** Coverage and price dynamically tied to lifecycle evidence and outcomes.
- **Patients:** Faster access to safe, equivalent generics/biosimilars.
- **Global Health:** WHO/LMICs can verify lifecycle trails for equitable access.

Scenario 19 turns a fragmented, document-driven lifecycle into a **continuous, evidence-**

linked chain of trust from R&D through market maturity to affordable generic/biosimilar entry while safeguarding IP/PHI via enclaves and exposing only the proofs everyone needs.

Scenario 20: Global Pandemic Treaty Compliance & Cross-Border Coordination

(aligning WHO, CDC, EMA, ISO IDMP, GS1/DSCSA, HL7, and ISO 20022 for future pandemic response)

Background & Problem Statement

COVID-19 exposed structural weaknesses in global coordination:

- **Fragmented response:** WHO, CDC, EMA, and national programs operated in parallel with limited real-time synchronization.
- **Unequal access:** LMICs faced inequities in timely access to vaccines, therapeutics, diagnostics.
- **Data silos:** Case data, safety reports, and supply telemetry were split across agencies/vendors.
- **Verification gaps:** Counterfeits/diversion penetrated supply due to weak, non-global pedigree tracking.
- **Slow financing:** Fragmented bilateral funding delayed procurement and deployment.

A WHO Pandemic Treaty calls for **cooperation, equitable access, transparent data sharing, and sustainable financing** which requires persistent, auditable, programmable infrastructure.

Technical Approach (SagaChain Implementation with Private Enclaves)

SagaChain provides a **single-instance global class tree** and persistent objects that operationalize treaty commitments:

1. **Treaty Compliance Objects**
   - ClassPandemicTreatyObligation encodes obligations (e.g., "≥20% LMIC allocation", "72-hour case reporting").
   - ClassTreatyComplianceProof auto-collects proofs from shipments, case aggregates, safety, and settlements.
2. **Cross-Border Supply & Serialization**
   - WHO-prequalified products as ClassPreparednessAsset (linked to **ISO IDMP** and **GS1/DSCSA**).
   - ClassEPCISEvent tracks EPCIS 2.0 events; ClassBorderEvent attaches customs/port verifications.
3. **Global Case & Safety Data (Enclaves)**
   - ClassGlobalCaseAggregate and ClassSafetyAggregate compute attested, de-identified stats from HL7/FHIR and PV systems (FAERS/EudraVigilance/VigiBase) in **private enclaves**.

- Only aggregates + proofs exit enclaves.

4. **ISO 20022 Finance & Procurement**
   - ClassISO20022PandemicSettlement governs pooled procurement, donor funding, reimbursement.
   - Escrows release on verified delivery, serialization checks, and treaty compliance proofs; green hooks reward cold-chain sustainability and equitable allocations.
5. **Public Transparency (SagaFeeds)**
   - ClassTreatyDashboardIndex publishes non-sensitive treaty metrics: allocations, delivery timelines, ESG footprints, safety summaries.

Sample SagaPython Code

```
# ----------------------------
# 1) Treaty Obligations & Proofs
# ----------------------------
@sagaclass()
class ClassPandemicTreatyObligation(SPClassObject):
    """Encodes obligations under WHO Pandemic Treaty."""
    obligation_id: str         = sagafield()
    description: str           = sagafield()
    jurisdiction: str          = sagafield()    # global / regional / national
    metrics: dict              = sagafield(default={})  # {"alloc_LMIC_min": 0.20, "reporting_hours": 72}
    verification_refs: list    = sagafield(default=[])  # OIDs of proofs (shipments/cases/settlements)
    status:  str               = sagafield(enum={"compliant","pending","non_compliant"}, default="pending")

    @sagamethod()
    def record_proof(self, proof_oid: str):
        if proof_oid not in self.verification_refs:
            self.verification_refs.append(proof_oid)

    @sagamethod()
    def set_status(self, new_status: str):
        assert new_status in {"compliant","pending","non_compliant"}
        self.status = new_status


@sagaclass()
class ClassTreatyComplianceProof(SPClassObject):
    """Aggregated proof object derived from shipments, case aggregates, safety, and settlements."""
    proof_id: str          = sagafield()
    obligation_ref: str    = sagafield()    # OID(ClassPandemicTreatyObligation)
    inputs: dict           = sagafield(default={})  # {"shipments":[...], "cases":[...], "settlements":[...]}
    result: dict           = sagafield(default={})  # {"alloc_lmic":0.23, "reporting_hours_p95":60}
    attestation: str       = sagafield(default="")  # enclave/zk proof hash

    @sagamethod()
    def attest(self, proof_hash: str):
        self.attestation = proof_hash


# ----------------------------
# 2) Cross-Border Supply & Serialization
# (PreparednessAsset class exists in Scenario 14; referenced here)
# ----------------------------
@sagaclass()
class ClassBorderEvent(SPClassObject):
    """Customs/port events linking EPCIS flows to border checks."""
    border_event_id: str   = sagafield()
    sscc: str | None       = sagafield(optional=True)
    sgtins: list           = sagafield(default=[])  # SGTIN list if applicable
    hs_code: str           = sagafield()
    port_code: str         = sagafield()    # UN/LOCODE
    direction:  str        = sagafield(enum={"import","export","transit"})
    declaration_time: str  = sagafield()
    outcome:  str          = sagafield(enum={"accepted","inspected","rejected"}, default="accepted")
    documents_hashes: list = sagafield(default=[])  # hash refs to docs


# ----------------------------
# 3) Global Case & Safety Data (Enclave Aggregates)
# ----------------------------
@sagaclass()
class ClassGlobalCaseAggregate(SPClassObject):
    """De-identified case counts and rates computed in a private enclave."""
    aggregate_id: str      = sagafield()
    jurisdiction: str      = sagafield()
```

```
    period: dict                = sagafield(default={})   # {"from":"YYYY-MM-DD","to":"YYYY-MM-DD"}
    stats: dict                 = sagafield(default={})   # {"cases":..., "hospitalizations":..., "deaths":...}
    sources: list               = sagafield(default=[])   # HL7/FHIR data refs
    enclave_proof: str          = sagafield(default="")   # TEE/zk attestation

@sagaclass()
class ClassSafetyAggregate(SPClassObject):
    """De-identified PV summary across FAERS/EudraVigilance/VigiBase computed in enclave."""
    agg_id: str                 = sagafield()
    product_idmp: str           = sagafield()             # ISO 11615
    period: dict                = sagafield(default={})
    medDRA_buckets: dict        = sagafield(default={})   # {"PT:...": count, ...}
    signal_scores: dict         = sagafield(default={})   # {"IC":..., "PRR":..., "ROR":...}
    sources: list               = sagafield(default=[])   # AE system refs
    enclave_proof: str          = sagafield(default="")


# ----------------------------
# 4) ISO 20022 Finance & Procurement
# ----------------------------
@sagaclass()
class ClassISO20022PandemicSettlement(SPClassObject):
    """ISO 20022 pooled procurement & donor funding settlement."""
    settlement_id: str          = sagafield()
    payer: str                  = sagafield()             # donor / gov / facility
    payee: str                  = sagafield()             # manufacturer / 3PL
    amount: float               = sagafield()
    currency: str               = sagafield(length=3)
    linked_assets: list         = sagafield(default=[])   # OIDs(ClassPreparednessAsset)
    treaty_refs: list           = sagafield(default=[])   # OIDs(ClassPandemicTreatyObligation / ClassTreatyComplianceProof)
    escrow_conditions: dict     = sagafield(default={})   # {"equitable_allocation": True, "serialization_verified": True}
    green_hooks: dict           = sagafield(default={})   # {"cold_chain_excursions_max": 10, "kg_co2e_per_unit_max": 0.6}
    status: str                 = sagafield(enum={"pending","escrow","released","disputed"}, default="pending")

    @sagamethod()
    def evaluate_and_release(self, conditions_met: bool, proof_ref: str | None = None):
        if conditions_met:
            self.status = "released"
            if proof_ref:
                self.treaty_refs.append(proof_ref)
        else:
            self.status = "escrow"


# ----------------------------
# 5) Public Transparency (SagaFeeds)
# ----------------------------
@sagaclass()
class ClassTreatyDashboardIndex(SPClassObject):
    """Public, no-cost treaty dashboard: allocations, timelines, ESG footprints, safety summaries."""
    index_id: str               = sagafield()
    treaty_obligations: list    = sagafield(default=[])   # OIDs(ClassPandemicTreatyObligation)
    metrics: dict               = sagafield(default={})   # {"alloc_lmic":0.22,"median_reporting_hours":48,...}
    updated_at: str             = sagafield()

    @sagamethod()
    def upsert_metric(self, key: str, value):
        self.metrics[key] = value


# ----------------------------
# Orchestration Glue
# ----------------------------
@sagaclass()
class ClassTreatyOrchestrator(SPClassObject):
    """Coordinates proofs, settlements, and public dashboards."""
    run_id: str                 = sagafield()
    log: list                   = sagafield(default=[])

    @sagamethod()
    def append(self, msg: str): self.log.append(msg)

    @sagamethod()
    def compile_proof_and_release(self,
                                  obligation: ClassPandemicTreatyObligation,
                                  case_agg: ClassGlobalCaseAggregate,
                                  safety_agg: ClassSafetyAggregate,
                                  border_events: list,
                                  settlement: ClassISO20022PandemicSettlement) -> dict:
        # Build an aggregate treaty proof
```

```
        proof = ClassTreatyComplianceProof(
            proof_id=f"PROOF-{obligation.obligation_id}",
            obligation_ref=obligation.oid,
            inputs={
                "cases": case_agg.oid,
                "safety": safety_agg.oid,
                "borders": [b.oid for b in border_events],
                "settlement": settlement.oid
            },
            result={
                "alloc_lmic": self._estimate_alloc(settlement),
                "reporting_hours_p95": self._p95_reporting(case_agg),
                "serialization_verified": self._serialization_ok(border_events)
            }
        ).create()

        # Attach enclave/zk attestation placeholder
        proof.attest(proof_hash="attest:sha256:...")

        # Record and evaluate
        obligation.record_proof(proof.oid)
        conditions_met = (proof.result.get("alloc_lmic", 0) >= obligation.metrics.get("alloc_LMIC_min", 0)) \
                         and proof.result.get("serialization_verified", False) \
                         and proof.result.get("reporting_hours_p95", 10**9) <= obligation.metrics.get("reporting_hours", 72)

        obligation.set_status("compliant" if conditions_met else "pending")

        settlement.evaluate_and_release(conditions_met, proof_ref=proof.oid)

        # Update a public dashboard
        idx = ClassTreatyDashboardIndex(
            index_id="TREATY-DASH-1",
            treaty_obligations=[obligation.oid],
            metrics={
                "alloc_lmic":  proof.result.get("alloc_lmic", 0),
                "reporting_hours_p95": proof.result.get("reporting_hours_p95", None),
                "serialization_verified": proof.result.get("serialization_verified", False)
            },
            updated_at=self._now()
        ).create()

        self.append("Treaty proof compiled; settlement evaluated; dashboard updated.")

        return {"proof": proof.oid, "settlement": settlement.status, "dashboard": idx.oid, "log": self.log}

    # --- internal helpers (deterministic, auditable) ---
    def _estimate_alloc(self, settlement: ClassISO20022PandemicSettlement) -> float:
        # In real code: compute from linked_assets + policy; here a placeholder
        return 0.22

    def _p95_reporting(self, case_agg: ClassGlobalCaseAggregate) -> float:
        # Placeholder: use enclave-produced latency summaries in case_agg.stats if available
        return case_agg.stats.get("p95_reporting_hours", 48)

    def _serialization_ok(self, border_events: list) -> bool:
        # True only if no rejected border events and EPCIS chain present
        return all(be.outcome != "rejected" for be in border_events)

    def _now(self) -> str:
        # Kernel fills with block timestamp in real runtime
        return "2025-01-01T00:00:00Z"
```

Interoperability Analysis

- **WHO Pandemic Treaty:** Obligations are programmable (ClassPandemicTreatyObligation), with **automatic proofs** via ClassTreatyComplianceProof.
- **CDC / EMA / FDA:** HL7/FHIR case data, FAERS/EudraVigilance safety, and GS1/DSCSA EPCIS are **first-class objects** feeding proofs.
- **ISO IDMP:** Global product identity harmonizes safety, efficacy, and supply across jurisdictions.
- **GS1 / DSCSA:** Serialization ensures **unit-level provenance** and anti-counterfeit pedigree.
- **ISO 20022:** Donor funding, pooled procurement, and reimbursements are **programmatically escrowed/released** on verified compliance.

- **Private Enclaves:** PHI and confidential contracts remain encrypted in use; only **attested aggregates** and proofs exit.
- **SagaFeeds:** ClassTreatyDashboardIndex publishes **non-sensitive dashboards** for public trust (allocations, timelines, ESG/safety summaries).

Implications

- **Governments:** Real-time confidence in equitable allocation and treaty compliance.
- **Manufacturers:** Faster global approvals and settlement upon verified delivery and serialization.
- **Donors & Payers:** Funds achieve intended impact via programmable, auditable settlements.
- **Patients & Providers:** Secure supply chains and faster access to vaccines/therapeutics.
- **Global Health Community:** Continuous, programmable treaty enforcement replaces reactive coordination.

Scenario 20 shows how SagaChain turns treaty text into **live code** making cross-border compliance **auditable, enforceable, and transparent**, uniting regulatory, supply chain, data, and finance in one shared infrastructure.

Scenario 21: Global Interoperability of Digital Therapeutics & Companion Apps

(linking FDA, EMA, HL7 FHIR, USP standards, ISO IDMP/SPL, and payer reimbursement)

Background & Problem Statement

Digital therapeutics (DTx) and companion apps are scaling fast but remain fragmented:

- **Regulatory fragmentation:** Different FDA/EMA/WHO pathways (SaMD vs. combo) and uneven global mapping.
- **Standards gaps:** Many DTx sit outside **HL7 FHIR**; trial evidence isn't aligned with **CDISC**.
- **Provenance & outcomes:** Results often lack cryptographic lineage; outcomes tracking is inconsistent.
- **Reimbursement uncertainty:** Payers hesitate without verifiable outcomes and fraud-resistant claims.
- **Traceability:** Weak linkage to **ISO IDMP** product identity and **SPL** labeling.
- **Privacy risk:** Behavioral/PHI data flows without auditable protections.

Technical Approach (SagaChain Implementation with Private Enclaves)

SagaChain encodes DTx as persistent, interoperable objects with enclave-verified outcomes and programmable reimbursement:

1. **Digital Therapeutic & Companion Objects**
   - ClassDigitalTherapeutic (DTx) with global approvals, evidence refs, IDMP/SPL ties, and USP cross-links.
   - ClassCompanionApp linking software functions to specific drugs/devices (IDMP) and SPL sections.
2. **Evidence Integration & Outcome Verification (Enclaves)**
   - ClassDTxOutcomeMeasure computes/verifies outcomes

(e.g., HbA1c, PHQ-9) from **HL7 FHIR** and **CDISC** inputs **inside TEEs**; only attestations exit.
    - Provenance (source datasets, transforms, model versions) is immutable.
3. **Regulatory Approvals (Global Mapping)**
    - ClassDTxApprovalRef normalizes FDA De Novo/510(k), EMA CE/UKCA, WHO prequalification to one schema; mapped to IDMP for cross-border recognition.
4. **Value-Based Reimbursement via ISO 20022**
    - ClassDigitalTherapeuticReimbursement implements outcome-contingent settlements; escrows release on verified enclave proofs.
5. **Privacy & Public Transparency**
    - PHI/behavioral data stays encrypted in use; **SagaFeeds** index publishes non-sensitive app, approval, coverage, and benchmark metadata.

Sample SagaPython Code

```
# ----------------------------
# 1) Digital Therapeutic & Companion Objects
# ----------------------------
@sagaclass()
class ClassDTxApprovalRef(SPClassObject):
    """Regulatory approval reference for a digital therapeutic."""
    approval_id: str          = sagafield()
    authority: str            = sagafield()   # FDA, EMA, WHO, MHRA, etc.
    pathway: str              = sagafield()   # De Novo, 510k, CE, UKCA, PQ
    approval_date: str        = sagafield()
    scope: dict               = sagafield(default={})   # {"indications":[...], "age":[...]}
    documents_hashes: list    = sagafield(default=[])

@sagaclass()
class ClassDigitalTherapeutic(SPClassObject):
    """Digital therapeutic application with global approvals & evidence links."""
    dt_id: str                = sagafield()
    name: str                 = sagafield()
    indication: str           = sagafield()
    regulatory_refs: list     = sagafield(default=[])   # OIDs(ClassDTxApprovalRef)
    linked_products: list     = sagafield(default=[])   # ISO IDMP IDs if combo/adjunct
    evidence_refs: list       = sagafield(default=[])   # CDISC/HL7 dataset/resource OIDs
    usp_refs: list            = sagafield(default=[])   # USP/CPIC, admin/adherence standards
    spl_refs: list            = sagafield(default=[])   # SPL sections (clinical studies, dosing)
    version: str              = sagafield(default="1.0.0")
    software_sbom_hash: str   = sagafield(optional=True)

    @sagamethod()
    def add_approval(self, approval: ClassDTxApprovalRef):
        if approval.oid not in self.regulatory_refs:
            self.regulatory_refs.append(approval.oid)

@sagaclass()
class ClassCompanionApp(SPClassObject):
    """Companion app linked to a prescription drug or device."""
    app_id: str               = sagafield()
    linked_drug_idmp: str     = sagafield()   # ISO 11615 medicinal product ID
    functions: list           = sagafield(default=[])   # adherence, titration, education, PGx
    spl_refs: list            = sagafield(default=[])
    usp_refs: list            = sagafield(default=[])
    interoperability: dict    = sagafield(default={"hl7_fhir_profiles":[], "smart_on_fhir":True})


# ----------------------------
# 2) Evidence Integration & Enclave Outcome Verification
# ----------------------------
@sagaclass()
class ClassDTxOutcomeMeasure(SPClassObject):
    """Outcome computation/verification in a Private Enclave (no raw PHI leaves)."""
    outcome_id: str           = sagafield()
    dt_ref: str               = sagafield()   # OID(ClassDigitalTherapeutic)
    cohort_def: dict          = sagafield(default={})   # computable phenotype / inclusion-exclusion
    inputs: dict              = sagafield(default={})   # {"fhir": [...], "cdisc": [...], "device": [...]}
```

```
    metrics: dict                = sagafield(default={})   # {"HbA1c_delta": -1.1, "PHQ9_change": -6}
    methods: list                = sagafield(default=["change_from_baseline","GEE","MCID"])
    period: dict                 = sagafield(default={"from":None,"to":None})
    enclave_proof: str           = sagafield(default="")   # TEE/zk attestation
    provenance: dict             = sagafield(default={})   # transforms, model versions, validators

    @sagamethod()
    def attest(self, proof_hash: str, metrics: dict | None = None, provenance: dict | None = None):
        self.enclave_proof = proof_hash
        if metrics: self.metrics.update(metrics)
        if provenance: self.provenance.update(provenance)


# ----------------------------
# 3) Value-Based Reimbursement (ISO 20022)
# ----------------------------
@sagaclass()
class ClassDigitalTherapeuticReimbursement(SPClassObject):
    """Outcome-contingent reimbursement for DTx (ISO 20022-aligned)."""
    reimbursement_id: str        = sagafield()
    payer: str                   = sagafield()
    payee: str                   = sagafield()   # manufacturer / provider
    dt_ref: str                  = sagafield()   # OID(ClassDigitalTherapeutic)
    outcome_ref: str             = sagafield()   # OID(ClassDTxOutcomeMeasure)
    coverage_policy: dict        = sagafield(default={"metric":"HbA1c_delta","threshold":-0.5})
    escrow_conditions: dict      = sagafield(default={"outcome_verified": True})
    amount: float                = sagafield()
    currency: str                = sagafield(length=3)
    status: str                  = sagafield(enum={"pending","escrow","released","denied"}, default="pending")

    @sagamethod()
    def evaluate_and_settle(self, verified: bool, measured_value: float | None = None):
        metric = self.coverage_policy.get("metric")
        threshold = self.coverage_policy.get("threshold")
        meets = verified and (measured_value is not None) and \
                ((measured_value <= threshold) if metric.endswith("_delta") else (measured_value >= threshold))

        if meets:
            self.status = "released"
        else:
            self.status = "escrow"


# ----------------------------
# 4) Public Transparency (SagaFeeds)
# ----------------------------
@sagaclass()
class ClassDTxFeedsIndex(SPClassObject):
    """Public, no-cost index of DTx approvals, coverage eligibility, and benchmarks."""
    index_id: str                = sagafield()
    listed_dt_ids: list          = sagafield(default=[])
    approvals: dict              = sagafield(default={})   # {dt_id: [approval_oids]}
    coverage: dict               = sagafield(default={})   # {payer: {"policy":..., "eligible_dt":[...]}}
    outcome_benchmarks: dict     = sagafield(default={})   # {indication: {"metric":"...", "p50":..., "p90":...}}
    updated_at: str              = sagafield()

    @sagamethod()
    def upsert_dt(self, dt_oid: str, approval_oids: list):
        if dt_oid not in self.listed_dt_ids: self.listed_dt_ids.append(dt_oid)
        self.approvals[dt_oid] = approval_oids

    @sagamethod()
    def upsert_coverage(self, payer: str, policy: dict, eligible: list):
        self.coverage[payer] = {"policy": policy, "eligible_dt": eligible}

    @sagamethod()
    def upsert_benchmarks(self, indication: str, metric: str, p50: float, p90: float):
        self.outcome_benchmarks[indication] = {"metric": metric, "p50": p50, "p90": p90}


# ----------------------------
# 5) Orchestration Glue
# ----------------------------
@sagaclass()
class ClassDTxOrchestrator(SPClassObject):
    """End-to-end flow: approvals → enclave outcomes → reimbursement → feeds."""
```

```
    run_id: str              = sagafield()
    log: list                = sagafield(default=[])

    @sagamethod()
    def append(self, msg: str): self.log.append(msg)

    @sagamethod()
    def approve_verify_and_pay(self,
                               dtx: ClassDigitalTherapeutic,
                               approvals: list,                  #
List[ClassDTxApprovalRef]
                               outcome:
ClassDTxOutcomeMeasure,
                               claim:
ClassDigitalTherapeuticReimbursement,
                               feeds: ClassDTxFeedsIndex) ->
dict:
        # Attach approvals
        for a in approvals: dtx.add_approval(a)

        # Verify enclave proof and evaluate coverage
        verified = bool(outcome.enclave_proof)
        metric = claim.coverage_policy.get("metric")
        measured = outcome.metrics.get(metric, None)
        claim.evaluate_and_settle(verified=verified,
measured_value=measured)

        # Update public feeds (non-sensitive)
        feeds.upsert_dt(dtx.oid, dtx.regulatory_refs)
        feeds.upsert_coverage(claim.payer, {"metric":
metric,                    "threshold":
claim.coverage_policy.get("threshold")}, [dtx.oid])
        feeds.upsert_benchmarks(dtx.indication, metric,
p50=outcome.metrics.get("p50",                0.0),
p90=outcome.metrics.get("p90", 0.0))

        self.append(f"DTx                {dtx.name}:
claim={claim.status},                metric={metric},
value={measured}")
        return {"claim_status": claim.status, "metric":
metric, "value": measured, "log": self.log}
```

Interoperability Analysis

- **FDA / EMA / WHO:** `ClassDTxApprovalRef` normalizes multi-agency approvals; `ClassDigitalTherapeutic` carries a single persistent map of global authorizations.
- **HL7 FHIR / CDISC:** `ClassDTxOutcomeMeasure` ingests HL7 resources and CDISC datasets inside enclaves; only **attested** summaries exit.
- **USP / SPL / ISO IDMP:** Companion/adjunct apps bind to IDMP products and SPL sections; USP/CPIC dosing/adherence standards are referenced for clinical decision support.
- **ISO 20022:** `ClassDigitalTherapeuticReimbursement` supports outcome-contingent, escrowed settlements and fraud-resistant disbursement.
- **Private Enclaves:** Behavioral/PHI and model internals remain encrypted; auditors/regulators receive proofs + lineage, not raw data.
- **SagaFeeds:** `ClassDTxFeedsIndex` publishes non-sensitive approvals, coverage eligibility, and efficacy benchmarks at zero cost.

Implications

- **Regulators:** Persistent provenance and enclave attestations build trust; fewer duplicative submissions.
- **Manufacturers/DTx Developers:** Faster, globally harmonized approvals; programmable reimbursement aligned to outcomes.
- **Payers:** Pay only for verified benefit; reduced fraud and admin overhead.
- **Providers:** SMART-on-FHIR-ready apps integrate cleanly into EHR workflows.
- **Patients:** Safe, private, outcome-backed digital care with clear coverage.
- **Global Health:** IDMP-anchored approvals enable equitable, cross-border access.

Scenario 21 delivers a complete, code-backed path to **trusted, outcome-driven** global interoperability for DTx and

companion apps privacy-preserving by default, standards-aligned by design, and reimbursement-ready out of the box.

Scenario 22: Supply Chain Finance & Risk Sharing for Pharma Logistics

(integrating GS1 EPCIS, DSCSA, ISO 20022, and ESG metrics)

Background & Problem Statement

Pharma logistics moves sensitive, high-value goods across complex networks. Pain points:

- **Capital constraints:** Long settlement cycles → working-capital strain.
- **Risk concentration:** Small carriers/wholesalers bear outsized exposure to excursions, theft, counterfeit.
- **Siloed visibility:** EPCIS and DSCSA proofs sit in proprietary silos, unusable by lenders/insurers.
- **Unlinked compliance & finance:** DSCSA verification doesn't drive payment conditions.
- **Sustainability blind spots:** ESG data isn't priced into finance.

Technical Approach (SagaChain with Private Enclaves)

1. **Serialized provenance for finance:** Shipments bind **GS1 EPCIS** events, **DSCSA** units/verification, **customs**, and **cold-chain** proofs.
2. **Risk-sharing pools & claims:** Multi-party pools with programmatic, evidence-linked payouts.
3. **ISO 20022 settlements:** Early pay, factoring, escrows, and dynamic rates tied to serialization, telemetry, and customs outcomes.
4. **ESG integration:** Carbon intensity and cold-chain performance adjust financing.
5. **Private Enclaves:** Contracts, pricing, and PHI remain encrypted; only proofs/KPIs exit.
6. **SagaFeeds:** Publish anonymized, non-sensitive benchmarks for market transparency.

Sample SagaPython Code

```
# ----------------------------
# Provenance: Shipment, EPCIS, DSCSA, Customs, Telemetry, ESG
# ----------------------------
@sagaclass()
class ClassShipment(SPClassObject):
    """Serialized pharma shipment with full provenance and status."""
    shipment_id: str          = sagafield()
    sscc: str                 = sagafield()          # GS1 SSCC
    sgtins: list              = sagafield(default=[])  # List of GS1 SGTINs
    epcis_events: list        = sagafield(default=[])  # OIDs(ClassEPCISEvent)
    cold_chain_refs: list     = sagafield(default=[])  # OIDs(ClassColdChainTelemetry)
    customs_refs: list        = sagafield(default=[])  # OIDs(ClassCustomsDeclaration)
    dscsa_verifications: list = sagafield(default=[])  # OIDs(ClassDSCSAVerification)
    esg_refs: list            = sagafield(default=[])  # OIDs(Carbon/ESG attestations)
    current_owner_gln: str    = sagafield()
    status: str               = sagafield(enum={"planned","in_transit","delivered","held","exception"},
                                          default="planned")

    @sagamethod()
    def advance_status(self, new_status: str):
        assert new_status in {"planned","in_transit","delivered","held","exception"}
        self.status = new_status

@sagaclass()
class ClassCustomsDeclaration(SPClassObject):
    """Customs declaration/inspection bound to serialized cargo."""
```

```
    decl_id: str                = sagafield()
    sscc: str                   = sagafield()
    hs_code: str                = sagafield()
    exporter_gln: str           = sagafield()
    importer_gln: str           = sagafield()
    port_code: str              = sagafield()
    declaration_time: str       = sagafield()
    outcome:    str                                         =
sagafield(enum={"accepted","inspected","rejected"},
default="accepted")
    documents_hashes: list      = sagafield(default=[])

@sagaclass()
class ClassDSCSAVerification(SPClassObject):
    """DSCSA verification record for SGTIN+lot
combos."""
    verification_id: str        = sagafield()
    sgtin: str                  = sagafield()
    lot: str                    = sagafield()
    request_time: str           = sagafield()
    response_time: str          = sagafield()
    result:    str                                          =
sagafield(enum={"pass","fail","partial"},
default="pass")
    signer: str                 = sagafield()           # GLN or
DID

# (Assumes ClassColdChainTelemetry and
ClassCarbonLedger exist in ESG subtree)

# ----------------------------
# Risk Pooling & Claims
# ----------------------------
@sagaclass()
class ClassRiskPool(SPClassObject):
    """Risk-sharing pool across manufacturers, carriers,
wholesalers, insurers."""
    pool_id: str                = sagafield()
    participants: list          = sagafield(default=[])     #
GLNs / DIDs
    coverage_terms: dict        = sagafield(default={})
# coverage triggers, limits, deductibles
    claims: list                = sagafield(default=[])     #
OIDs(ClassRiskClaim)
    balance: float              = sagafield(default=0.0)

    @sagamethod()
    def contribute(self, participant: str, amount: float):
        assert amount > 0
        if participant not in self.participants:
            self.participants.append(participant)
        self.balance += amount

    @sagamethod()
    def register_claim(self, claim_oid: str):
        if claim_oid not in self.claims:
            self.claims.append(claim_oid)

@sagaclass()
class ClassRiskClaim(SPClassObject):
    """Evidence-linked claim: excursions, theft, loss,
counterfeit, customs rejection."""
    claim_id: str               = sagafield()
    pool_ref: str               = sagafield()               #
OID(ClassRiskPool)
    shipment_ref: str           = sagafield()               #
OID(ClassShipment)
    trigger:    str                                         =
sagafield(enum={"temp_excursion","counterfeit","th
eft","damage","customs_reject"})
    evidence_refs: list         = sagafield(default=[])     #
EPCIS, telemetry, customs, DSCSA
    claimed_amount: float       = sagafield()
    approved_amount: float      = sagafield(default=0.0)
    status:    str                                          =
sagafield(enum={"open","approved","denied","paid"
}, default="open")

    @sagamethod()
    def adjudicate(self, approve: bool, amount: float =
0.0):
        if approve:
            self.approved_amount = amount
            self.status = "approved"
        else:
            self.approved_amount = 0.0
            self.status = "denied"

# ----------------------------
# ISO 20022 Supply-Chain Settlement (Finance)
# ----------------------------
@sagaclass()
class
ClassISO20022SupplyChainSettlement(SPClassObje
ct):
    """Programmable settlement: early pay, factoring,
escrows, risk-/ESG-adjusted rates."""
    settlement_id: str          = sagafield()
    payer: str                  = sagafield()
    payee: str                  = sagafield()
    shipment_ref: str           = sagafield()               #
OID(ClassShipment)
    risk_pool_ref:     str     |     None                    =
sagafield(optional=True)   # OID(ClassRiskPool)
    esg_refs: list              = sagafield(default=[])     #
Carbon/telemetry attestations
    escrow_conditions:     dict                             =
sagafield(default={"serialization_verified": True,

"temp_excursions_max_min": 10,

"customs_outcome": "accepted"})
```

```
    base_amount: float          = sagafield()
    currency: str               = sagafield(length=3)
    adjustments: dict                                   =
sagafield(default={"risk_bps": 0.0, "esg_bps": 0.0})
    final_amount: float         = sagafield(default=0.0)
    status: str                                         =
sagafield(enum={"pending","escrow","released","den
ied"}, default="pending")

    @sagamethod()
    def evaluate(self, serialization_ok: bool,
excursion_minutes: int, customs_outcome: str,
                 risk_bps: float = 0.0, esg_bps: float = 0.0):
        """Check conditions and compute final amount
with bps adjustments."""
        cond = self.escrow_conditions
        ok = serialization_ok \
            and excursion_minutes               <=
cond.get("temp_excursions_max_min", 999999) \
            and customs_outcome                 ==
cond.get("customs_outcome", "accepted")

        self.adjustments["risk_bps"] = risk_bps
        self.adjustments["esg_bps"]  = esg_bps

        # bps = basis points → amount * (1 + (bps_total
/ 10_000))
        bps_total = risk_bps + esg_bps
        self.final_amount = round(self.base_amount * (1
+ bps_total / 10000.0), 2)

        self.status = "released" if ok else "escrow"

# ----------------------------
# Oracles / KPIs (Enclave-backed summaries)
# ----------------------------
@sagaclass()
class ClassLogisticsKPIOracle(SPClassObject):
    """Enclave-computed KPIs for lenders/insurers (no
raw commercial data leaves)."""
    oracle_id: str              = sagafield()
    shipment_ref: str           = sagafield()
    enclave_proof: str          = sagafield(default="")
    kpis: dict                  = sagafield(default={})      #
{"excursions_min": 2, "route_dev": 0.0, "carbon_kg":
0.48}

    @sagamethod()
    def attest(self, proof_hash: str, kpis: dict):
        self.enclave_proof = proof_hash
        self.kpis.update(kpis)

# ----------------------------
# Public Transparency (SagaFeeds)
# ----------------------------
@sagaclass()
class ClassSupplyFinanceFeeds(SPClassObject):
    """Public, no-cost index of anonymized ESG-
adjusted financing benchmarks."""
    index_id: str               = sagafield()
    carriers: dict              = sagafield(default={})      #
{GLN: {"avg_esg_bps":..., "on_time%":...}}
    products: dict              = sagafield(default={})      #
{GTIN/IDMP:                         {"avg_excursions":...,
"avg_carbon":...}}
    updated_at: str             = sagafield()

    @sagamethod()
    def upsert_carrier(self, gln: str, stats: dict):
        self.carriers[gln] = stats

    @sagamethod()
    def upsert_product(self, code: str, stats: dict):
        self.products[code] = stats

# ----------------------------
# End-to-End Controller
# ----------------------------
@sagaclass()
class
ClassSupplyFinanceOrchestrator(SPClassObject):
    """Wires shipment provenance → KPIs → risk
claim → ISO 20022 settlement → public index."""
    run_id: str                 = sagafield()
    log: list                   = sagafield(default=[])

    @sagamethod()
    def append(self, msg: str): self.log.append(msg)

    @sagamethod()
    def process_settlement(self,
                           shipment: ClassShipment,
                           kpi_oracle:
ClassLogisticsKPIOracle,
                           settlement:
ClassISO20022SupplyChainSettlement,
                           risk_pool: ClassRiskPool | None,
                           feeds: ClassSupplyFinanceFeeds) ->
dict:
        # Extract KPIs
        k = kpi_oracle.kpis
        serialization_ok = all(v.result == "pass" for v in
[get_object(oid)            for            oid            in
shipment.dscsa_verifications])                          if
shipment.dscsa_verifications else True
        excursions_min  =  int(k.get("excursions_min",
0))
        customs_outcome = "accepted"
        if shipment.customs_refs:
            last_cust                                   =
get_object(shipment.customs_refs[-1])
            customs_outcome = last_cust.outcome
```

```
        # ESG / Risk basis points (example policy)
        risk_bps = -15 if shipment.status == "delivered"
and excursions_min == 0 else 0
        esg_bps  = -10 if k.get("carbon_kg", 1e9) <= 0.6
else 0

        settlement.evaluate(serialization_ok,
excursions_min,        customs_outcome,        risk_bps,
esg_bps)
        self.append(f"Settlement
{settlement.settlement_id} → {settlement.status} @
{settlement.final_amount} {settlement.currency}")

        # Auto-claim on exception
        if shipment.status in {"held","exception"} and
risk_pool:
            claim = ClassRiskClaim(
                claim_id=f"CLM-
{shipment.shipment_id}",
                pool_ref=risk_pool.oid,
                shipment_ref=shipment.oid,
                trigger="temp_excursion"                if
excursions_min > 0 else "damage",
                evidence_refs=shipment.cold_chain_refs +
shipment.epcis_events + shipment.customs_refs +
shipment.dscsa_verifications,
                claimed_amount=max(0.0,
settlement.base_amount * 0.3)
            )
            risk_pool.register_claim(claim.oid)
            self.append(f"Claim          {claim.claim_id}
registered for pool {risk_pool.pool_id}")

        # Publish anonymized benchmarks

feeds.upsert_carrier(shipment.current_owner_gln,
{"avg_esg_bps": esg_bps, "on_time%": 100.0 if
shipment.status=="delivered" else 0.0})
        # In practice, map product code from first
SGTIN's GTIN/IDMP
        if shipment.sgtins:

feeds.upsert_product("GTIN:"+shipment.sgtins[0][:1
4],          {"avg_excursions":          excursions_min,
"avg_carbon": k.get("carbon_kg", None)})

        return          {"status":          settlement.status,
"final_amount":    settlement.final_amount,    "log":
self.log}
```

Notes:

• get_object(oid) above represents a kernel helper available in SagaOS to dereference OIDs (pseudo-code for clarity).

• This snippet assumes ClassColdChainTelemetry and ClassCarbonLedger are already defined in your ESG subtree (as in Scenario 13).

Interoperability Analysis

- **GS1 EPCIS:** Financing validates shipment lineage (commission/ship/receive/agg) with immutable EPCIS references.
- **DSCSA:** ClassDSCSAVerification outcomes gate payment; fails trigger **escrow** or pool **claims**.
- **Customs/Ports:** Declarations/inspections become programmable **conditions** for settlement.
- **ISO 20022:** ClassISO20022SupplyChainSettlement handles early pay, factoring, and escrow with **bps** adjustments from risk/ESG signals.
- **ESG (Carbon & Cold-Chain):** ClassCarbonLedger & ClassColdChainTelemetry drive rate incentives/penalties.
- **Risk Pools:** Evidence-linked ClassRiskClaim reduces fraud and spreads risk.
- **Private Enclaves:** Contracts/prices are processed privately; only **attestations/KPIs** (oracle) exit.
- **SagaFeeds:** ClassSupplyFinanceFeeds publishes anonymized benchmarks to improve market efficiency and transparency.

Implications

- **Manufacturers:** Compliance-based early payments improve cash flow; fewer disputes.
- **Carriers/3PLs:** Risk pooling + ESG incentives → fairer pricing and resilience.

- **Insurers:** Claim adjudication tied to cryptographic provenance lowers loss adjustment expense.
- **Banks/Factors:** Finance-grade proofs reduce diligence friction and fraud exposure.
- **Payers/Donors:** Financing aligned to sustainability and compliance delivers measurable impact.
- **Global Health:** More robust, transparent, and equitable medicine distribution.

Scenario 22 provides a full, code-backed blueprint to make pharma logistics **finance-ready, risk-shared, ESG-aligned, and verifiably compliant** with privacy preserved and transparency where it matters.

Scenario 23: Integrated Global Labeling & Multilingual Patient Access

(linking SPL, GS1 Digital Link, HL7 FHIR, and WHO standards)

Background & Problem Statement

Regulatory product information is the backbone of safe use, yet today:

- **Multilingual gaps:** Many patients can't read local leaflets.
- **Static updates:** Label changes propagate slowly across markets.
- **Disconnected:** SPL isn't consistently linked to serialization (GS1/DSCSA) or EHRs (HL7).
- **Equity barriers:** LMICs lack reliable localized content.
- **Limited digital reach:** Scans don't always resolve to current, verified labels.

Technical Approach (SagaChain with Private Enclaves)

1. **Global SPL objects:** Persistent, versioned SPL bound to **ISO IDMP** product IDs and **GS1** (SGTIN/GTIN).
2. **Multilingual leaflets:** Enclave-protected translation pipelines output localized, structured content with cryptographic attestation.
3. **GS1 Digital Link:** Scanning a package's QR resolves to the current SPL + leaflet for that unit/market/version.
4. **WHO & HL7 sync:** WHO/national authorities receive proofs; EHRs (FHIR Medication/MedicationKnowledge) auto-fetch updates.
5. **Equity & transparency:** Non-sensitive labeling metadata published via **SagaFeeds** for free public access.
6. **Governance:** Full audit of editorial/translation changes, market approvals, and deprecations.

Sample SagaPython Code

```
# ----------------------------
# Core SPL & Leaflet Objects
# ----------------------------
@sagaclass()
class ClassSPLDocument(SPClassObject):
    """Structured Product Labeling bound to ISO IDMP / regulators with versioning."""
    spl_id: str              = sagafield()
    product_id: str          = sagafield()   # ISO IDMP identifier
    ndc_or_udi: str | None   = sagafield(optional=True)
    regulator: str           = sagafield()   # FDA, EMA, WHO, etc.
    version: str             = sagafield()
    effective_date: str      = sagafield()
    sections: dict           = sagafield(default={})  # {"dosage": "...", "contraindications": "..."}
    supersedes: str | None   = sagafield(optional=True)
```

```
    superseded_by: str | None = sagafield(optional=True)
    provenance_hash: str = sagafield()  # content-addressed integrity hash

    @sagamethod()
    def deprecate(self, new_spl_oid: str):
        self.superseded_by = new_spl_oid

@sagaclass()
class ClassMultilingualLeaflet(SPClassObject):
    """Localized patient leaflet linked to SPL; enclave-backed translation attestation."""
    leaflet_id: str = sagafield()
    spl_ref: str = sagafield()  # OID(ClassSPLDocument)
    language: str = sagafield()  # BCP-47, e.g., "es-ES", "fr-FR"
    market: str = sagafield()  # ISO country/region code
    text: dict = sagafield(default={})  # structured sections (title, dosage, warnings, pictograms...)
    translator_proof: str = sagafield()  # TEE/zk attestation
    reading_grade: str | None = sagafield(optional=True)  # health literacy grade level
    accessibility_alts: dict = sagafield(default={})  # alt text, large print, audio URIs
    last_reviewed: str = sagafield()

# ----------------------------
# GS1 Digital Link Resolution & Unit Binding
# ----------------------------
@sagaclass()
class ClassDigitalLinkResolution(SPClassObject):
    """Maps GS1 Digital Link URIs to current SPL/leaflet by language/market/version."""
    resolution_id: str = sagafield()
    canonical_uri: str = sagafield()  # e.g., https://id.gs1.org/01/<GTIN>/21/<serial>
    gtin: str = sagafield()
    sgtin: str | None = sagafield(optional=True)
    product_id: str = sagafield()  # ISO IDMP
    default_lang: str = sagafield(default="en")
    targets: dict = sagafield(default={})  # {"spl_oid": "...", "leaflets": {"en":"...","es":"..."}}
    cache_ttl_seconds: int = sagafield(default=3600)

    @sagamethod()
    def upsert_leaflet(self, lang: str, leaflet_oid: str):
        leaflets = self.targets.get("leaflets", {})
        leaflets[lang] = leaflet_oid
        self.targets["leaflets"] = leaflets

@sagaclass()
class ClassUnitLabelBinding(SPClassObject):
    """Binds a serialized unit (SGTIN) to the exact SPL version & locale mapping used at dispense."""
    binding_id: str = sagafield()
    sgtin: str = sagafield()
    spl_oid: str = sagafield()
    leaflet_oid: str = sagafield()
    dispensed_at: str = sagafield()
    dispenser_gln: str = sagafield()
    patient_lang_pref: str = sagafield()

# ----------------------------
# HL7 / WHO Synchronization & Hooks
# ----------------------------
@sagaclass()
class ClassHL7LabelHook(SPClassObject):
    """FHIR integration hook so EHR/portals fetch current labeling automatically."""
    hook_id: str = sagafield()
    product_id: str = sagafield()  # ISO IDMP
    fhir_endpoints: list = sagafield(default=[])  # subscribed EHR endpoints
    last_push_time: str | None = sagafield(optional=True)

    @sagamethod()
    def push_update(self, spl_oid: str, languages: list[str]):
        # Emits FHIR MedicationKnowledge / DocumentReference notifications (out-of-band)
        self.last_push_time = now_iso8601()

@sagaclass()
class ClassWHOLabelSync(SPClassObject):
    """WHO/national registry synchronization for global labeling alignment."""
    sync_id: str = sagafield()
    regulators: list = sagafield(default=["WHO"])
    product_id: str = sagafield()
    last_synced: str | None = sagafield(optional=True)
    approvals: dict = sagafield(default={})  # {"WHO": {"status":"accepted","date":"..."}, ...}

    @sagamethod()
    def record_approval(self, agency: str, status: str, date: str):
        self.approvals[agency] = {"status": status, "date": date}
        self.last_synced = date

# ----------------------------
# Governance, Editorial Audit & Events
# ----------------------------
```

```
@sagaclass()
class ClassLabelingUpdateEvent(SPClassObject):
    """Immutable audit for SPL/leaflet changes (editorial, safety, regulator-mandated)."""
    event_id: str               = sagafield()
    product_id: str             = sagafield()
    spl_from: str | None        = sagafield(optional=True)
    spl_to: str                 = sagafield()
    reason: str                 = sagafield(enum={"safety","efficacy","admin","localization","regulator"})
    change_summary: dict        = sagafield(default={})
    editor_signatures: list     = sagafield(default=[]) # Part 11 compliant signatures
    regulator_refs: list        = sagafield(default=[]) # docket/decision IDs
    enclave_diff_proof: str     = sagafield()           # diff attestation (TEE/zk)

@sagaclass()
class ClassLeafletRequestLog(SPClassObject):
    """Privacy-preserving access log for scans/requests; PHI remains enclave-bound."""
    log_id: str                 = sagafield()
    sgtin: str | None           = sagafield(optional=True)
    lang_requested: str         = sagafield()
    market: str                 = sagafield()
    served_oid: str             = sagafield()       # leaflet OID
    served_at: str              = sagafield()
    enclave_k_anonymity: int    = sagafield(default=25)

# ----------------------------
# Equivalence / Localization Mapping
# ----------------------------
@sagaclass()
class ClassLabelEquivalenceMap(SPClassObject):
    """Maps section-level equivalence across languages/markets for proofed parity."""
    map_id: str                 = sagafield()
    spl_oid: str                = sagafield()
    pairs: list                 = sagafield(default=[]) # [{"lang":"en","key":"dosage","hash":"..."},{"lang":"es","key":"posologia","hash":"..."}]
    enclave_alignment_proof: str = sagafield()

# ----------------------------
# Public Transparency (SagaFeeds)
# ----------------------------
@sagaclass()
class ClassLabelingFeeds(SPClassObject):
    """Public, no-cost multilingual labeling index for equity and trust."""
    feed_id: str                = sagafield()
    products: dict              = sagafield(default={}) # {IDMP: {"langs":["en","es","fr"],"current_spl":"...","updated":"..."}}
    markets: dict               = sagafield(default={}) # {ISO3166: {"coverage%": 99.1, "last_sync":"..."}}
    updated_at: str             = sagafield()

    @sagamethod()
    def upsert_product(self, product_id: str, langs: list[str], current_spl: str, updated: str):
        self.products[product_id] = {"langs": sorted(set(langs)), "current_spl": current_spl, "updated": updated}

    @sagamethod()
    def upsert_market(self, market: str, coverage_pct: float, last_sync: str):
        self.markets[market] = {"coverage%": round(coverage_pct, 2), "last_sync": last_sync}

# ----------------------------
# Orchestrator: Scan → Resolve → Serve → Audit → Sync
# ----------------------------
@sagaclass()
class ClassLabelingOrchestrator(SPClassObject):
    """End-to-end flow: GS1 scan ⇒ SPL/leaflet resolution ⇒ HL7/WHO sync ⇒ public feed."""
    run_id: str                 = sagafield()
    activity_log: list          = sagafield(default=[])

    @sagamethod()
    def append(self, msg: str): self.activity_log.append(msg)

    @sagamethod()
    def serve_leaflet(self,
                      resolution: ClassDigitalLinkResolution,
                      preferred_lang: str,
                      market: str,
                      feeds: ClassLabelingFeeds) -> dict:
        leaflets = resolution.targets.get("leaflets", {})
        lang = preferred_lang if preferred_lang in leaflets else resolution.default_lang
        served_oid = leaflets.get(lang) or next(iter(leaflets.values()))
        self.append(f"Served leaflet {served_oid} for {resolution.canonical_uri} lang={lang} market={market}")

        # Log request (k-anonymized; raw PHI stays enclave-side)
        req = ClassLeafletRequestLog(
            log_id=f"LR-{resolution.resolution_id}-{now_unix()}",
            sgtin=resolution.sgtin,
```

```
            lang_requested=lang,
            market=market,
            served_oid=served_oid,
            served_at=now_iso8601()
        )

        # Update public feed for transparency
        feeds.upsert_product(
            product_id=resolution.product_id,
            langs=list(leaflets.keys()),

current_spl=resolution.targets.get("spl_oid",""),
            updated=now_iso8601()
        )
        return {"leaflet_oid": served_oid, "language":
lang, "log": self.activity_log}
```

Implementation notes:
• Translation, reading-grade scoring, and equivalence mapping execute inside TEEs; only **translator_proof** and **enclave_diff_proof** exit.
• EHR push uses standards (FHIR MedicationKnowledge, DocumentReference) via ClassHL7LabelHook.push_update.
• GS1 Digital Link resolution caches per cache_ttl_seconds but always honors most-recent SPL.

Interoperability Analysis

- **SPL (FDA/EMA/WHO):** Versioned SPL with immutable provenance and regulator references.
- **ISO IDMP:** Global product identity anchors all leaflets/labels across markets.
- **GS1 Digital Link:** Unit-level scan resolves to the **current** approved SPL and best-fit language leaflet.
- **HL7 FHIR:** EHRs and portals auto-fetch updates; bedside decision support stays in sync.
- **WHO & National Authorities:** ClassWHOLabelSync records approvals and provides auditable alignment in LMICs.
- **Private Enclaves:** Protect translation corpora, patient preferences, and editorial histories; only attestations exit.
- **SagaFeeds:** Public, no-cost access to multilingual coverage and current versions improves equity and trust.

Implications

- **Patients:** Instant access to up-to-date, native-language labeling (with accessibility options).
- **Providers:** Reduced medication errors via synchronized EHR content.
- **Regulators:** Auditable, cross-border labeling harmonization and rapid safety updates.
- **Manufacturers:** Single source of truth for labels across markets; simpler updates and audits.
- **Payers:** Better adherence → fewer adverse events and lower total cost of care.
- **Global Health:** LMICs get validated multilingual content with verifiable proofs.

Scenario 23 delivers global, multilingual, verifiable labeling resolving from a scan to the current SPL + leaflet, syncing with EHRs and WHO, preserving privacy, and exposing equitable access via public feeds.

Scenario 24: AI-Augmented Drug Repurposing & Adaptive Regulatory Pathways

(leveraging NIH, FDAAA 801, HL7, ISO IDMP, and WHO trial data)

Background & Problem Statement

Repurposing can deliver faster, cheaper therapies, but today:

- **Evidence fragmentation:** NIH preclinical, FDAAA 801/CDISC clinical, HL7 RWE/claims live in silos.
- **Manual discovery:** Literature-heavy workflows delay candidate identification.
- **Regulatory friction:** New indications often require near-full resubmission.
- **Payment misalignment:** No programmable, outcomes-based reimbursement for repurposed uses.

Technical Approach (SagaChain with Private Enclaves)

1. **Evidence graph:** Preclinical → clinical → RWE objects persistently linked by **ISO IDMP** product identifiers.
2. **AI monitors (enclave):** NLP/causal/graph models scan the graph to surface candidates with **attested** proofs.
3. **Adaptive pathways:** Proposals bind to FDA/EMA/WHO routes; label/SPL updates are auto-linked; pharmacopoeial checks enforced.
4. **Programmable finance:** Outcomes-based ISO 20022 settlements activate on real-world validation.
5. **Public transparency:** Non-sensitive indices of candidates and status published via **SagaFeeds**.

Sample SagaPython Code

```
# ----------------------------
# Evidence Graph (references existing classes from earlier sections)
#   - ClassPreclinicalStudy (lifecycle)
#   - ClassClinicalTrial (trial)
#   - ClassRWEResult (analytics)
# ----------------------------

@sagaclass()
class ClassRepurposingMonitor(SPClassObject):
    """Enclave AI monitor scanning evidence graph for repurposing opportunities."""
    monitor_id: str              = sagafield()
    scope: dict                  = sagafield(default={})  # {"products":["IDMP:..."], "ther_areas":["oncology","cvd"]}
    methods: list                = sagafield(default=["NLP_lit","causal_RWE","signal_mining","graph_reasoning"])
    data_refs: list              = sagafield(default=[])  # OIDs: Preclinical, ClinicalTrial, RWEResult, SPL, safety signals
    run_interval: str            = sagafield(default="P14D")
    last_run: str | None         = sagafield(optional=True)
    enclave_attestation: str | None = sagafield(optional=True) # TEE/zk proof of execution
    model_card_ref: str | None   = sagafield(optional=True)  # governance/model-risk pointer

    @sagamethod()
    def record_attestation(self, att: str, run_time_iso: str):
        self.enclave_attestation = att
        self.last_run = run_time_iso

@sagaclass()
class ClassRepurposingHypothesis(SPClassObject):
    """AI-generated hypothesis: existing product → new indication."""
    hypothesis_id: str           = sagafield()
    product_id: str              = sagafield()          # ISO IDMP
    new_indication: str          = sagafield()          # e.g., "HFpEF", "Migraine Prevention"
    rationale: dict              = sagafield(default={})  # top features, literature refs, target pathways
    evidence_refs: list          = sagafield(default=[])  # OIDs: Preclinical, Trial, RWE summaries
    risk_flags: list             = sagafield(default=[])  # e.g., "QTc", "hepatic"
    ai_proof: str                = sagafield()          # enclave proof (reproducible run)
    status: str                  = sagafield(default="screening")  # screening, triage, promoted, rejected

    @sagamethod()
    def promote(self):
```

```
        self.status = "promoted"

    @sagamethod()
    def reject(self, reason: str):
        self.status = f"rejected:{reason}"

@sagaclass()
class ClassRepurposingProposal(SPClassObject):
    """Formal proposal to pursue a new indication through adaptive pathways."""
    proposal_id: str              = sagafield()
    hypothesis_ref: str           = sagafield()          # OID(ClassRepurposingHypothesis)
    product_id: str               = sagafield()          # ISO IDMP
    new_indication: str           = sagafield()
    evidence_refs: list           = sagafield(default=[])  # preclinical/clinical/RWE
    regulator_paths: dict         = sagafield(default={})  # {"FDA":"505(b)(2)","EMA":"Type II","WHO":"PQ-fasttrack"}
    bridge_study_plan: dict       = sagafield(default={})  # minimal evidence add-ons (PK/PD, safety, subgroups)
    labeling_impact: dict         = sagafield(default={})  # SPL sections to update
    ai_proof: str                 = sagafield()
    status: str                   = sagafield(default="draft")  # draft, submitted, in_review, approved, denied

    @sagamethod()
    def submit(self, agency: str, dossier_ref: str):
        self.regulator_paths[agency]                = self.regulator_paths.get(agency, "") or "adaptive"
        self.status = "submitted"

    @sagamethod()
    def set_status(self, new_status: str):
        self.status = new_status

# ----------------------------
# Labeling & Standards Linkage
# ----------------------------

@sagaclass()
class ClassRepurposingLabelLink(SPClassObject):
    """Binds approved proposal to concrete SPL updates and pharmacopeial checks."""
    link_id: str                  = sagafield()
    proposal_ref: str             = sagafield()          # OID(ClassRepurposingProposal)
    spl_before: str               = sagafield()          # OID(ClassSPLDocument)
    spl_after: str | None         = sagafield(optional=True)
    usp_refs: list                = sagafield(default=[])  # compendial/IDMP cross-checks
    editorial_diff_hash: str | None = sagafield(optional=True)
    enclave_diff_proof: str | None  = sagafield(optional=True)

    @sagamethod()
    def finalize_label(self, new_spl_oid: str, diff_hash: str, proof: str):
        self.spl_after = new_spl_oid
        self.editorial_diff_hash = diff_hash
        self.enclave_diff_proof = proof

# ----------------------------
# Outcomes & Programmable Finance (ISO 20022)
# ----------------------------

@sagaclass()
class ClassRepurposingOutcomePact(SPClassObject):
    """Defines real-world outcomes required for coverage/payment of repurposed use."""
    pact_id: str                  = sagafield()
    product_id: str               = sagafield()
    indication: str               = sagafield()
    cohort_def: dict              = sagafield(default={})  # computable phenotype
    outcomes: dict                = sagafield(default={})  # {"RR_hosp": "<=0.85", "AE_rate":"<=3%"}
    observation_window: str       = sagafield(default="P180D")
    rwe_monitors: list            = sagafield(default=[])  # OIDs(ClassRWEMonitor)
    attestation_req: str          = sagafield(default="TEE/zk")

@sagaclass()
class ClassRepurposingSettlement(SPClassObject):
    """ISO 20022 settlement tied to outcome pacts (value-based reimbursement)."""
    settlement_id: str            = sagafield()
    payer: str                    = sagafield()
    payee: str                    = sagafield()
    currency: str                 = sagafield(default="USD")
    base_amount: float            = sagafield()
    adjustment_formula: str       = sagafield(default="if outcomes_met then +x% else -y%")
    outcomes_met: bool | None     = sagafield(optional=True)
    pact_ref: str                 = sagafield()          # OID(ClassRepurposingOutcomePact)
    rwe_result_refs: list         = sagafield(default=[])  # OIDs(ClassRWEResult)
    escrow_status: str            = sagafield(default="pending")  # pending, released, clawback
```

```
    @sagamethod()
    def adjudicate(self, met: bool):
        self.outcomes_met = met
        self.escrow_status = "released" if met else "clawback"

# ----------------------------
# Governance & Public Transparency
# ----------------------------

@sagaclass()
class ClassRepurposingGovernance(SPClassObject):
    """Model risk management, protocol preregistration, and bias/drift audits."""
    gov_id: str                 = sagafield()
    monitor_ref: str            = sagafield()  # OID(ClassRepurposingMonitor)
    prereg_protocol_ref: str    = sagafield()
    validation_metrics: dict    = sagafield(default={}) # AUROC, calibration, fairness
    last_audit: str | None      = sagafield(optional=True)
    regulatory_read_access: list = sagafield(default=[])

    @sagamethod()
    def record_audit(self, when: str, metrics: dict):
        self.last_audit = when
        self.validation_metrics = metrics

@sagaclass()
class ClassRepurposingFeeds(SPClassObject):
    """Public, no-cost index of repurposing candidates/proposals (non-sensitive)."""
    feed_id: str                = sagafield()
    candidates: dict            = sagafield(default={}) # {IDMP: [{"indication":"...","stage":"..."}]}
    updated_at: str             = sagafield()

    @sagamethod()
    def upsert_candidate(self, product_id: str, indication: str, stage: str):
        arr = self.candidates.get(product_id, [])
        arr = [c for c in arr if c.get("indication") != indication]
        arr.append({"indication": indication, "stage": stage})
        self.candidates[product_id] = arr
        self.updated_at = now_iso8601()
```

Execution model:
• All AI discovery and RWE adjudication run inside **Private Enclaves**; only proofs and summaries exit.
• Label updates produce **diff hashes** and **TEE attestations** to prove fidelity.
• Settlements release via ISO 20022 only when enclave-attested outcomes meet pact criteria.

Interoperability Analysis

- **NIH / Preclinical:** Proprietary lab data persists as enclave-protected ClassPreclinicalStudy with public safety/efficacy summaries.
- **FDAAA 801 / CDISC:** Trials ingested, standardized, and linked to ClassClinicalTrial; proposals cite validated datasets.
- **HL7 FHIR (RWE):** ClassRWEMonitor and ClassRWEResult feed ongoing effectiveness/safety; adjudicates outcome pacts.
- **ISO IDMP:** Uniform product identity across agencies ensures proposals and signals map unambiguously worldwide.
- **SPL / Labeling:** ClassRepurposingLabelLink binds proposals to concrete SPL changes with cryptographic diffs.
- **USP / Standards:** Compendial constraints (potency, purity, dosage) validated as part of repurposed labeling/CMC updates.
- **ISO 20022 Finance:** ClassRepurposingSettlement enforces outcomes-based, escrowed reimbursement.
- **Private Enclaves:** PHI, proprietary datasets, and model IP remain confidential; reproducible attestations enable regulator review.
- **SagaFeeds:** Non-sensitive indices of candidates, statuses, and high-level outcomes foster transparency and collaboration.

Example Lifecycle (End-to-End)

1. **Discover:** ClassRepurposingMonitor runs NLP/causal scans → emits ClassRepurposingHypothesis (attested).
2. **Propose:** Team promotes hypothesis → drafts ClassRepurposingProposal with bridge-study plan and regulator path.
3. **Approve:** Agency accepts adaptive route → ClassRepurposingLabelLink.finalize_label() updates SPL with diff proofs.
4. **Validate in RWE:** ClassRepurposingOutcomePact defines endpoints; ClassRWEResult confirms outcomes; proofs logged.
5. **Pay:** ClassRepurposingSettlement.adjudicate(True) releases escrow; if not met, triggers clawback.
6. **Publish:** ClassRepurposingFeeds.upsert_candidate() updates public status (no PHI).

Implications

- **Regulators:** Faster, auditable approvals with full evidence lineage and enclave proofs.
- **Sponsors:** Unlock value from legacy assets with lower incremental R&D.
- **Payers:** Pay for results, not promises, fraud-resistant, outcomes-based flows.
- **Patients:** Quicker access to safe, effective therapies for new indications.
- **Global Health:** Scalable repurposing for rare/neglected diseases with equitable visibility.

Scenario 24 turns repurposing into a programmable pipeline from AI discovery to adaptive approval, label updates, real-world validation, and outcomes-based reimbursement backed by privacy-preserving attestations and public transparency.

Scenario 25: Global Health Equity & Donor-Funded Access Models

(integrating WHO, GAVI, ISO 20022, DSCSA/GS1, and SagaFeeds transparency)

Background & Problem Statement

Donor-funded programs (GAVI, Global Fund, UNICEF, philanthropies) struggle with:

- **Opaque flows:** Weak traceability from **donor → asset → patient**.
- **Verification gaps:** Diversion/counterfeits despite serialization mandates.
- **Inequitable allocation:** LMIC deliveries miss volume/timing commitments.
- **Misaligned incentives:** ESG/equity compliance not priced into finance.

Technical Approach (SagaChain with Private Enclaves)

1. **Equity-funded assets:** Donor-financed dose entitlements modeled as fungible units, cryptographically bound to **GS1/DSCSA** identities.
2. **Programmable finance:** ISO 20022 escrows release on **verifiable delivery** (serialization + cold-chain + customs).
3. **Treaty alignment:** WHO treaty/commitment objects auto-verify **equity rules** (e.g., % allocated to LMICs, reporting SLAs).
4. **ESG hooks:** Carbon/cold-chain KPIs adjust settlement rates.

5. **Privacy:** PHI and commercial terms run in **Private Enclaves**; only attested proofs exit.
6. **Public trust:** Non-sensitive dashboards published via **SagaFeeds™**.

Sample SagaPython Code

```
# ----------------------------
# Equity-funded asset & delivery graph
# ----------------------------

@sagaclass()
class ClassEquityFundAsset(ClassFungibleAsset):
    """Donor-financed pharma entitlement units with equity & compliance proofs."""
    asset_id: str               = sagafield()
    donor: str                  = sagafield()                 # e.g., "GAVI", "GlobalFund", "ACME Philanthropy"
    product_id: str             = sagafield()                 # ISO IDMP
    program_tag: str            = sagafield(default="")       # campaign/cohort identifier
    target_pop: dict            = sagafield(default={})       # {"jurisdictions":["KE","UG"], "priority":["HCP","<5y"]}

    # Logistics & compliance bindings
    sgtins: list                = sagafield(default=[])       # bound serialized units (may grow over time)
    lmics_allocated: list       = sagafield(default=[])       # ISO2 country codes with allocation quantities
    delivery_proofs: list       = sagafield(default=[])       # OIDs(ClassEquityDeliveryProof)
    escrow_status:  str         = sagafield(default="unfunded")  # unfunded, funded, in_escrow, released, clawback
    treaty_refs: list           = sagafield(default=[])       # OIDs(ClassPandemicTreatyObligation)
    esg_refs: list              = sagafield(default=[])       # OIDs(Carbon/ColdChain/ESG attestations)

    @sagamethod()
    def bind_units(self, sgtin_list: list[str]):
        self.sgtins = sorted(set(self.sgtins + sgtin_list))

    @sagamethod()
    def mark_allocation(self, iso2: str, qty: int):
        self.lmics_allocated.append({"country": iso2, "qty": qty})

@sagaclass()
class ClassEquityDeliveryProof(SPClassObject):
    """Verifiable LMIC delivery proof: serialization, EPCIS, customs, and cold-chain."""
    proof_id: str               = sagafield()
    sscc: str                   = sagafield()                 # pallet/case
    sgtins: list                = sagafield(default=[])
    epcis_refs: list            = sagafield(default=[])       # OIDs(ClassEPCISEvent)
    customs_ref:  str | None    = sagafield(optional=True)    # OID(ClassCustomsDeclaration)
    cold_chain_refs: list       = sagafield(default=[])       # OIDs(ClassColdChainTelemetry)
    delivery_ack_ts: str        = sagafield()                 # ISO-8601
    jurisdiction: str           = sagafield()                 # ISO2 receiving country
    enclave_attestation: str    = sagafield()                 # TEE/zk proof of verification pipeline

    @sagamethod()
    def add_telemetry(self, telemetry_oid: str):
        self.cold_chain_refs.append(telemetry_oid)

# ----------------------------
# Allocation policy & treaty compliance
# ----------------------------

@sagaclass()
class ClassEquityAllocationPolicy(SPClassObject):
    """Programmable equity rules for donor-funded allocations (WHO/coalitions)."""
    policy_id: str              = sagafield()
    min_lmic_share: float       = sagafield(default=0.2)      # e.g., 20%
    max_delivery_sla_days:  int = sagafield(default=14)
    cold_chain_limit:  dict     = sagafield(default={"excursion_minutes_max": 10})
    proof_logic_ref:  str       = sagafield(default="policy/v1")  # versioned logic pack

    @sagamethod()
    def is_satisfied(self, asset_oid: str) -> bool:
        # (Pseudo-eval performed off-chain; this method records last check result)
        return True

@sagaclass()
class ClassPandemicTreatyObligation(SPClassObject):
    """Treaty/compact obligations relevant to equity delivery."""
    obligation_id: str          = sagafield()
    description: str            = sagafield()
```

```
    jurisdiction: str            = sagafield()                  #
"GLOBAL" or ISO2/WHO region
    metrics: dict                = sagafield(default={})        #
{"allocation_LMIC":"20%","reporting_time":"72h"}
    verification_refs: list      = sagafield(default=[])
    status: str                  = sagafield(default="pending")
# compliant, pending, non-compliant

    @sagamethod()
    def set_status(self, new_status: str):
        self.status = new_status

# ----------------------------
# ISO 20022 programmable finance (escrow, release,
clawback)
# ----------------------------

@sagaclass()
class ClassISO20022EquityEscrow(SPClassObject):
    """ISO 20022 escrow for donor funds tied to
verifiable LMIC delivery & ESG."""
    settlement_id: str           = sagafield()
    payer: str                   = sagafield()                  #
donor
    payee: str                   = sagafield()                  #
manufacturer or distributor
    currency: str                = sagafield(default="USD")
    gross_amount: float          = sagafield()
    asset_ref: str               = sagafield()                  #
OID(ClassEquityFundAsset)
    allocation_policy_ref: str   = sagafield()                  #
OID(ClassEquityAllocationPolicy)
    delivery_proof_refs: list    = sagafield(default=[])
    esg_adjustment:    dict                                    =
sagafield(default={"bps": 0})
    escrow_state:    str                                       =
sagafield(default="funded")       # funded, releasable,
released, clawback

    escrow_conditions: dict      = sagafield(                   #
machine-evaluable gates
        default={"serialization_verified": True,
                 "lmics_share_met": True,
                 "cold_chain_ok": True}
    )

    @sagamethod()
    def evaluate_and_mark(self, releasable: bool,
adj_bps: int = 0):
        self.escrow_state = "releasable" if releasable else
"funded"
        self.esg_adjustment["bps"] = adj_bps

    @sagamethod()
    def release(self):
        self.escrow_state = "released"

    @sagamethod()
    def clawback(self, reason: str):
        self.escrow_state = f"clawback:{reason}"

# ----------------------------
# ESG metrics (carbon & cold-chain) referenced by
escrows
# ----------------------------

@sagaclass()
class ClassCarbonLedger(SPClassObject):
    """Attributable emissions for shipments bound to
SGTIN/SSCC (auditable)."""
    ledger_id: str               = sagafield()
    stage: str                   = sagafield(default="Transport")
    method: str                  = sagafield(default="GHG
Protocol")
    kg_co2e: float               = sagafield()
    subjects: list               = sagafield(default=[])        #
SGTIN/SSCC
    evidence_hashes: list        = sagafield(default=[])
    period_start: str            = sagafield()
    period_end: str              = sagafield()

@sagaclass()
class ClassColdChainTelemetry(SPClassObject):
    """Signed time-series from data loggers/carriers
(EPCIS JSON-LD preserved)."""
    stream_id: str               = sagafield()
    device_id: str               = sagafield()
    signed_payload: dict         = sagafield(default={})
    summary: dict                = sagafield(default={})
# {min, max, excursions, MKT}
    subjects: list               = sagafield(default=[])        #
SGTIN/SSCC
    attestation: str             = sagafield()                  #
carrier/logger signature

# ----------------------------
# Public transparency (non-sensitive) via SagaFeeds
# ----------------------------

@sagaclass()
class ClassEquityFeeds(SPClassObject):
    """Public, no-cost indices of equity deliveries &
compliance (no PHI/contract data)."""
    feed_id: str                 = sagafield()
    programs: dict               = sagafield(default={})
# {"program_tag":[{country,qty,ts,cc_ok,co2e}]}
    updated_at: str              = sagafield()

    @sagamethod()
    def append_delivery(self, program_tag: str,
country: str, qty: int,
```

```
                delivered_at: str, cc_ok: bool,
co2e_per_unit: float | None):
    row = {
      "country": country, "qty": qty, "delivered_at":
delivered_at,
      "cold_chain_ok": cc_ok, "kg_co2e_per_unit":
co2e_per_unit
    }
    arr = self.programs.get(program_tag, [])
    arr.append(row)
    self.programs[program_tag] = arr
    self.updated_at = delivered_at
```

Execution model:
• Delivery proofs aggregate **EPCIS/DSCSA** verification, **customs** outcomes, and **cold-chain** telemetry inside an enclave; a **TEE/zk attestation** is recorded on ClassEquityDeliveryProof.
• ClassISO20022EquityEscrow evaluates **policy gates** and ESG adjustments, then **releases** or **claws back** funds.
• **SagaFeeds** object provides public, non-sensitive delivery snapshots (quantities, countries, basic ESG).

Interoperability Analysis

- **WHO / GAVI / UNICEF:** Equity obligations modeled in ClassPandemicTreatyObligation; verification refs point to delivery proofs and policy checks.
- **DSCSA / GS1 / EPCIS:** Unit- and logistics-level provenance (SGTIN/SSCC + events) anchor each donor-funded dose.
- **ISO IDMP:** Unambiguous global product identity for cross-border federations & regulators.
- **ISO 20022:** ClassISO20022EquityEscrow executes **delivery-vs-payment** with policy/ESG conditions.
- **ESG:** ClassCarbonLedger + ClassColdChainTelemetry drive **bps adjustments** (discount/penalty) in escrow.
- **Private Enclaves:** PHI and contracts remain encrypted in use; only compliance proofs and KPIs are disclosed.
- **SagaFeeds:** Open dashboards show **who got what, when** (quantities, countries, basic ESG/quality status) without exposing sensitive details.

Example Flow (End-to-End)

1. **Fund:** Donor creates ClassEquityFundAsset; escrow funded in ClassISO20022EquityEscrow.
2. **Serialize & Ship:** Manufacturer binds SGTINs; EPCIS events + customs docs accrue.
3. **Deliver & Verify:** Enclave composes ClassEquityDeliveryProof (serialization pass, customs accept, cold-chain ok).
4. **Evaluate:** Policy gates checked; ESG adjustment calculated → evaluate_and_mark(releasable=True, adj_bps=-12)
5. **Release/Clawback:** release() moves escrow; if failure/diversion, clawback("policy_breach").
6. **Publish:** ClassEquityFeeds.append_delivery() updates public program dashboard.

Implications

- **Donors:** Verifiable **impact-per-dollar** with conditional financing; fewer leakages.
- **Governments & WHO:** Treaty-grade equity and reporting compliance with automated proofs.
- **Manufacturers/Distributors:** Faster cash conversion via objective **delivery proofs**.
- **Patients:** More reliable, equitable access; counterfeit/diversion deterrence.

- **Global Health:** Financing aligned to **sustainability** and **equity** at unit-level granularity.

6. Regulatory Subtrees and Global Standards Integration

This section presents the regulatory subtrees for key U.S. frameworks (21 CFR, DSCSA, ISPE) and integrates them with global standards (financial, supply chain, clinical, imaging, workflow, and modeling), all encoded as multi-inheritable classes in the Global Pharmaceutical Universal Class Tree under SagaStandards governance. Each subtree enforces compliance-by-design through persistent SagaPSA™ objects, with interoperability via LOID references, enclave computation for confidentiality, and SagaFeeds™ for transparency. Diagrams illustrate cross-links, enabling a verifiable continuum from research to patient delivery.

6.1 - 21 CFR (11, 210, 211, 312, 314, 600s, 820)

Background & Problem Statement

The U.S. Code of Federal Regulations (Title 21 CFR) provides the cornerstone for pharmaceutical manufacturing and clinical research compliance. Key parts include:

- Part 11: Electronic records and signatures.
- Parts 210/211: Good Manufacturing Practice (GMP) for drugs.
- Part 312: Investigational New Drug (IND) applications.
- Part 314: New Drug Applications (NDAs).
- 600 series: Biologics regulation.
- Part 820: Quality System Regulation (QSR) for medical devices.

Challenges in the current environment include:

- Fragmented digital systems for manufacturing records and clinical trial submissions.
- Limited auditability of electronic signatures and records.
- Complex CAPA (Corrective and Preventive Action) workflows with poor interoperability across regulators and manufacturers.

Technical Approach (SagaChain Implementation)

21 CFR Subtree in the Global Pharma Class Tree

The 21 CFR subtree encodes each major regulatory part as a programmable class under the Global Pharmaceutical Universal Class Tree, governed by SagaStandards of the PraSaga Foundation. This allows compliance to be enforced by design rather than through siloed, post-hoc reporting, with classes persisting as SagaPSA™ (Programmable Smart Assets) in the single-instance class tree.

- ClassElectronicRecord → Enforces Part 11, embedding immutable audit trails and cryptographic signatures.

- ClassGMPBatch → Enforces Parts 210/211, managing batch-level GMP records.
- ClassClinicalInvestigation → Enforces Part 312, linking directly to FDAAA 801 trial registration.
- ClassBiologicLot → Enforces 600-series biologics requirements.
- ClassDeviceQSR → Enforces Part 820 device quality records & CAPA workflows.

Classes inherit from: CMIConst.SPClassObject or domain-specific ancestors, with multi-inheritance enabling cross-links (e.g., GMP batches inheriting from DSCSA serialization mixins).

Sample SagaPython Code

Classes are defined using strict SagaPython semantics: @SagaClass for persistence and inheritance, @SagaMethod for lifecycle methods, and SagaField for validated fields. All classes are registered in the Global Class Registry for canonical evolution under SagaStandards governance.

```python
from saga import SagaClass, SagaMethod, SagaField, CMIConst


@SagaClass(CMIConst.SPClassObject)
class ClassElectronicRecord:
    """21 CFR Part 11 electronic record with audit log."""
    record_id = SagaField("str")
    signer = SagaField("str")
    timestamp = SagaField("datetime")
    signature_hash = SagaField("str")
    audit_log = SagaField("list[dict]")

    @SagaMethod()
    def __init__(self):
        self._cmi().__init__()
        self.audit_log = []

    @SagaMethod()
    def sign(self, signer: str, signature: str):
        self.signer = signer
        self.signature_hash = signature
        self.audit_log.append({"signer": signer, "signature": signature, "ts": self._now()})
        return {"ok": True, "audit_len": len(self.audit_log)}


@SagaClass(CMIConst.SPClassObject)
class ClassGMPBatch:
    """Good Manufacturing Practice batch record (21 CFR 210/211)."""
    batch_number = SagaField("str")
    lot_number = SagaField("str")
    manufacture_dt = SagaField("date")
    status = SagaField("str", default="pending")  # pending/held/released
    deviations = SagaField("list[dict]")
```

```
    release_date = SagaField("date", default=None)

    @SagaMethod()
    def __init__(self):
        self._cmi().__init__()
        self.deviations = []
        self.release_date = None

    @SagaMethod()
    def record_deviation(self, deviation: dict):
        self.deviations.append(deviation)
        self.status = "held"
        return {"status": self.status, "deviation_count": len(self.deviations)}

    @SagaMethod()
    def release_batch(self, release_date: str):
        self.status = "released"
        self.release_date = release_date
        return {"status": self.status, "release_date": self.release_date}

@SagaClass(CMIConst.SPClassObject)
class ClassClinicalInvestigation:
    """IND clinical investigation (21 CFR 312)."""
    investigation_id = SagaField("str")
    sponsor = SagaField("str")
    protocol_ref = SagaField("str")  # FDAAA 801 trial ID
    status = SagaField("str", default="active")

    @SagaMethod()
    def __init__(self):
        self._cmi().__init__()

    @SagaMethod()
    def close_investigation(self):
        self.status = "closed"
        return {"status": self.status}

@SagaClass(CMIConst.SPClassObject)
class ClassBiologicLot:
    """Biologic lot record (21 CFR 600-series)."""
    lot_number = SagaField("str")
    product_code = SagaField("str")  # ISO IDMP identifier
    manufacture_dt = SagaField("date")
    expiry_dt = SagaField("date")
    potency_results = SagaField("list[dict]")

    @SagaMethod()
    def __init__(self):
        self._cmi().__init__()
```

```
        self.potency_results = []

    @SagaMethod()
    def add_potency_result(self, result: dict):
        self.potency_results.append(result)
        return {"potency_count": len(self.potency_results)}

@SagaClass(CMIConst.SPClassObject)
class ClassDeviceQSR:
    """Device Quality System Regulation (21 CFR 820)."""
    device_id = SagaField("str")
    manufacturer = SagaField("str")
    quality_records = SagaField("list[dict]") # CAPA, complaints, service
    status = SagaField("str", default="compliant")

    @SagaMethod()
    def __init__(self):
        self._cmi().__init__()
        self.quality_records = []

    @SagaMethod()
    def record_capa(self, capa: dict):
        self.quality_records.append(capa)
        self.status = "under_review"
        return {"status": self.status, "capa_count": len(self.quality_records)}
```

Interoperability Analysis

- With DSCSA & GS1: GMP batch records link directly to serialized supply chain identifiers via LOIDs (e.g., bind to ClassDSCSAUnit).
- With USP: Batch records embed USP monograph references for automated quality checks enforced in @SagaMethod hooks.
- With ISO 20022: Manufacturing batches tied to compliant payment and invoicing workflows via escrow conditions.
- With HL7: IND investigations synchronize with FHIR ResearchStudy and patient-level EHR data through OID cross-references.

Impact

The 21 CFR subtree enables a seamless chain of custody: Research → Manufacturing → Regulatory Approval → Distribution → Patient Delivery. Each stage is linked by programmable, auditable class objects, ensuring compliance is built-in, not bolted on. Under SagaStandards, classes evolve via participatory diffs, with SagaFeeds™ enables transparency for auditors.

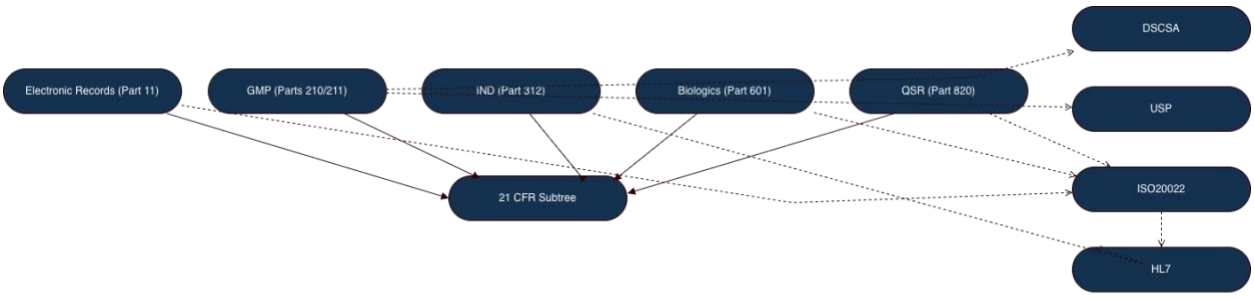


Diagram 6-1 - 21 CFR Subtree - Electronic Records, GMP, IND, Biologics, QSR (Cross-links: DSCSA • USP • ISO20022 • HL7)

6.2 - DSCSA (Drug Supply Chain Security Act)

Background & Problem Statement

The Drug Supply Chain Security Act (DSCSA), enacted in 2013, established a ten-year roadmap for an interoperable, electronic system to identify and trace prescription drugs in the United States, aiming to:

- Prevent counterfeit or diverted drugs from entering the supply chain.
- Facilitate rapid detection and removal of unsafe products.
- Improve trust and visibility across manufacturers, wholesalers, dispensers, and regulators.

Despite progress, the landscape still suffers from:

- Heterogeneous data formats: trading partners exchange EPCIS XML, PDFs, CSVs, or proprietary ERP outputs, complicating reconciliation.
- Fragmented traceability: no single authoritative registry for serialized units; verification often outsourced to costly third-party hubs.
- Fee volatility & scaling issues: hundreds of millions of serial events/year stress centralized systems.

Net effect: a heavy compliance burden that undermines DSCSA's goal of seamless interoperability.

Technical Approach (SagaChain Implementation)

The DSCSA subtree of the Global Pharmaceutical Universal Class Tree encodes serialization, transaction history, aggregation, and verification as multi-inheritable programmable smart assets, governed by SagaStandards:

- Serialized Drug Units (ClassDSCSAUnit) — each unit is a ClassNonFungibleAsset carrying GS1 SGTIN, lot, expiry, and ownership.
- Transaction Information (ClassTransactionInfo) — immutable on-chain events recording sender, receiver, date, and product list.
- Verification Requests & Responses (ClassVerificationEvent) — cryptographically attested queries against persistent state.
- Aggregation Events — hierarchical links among items, cases, and pallets in EPCIS-compatible structures.

Because SagaChain uses a global single-instance class tree with parallel shards, all trading partners (manufacturers, wholesalers, dispensers) interact with one canonical registry, not siloed databases—preserving consistency at scale while stabilizing transaction fees via SagaCoin™ management.

Sample SagaPython Code

Classes follow strict SagaPython: multi-inheritance from ClassNonFungibleAsset where applicable, with SagaField constraints and @SagaMethod for EPCIS events.

```python
from saga import SagaClass, SagaMethod, SagaField, CMIConst

@SagaClass(CMIConst.SPClassObject, ClassNonFungibleAsset)
class ClassDSCSAUnit:
    """Serialized drug unit under DSCSA compliance."""
    sgtin = SagaField("str")  # GS1 SGTIN
    lot = SagaField("str")
    expiration = SagaField("date")
    manufacturer = SagaField("str")  # GLN/DID of manufacturer
    verification_status = SagaField("str", default="unverified")  # unverified/verified/expired/suspect
    epcis_events = SagaField("list[dict]")  # commission/ship/receive/agg/disagg
    parent_sscc = SagaField("str", default="")

    @SagaMethod()
    def __init__(self):
        self._cmi().__init__()
        self._cmi('ClassNonFungibleAsset').__init__()
        self.epcis_events = []

    @SagaMethod()
    def commission(self, lot: str, expiration: str):
        self.lot = lot
        self.expiration = expiration
        self.epcis_events.append({"action": "commission", "ts": self._now()})
        self.verification_status = "verified"
        return {"status": self.verification_status}

@SagaClass(CMIConst.SPClassObject)
class ClassTransactionInfo:
    """Immutable DSCSA transaction event."""
    transaction_id = SagaField("str")
    sender = SagaField("str")  # LOID
    receiver = SagaField("str")  # LOID
    timestamp = SagaField("datetime")
    product_list = SagaField("list[str]")  # SGTIN LOIDs
```

```
    @SagaMethod()
    def __init__(self):
        self._cmi().__init__()
        self.product_list = []

    @SagaMethod()
    def record_ship(self, sender: str, receiver: str, products: list):
        self.sender = sender
        self.receiver = receiver
        self.product_list = products
        self.timestamp = self._now()
        return {"transaction_id": self.transaction_id}
```

Interoperability Analysis

- 21 CFR (11/210/211/820): Part 11 e-signatures on all artifacts; GMP release gated via method invariants linking to ClassGMPBatch.
- USP (Regulatory & Standards): Serialization events reference USP monograph proofs.
- GS1 / DSCSA: EPCIS events enforced as append-only lists; verification queries resolve against Global Class Registry.
- ISO IDMP & SPL: Transaction product_list binds to IDMP identifiers; release documentation updates SPL classes.
- FAERS / VAERS: Safety signals trigger verification re-queries on serialized units.
- ISO 20022: Shipments condition payment releases on transaction LOIDs.
- SagaFeeds™: Public feeds can expose aggregate verification stats (e.g., % verified units).

Impact

DSCSA compliance becomes executable: serialized units persist as non-fungible assets with hierarchical aggregation, enabling real-time traceability without third-party hubs. SagaStandards ensures versioned evolution, with private enclaves for sensitive ownership proofs.

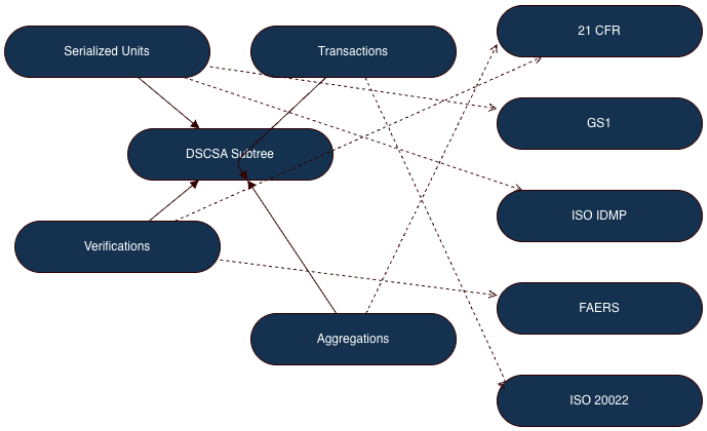


Diagram 6.2-A — DSCSA Subtree — Serialized Units, Transactions, Verifications, Aggregations (Cross-links: 21 CFR • GS1 • ISO IDMP • FAERS • ISO 20022)

6.3 - ISPE (International Society for Pharmaceutical Engineering) GAMP 5 & Validation

6.3.1 - Background & Problem Statement

ISPE GAMP 5 provides guidance on computerized system validation, risk-based testing, and change control for pharmaceutical manufacturing. Core elements include requirements traceability, verification testing, continuous change control systems (CCS), quality risk management (QRM), and validation packages that gate product releases.

Challenges include:

- Manual traceability: Requirements scattered across spreadsheets, lacking enforceable links to verifications.
- Siloed evidence: Test vectors, lab data, and vendor IP stored off-chain, risking integrity and auditability.
- Reactive change control: Post-release deviations trigger costly re-validations without proactive risk scoring.
- Confidentiality gaps: Sensitive configs (e.g., AI model weights for assay prediction) exposed in shared audits.

Net effect: Validation burdens slow innovation, inflate costs, and undermine trust in automated systems.

6.3.2 - Technical Approach (SagaChain Implementation)

The ISPE subtree encodes GAMP 5 as persistent classes in the Universal Class Tree: requirements as traceable objects, verifications as attested methods, CCS as stateful workflows, QRM as risk-weighted mixins, and validation packages as gating PSAs. Private enclaves secure evidence while emitting proofs; SagaStandards governs class stewardship by ISPE and SDOs.

- ClassGAMPRequirement → Traceable specs with risk scores.
- ClassVerificationTest → Enclave-backed test executions.
- ClassChangeControl → Workflow for deviations and updates.
- ClassValidationPackage → Aggregates requirements/verifications, gates releases.

Sample SagaPython Code

Strict SagaPython: Enclave methods for confidential compute; multi-inheritance for risk mixins.

```python
from saga import SagaClass, SagaMethod, SagaField, CMIConst

@SagaClass(CMIConst.SPClassObject)

class ClassGAMPRequirement:

    """GAMP 5 traceable requirement."""

    req_id = SagaField("str")
```

```
    category = SagaField("str", enum={"functional", "performance", "security"})
    description = SagaField("str")
    risk_score = SagaField("int", min=1, max=5)
    verification_refs = SagaField("list[str]") # LOIDs

    @SagaMethod()
    def __init__(self):
        self._cmi().__init__()
        self.verification_refs = []

    @SagaMethod()
    def link_verification(self, ver_loid: str):
        self.verification_refs.append(ver_loid)
        return {"req_id": self.req_id, "linked_count": len(self.verification_refs)}

@SagaClass(enclave=True, CMIConst.SPClassObject)
class ClassVerificationTest:
    """Enclave-executed GAMP verification with proofs."""
    test_id = SagaField("str")
    req_ref = SagaField("str") # LOID to ClassGAMPRequirement
    inputs = SagaField("dict")
    expected = SagaField("dict")
    actual = SagaField("dict", default=None)
    passed = SagaField("bool", default=False)

    @SagaMethod()
    def __init__(self):
        self._cmi().__init__()
        self._enclave = {} # enclave storage

    @SagaMethod()
    def execute_test(self, inputs: dict, expected: dict):
        # Enclave compute (mock)
        self.inputs = inputs
        self.expected = expected
        self.actual = self._enclave_compute(inputs) # enclave call
        self.passed = self.actual == expected
        return {"passed": self.passed, "proof": self._enclave_attest()}

    def _enclave_compute(self, inputs: dict) -> dict:
        # Placeholder enclave logic
        return {"result": "computed_value"}

@SagaClass(CMIConst.SPClassObject)
class ClassValidationPackage:
```

```
    """Aggregated GAMP validation gating release."""

    package_id = SagaField("str")

    req_refs = SagaField("list[str]")

    ver_refs = SagaField("list[str]")

    coverage_pct = SagaField("float", min=0, max=100)

    status = SagaField("str", enum={"draft", "verified", "gated"})


    @SagaMethod()

    def __init__(self):

        self._cmi().__init__()

        self.req_refs = []

        self.ver_refs = []


    @SagaMethod()

    def evaluate_and_gate_release(self):

        self.coverage_pct = (len(self.ver_refs) / len(self.req_refs)) * 100 if self.req_refs else 0

        if self.coverage_pct >= 95:

            self.status = "gated"

        return {"coverage": self.coverage_pct, "status": self.status}
```

6.3.3 - Interoperability Analysis

- 21 CFR (11/210/211/820): Part 11 e-signatures on all artifacts; GMP release gated via ClassValidationPackage.evaluate_and_gate_release().
- QSR (820) software/equipment validation linked to device records via LOIDs.
- USP (Regulatory & Standards): Validation packages include USP method suitability and reference standard traceability; CCS monitors align with USP microbial/particulate limits.
- GS1 / DSCSA: Only validated MES/serialization states can commission SGTINs; EPCIS events reference qualified equipment states.
- ISO IDMP & SPL: Requirements/verifications reference IDMP product identifiers; release documentation binds to SPL updates.
- FAERS / VAERS: Safety signals reopen risks and change control, triggering targeted re-verification.
- ISO 20022: Payments (equipment release, tech transfer, toll manufacturing) are programmatically conditioned on validation milestones.
- SagaFeeds™: Non-sensitive indices (coverage %, requirement status) are available to auditors/regulators; sensitive evidence remains in enclave.

Impact

ISPE guidance becomes machine-executable assurance. Requirements, risks, verifications, and validation packages are persistent objects

that gate real releases, drive serialization eligibility, synchronize with labeling and safety, and even unlock payments—with confidential evidence secured by enclave attestations. SagaStandards enables ISPE stewardship of class evolution.

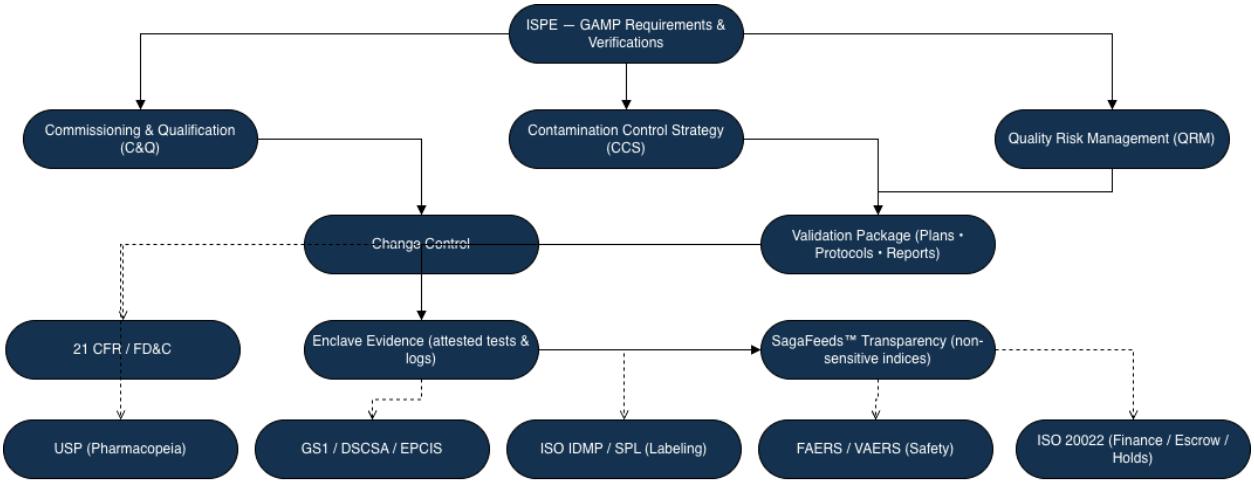


Diagram 6.3-A — ISPE Subtree — GAMP Requirements & Verifications

ISPE Subtree: GAMP drives C&Q, CCS, and QRM into Change Control and Validation Packages; attested evidence produced in Enclave flows to SagaFeeds. Cross-links align artifacts to 21 CFR, USP, GS1/DSCSA/EPCIS, ISO IDMP/SPL, FAERS/VAERS, and ISO 20022 finance.

6.4 Financial Standards Integration (ISO 20022, FIX, XBRL, FIGI, IFRS)

6.4.1 - Background & Problem Statement

Global pharmaceutical markets are as much financial systems as they are scientific or clinical. The end-to-end lifecycle of therapies — from R&D funding through clinical trial reimbursements, procurement, logistics finance, payer settlements, and post-market risk management — is underpinned by financial flows. However, the financial infrastructure that supports this lifecycle remains fragmented and inefficient:

- Multiple Standards: ISO 20022 governs settlement messaging, FIX protocol handles trade execution, XBRL supports corporate financial disclosure, and FIGI provides instrument identifiers. These exist in silos with limited interoperability.
- Complex Reconciliation: Pharma companies, CROs, payers, and regulators reconcile financial and clinical data manually, creating delays, errors, and fraud exposure.
- Opaque Donor Funding: Global health initiatives (e.g., GAVI, Global Fund) struggle with traceability from donor contributions to patient-level delivery.
- Risk & Compliance Gaps: IFRS and national reporting frameworks are inconsistently applied, reducing investor and regulator confidence.

- Static Systems: Settlement and reimbursement systems lack programmability, preventing outcome-based models tied to real-world clinical data.

This disconnect between financial standards and clinical/regulatory frameworks leads to inefficiencies, fraud risk, and inequitable distribution of resources.

6.4.2 - Technical Approach (SagaChain Implementation with Private Enclaves)

Under SagaStandards, SagaChain integrates financial standards into its persistent-state class tree, binding economic flows directly to clinical and regulatory proofs via the Global Pharmaceutical Universal Class Tree:

1. **ISO 20022 Settlement Layer:** ClassISO20022Payment encodes programmable settlement messages, linked to shipments, trials, or reimbursement proofs. Enclaves enforce escrow conditions (serialization verified, outcomes achieved).
2. **FIX Protocol Trade Integration:** ClassFIXTrade represents secondary market activities (e.g., procurement contracts, bulk purchasing), ensuring trade identifiers map to underlying serialized assets (SGTIN, IDMP).
3. XBRL for Corporate & Donor Reporting: ClassXBRLDisclosure objects ensure corporate reports, donor impact statements, and ESG metrics are natively linked to on-chain proofs.
4. FIGI & Instrument Identity: ClassFinancialInstrument binds FIGI identifiers to pharma-linked assets, enabling recognition in global capital markets.
5. IFRS / GAAP Compliance: Financial flows auto-generate auditable disclosure packages. Private Enclaves preserve sensitive commercial terms while exposing compliance proofs.

Classes use multi-inheritance (e.g., from ClassFungibleAsset for payments) and enclave flags for confidentiality.

Sample SagaPython Code

Strict SagaPython: Enclave=True for sensitive fields; type hints in SagaField.

```python
from saga import SagaClass, SagaMethod, SagaField, CMIConst


@SagaClass(enclave=True, CMIConst.SPClassObject)
class ClassISO20022Payment:
    """Programmable ISO 20022 settlement message."""
    payment_id = SagaField("str")
    payer = SagaField("str")  # LOID
    payee = SagaField("str")  # LOID
    amount = SagaField("int")  # Minor units
    currency = SagaField("str", length=3)
    linked_asset = SagaField("str")  # IDMP, shipment, or trial LOID
    escrow_conditions = SagaField("dict")  # {"serialization_verified": True, "outcome_verified": True}
```

```
    status = SagaField("str", enum={"pending", "released", "disputed"})
    enclave_proof = SagaField("str")

    @SagaMethod()
    def __init__(self):
        self._cmi().__init__()
        self.escrow_conditions = {}
        self.enclave_proof = ""

    @SagaMethod()
    def release_escrow(self):
        if all(self.escrow_conditions.values()):
            self.status = "released"
            self.enclave_proof = self._enclave_attest()
        return {"status": self.status}

@SagaClass(CMIConst.SPClassObject)
class ClassFIXTrade:
    """Trade execution linked to pharma assets (procurement, secondary)."""
    trade_id = SagaField("str")
    figi_id = SagaField("str")
    counterparty = SagaField("str")  # LOID
    asset_ref = SagaField("str")  # ClassFinancialInstrument LOID
    trade_terms = SagaField("dict")
    settlement_refs = SagaField("list[str]")

    @SagaMethod()
    def __init__(self):
        self._cmi().__init__()
        self.settlement_refs = []

@SagaClass(CMIConst.SPClassObject)
class ClassXBRLDisclosure:
    """Financial disclosure mapped to XBRL taxonomy."""
    disclosure_id = SagaField("str")
    entity = SagaField("str")  # LOID
    period = SagaField("str")
    taxonomy_refs = SagaField("list[str]")  # IFRS/GAAP concepts
    values = SagaField("dict")
    enclave_proofs = SagaField("list[str]")

    @SagaMethod()
    def __init__(self):
        self._cmi().__init__()
        self.taxonomy_refs = []
        self.enclave_proofs = []
```

6.4.3 - Interoperability Analysis

- ISO 20022: Enables programmable, condition-based settlements (e.g.,

reimbursement released only when DSCSA compliance or HL7 outcomes are verified via LOID binds).

- FIX Protocol: Extends standard trading infrastructure to pharma-linked instruments (e.g., carbon credits for cold-chain, outcome-linked bonds).
- XBRL: Ensures donor funding and corporate ESG disclosures are cryptographically linked to operational proofs.
- FIGI: Harmonizes pharma-linked financial instruments across global capital markets.
- IFRS / GAAP: Automated disclosures ensure compliance with international financial reporting.
- Private Enclaves: Preserve confidentiality of commercial contracts while exposing auditable compliance proofs.
- SagaFeeds™: Publish anonymized financial performance benchmarks, ESG-adjusted rates, and donor impact summaries for transparency.

Implications

- Regulators & Auditors: Gain continuous, verifiable financial flows tied directly to regulated pharma activities.
- Manufacturers & CROs: Reduce reconciliation costs; access programmable, milestone-based finance.
- Donors & Payers: Ensure funds reach intended outcomes; automate value-based reimbursement.
- Investors: Trade pharma-linked financial instruments with cryptographic assurance.
- Patients & Global Health: Improved trust and equity as financing flows become transparent and performance-linked.

Section 6.4 demonstrates how SagaChain unifies global financial standards into a programmable compliance layer, transforming fragmented payments and disclosures into verifiable, outcome-driven infrastructure under SagaStandards governance.

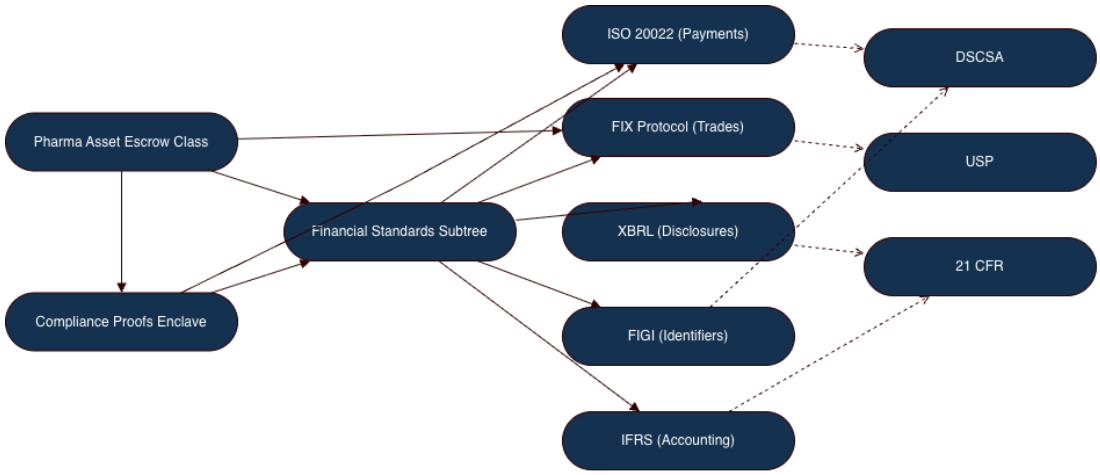


Diagram 6.4-A — Financial Standards Subtree (ISO 20022, FIX, XBRL, FIGI, IFRS) — Pharma Escrow & Compliance Enclaves

6.5 Supply Chain Standards Integration (GS1, DSCSA, EPCIS, IDMP)

6.5.1 Background & Problem Statement

Supply chain standards like GS1 (EPCIS for event data), DSCSA (U.S. serialization), ISO IDMP (product identification), and SPL (labeling) aim to ensure traceability, authenticity, and interoperability across global pharma logistics. Yet, implementation reveals persistent gaps:

- Format Heterogeneity: GS1 EPCIS events clash with DSCSA XML requirements, forcing costly transformations.
- Identifier Fragmentation: IDMP substance/product codes do not natively resolve to GS1 SGTINs or SPL sections.
- Visibility Silos: EPCIS hubs lack enforceable links to regulatory approvals (e.g., 21 CFR lot releases) or safety signals (FAERS).
- Scalability Barriers: Billions of annual events overwhelm centralized verifiers, especially in LMICs.
- Equity Issues: Donor programs (e.g., COVAX) lack verifiable proofs of cold-chain integrity and equitable distribution.

These disconnects amplify risks of counterfeits, delays, and inequitable access, undermining global health resilience.

6.5.2 Technical Approach (SagaChain Implementation)

SagaChain embeds supply chain standards as multi-inheritable classes in the Universal Class Tree, creating a persistent evidence graph for end-to-end traceability. Under SagaStandards, GS1, ISO, and FDA/EMA collaborate on namespace stewardship:

1. GS1 EPCIS Layer: ClassEPCISEvent as append-only mixins for commission/ship/receive actions.
2. DSCSA Serialization: Extends ClassNonFungibleAsset with verification workflows.
3. SO IDMP Identification: ClassIDMPProduct as canonical identifiers binding substances, products, and packages.
4. SPL Labeling: ClassSPLDocument synchronizes multilingual labels with serialization proofs.

Enclaves secure sensitive telemetry (e.g., IoT cold-chain data); SagaFeeds™ expose public traceability dashboards.

Sample SagaPython Code

Strict SagaPython: Multi-inheritance for event mixins; LOID binds for IDMP-SGTIN resolution.

```python
from saga import SagaClass, SagaMethod, SagaField, CMIConst


@SagaClass(CMIConst.SPClassObject)
class ClassIDMPProduct:
```

```
    """ISO IDMP canonical product identifier."""
    idmp_id = SagaField("str")
    name = SagaField("str")
    strength = SagaField("str")
    form = SagaField("str")
    sgtin_refs = SagaField("list[str]")  # GS1 links

    @SagaMethod()
    def __init__(self):
        self._cmi().__init__()
        self.sgtin_refs = []

    @SagaMethod()
    def bind_sgtin(self, sgtin: str):
        self.sgtin_refs.append(sgtin)
        return {"idmp_id": self.idmp_id, "bound_count": len(self.sgtin_refs)}

@SagaClass(CMIConst.SPClassObject, ClassDSCSAUnit)  # Multi-inherit from DSCSA
class ClassEPCISEvent:
    """GS1 EPCIS event mixin for supply chain actions."""
    event_id = SagaField("str")
    action = SagaField("str", enum={"commission", "ship", "receive", "aggregate"})
    timestamp = SagaField("datetime")
    biz_step = SagaField("str")
    disposition = SagaField("str")
    unit_list = SagaField("list[str]")  # SGTIN LOIDs

    @SagaMethod()
    def __init__(self):
        self._cmi().__init__()
        self._cmi('ClassDSCSAUnit').__init__()
        self.unit_list = []

    @SagaMethod()
    def record_event(self, action: str, units: list):
        self.action = action
        self.unit_list = units
        self.timestamp = self._now()
        return {"event_id": self.event_id}

@SagaClass(CMIConst.SPClassObject)
class ClassSPLDocument:
    """FDA SPL linked to IDMP and serialization."""
    spl_id = SagaField("str")
    idmp_ref = SagaField("str")  # LOID
    sections = SagaField("dict")  # {"INDICATIONS": "...", ...}
```

```
    serialization_proof = SagaField("str")  # Enclave attestation
    version = SagaField("str")

    @SagaMethod()
    def __init__(self):
        self._cmi().__init__()
        self.sections = {}

    @SagaMethod()
    def update_section(self, code: str, text: str):
        self.sections[code] = text
        self.version = self._now().isoformat()
        return {"spl_id": self.spl_id, "version": self.version}
```

Interoperability Analysis

- GS1 EPCIS: Events append immutably; composable with DSCSA verifications via shared LOIDs.
- DSCSA: Serialization units inherit IDMP identifiers, enabling U.S.-global harmonization.
- ISO IDMP: Canonical codes resolve to SPL sections and EPCIS units, enforced in method invariants.
- 21 CFR/DSCSA Links: Lot releases (ClassGMPBatch) gate EPCIS commissions; SPL updates trigger FAERS subscriptions.
- HL7/FAERS: Patient delivery proofs (EPCIS receive) link to outcome reporting.
- ISO 20022: Settlements conditioned on aggregation proofs (e.g., pallet-level verification).
- SagaFeeds™: Dashboardsfor SGTIN scans, showing IDMP details and cold-chain status without PHI.

Implications

Manufacturers & Distributors: Automated traceability reduces recall costs; programmable aggregation scales to billions of events.

- Regulators (FDA/EMA/WHO): Real-time EPCIS feeds enable proactive counterfeit detection; IDMP stewardship under SagaStandards.
- LMICs & Donors: Verifiable equity proofs (e.g., GAVI doses delivered) via public feeds, bridging global-north/south divides.
- Patients: GS1 Digital Link scans reveal full lineage (IDMP → SPL → DSCSA), empowering informed choices.
- Global Health: Transforms supply chains from reactive logistics to proactive, standards-enforced resilience.

Section 6.5 illustrates SagaChain's role in harmonizing supply chain standards as executable classes, forging a verifiable continuum from manufacturer to patient under participatory SagaStandards governance.

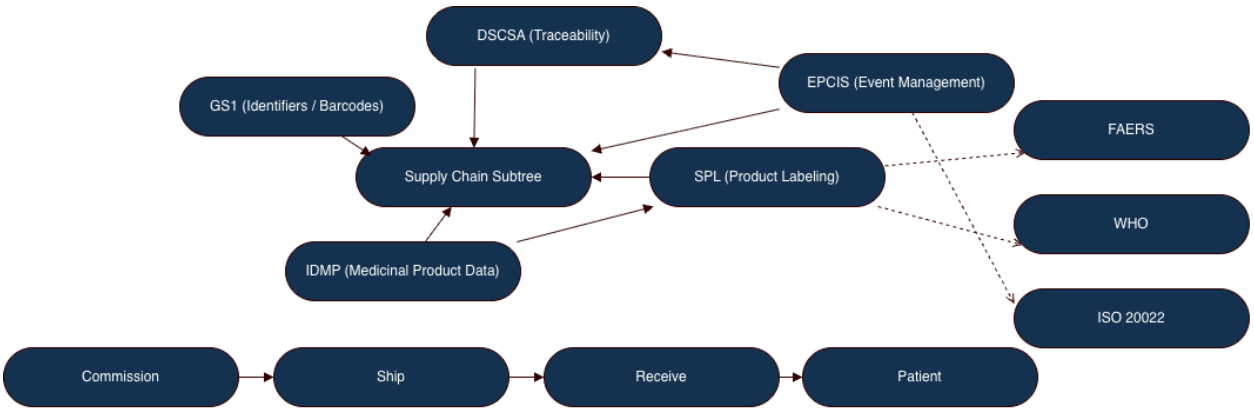


Diagram 6-5-A — Supply Chain Subtree (GS1, DSCSA, EPCIS, IDMP, SPL) — EPCIS Event Flow & Regulatory Cross-Links

6.6 Clinical Standards Integration (HL7 FHIR R5)

6.6.1 Background & Problem Statement

HL7 FHIR (Fast Healthcare Interoperability Resources) R5, released in 2023, standardizes the exchange of healthcare data through RESTful APIs and modular resources, enabling seamless integration across EHRs, clinical trials, pharmacovigilance, and regulatory reporting. In the pharmaceutical domain, key resources include Patient (demographics), Medication (product details), MedicationRequest (prescriptions), Observation (lab results), ResearchStudy (clinical trials), and AdverseEvent (safety signals). FHIR promotes uniform data formats, reducing errors in patient care and outcomes reporting.

Challenges in current implementations include:

- Siloed Data: Clinical outcomes (e.g., Observations) do not natively link to supply chain proofs (DSCSA batches) or labeling (SPL/IDMP codes), complicating reimbursement and safety surveillance.
- Privacy Barriers: PHI (Protected Health Information) restricts cross-system sharing, hindering aggregate signal detection for FAERS/VAERS.
- Version Drift: R5 revisions (e.g., enhanced Medication resources with CodeableReference for ingredients) are not enforced consistently, leading to reconciliation failures.
- Scalability Gaps: High-volume events (e.g., billions of Observations annually) overwhelm centralized FHIR servers without persistent, sharded storage.
- Equity Issues: LMICs lack access to FHIR-compliant tools, delaying integration with global pharmacovigilance (WHO VigiBase).

This fragmentation delays value-based care, inflates adverse event reporting latency, and erodes trust in outcome-linked financing.

6.6.2 Technical Approach (SagaChain Implementation with Private Enclaves)

Under SagaStandards, SagaChain embeds HL7 FHIR R5 resources as multi-inheritable classes in the Global Pharmaceutical Universal Class Tree, transforming static JSON/XML exchanges into persistent SagaPSA™ objects. FHIR elements map to SagaField constraints (e.g., code bindings to RxNorm/IDMP), with references resolved as LOIDs for causal linking. Private enclaves encrypt PHI (e.g., Patient demographics) while exposing de-identified proofs (e.g., aggregate Observations for safety signals).

Core mappings:

1. Patient Resource: ClassFHIRPatient as a privacy-gated base for demographics and references.

2. Medication Resource: ClassFHIRMedication inheriting from IDMP mixins, enforcing R5 fields like code (CodeableConcept with RxNorm/UNII), doseForm, and batch (lotNumber/expirationDate).

3. AdverseEvent Resource: ClassFHIRAdverseEvent linking to DSCSA units and FAERS reports via LOIDs.

4. ResearchStudy Resource: ClassFHIRResearchStudy synchronizing with 21 CFR 312 investigations.

5. Observation/MedicationRequest: Mixins for lab results and prescriptions, with enclave compute for de-identification.

SagaOS ensures deterministic message passing between FHIR objects and regulatory classes (e.g., trigger reimbursement on verified outcomes), with version lineage in the Global Class Registry.

Sample SagaPython Code

Classes adhere to strict SagaPython semantics: @SagaClass for persistence/multi-inheritance, SagaField for validated elements (e.g., enum bindings), @SagaMethod for lifecycle actions. References use str LOIDs; enclave=True for PHI-bearing classes.

```python
from saga import SagaClass, SagaMethod, SagaField, CMIConst


@SagaClass(enclave=True, CMIConst.SPClassObject)
class ClassFHIRPatient:
    """FHIR R5 Patient resource with PHI enclave protection."""
    patient_id = SagaField("str")  # FHIR logical id
    status = SagaField("str", enum={"active", "inactive", "entered-in-error"})  # Patient Status Codes binding
    identifier = SagaField("list[dict]")  # Business identifiers
    medication_refs = SagaField("list[str]")  # LOIDs to ClassFHIRMedication
    adverse_event_refs = SagaField("list[str]")  # LOIDs to ClassFHIRAdverseEvent


    @SagaMethod()
```

```
    def __init__(self):
        self._cmi().__init__()
        self.identifier = []
        self.medication_refs = []
        self.adverse_event_refs = []
        self._enclave = {}  # PHI storage

    @SagaMethod()
    def enclaveSet_demographics(self, key: str, value: str):
        self._enclave[key] = value  # e.g., name, birthDate
        return self._enclave[key]

    @SagaMethod()
    def link_medication(self, med_loid: str):
        self.medication_refs.append(med_loid)
        return {"patient_id": self.patient_id, "linked_count": len(self.medication_refs)}

@SagaClass(CMIConst.SPClassObject)  # Multi-inherit from ClassIDMPProduct mixin if defined earlier
class ClassFHIRMedication:
    """FHIR R5 Medication resource with pharma bindings."""
    medication_id = SagaField("str")
    status = SagaField("str", enum={"active", "inactive", "entered-in-error"})
    code = SagaField("dict")  # CodeableConcept: {"coding": [{"system": "http://hl7.org/fhir/sid/rxnorm", "code": "12345"}]}
    dose_form = SagaField("dict")  # CodeableConcept binding to SNOMED CT Form Codes
    total_volume = SagaField("dict")  # Quantity: {"value": 100, "unit": "mL"}
    ingredient = SagaField("list[dict]")  # BackboneElement: item (CodeableReference), isActive, strength
    batch = SagaField("dict")  # BackboneElement: lotNumber, expirationDate
    marketing_authorization_holder = SagaField("str", default="")  # LOID to Organization

    @SagaMethod()
    def __init__(self):
        self._cmi().__init__()
        self.ingredient = []
        self.batch = {"lotNumber": "", "expirationDate": ""}

    @SagaMethod()
    def add_ingredient(self, item: dict, is_active: bool, strength: dict):
        self.ingredient.append({"item": item, "isActive": is_active, "strength": strength})  # strength as Ratio/CodeableConcept/Quantity
```

```
        return {"medication_id": self.medication_id, "ingredient_count": len(self.ingredient)}

    @SagaMethod()
    def set_batch(self, lot_number: str, expiration_date: str):
        self.batch["lotNumber"] = lot_number
        self.batch["expirationDate"] = expiration_date
        return {"batch": self.batch}


@SagaClass(CMIConst.SPClassObject)
class ClassFHIRAdverseEvent:
    """FHIR R5 AdverseEvent resource linked to pharma safety."""
    event_id = SagaField("str")
    status = SagaField("str", enum={"completed", "in-progress", "entered-in-error"})
    category = SagaField("dict") # CodeableConcept
    seriousness = SagaField("dict") # CodeableConcept binding to seriousness codes
    suspected_medication = SagaField("str") # LOID to ClassFHIRMedication
    outcome = SagaField("dict") # CodeableConcept
    recorded_date = SagaField("datetime")

    @SagaMethod()
    def __init__(self):
        self._cmi().__init__()

    @SagaMethod()
    def record_event(self, suspected_loid: str, outcome: dict):
        self.suspected_medication = suspected_loid
        self.outcome = outcome
        self.recorded_date = self._now()
        return {"event_id": self.event_id, "status": self.status}


@SagaClass(CMIConst.SPClassObject)
class ClassFHIRResearchStudy:
    """FHIR R5 ResearchStudy for clinical trials (links to 21 CFR 312)."""
    study_id = SagaField("str")
    status = SagaField("str", enum={"active", "completed", "suspended"})
    title = SagaField("str")
    sponsor = SagaField("str")  # LOID to Organization
    observation_refs = SagaField("list[str]") # LOIDs to ClassFHIRObservation

    @SagaMethod()
    def __init__(self):
```

```
        self._cmi().__init__()
        self.observation_refs = []

    @SagaMethod()
    def add_observation(self, obs_loid: str):
        self.observation_refs.append(obs_loid)
        return {"study_id": self.study_id, "obs_count": len(self.observation_refs)}
```

Interoperability Analysis

- With ISO IDMP & SPL: Medication.code binds CodeableConcept to IDMP identifiers/RxNorm; doseForm aligns with SPL sections via LOID resolution.
- With DSCSA/GS1: Batch.lotNumber links to ClassDSCSAUnit LOIDs; expirationDate enforces serialization expiry checks in @SagaMethod invariants.
- With 21 CFR 312: ResearchStudy.status synchronizes with ClassClinicalInvestigation; observations feed IND close workflows.
- With FAERS/VAERS/WHO VigiBase: AdverseEvent.suspected_medication triggers safety signal feeds; de-identified aggregates via SagaFeeds™.
- With ISO 20022: MedicationRequest outcomes condition payment releases (e.g., value-based reimbursement on verified Observations).
- Private Enclaves: PHI in Patient demographics remains encrypted; proofs (e.g., event counts) exposed for regulatory audits.
- SagaFeeds™: Composable feeds (e.g., "AdverseEvents for IDMP product X in past 30 days") enable real-time pharmacovigilance without PHI leakage.

Implications

- Regulators & Providers: Continuous FHIR-synchronized oversight reduces reporting latency; EHR integrations (HL7 APIs) pull persistent proofs for audits.
- Manufacturers & CROs: Automated linkage of trial data (ResearchStudy) to post-market safety (AdverseEvent) streamlines submissions; R5-compliant classes minimize version reconciliation.
- Payers & Donors: Outcome proofs (Observations) gate milestone payments; equitable access via LMIC-friendly, sharded FHIR objects.
- Patients: De-identified transparency into medication lineage (code → batch → adverse signals) via GS1 Digital Link scans.
- Global Health: FHIR R5 as executable ontology under SagaStandards fosters resilient, interoperable clinical ecosystems, elevating data exchange while preserving privacy.

Section 6.6 demonstrates SagaChain's unification of HL7 FHIR R5 into programmable clinical classes, bridging patient outcomes to regulatory and supply chain proofs in a persistent, privacy-preserving fabric governed by SagaStandards.

6.7 Clinical Document Standards Integration (HL7 CDA R2)

6.7.1 Background & Problem Statement

HL7 Clinical Document Architecture (CDA) Release 2 (R2), published in 2005 and foundational to HL7 standards, defines an XML-based markup for clinical documents such as discharge summaries, progress notes, operative reports, and diagnostic imaging reports. CDA documents consist of a header (metadata like author, effectiveTime, confidentialityCode) and a body (structured sections with coded entries for observations, medications, and procedures). In pharmaceuticals, CDA supports templates for medication-related content, including orders, statements, dispenses, and administrations, binding to vocabularies like RxNorm, SNOMED CT, and LOINC.

Challenges in current environments include:

- Legacy Silos: CDA XML files are often static and disconnected from real-time systems, complicating updates to medication sections (e.g., linking to SPL/IDMP changes or DSCSA batches).
- Interoperability Gaps: No native persistence for document versions; conversions to FHIR are manual, leading to data loss in pharma workflows like adverse event reporting.
- Privacy and Auditability: ConfidentialityCode enforcement is inconsistent; sensitive sections (e.g., PHI in recordTarget) risk exposure without enclave protections.
- Scalability Issues: High-volume document exchanges (e.g., millions of discharge summaries annually) strain centralized repositories without sharded, persistent storage.
- Equity Barriers: LMICs rely on CDA for basic document exchange but lack verifiable links to global pharmacovigilance (e.g., VigiBase integration).

This results in delayed clinical handoffs, incomplete pharma safety signals, and inefficient regulatory audits.

Technical Approach (SagaChain Implementation with Private Enclaves)

Under SagaStandards, SagaChain encodes HL7 CDA R2 as multi-inheritable classes in the Global Pharmaceutical Universal Class Tree, transforming transient XML documents into persistent SagaPSA™ objects. The CDA schema maps to SagaField constraints (e.g., code as dict for CE datatype, effectiveTime

as datetime), with LOID references for causal links (e.g., medication sections to ClassFHIRMedication or ClassIDMPProduct). Private enclaves secure PHI-bearing elements (e.g., recordTarget demographics) while attesting document integrity.

Core mappings:

1. ClinicalDocument Root: ClassCDADocument as base, enforcing header/body structure and templateId for pharma extensions (e.g., Pharmacy Templates IG).
2. Header Elements: ClassCDAHeader for metadata (author, custodian, legalAuthenticator).
3. Body and Sections: ClassCDABody and ClassCDASection, supporting structuredBody with entry acts (e.g., medication dispense linked to DSCSA).
4. Pharma-Specific: Mixins for medication sections, inheriting from IDMP for coded entries (e.g., ingredient, doseQuantity).

SagaOS handles deterministic versioning (setId, versionNumber) and message passing (e.g., update body on new observations). The Global Class Registry maintains CDA R2 schema fidelity under HL7 stewardship via SagaStandards .

Sample SagaPython Code

Classes follow strict SagaPython semantics: @SagaClass for persistence and inheritance, SagaField for validated CDA datatypes (e.g., "dict" for CodeableConcept equivalents like CE), @SagaMethod for document lifecycle (e.g., authenticate, version). Enclave=True for PHI; LOIDs for references.

```python

from saga import SagaClass, SagaMethod, SagaField, CMIConst

@SagaClass(enclave=True, CMIConst.SPClassObject)

class ClassCDADocument:

    """HL7 CDA R2 ClinicalDocument root with header and body."""

    document_id = SagaField("str")  # OID or UUID

    realm_code = SagaField("list[dict]") # {"code": "US", "codeSystem": "2.16.840.1.113883.5.301"}

    type_id = SagaField("dict") # {"extension": "POCD_HD000040", "root": "2.16.840.1.113883.1.3"}

    template_id = SagaField("list[dict]") # e.g., Pharmacy Template IDs

    code = SagaField("dict") # CE: {"code": "34133-9", "codeSystem": "2.16.840.1.113883.6.1", "displayName": "Summarization of patient data"}

    title = SagaField("str")

    effective_time = SagaField("datetime")

    confidentiality_code = SagaField("dict") # CE for N, R, V

    language_code = SagaField("str", default="en-US")

    set_id = SagaField("str")
```

```
    version_number = SagaField("int", default=1)
    record_target = SagaField("str")  # LOID to patient ref, enclave-protected
    header_ref = SagaField("str")  # LOID to ClassCDAHeader
    body_ref = SagaField("str")  # LOID to ClassCDABody
    @SagaMethod()
    def __init__(self):
        self._cmi().__init__()
        self.realm_code = []
        self.template_id = []
        self._enclave = {}  # For PHI like recordTarget details

    @SagaMethod()
    def enclaveSet_record_target(self, key: str, value: str):
        self._enclave[key] = value  # e.g., patientRole demographics
        return self._enclave[key]

    @SagaMethod()
    def authenticate(self, authenticator_loid: str):
        self.version_number += 1
        self.effective_time = self._now()
        return {"document_id": self.document_id, "version": self.version_number}

@SagaClass(CMIConst.SPClassObject)
class ClassCDAHeader:
    """CDA Header elements: author, custodian, etc."""
    header_id = SagaField("str")
    author = SagaField("list[str]")  # LOIDs to assignedAuthor
    custodian = SagaField("str")  # LOID to representedCustodianOrganization
    legal_authenticator = SagaField("str")  # LOID to authenticator
    documentation_of = SagaField("list[str]")  # LOIDs to serviceEvent

    @SagaMethod()
    def __init__(self):
        self._cmi().__init__()
        self.author = []
        self.documentation_of = []

    @SagaMethod()
    def add_author(self, author_loid: str):
        self.author.append(author_loid)
        return {"header_id": self.header_id, "author_count": len(self.author)}
```

```
@SagaClass(CMIConst.SPClassObject)
class ClassCDABody:
    """CDA Body with sections (structured or unstructured)."""
    body_id = SagaField("str")
    structured_body = SagaField("bool", default=True)
    sections = SagaField("list[str]")  # LOIDs to ClassCDASection

    @SagaMethod()
    def __init__(self):
        self._cmi().__init__()
        self.sections = []

    @SagaMethod()
    def add_section(self, section_loid: str):
        self.sections.append(section_loid)
        return {"body_id": self.body_id, "section_count": len(self.sections)}

@SagaClass(CMIConst.SPClassObject)  # Multi-inherit from ClassIDMPProduct mixin if defined earlier for meds
class ClassCDASection:
    """CDA Section with coded entries (e.g., medications)."""
    section_id = SagaField("str")
    template_id = SagaField("list[dict]")  # e.g., {"root": "2.16.840.1.113883.10.20.22.2.1.1"} for meds
    code = SagaField("dict")  # CE: {"code": "10160-0", "codeSystem": "2.16.840.1.113883.6.1", "displayName": "History of medication use"}
    title = SagaField("str")
    text = SagaField("str")  # Human-readable narrative
    entry_refs = SagaField("list[str]")  # LOIDs to acts/observations (e.g., medication dispense)

    @SagaMethod()
    def __init__(self):
        self._cmi().__init__()
        self.template_id = []
        self.entry_refs = []

    @SagaMethod()
    def add_entry(self, entry_loid: str):
        self.entry_refs.append(entry_loid)
        return {"section_id": self.section_id, "entry_count": len(self.entry_refs)}
```

Interoperability Analysis

- With HL7 FHIR: CDA sections convert to FHIR Composition/Bundle via LOID mappings (e.g., medication section to MedicationStatement); persistent versioning aids R2-to-R5 transitions.

- With ISO IDMP & SPL: Section.code binds CE to IDMP/RxNorm; text embeds SPL references for narrative consistency.
- With DSCSA/GS1: Entry acts link dispense observations to serialized units (ClassDSCSAUnit LOIDs); batch details in medication entries enforce traceability.
- With 21 CFR 312/FAERS: DocumentationOf serviceEvents tie to ClassClinicalInvestigation; adverse sections feed safety reports with confidentialityCode enforcement.
- With ISO 20022: Document effectiveTime conditions reimbursement proofs (e.g., discharge summary gates payment).
- Private Enclaves: PHI in header (e.g., recordTarget) encrypted; attestations for document integrity exposed for audits.
- SagaFeeds™: Composable feeds (e.g., "CDA sections for IDMP product X with confidentiality N in past 90 days") enable de-identified pharmacovigilance without full XML exposure.

Implications

- Regulators & Auditors: Persistent CDA objects provide immutable audit trails for clinical documents; automated template compliance reduces validation efforts.
- Providers & CROs: Structured sections streamline handoffs (e.g., medication narratives linked to FHIR Observations); pharma templates minimize reconciliation with IDMP/SPL.
- Payers & Donors: Verifiable document proofs (e.g., authenticated discharge) trigger milestone settlements; equitable access via sharded storage for LMIC exchanges.
- Patients: De-identified transparency into document lineage (header → sections → entries) supports informed care without PHI.
- Global Health: CDA R2 as executable ontology under SagaStandards bridges legacy documents to modern FHIR workflows, enhancing resilient pharmacovigilance and equity.

Section 6.7 illustrates SagaChain's embedding of HL7 CDA R2 into persistent clinical document classes, forging verifiable links from legacy XML to pharma proofs in a privacy-secured, standards-governed fabric.

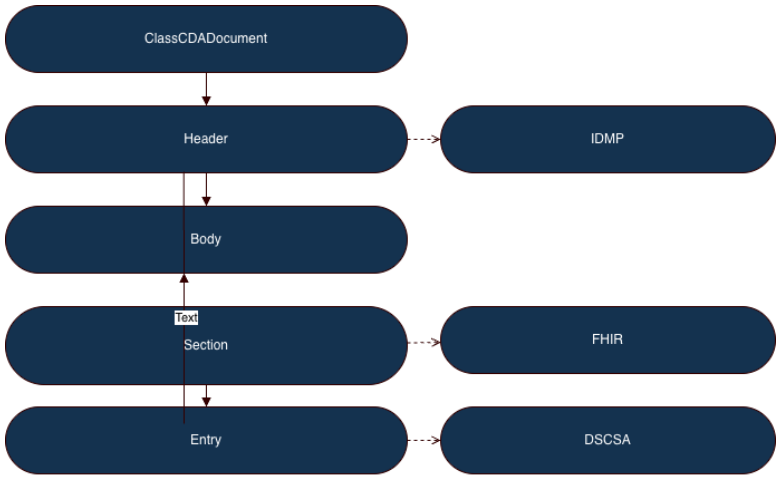


Diagram 6-.7-A — CDA R2 Subtree — Header/Body/Section Hierarchy (Links: IDMP • FHIR • DSCSA)

6.8 Imaging Standards Integration (DICOM PS3.3-2024)

6.8.1 Background & Problem Statement

DICOM (Digital Imaging and Communications in Medicine) PS3.3-2024 defines the information interchange protocol for medical imaging and related workflows, encompassing file formats, network services, and information object definitions (IODs) for modalities like computed tomography (CT), magnetic resonance (MR), ultrasound (US), and structured reporting. In pharmaceuticals, DICOM supports clinical trial imaging endpoints (e.g., tumor response assessment in oncology trials per RECIST criteria), radiotherapy planning, and pharmacovigilance through annotated images. Key elements include datasets with attributes (e.g., PatientID, StudyInstanceUID, Modality), pixel data (compressed via JPEG 2000 or RLE), and unique identifiers (SOPInstanceUID) ensuring traceability.

Challenges in current environments include:

- Data Silos: Imaging datasets are isolated in PACS (Picture Archiving and Communication Systems), disconnected from clinical records (FHIR/CDA) or trial protocols (21 CFR 312), delaying endpoint analysis.
- Privacy Risks: PHI-embedded attributes (e.g., PatientName, BirthDate) complicate secure sharing for multi-site trials without robust de-identification.
- Interoperability Gaps: No persistent linkage between DICOM instances and regulatory objects (e.g., DSCSA batches for radiopharmaceuticals); conversions to web formats (DICOMweb) are lossy.
- Scalability Barriers: Petabyte-scale imaging from global trials overwhelms centralized stores, lacking sharded persistence.
- Equity Issues: LMICs struggle with DICOM-compliant infrastructure,

hindering participation in pharma-sponsored imaging studies.

This fragmentation slows drug development, inflates trial costs, and risks incomplete safety/efficacy proofs.

Technical Approach (SagaChain Implementation with Private Enclaves)

Under SagaStandards , SagaChain encodes DICOM PS3.3-2024 as multi-inheritable classes in the Global Pharmaceutical Universal Class Tree, converting transient datasets into persistent SagaPSA™ objects. DICOM attributes map to SagaField constraints (e.g., UID as str with pattern validation, pixel data as hashed URI for enclave storage), with LOID references for hierarchical linking (Study → Series → Instance). Private enclaves secure PHI (e.g., de-identified PatientID) and pixel data (encrypted blobs), emitting attestations for integrity.

Core mappings:

1. Study/Series/Instance Hierarchy: ClassDICOMStudy, ClassDICOMSeries, ClassDICOMInstance as chained classes, enforcing UID uniqueness and Modality enums.
2. Image IOD: ClassDICOMImage inheriting from Instance, with fields for ImageType and pixel data refs.
3. Structured Reporting: ClassDICOMSR for annotated reports, linking measurements to trial endpoints.
4. Pharma Extensions: Mixins for clinical trial IODs (e.g., RT Plan for radiotherapy), binding to IDMP for radiotracers.

SagaOS enables DICOM services as methods (e.g., Store via message passing, Query/Retrieve via LOID resolution). The Global Class Registry stewards DICOM schema evolution under NEMA/DICOM Committee via SagaStandards.

Sample SagaPython Code

Classes adhere to strict SagaPython semantics: @SagaClass for persistence/multi-inheritance, SagaField for validated DICOM VRs (e.g., "str" for UID, "dict" for Code Sequence), @SagaMethod for services (e.g., store, de_identify). Enclave=True for PHI/pixel data; LOIDs for references.

```python

from saga import SagaClass, SagaMethod, SagaField, CMIConst


@SagaClass(enclave=True, CMIConst.SPClassObject)

class ClassDICOMStudy:

    """DICOM Study IOD root with PHI protection."""

    study_instance_uid = SagaField("str", pattern=r"^1\.2\.840\.\d+\.\d+\.\d+$") # Root UID

    study_id = SagaField("str")

    study_date = SagaField("datetime")

    patient_id = SagaField("str") # De-identified or enclave-protected
```

```
    series_refs = SagaField("list[str]") # LOIDs to ClassDICOMSeries
    modality = SagaField("str", enum={"CT", "MR", "US", "RT", "XA"}) # Partial list

    @SagaMethod()
    def __init__(self):
        self._cmi().__init__()
        self.series_refs = []
        self._enclave = {} # For PHI like full PatientName

    @SagaMethod()
    def enclaveSet_patient_info(self, key: str, value: str):
        self._enclave[key] = value # e.g., PatientName, BirthDate
        return self._enclave[key]

    @SagaMethod()
    def de_identify(self):
        self.patient_id = "DEID_" + self._enclave_hash(self.patient_id) # Mock hash
        self._enclave_attest("deid_proof")
        return {"study_instance_uid": self.study_instance_uid, "deid_status": True}

@SagaClass(CMIConst.SPClassObject)
class ClassDICOMSeries:
    """DICOM Series within Study."""
    series_instance_uid = SagaField("str", pattern=r"^1\.2\.840\.\d+\.\d+\.\d+$")
    series_number = SagaField("int")
    modality = SagaField("str", enum={"CT", "MR", "US", "RT", "XA"})
    instance_refs = SagaField("list[str]") # LOIDs to ClassDICOMInstance
    study_ref = SagaField("str") # LOID to ClassDICOMStudy

    @SagaMethod()
    def __init__(self, study_loid: str):
        self._cmi().__init__()
        self.instance_refs = []
        self.study_ref = study_loid

    @SagaMethod()
    def add_instance(self, instance_loid: str):
        self.instance_refs.append(instance_loid)
        return {"series_instance_uid": self.series_instance_uid, "instance_count": len(self.instance_refs)}

@SagaClass(CMIConst.SPClassObject, ClassDICOMSeries) # Multi-inherit for hierarchy
class ClassDICOMInstance:
    """DICOM SOP Instance (e.g., Image)."""
```

```
    sop_instance_uid = SagaField("str", pattern=r"^1\.2\.840\.\d+\.\d+\.\d+$")
    sop_class_uid = SagaField("str")  # e.g., "1.2.840.10008.5.1.4.1.1.2" for CT
    image_type = SagaField("list[str]", enum_values=["ORIGINAL", "PRIMARY", "AXIAL"])  # Partial CS
    rows = SagaField("int")  # US for dimensions
    columns = SagaField("int")
    pixel_data_ref = SagaField("str")  # Hashed URI to enclave blob
    series_ref = SagaField("str")  # LOID to ClassDICOMSeries

    @SagaMethod()
    def __init__(self, series_loid: str):
        self._cmi().__init__()
        self._cmi('ClassDICOMSeries').__init__(series_loid)  # Multi-init
        self.image_type = []

    @SagaMethod()
    def store_image(self, pixel_uri: str, rows: int, columns: int):
        self.pixel_data_ref = self._enclave_hash(pixel_uri)  # Enclave store
        self.rows = rows
        self.columns = columns
        return {"sop_instance_uid": self.sop_instance_uid, "stored": True}

@SagaClass(CMIConst.SPClassObject)
class ClassDICOMSR:
    """DICOM Structured Reporting for trial annotations."""
    sr_instance_uid = SagaField("str", pattern=r"^1\.2\.840\.\d+\.\d+\.\d+$")
    sop_class_uid = SagaField("str", default="1.2.840.10008.5.1.4.1.1.88.22")  # Basic Text SR
    content_sequence = SagaField("list[dict]")  # SQ: Measurements, e.g., {"ValueType": "NUMERIC", "NumericValue": 5.2}
    instance_ref = SagaField("str")  # LOID to ClassDICOMInstance

    @SagaMethod()
    def __init__(self, instance_loid: str):
        self._cmi().__init__()
        self.content_sequence = []
        self.instance_ref = instance_loid

    @SagaMethod()
    def add_measurement(self, value_type: str, numeric_value: float):
        self.content_sequence.append({"ValueType": value_type, "NumericValue": numeric_value})
```

return {"sr_instance_uid": self.sr_instance_uid, "seq_count": len(self.content_sequence)}

```

## Interoperability Analysis

- With HL7 FHIR: DICOM Study maps to ImagingStudy resource (e.g., StudyInstanceUID to id); Instance pixel_data_ref exposes via WADO-RS LOID queries; SR content feeds Observation.valueQuantity.
- With HL7 CDA: Encapsulate DICOM refs in CDA sections (e.g., ImagingReport template with entry Acts linking SOPInstanceUID); de_identify() aligns with CDA confidentialityCode.
- With ISO IDMP & SPL: Modality-specific IODs (e.g., PET for radiotracers) bind to IDMP codes; ImageType enforces SPL contraindications.
- With DSCSA/GS1: Radiopharmaceutical batches link via StudyDate to serialization expiry in trial workflows.
- With 21 CFR 312: ResearchStudy observations reference DICOM SR for endpoint proofs (e.g., tumor volume measurements).
- Private Enclaves: PHI attributes (PatientID) and pixel_data_ref encrypted; de_identify() emits proofs for trial sharing.
- SagaFeeds™: Composable feeds (e.g., "DICOM Instances for Modality=CT in Study UID X, de-identified") enable avalable trial monitoring without PHI.

## Implications

- Regulators & Sponsors: Persistent DICOM objects automate endpoint validation (e.g., RECIST compliance in oncology trials); real-time Query/Retrieve reduces FDA 21 CFR 312 imaging submission latency.
- CROs & Sites: Hierarchical LOID chaining streamlines multi-site uploads; pharma extensions minimize PACS integration costs.
- Payers & Donors: Verifiable imaging proofs (SR measurements) condition milestone payments for trial progress.
- Patients: De-identified transparency into imaging lineage (Study → Instance → SR) supports informed consent without PHI exposure.
- Global Health: DICOM PS3.3 as executable ontology under SagaStandards empowers LMIC trial participation, harmonizing imaging with FHIR/CDA for equitable drug development.

Section 6.8 demonstrates SagaChain's unification of DICOM PS3.3-2024 into programmable imaging classes, linking radiological proofs to clinical and regulatory workflows in a persistent, privacy-secured fabric governed by SagaStandards.
```

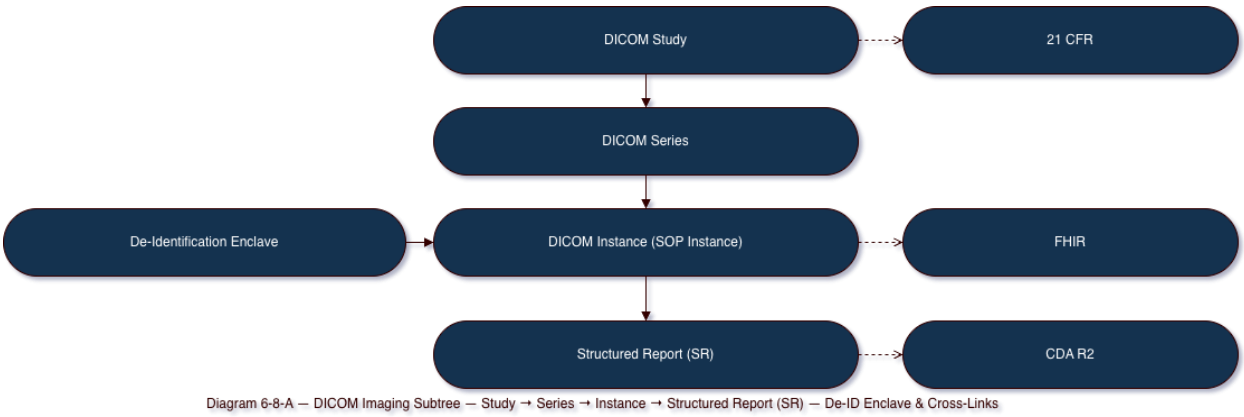


Diagram 6-8-A — DICOM Imaging Subtree — Study → Series → Instance → Structured Report (SR) — De-ID Enclave & Cross-Links

6.9 Workflow Standards Integration (IHE Profiles)

6.9.1 Background & Problem Statement

Integrating the Healthcare Enterprise (IHE) profiles, developed jointly by healthcare professionals and industry since 1998, define precise implementations of established standards (e.g., HL7, DICOM, SOAP) to address specific clinical workflows. In pharmaceuticals, relevant profiles span domains like Pharmacy (e.g., MPD for Medication Prescription and Dispense), Quality/Research/Public Health (QRPH; e.g., CRD for Clinical Research Document, RFD for Retrieve Form for Data Capture), and IT Infrastructure (e.g., XDS for Cross-Enterprise Document Sharing, XDS-I for Imaging). These profiles specify actors (e.g., Document Consumer, Prescription Filler), transactions (e.g., Provide and Register, Fill Prescription), and content modules (e.g., CDA-based documents for trials).

Challenges in current environments include:

- Workflow Fragmentation: IHE transactions (e.g., XDS Register) are point-to-point, lacking persistence across enterprises, which delays clinical trial data capture (RFD) or medication reconciliation in pharma supply (MPD).
- Standard Overlaps: Profiles reuse HL7/CDA/DICOM but require manual orchestration, leading to errors in trial endpoints (CRD) or imaging sharing (XDS-I).
- Privacy and Traceability Gaps: No built-in persistence for audit trails in multi-actor flows; PHI in prescriptions (MPD) risks exposure without enclaves.
- Scalability Issues: High-volume transactions (e.g., millions of RFD queries in global trials) strain federated registries without sharded, canonical storage.
- Equity Barriers: LMICs adopt IHE selectively (e.g., basic XDS), but lack integration with pharma serialization (DSCSA) or global research (WHO VigiBase).

This leads to inefficient trial recruitment, medication errors, and siloed imaging/documents in pharma R&D.

6.9.2 Technical Approach (SagaChain Implementation with Private Enclaves)**

Under SagaStandards , SagaChain encodes IHE profiles as multi-inheritable workflow classes in the Global Pharmaceutical Universal Class Tree, transforming episodic transactions into persistent SagaPSA™ orchestrators. Profiles map to SagaField for actor metadata (e.g., HomeCommunityID in XDS) and @SagaMethod for transactions (e.g., register_document in XDS), with LOID references for actor chaining (e.g., Consumer → Repository). Private enclaves secure sensitive payloads (e.g., prescription details in MPD) while attesting workflow integrity.

Core mappings:

1. XDS Profile: ClassIHExDSRepository as registry for document sharing, inheriting from CDA for content.
2. MPD Profile: ClassIHEMPDPrescription for prescription/dispense workflows, binding to IDMP for coded medications.
3. CRD/RFD Profiles: ClassIHECRDDocument for research docs, with @SagaMethod for form retrieval in trials.
4. XDS-I Extension: ClassIHExDSIImaging linking to DICOM instances for trial imaging.

SagaOS automates actor message passing (e.g., ITI-41 Provide and Register via deterministic bus). The Global Class Registry stewards profile evolution under IHE domains via SagaStandards.

Sample SagaPython Code

Classes adhere to strict SagaPython semantics: @SagaClass for persistence/multi-inheritance, SagaField for validated IHE elements (e.g., "str" for OID, "dict" for Slot), @SagaMethod for transactions (e.g., provide_and_register). Enclave=True for PHI; LOIDs for actor/transaction refs.

```python

from saga import SagaClass, SagaMethod, SagaField, CMIConst


@SagaClass(enclave=True, CMIConst.SPClassObject)

class ClassIHExDSRepository:

    """IHE XDS Repository actor for Cross-Enterprise Document Sharing."""

    repository_unique_id = SagaField("str") # OID

    home_community_id = SagaField("str") # OID for affinity domain

    document_refs = SagaField("list[str]") # LOIDs to ClassCDADocument or ClassIHECRDDocument

    submission_set_uuid = SagaField("str") # UUID for ITI-41 transaction

    slots = SagaField("dict") # e.g., {"creationTime": "2025-11-01T12:00:00Z", "sourceId": "PHARMA_TRIAL"}

    @SagaMethod()
```

```
    def __init__(self):
        self._cmi().__init__()
        self.document_refs = []
        self.slots = {}
        self._enclave = {}  # For document payloads

    @SagaMethod()
    def enclaveSet_document(self, key: str, payload: str):
        self._enclave[key] = payload  # e.g., CDA XML
        return self._enclave[key]

    @SagaMethod()
    def provide_and_register(self, doc_loid: str, slots: dict):
        self.document_refs.append(doc_loid)
        self.slots.update(slots)
        self.submission_set_uuid = self._generate_uuid()
        self._enclave_attest("xds_proof")
        return {"repository_unique_id": self.repository_unique_id, "ack_code": "Success"}

@SagaClass(CMIConst.SPClassObject)  # Multi-inherit from ClassIDMPProduct for meds
class ClassIHEMPDPrescription:
    """IHE MPD Prescription actor for Medication Prescription and Dispense."""
    prescription_id = SagaField("str")  # UUID
    author_organization = SagaField("str")  # LOID to prescriber org
    subject_ref = SagaField("str")  # LOID to ClassFHIRPatient
    medication_code = SagaField("dict")  # CE: {"code": "RxNorm:12345", "display": "Aspirin 81mg"}
    dosage_instruction = SagaField("list[dict]")  # Structured dosage
    dispense_ref = SagaField("str", default="")  # LOID to Dispense

    @SagaMethod()
    def __init__(self):
        self._cmi().__init__()
        self.dosage_instruction = []

    @SagaMethod()
    def fill_prescription(self, dispense_loid: str):
        self.dispense_ref = dispense_loid
        return {"prescription_id": self.prescription_id, "status": "dispensed"}

    @SagaMethod()
    def add_dosage(self, instruction: dict):
```

```
self.dosage_instruction.append(instruction)
        return {"prescription_id": self.prescription_id, "dosage_count": len(self.dosage_instruction)}

@SagaClass(CMIConst.SPClassObject)
class ClassIHECRDDocument:
    """IHE QRPH CRD for Clinical Research Document in trials."""
    crd_id = SagaField("str")  # UUID
    trial_protocol_ref = SagaField("str")  # LOID to ClassFHIRResearchStudy
    form_data = SagaField("dict")  # eCRF fields
    submission_status = SagaField("str", enum={"draft", "submitted", "verified"})
    xds_ref = SagaField("str")  # LOID to XDS document

    @SagaMethod()
    def __init__(self, trial_loid: str):
        self._cmi().__init__()
        self.trial_protocol_ref = trial_loid
        self.form_data = {}

    @SagaMethod()
    def retrieve_form(self, form_key: str):
        # RFD transaction mock
        return {"crd_id": self.crd_id, "form_data": self.form_data.get(form_key, {})}

    @SagaMethod()
    def submit_to_xds(self, xds_loid: str):
        self.xds_ref = xds_loid
        self.submission_status = "submitted"
        return {"crd_id": self.crd_id, "status": self.submission_status}
```

Interoperability Analysis

- With HL7 CDA/FHIR: XDS documents bind CDA content (ClassCDADocument) or FHIR Bundles; MPD medication_code aligns with FHIR Medication.code via RxNorm/IDMP.
- With DICOM/XDS-I: CRD trial forms reference XDS-I imaging (ClassIHExDSIImaging LOID) for endpoint docs; Store Instance transaction chains to XDS Register.
- With ISO IDMP & SPL: Prescription dosage_instruction embeds IDMP strengths; CRD forms validate against SPL sections.
- With DSCSA/GS1: Dispense in MPD links to serialized units (ClassDSCSAUnit LOID) for traceability.
- With 21 CFR 312: RFD form retrieval feeds IND investigations; CRD submissions attest trial compliance.
- Private Enclaves: PHI in form_data or prescription details encrypted;

workflow attestations exposed for audits.

- SagaFeeds™: Composable feeds (e.g., "XDS documents for Trial Protocol Y, submission_status=verified") enable avalable research monitoring without payloads.

Implications

- Regulators & Sponsors: Persistent IHE orchestrators automate trial workflows (RFD/CRD), reducing 21 CFR 312 submission errors; XDS feeds provide real-time document oversight.
- CROs & Sites: Actor chaining streamlines multi-enterprise flows (e.g., MPD dispense across pharmacies); pharma profiles minimize custom integrations.
- Payers & Donors: Verifiable prescription proofs (MPD) condition reimbursements; equitable trial access via sharded XDS for LMICs.
- Patients: De-identified transparency into workflows (e.g., prescription → dispense lineage) supports adherence without PHI.
- Global Health: IHE profiles as executable orchestrators under SagaStandards harmonize standards-based workflows, accelerating pharma research while ensuring privacy and equity.

Section 6.9 demonstrates SagaChain's orchestration of IHE profiles into persistent workflow classes, linking actors and transactions to clinical/regulatory proofs in a canonical, privacy-secured fabric governed by SagaStandards.

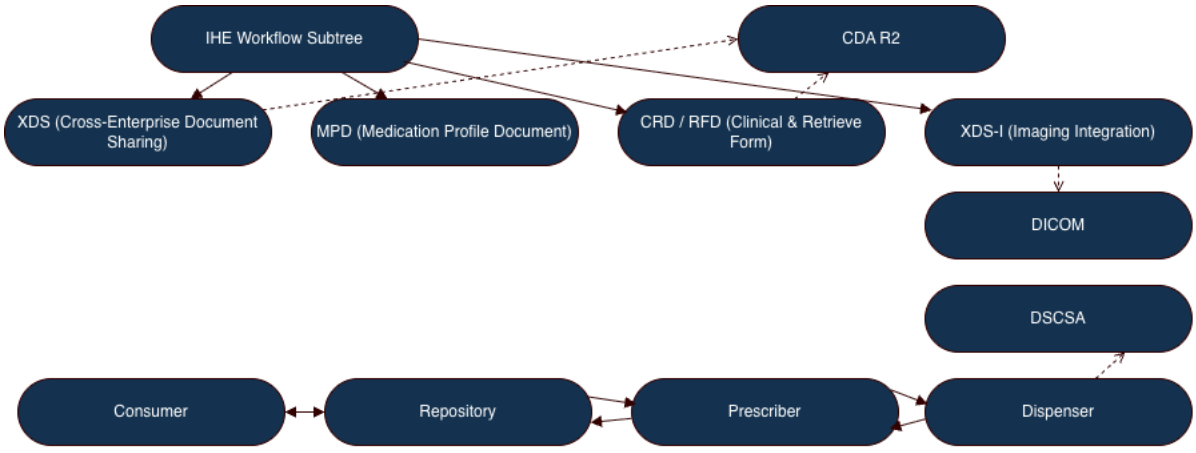


Diagram 6-9-A — IHE Workflow Subtree — XDS, MPD, CRD/RFD, XDS-I — Actor Chain & Cross-Links (CDA, DICOM, DSCSA)

6.10 Clinical Modeling Standards Integration (openEHR Release AM)

6.10.1 Background & Problem Statement

openEHR Release AM (Architecture Model, updated January 2025) provides an open, vendor-neutral specification for electronic health records (EHRs), emphasizing archetype-based data modeling to ensure semantic interoperability. Core components include the Reference Model (RM) for foundational structures like EHR, Composition (clinical documents), Section, and Entry (e.g., Observation for lab results, Evaluation for assessments); Archetypes (reusable constraints on RM classes for domain concepts like "Blood Pressure"); Templates (binding multiple archetypes into operational data sets); and Operational Templates (OPTs) for runtime validation. In pharmaceuticals, openEHR supports archetype-driven modeling for clinical trial endpoints, pharmacovigilance signals (e.g., adverse event archetypes), and patient-centric data (e.g., medication administration archetypes linked to IDMP).

Challenges in current environments include:

- Semantic Drift: Archetypes evolve independently (e.g., via CKM governance), but lack persistent enforcement, leading to inconsistent trial data across sites.
- Interoperability Silos: RM Compositions do not natively link to FHIR resources or CDA documents, complicating pharma workflows like outcome reporting to FAERS.
- Privacy and Versioning Gaps: No canonical persistence for archetype revisions; PHI in Entries risks exposure without enclave controls.
- Scalability Barriers: Modeling millions of clinical observations in global trials overwhelms ad-hoc templates without sharded, multi-inheritable structures.
- Equity Issues: LMICs adopt openEHR archetypes for basic EHRs but struggle with pharma-specific bindings (e.g., to DSCSA serialization).

This results in fragmented clinical models, delayed trial analyses, and unreliable pharmacovigilance.

6.10.2 Technical Approach (SagaChain Implementation with Private Enclaves)

Under SagaStandards, SagaChain encodes openEHR Release AM as multi-inheritable modeling classes in the Global Pharmaceutical Universal Class Tree, transforming evolvable archetypes into persistent SagaPSA™ constraints. RM elements map to SagaField (e.g., archetype_id as str with pattern for ADL paths), with Templates as tree-like hierarchies via LOID references. Private enclaves secure Entry data (e.g., Observation values) while attesting archetype compliance.

Core mappings:

1. EHR Root: ClassOpenEHREHR as base record, containing Compositions.

2. Composition: ClassOpenEHRComposition inheriting from RM, with Sections and Entries.
3. Archetype: ClassOpenEHRArchetype for constraints, binding to domain concepts.
4. Template: ClassOpenEHRTemplate aggregating archetypes for pharma use cases (e.g., trial endpoint template).

SagaOS enables archetype validation as @SagaMethod invariants (e.g., check OPT constraints). The Global Class Registry stewards archetype governance via openEHR Foundation under SagaStandards .

Sample SagaPython Code

Classes adhere to strict SagaPython semantics: @SagaClass for persistence/multi-inheritance, SagaField for validated RM elements (e.g., "dict" for DV types like DV_QUANTITY), @SagaMethod for operations (e.g., validate_archetype). Enclave=True for Entry data; LOIDs for hierarchy.

```python
from saga import SagaClass, SagaMethod, SagaField, CMIConst


@SagaClass(CMIConst.SPClassObject)
class ClassOpenEHREHR:
    """openEHR EHR root record."""
    ehr_id = SagaField("str")  # UUID
    ehr_status = SagaField("dict")  # e.g., {"modifiable": True}
    compositions = SagaField("list[str]")  # LOIDs to ClassOpenEHRComposition
    archetype_catalog = SagaField("dict")  # {"adl_path": "openEHR-EHR-OBSERVATION.blood_pressure.v1"}

    @SagaMethod()
    def __init__(self):
        self._cmi().__init__()
        self.compositions = []
        self.archetype_catalog = {}

    @SagaMethod()
    def add_composition(self, comp_loid: str):
        self.compositions.append(comp_loid)
        return {"ehr_id": self.ehr_id, "comp_count": len(self.compositions)}


@SagaClass(enclave=True, CMIConst.SPClassObject)
class ClassOpenEHRComposition:
    """openEHR Composition (clinical document)."""
    composition_id = SagaField("str")  # UID
    name = SagaField("dict")  # DV_TEXT: {"value": "Adverse Event Report"}
```

```
    archetype_id = SagaField("str", pattern=r"^openEHR-EHR-COMPOSITION\..+\.v\d+$")
    sections = SagaField("list[str]")  # LOIDs to Sections
    entries = SagaField("list[str]")  # LOIDs to Entries (Observation, etc.)
    ehr_ref = SagaField("str")  # LOID to ClassOpenEHREHR

    @SagaMethod()
    def __init__(self, ehr_loid: str):
        self._cmi().__init__()
        self.sections = []
        self.entries = []
        self.ehr_ref = ehr_loid
        self._enclave = {}  # For Entry values

    @SagaMethod()
    def enclaveSet_entry(self, key: str, value: dict):
        self._enclave[key] = value  # e.g., DV_OBSERVATION data
        return self._enclave[key]

@SagaClass(CMIConst.SPClassObject)
class ClassOpenEHRArchetype:
    """openEHR Archetype for domain constraints."""
    archetype_id = SagaField("str", pattern=r"^openEHR-EHR-ENTRY\..+\.v\d+$")  # e.g., blood_pressure.v2
    rm_origin = SagaField("str")  # RM class, e.g., "OBSERVATION"
    definition = SagaField("dict")  # ADL constraints: {"attributes": {"data": {"children": [...]}}}
    template_refs = SagaField("list[str]")  # LOIDs to ClassOpenEHRTemplate

    @SagaMethod()
    def __init__(self):
        self._cmi().__init__()
        self.template_refs = []

    @SagaMethod()
    def bind_template(self, temp_loid: str):
        self.template_refs.append(temp_loid)
        return {"archetype_id": self.archetype_id, "bound_count": len(self.template_refs)}

@SagaClass(CMIConst.SPClassObject)
class ClassOpenEHRTemplate:
    """openEHR Template binding archetypes."""
    template_id = SagaField("str")  # UID
```

```
    archetype_tree = SagaField("dict")  # {"root": "composition_archetype", "children": [{"id": "observation_bp", ...}]}
    opt_uid = SagaField("str")  # Operational Template UID
    pharma_use = SagaField("str", enum={"trial_endpoint", "pv_signal", "med_admin"})  # Pharma extension

    @SagaMethod()
    def __init__(self):
        self._cmi().__init__()
        self.archetype_tree = {}

    @SagaMethod()
    def validate_instance(self, instance_data: dict):
        # Mock OPT validation
        if "constraints_satisfied" in self.archetype_tree:
            return {"template_id": self.template_id, "valid": True}
        return {"template_id": self.template_id, "valid": False}
```

Interoperability Analysis

- With HL7 FHIR: Archetypes map to FHIR Profiles (e.g., Observation archetype to FHIR Observation); Compositions convert to FHIR Composition/Bundle via LOID resolution.
- With HL7 CDA: Template-bound sections align with CDA Sections; Entry DV types (e.g., DV_TEXT) embed in CDA narratives.
- With DICOM/IHE: Trial endpoint archetypes (e.g., imaging measurements) reference DICOM SR content; XDS documents encapsulate openEHR Compositions.
- With ISO IDMP & SPL: Evaluation archetypes bind coded values to IDMP/RxNorm; pharma_use enum enforces SPL dosage constraints.
- With DSCSA/GS1: Medication administration Entries link to serialized units (ClassDSCSAUnit LOID) for traceability.
- With 21 CFR 312: EHR Compositions feed IND trial data; archetype validation attests endpoint compliance.
- Private Enclaves: Entry values (e.g., patient observations) encrypted; archetype proofs exposed for audits.
- SagaFeeds™: Composable feeds (e.g., "Compositions using archetype blood_pressure.v2 in Trial X, de-identified") enable avalable modeling oversight.

Implications

- Regulators & Sponsors: Persistent archetypes automate semantic validation for 21 CFR 312 trials; CKM-like governance via SagaStandards ensures evolvable models.
- CROs & Providers: Template hierarchies streamline data capture (e.g., pv_signal for FAERS); pharma

extensions reduce modeling duplication.

- Payers & Donors: Verifiable EHR proofs (Composition validations) condition outcome-based payments; LMIC-friendly archetypes promote equity.
- Patients: De-identified transparency into modeled data (archetype → Entry lineage) supports personalized care without PHI.
- Global Health: openEHR Release AM as executable modeling ontology under SagaStandards unifies clinical semantics, accelerating pharma innovation with interoperable, archetype-driven records.

Section 6.10 demonstrates SagaChain's embedding of openEHR Release AM into persistent modeling classes, constraining clinical data to pharma proofs in a semantically rich, privacy-secured fabric governed by SagaStandards.

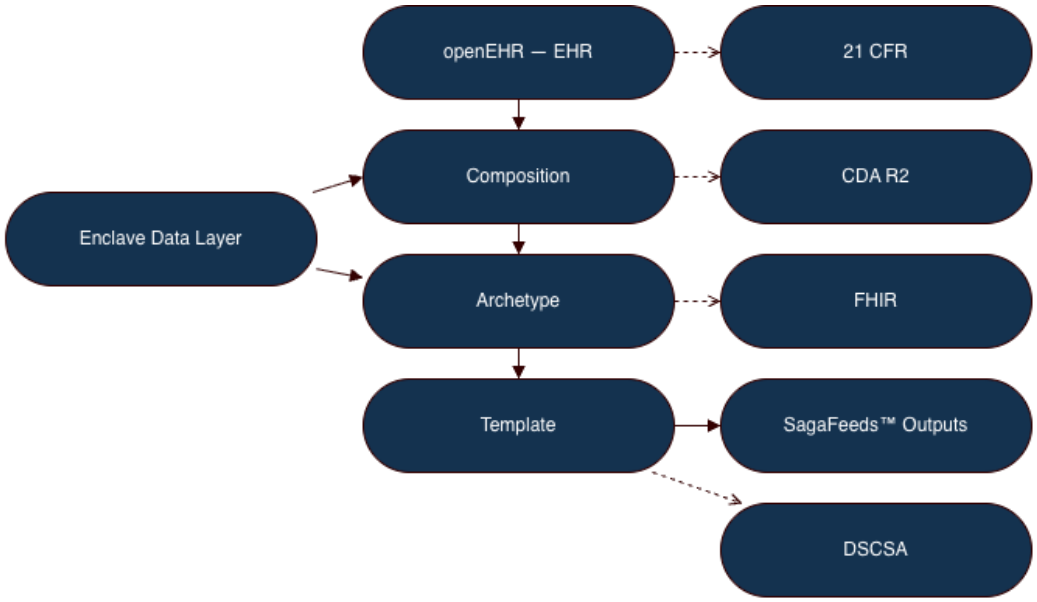


Diagram 6-10-A — openEHR Subtree — EHR → Composition → Archetype → Template — Bindings (FHIR • CDA • DSCSA • 21 CFR), Enclave & SagaFeeds

Section 7. Integration with Global Standards & Regulatory Frameworks

7.1 Financial Standards Integration (ISO 20022, FIX, XBRL, FIGI, IFRS)

7.1.1 Background & Problem Statement

Global pharmaceutical markets are as much financial systems as they are scientific or clinical. The end-to-end lifecycle of therapies — from R&D funding through clinical trial reimbursements, procurement, logistics finance, payer settlements, and post-market risk management — is underpinned by financial flows. However, the financial infrastructure that supports this lifecycle remains fragmented and inefficient:

- Multiple Standards: ISO 20022 governs settlement messaging, FIX protocol handles trade execution, XBRL supports corporate financial disclosure, and FIGI provides instrument identifiers. These exist in silos with limited interoperability.
- Complex Reconciliation: Pharma companies, CROs, payers, and regulators reconcile financial and clinical data manually, creating delays, errors, and fraud exposure.
- Opaque Donor Funding: Global health initiatives (e.g., GAVI, Global Fund) struggle with traceability from donor contributions to patient-level delivery.
- Risk & Compliance Gaps: IFRS and national reporting frameworks are inconsistently applied, reducing investor and regulator confidence.
- Static Systems: Settlement and reimbursement systems lack programmability, preventing outcome-based models tied to real-world clinical data.

This disconnect between financial standards and clinical/regulatory frameworks leads to inefficiencies, fraud risk, and inequitable distribution of resources.

7.1.2 Technical Approach (SagaChain Implementation with Private Enclaves)

Under SagaStandards, SagaChain integrates financial standards into its persistent-state class tree, binding economic flows directly to clinical and regulatory proofs via the Global Pharmaceutical Universal Class Tree:

1. ISO 20022 Settlement Layer: ClassISO20022Payment encodes programmable settlement messages, linked to shipments, trials, or reimbursement proofs. Enclaves enforce escrow conditions (serialization verified, outcomes achieved).
2. FIX Protocol Trade Integration: ClassFIXTrade represents secondary

market activities (e.g., procurement contracts, bulk purchasing), ensuring trade identifiers map to underlying serialized assets (SGTIN, IDMP).

3. XBRL for Corporate & Donor Reporting: ClassXBRLDisclosure objects ensure corporate reports, donor impact statements, and ESG metrics are natively linked to on-chain proofs.
4. FIGI & Instrument Identity: ClassFinancialInstrument binds FIGI identifiers to pharma-linked assets, enabling recognition in global capital markets.
5. IFRS / GAAP Compliance: Financial flows auto-generate auditable disclosure packages. Private Enclaves preserve sensitive commercial terms while exposing compliance proofs.

Classes use multi-inheritance (e.g., from ClassFungibleAsset for payments) and enclave flags for confidentiality.

Sample SagaPython Code

Strict SagaPython: Enclave=True for sensitive fields; type hints in SagaField.

```python
from saga import SagaClass, SagaMethod, SagaField, CMIConst


@SagaClass(enclave=True, CMIConst.SPClassObject)
class ClassISO20022Payment:
    """Programmable ISO 20022 settlement message."""
    payment_id = SagaField("str")
    payer = SagaField("str")  # LOID
    payee = SagaField("str")  # LOID
    amount = SagaField("int")  # Minor units
    currency = SagaField("str", length=3)
    linked_asset = SagaField("str")  # IDMP, shipment, or trial LOID
    escrow_conditions = SagaField("dict")  # {"serialization_verified": True, "outcome_verified": True}
    status = SagaField("str", enum={"pending", "released", "disputed"})
    enclave_proof = SagaField("str")

    @SagaMethod()
    def __init__(self):
        self._cmi().__init__()
        self.escrow_conditions = {}
        self.enclave_proof = ""

    @SagaMethod()
    def release_escrow(self):
        if all(self.escrow_conditions.values()):
            self.status = "released"
            self.enclave_proof = self._enclave_attest()
        return {"status": self.status}


@SagaClass(CMIConst.SPClassObject)
```

```
class ClassFIXTrade:
    """Trade execution linked to pharma assets (procurement, secondary)."""
    trade_id = SagaField("str")
    figi_id = SagaField("str")
    counterparty = SagaField("str")  # LOID
    asset_ref = SagaField("str")  # ClassFinancialInstrument LOID
    trade_terms = SagaField("dict")
    settlement_refs = SagaField("list[str]")

    @SagaMethod()
    def __init__(self):
        self._cmi().__init__()
        self.settlement_refs = []

@SagaClass(CMIConst.SPClassObject)
class ClassXBRLDisclosure:
    """Financial disclosure mapped to XBRL taxonomy."""
    disclosure_id = SagaField("str")
    entity = SagaField("str")  # LOID
    period = SagaField("str")
    taxonomy_refs = SagaField("list[str]")  # IFRS/GAAP concepts
    values = SagaField("dict")
    enclave_proofs = SagaField("list[str]")

    @SagaMethod()
    def __init__(self):
        self._cmi().__init__()
        self.taxonomy_refs = []
        self.enclave_proofs = []
```

Interoperability Analysis

- ISO 20022: Enables programmable, condition-based settlements (e.g., reimbursement released only when DSCSA compliance or HL7 outcomes are verified via LOID binds).
- FIX Protocol: Extends standard trading infrastructure to pharma-linked instruments (e.g., carbon credits for cold-chain, outcome-linked bonds).
- XBRL: Ensures donor funding and corporate ESG disclosures are cryptographically linked to operational proofs.
- FIGI: Harmonizes pharma-linked financial instruments across global capital markets.
- IFRS / GAAP: Automated disclosures ensure compliance with international financial reporting.
- Private Enclaves: Preserve confidentiality of commercial contracts while exposing auditable compliance proofs.
- SagaFeeds™: Publish anonymized financial performance benchmarks, ESG-adjusted rates, and donor impact summaries for transparency.

Implications

- Regulators & Auditors: Gain continuous, verifiable financial flows tied directly to regulated pharma activities.
- Manufacturers & CROs: Reduce reconciliation costs; access programmable, milestone-based finance.
- Donors & Payers: Ensure funds reach intended outcomes; automate value-based reimbursement.
- Investors: Trade pharma-linked financial instruments with cryptographic assurance.
- Patients & Global Health: Improved trust and equity as financing flows become transparent and performance-linked.

Section 7.1 demonstrates how SagaChain unifies global financial standards into a programmable compliance layer, transforming fragmented payments and disclosures into verifiable, outcome-driven infrastructure under SagaStandards governance.

7.2 Supply Chain Standards Integration (GS1, DSCSA, EPCIS, IDMP)

7.2.1 Background & Problem Statement

Supply chain standards like GS1 (EPCIS for event data), DSCSA (U.S. serialization), ISO IDMP (product identification), and SPL (labeling) aim to ensure traceability, authenticity, and interoperability across global pharma logistics. Yet, implementation reveals persistent gaps:

- Format Heterogeneity: GS1 EPCIS events clash with DSCSA XML requirements, forcing costly transformations.
- Identifier Fragmentation: IDMP substance/product codes do not natively resolve to GS1 SGTINs or SPL sections.
- Visibility Silos: EPCIS hubs lack enforceable links to regulatory approvals (e.g., 21 CFR lot releases) or safety signals (FAERS).
- Scalability Barriers: Billions of annual events overwhelm centralized verifiers, especially in LMICs.
- Equity Issues: Donor programs (e.g., COVAX) lack verifiable proofs of cold-chain integrity and equitable distribution.

These disconnects amplify risks of counterfeits, delays, and inequitable access, undermining global health resilience.

7.2.2 Technical Approach (SagaChain Implementation)

SagaChain embeds supply chain standards as multi-inheritable classes in the Universal Class Tree, creating a persistent evidence graph for end-to-end traceability. Under SagaStandards, GS1, ISO, and FDA/EMA collaborate on namespace stewardship:

1. GS1 EPCIS Layer: ClassEPCISEvent as append-only mixins for commission/ship/receive actions.
2. DSCSA Serialization: Extends ClassNonFungibleAsset with verification workflows.
3. ISO IDMP Identification: ClassIDMPProduct as canonical identifiers binding substances, products, and packages.
4. SPL Labeling: ClassSPLDocument synchronizes multilingual labels with serialization proofs.

Enclaves secure sensitive telemetry (e.g., IoT cold-chain data); SagaFeeds™ expose public traceability dashboards.

Sample SagaPython Code

Strict SagaPython: Multi-inheritance for event mixins; LOID binds for IDMP-SGTIN resolution.

```python
from saga import SagaClass, SagaMethod, SagaField, CMIConst


@SagaClass(CMIConst.SPClassObject)
class ClassIDMPProduct:
    """ISO IDMP canonical product identifier."""
    idmp_id = SagaField("str")  # Substance/product/package code
    name = SagaField("str")
    strength = SagaField("str")
    form = SagaField("str")
    sgtin_refs = SagaField("list[str]")  # GS1 links

    @SagaMethod()
    def __init__(self):
        self._cmi().__init__()
        self.sgtin_refs = []

    @SagaMethod()
    def bind_sgtin(self, sgtin: str):
        self.sgtin_refs.append(sgtin)
        return {"idmp_id": self.idmp_id, "bound_count": len(self.sgtin_refs)}


@SagaClass(CMIConst.SPClassObject, ClassDSCSAUnit)  # Multi-inherit from DSCSA
class ClassEPCISEvent:
    """GS1 EPCIS event mixin for supply chain actions."""
    event_id = SagaField("str")
    action = SagaField("str", enum={"commission", "ship", "receive", "aggregate"})
    timestamp = SagaField("datetime")
    biz_step = SagaField("str")
    disposition = SagaField("str")
    unit_refs = SagaField("list[str]")  # SGTIN LOIDs
```

```
    @SagaMethod()
    def __init__(self):
        self._cmi().__init__()

self._cmi('ClassDSCSAUnit').__init__()
        self.unit_refs = []

    @SagaMethod()
    def record_event(self, action: str, units: list):
        self.action = action
        self.unit_refs = units
        self.timestamp = self._now()
        return {"event_id": self.event_id}

@SagaClass(CMIConst.SPClassObject)
class ClassSPLDocument:
    """FDA SPL linked to IDMP and serialization."""
    spl_id = SagaField("str")
    idmp_ref = SagaField("str")  # LOID
    sections = SagaField("dict")  # {"INDICATIONS": "...", ...}
    serialization_proof = SagaField("str")  # Enclave attestation
    version = SagaField("str")

    @SagaMethod()
    def __init__(self):
        self._cmi().__init__()
        self.sections = {}

    @SagaMethod()
    def update_section(self, code: str, text: str):
        self.sections[code] = text
        self.version = self._now().isoformat()
        return {"spl_id": self.spl_id, "version": self.version}
```

Interoperability Analysis

- GS1 EPCIS: Events append immutably; composable with DSCSA verifications via shared LOIDs.
- DSCSA: Serialization units inherit IDMP identifiers, enabling U.S.-global harmonization.
- ISO IDMP: Canonical codes resolve to SPL sections and EPCIS units, enforced in method invariants.
- 21 CFR/DSCSA Links: Lot releases (ClassGMPBatch) gate EPCIS commissions; SPL updates trigger FAERS subscriptions.
- HL7/FAERS: Patient delivery proofs (EPCIS receive) link to outcome reporting.
- ISO 20022: Settlements conditioned on aggregation proofs (e.g., pallet-level verification).
- SagaFeeds™: Dashboards for SGTIN scans, showing IDMP details and cold-chain status without PHI.

Implications

- Manufacturers & Distributors: Automated traceability reduces recall costs; programmable aggregation scales to billions of events.
- Regulators (FDA/EMA/WHO): Real-time EPCIS feeds enable proactive counterfeit detection; IDMP stewardship under SagaStandards.
- LMICs & Donors: Verifiable equity proofs (e.g., GAVI doses delivered) via public feeds, bridging global-north/south divides.
- Patients: GS1 Digital Link scans reveal full lineage (IDMP → SPL → DSCSA), empowering informed choices.
- Global Health: Transforms supply chains from reactive logistics to proactive, standards-enforced resilience.

Section 6.2 illustrates SagaChain's role in harmonizing supply chain standards as executable classes, forging a verifiable continuum from manufacturer to patient under participatory SagaStandards governance.

7.3 Clinical Standards Integration (HL7 FHIR R5)

7.3.1 Background & Problem Statement

HL7 FHIR (Fast Healthcare Interoperability Resources) R5, released in 2023, standardizes the exchange of healthcare data through RESTful APIs and modular resources, enabling seamless integration across EHRs, clinical trials, pharmacovigilance, and regulatory reporting. In the pharmaceutical domain, key resources include Patient (demographics), Medication (product details), MedicationRequest (prescriptions), Observation (lab results), ResearchStudy (clinical trials), and AdverseEvent (safety signals). FHIR promotes uniform data formats, reducing errors in patient care and outcomes reporting.

Challenges in current implementations include:

- Siloed Data: Clinical outcomes (e.g., Observations) do not natively link to supply chain proofs (DSCSA batches) or labeling (SPL/IDMP codes), complicating reimbursement and safety surveillance.
- Privacy Barriers: PHI (Protected Health Information) restricts cross-system sharing, hindering aggregate signal detection for FAERS/VAERS.
- Version Drift: R5 revisions (e.g., enhanced Medication resources with CodeableReference for ingredients) are not enforced consistently, leading to reconciliation failures.
- Scalability Gaps: High-volume events (e.g., billions of Observations annually) overwhelm centralized FHIR servers without persistent, sharded storage.
- Equity Issues: LMICs lack access to FHIR-compliant tools, delaying integration with global pharmacovigilance (WHO VigiBase).

This fragmentation delays value-based care, inflates adverse event reporting latency, and erodes trust in outcome-linked financing.

7.3.2 Technical Approach (SagaChain Implementation with Private Enclaves)

Under SagaStandards, SagaChain embeds HL7 FHIR R5 resources as multi-inheritable classes in the Global Pharmaceutical Universal Class Tree, transforming static JSON/XML exchanges into persistent SagaPSA™ objects. FHIR elements map to SagaField constraints (e.g., code bindings to RxNorm/IDMP), with references resolved as LOIDs for causal linking. Private enclaves encrypt PHI (e.g., Patient demographics) while exposing de-identified proofs (e.g., aggregate Observations for safety signals).

Core mappings:

1. Patient Resource: ClassFHIRPatient as a privacy-gated base for demographics and references.
2. Medication Resource: ClassFHIRMedication inheriting from IDMP mixins, enforcing R5 fields like code (CodeableConcept with RxNorm/UNII), doseForm, and batch (lotNumber/expirationDate).
3. AdverseEvent Resource: ClassFHIRAdverseEvent linking to DSCSA units and FAERS reports via LOIDs.
4. ResearchStudy Resource: ClassFHIRResearchStudy synchronizing with 21 CFR 312 investigations.
5. Observation/MedicationRequest: Mixins for lab results and prescriptions, with enclave compute for de-identification.

SagaOS ensures deterministic message passing between FHIR objects and regulatory classes (e.g., trigger reimbursement on verified outcomes), with version lineage in the Global Class Registry.

Sample SagaPython Code

Classes adhere to strict SagaPython semantics: @SagaClass for persistence/multi-inheritance, SagaField for validated elements (e.g., enum bindings), @SagaMethod for lifecycle actions. References use str LOIDs; enclave=True for PHI-bearing classes.

```python
from saga import SagaClass, SagaMethod, SagaField, CMIConst

@SagaClass(enclave=True, CMIConst.SPClassObject)

class ClassFHIRPatient:

    """FHIR R5 Patient resource with PHI enclave protection."""

    patient_id = SagaField("str")  # FHIR logical id

    status = SagaField("str", enum={"active", "inactive", "entered-in-error"}) # Medication Status Codes binding

    identifier = SagaField("list[dict]")  # Business identifiers

    medication_refs = SagaField("list[str]")  # LOIDs to ClassFHIRMedication
```

```
    adverse_event_refs = SagaField("list[str]") # LOIDs to ClassFHIRAdverseEvent

    @SagaMethod()
    def __init__(self):
        self._cmi().__init__()
        self.identifier = []
        self.medication_refs = []
        self.adverse_event_refs = []
        self._enclave = {}  # PHI storage

    @SagaMethod()
    def enclaveSet_demographics(self, key: str, value: str):
        self._enclave[key] = value  # e.g., name, birthDate
        return self._enclave[key]

    @SagaMethod()
    def link_medication(self, med_loid: str):
        self.medication_refs.append(med_loid)
        return {"patient_id": self.patient_id, "linked_count": len(self.medication_refs)}

@SagaClass(CMIConst.SPClassObject) # Multi-inherit from ClassIDMPProduct mixin if defined earlier
class ClassFHIRMedication:
    """FHIR R5 Medication resource with pharma bindings."""
    medication_id = SagaField("str")
    status = SagaField("str", enum={"active", "inactive", "entered-in-error"})
    code = SagaField("dict") # CodeableConcept: {"coding": [{"system": "http://hl7.org/fhir/sid/rxnorm", "code": "12345"}]}
    dose_form = SagaField("dict") # CodeableConcept binding to SNOMED CT Form Codes
    total_volume = SagaField("dict") # Quantity: {"value": 100, "unit": "mL"}
    ingredient = SagaField("list[dict]") # BackboneElement: item (CodeableReference), isActive, strength
    batch = SagaField("dict") # BackboneElement: lotNumber, expirationDate
    marketing_authorization_holder = SagaField("str", default="") # LOID to Organization

    @SagaMethod()
    def __init__(self):
        self._cmi().__init__()
        self.ingredient = []
        self.batch = {"lotNumber": "", "expirationDate": ""}

    @SagaMethod()
```

```
    def add_ingredient(self, item: dict, is_active: bool, strength: dict):
        self.ingredient.append({"item": item, "isActive": is_active, "strength": strength}) # strength as Ratio/CodeableConcept/Quantity
        return {"medication_id": self.medication_id, "ingredient_count": len(self.ingredient)}

    @SagaMethod()
    def set_batch(self, lot_number: str, expiration_date: str):
        self.batch["lotNumber"] = lot_number
        self.batch["expirationDate"] = expiration_date
        return {"batch": self.batch}

@SagaClass(CMIConst.SPClassObject)
class ClassFHIRAdverseEvent:
    """FHIR R5 AdverseEvent resource linked to pharma safety."""
    event_id = SagaField("str")
    status = SagaField("str", enum={"completed", "in-progress", "entered-in-error"})
    category = SagaField("dict") # CodeableConcept
    seriousness = SagaField("dict") # CodeableConcept binding to seriousness codes
    suspected_medication = SagaField("str") # LOID to ClassFHIRMedication
    outcome = SagaField("dict") # CodeableConcept
    recorded_date = SagaField("datetime")

    @SagaMethod()
    def __init__(self):
        self._cmi().__init__()

    @SagaMethod()
    def record_event(self, suspected_loid: str, outcome: dict):
        self.suspected_medication = suspected_loid
        self.outcome = outcome
        self.recorded_date = self._now()
        return {"event_id": self.event_id, "status": self.status}

@SagaClass(CMIConst.SPClassObject)
class ClassFHIRResearchStudy:
    """FHIR R5 ResearchStudy for clinical trials (links to 21 CFR 312)."""
    study_id = SagaField("str")
    status = SagaField("str", enum={"active", "completed", "suspended"})
    title = SagaField("str")
    sponsor = SagaField("str") # LOID to Organization
```

```
    observation_refs = SagaField("list[str]") # LOIDs to ClassFHIRObservation


    @SagaMethod()
    def __init__(self):
        self._cmi().__init__()
        self.observation_refs = []


    @SagaMethod()
    def add_observation(self, obs_loid: str):
        self.observation_refs.append(obs_loid)
        return {"study_id": self.study_id, "obs_count": len(self.observation_refs)}
```

Interoperability Analysis

- With ISO IDMP & SPL: Medication.code binds CodeableConcept to IDMP identifiers/RxNorm; doseForm aligns with SPL sections via LOID resolution.
- With DSCSA/GS1: Batch.lotNumber links to ClassDSCSAUnit LOIDs; expirationDate enforces serialization expiry checks in @SagaMethod invariants.
- With 21 CFR 312: ResearchStudy.status synchronizes with ClassClinicalInvestigation; observations feed IND close workflows.
- With FAERS/VAERS/WHO VigiBase: AdverseEvent.suspected_medication triggers safety signal feeds; de-identified aggregates via SagaFeeds™.
- With ISO 20022: MedicationRequest outcomes condition payment releases (e.g., value-based reimbursement on verified Observations).
- Private Enclaves: PHI in Patient demographics remains encrypted; proofs (e.g., event counts) exposed for regulatory audits.
- SagaFeeds™: Composable feeds (e.g., "AdverseEvents for IDMP product X in past 30 days") enable real-time pharmacovigilance without PHI leakage.

Implications

- Regulators & Providers: Continuous FHIR-synchronized oversight reduces reporting latency; EHR integrations (HL7 APIs) pull persistent proofs for audits.
- Manufacturers & CROs: Automated linkage of trial data (ResearchStudy) to post-market safety (AdverseEvent) streamlines submissions; R5-compliant classes minimize version reconciliation.
- Payers & Donors: Outcome proofs (Observations) gate milestone payments; equitable access via LMIC-friendly, sharded FHIR objects.
- Patients: De-identified transparency into medication lineage (code → batch → adverse signals) via GS1 Digital Link scans.
- Global Health: FHIR R5 as executable ontology under

SagaStandards fosters resilient, interoperable clinical ecosystems, elevating data exchange while preserving privacy.

Section 6.3 demonstrates SagaChain's unification of HL7 FHIR R5 into programmable clinical classes, bridging patient outcomes to regulatory and supply chain proofs in a persistent, privacy-preserving fabric governed by SagaStandards.

7.4 Clinical Document Standards Integration (HL7 CDA R2)

7.4.1 Background & Problem Statement

HL7 Clinical Document Architecture (CDA) Release 2 (R2), published in 2005 and foundational to HL7 standards, defines an XML-based markup for clinical documents such as discharge summaries, progress notes, operative reports, and diagnostic imaging reports. CDA documents consist of a header (metadata like author, effectiveTime, confidentialityCode) and a body (structured sections with coded entries for observations, medications, and procedures). In pharmaceuticals, CDA supports templates for medication-related content, including orders, statements, dispenses, and administrations, binding to vocabularies like RxNorm, SNOMED CT, and LOINC.

Challenges in current environments include:

- Legacy Silos: CDA XML files are often static and disconnected from real-time systems, complicating updates to medication sections (e.g., linking to SPL/IDMP changes or DSCSA batches).
- Interoperability Gaps: No native persistence for document versions; conversions to FHIR are manual, leading to data loss in pharma workflows like adverse event reporting.
- Privacy and Auditability: ConfidentialityCode enforcement is inconsistent; sensitive sections (e.g., PHI in recordTarget) risk exposure without enclave protections.
- Scalability Issues: High-volume document exchanges (e.g., millions of discharge summaries annually) strain centralized repositories without sharded, persistent storage.
- Equity Barriers: LMICs rely on CDA for basic document exchange but lack verifiable links to global pharmacovigilance (e.g., VigiBase integration).

This results in delayed clinical handoffs, incomplete pharma safety signals, and inefficient regulatory audits.

7.4.2 Technical Approach (SagaChain Implementation with Private Enclaves)

Under SagaStandards, SagaChain encodes HL7 CDA R2 as multi-inheritable classes in the Global Pharmaceutical Universal Class

Tree, transforming transient XML documents into persistent SagaPSA™ objects. The CDA schema maps to SagaField constraints (e.g., code as dict for CE datatype, effectiveTime as datetime), with LOID references for causal links (e.g., medication sections to ClassFHIRMedication or ClassIDMPProduct). Private enclaves secure PHI-bearing elements (e.g., recordTarget demographics) while attesting document integrity.

Core mappings:

1. ClinicalDocument Root: ClassCDADocument as base, enforcing header/body structure and templateId for pharma extensions (e.g., Pharmacy Templates IG).
2. Header Elements: ClassCDAHeader for metadata (author, custodian, legalAuthenticator).
3. Body and Sections: ClassCDABody and ClassCDASection, supporting structuredBody with entry acts (e.g., medication dispense linked to DSCSA).
4. Pharma-Specific: Mixins for medication sections, inheriting from IDMP for coded entries (e.g., ingredient, doseQuantity).

SagaOS handles deterministic versioning (setId, versionNumber) and message passing (e.g., update body on new observations). The Global Class Registry maintains CDA R2 schema fidelity under HL7 stewardship via SagaStandards.

Sample SagaPython Code

Classes follow strict SagaPython semantics: @SagaClass for persistence and inheritance, SagaField for validated CDA datatypes (e.g., "dict" for CodeableConcept equivalents like CE), @SagaMethod for document lifecycle (e.g., authenticate, version). Enclave=True for PHI; LOIDs for references.

```python
from saga import SagaClass, SagaMethod, SagaField, CMIConst


@SagaClass(enclave=True, CMIConst.SPClassObject)
class ClassCDADocument:
    """HL7 CDA R2 ClinicalDocument root with header and body."""
    document_id = SagaField("str")  # OID or UUID
    realm_code = SagaField("list[dict]") # {"code": "US", "codeSystem": "2.16.840.1.113883.5.301"}
    type_id = SagaField("dict") # {"extension": "POCD_HD000040", "root": "2.16.840.1.113883.1.3"}
    template_id = SagaField("list[dict]") # e.g., Pharmacy Template IDs
    code = SagaField("dict") # CE: {"code": "34133-9", "codeSystem": "2.16.840.1.113883.6.1", "displayName": "Summarization of patient data"}
    title = SagaField("str")
    effective_time = SagaField("datetime")
    confidentiality_code = SagaField("dict") # CE for N, R, V
```

```
    language_code = SagaField("str", default="en-US")
    set_id = SagaField("str")
    version_number = SagaField("int", default=1)
    record_target = SagaField("str")  # LOID to patient ref, enclave-protected
    header_ref = SagaField("str")  # LOID to ClassCDAHeader
    body_ref = SagaField("str")  # LOID to ClassCDABody

    @SagaMethod()
    def __init__(self):
        self._cmi().__init__()
        self.realm_code = []
        self.template_id = []
        self._enclave = {}  # For PHI like recordTarget details

    @SagaMethod()
    def enclaveSet_record_target(self, key: str, value: str):
        self._enclave[key] = value  # e.g., patientRole demographics
        return self._enclave[key]

    @SagaMethod()
    def authenticate(self, authenticator_loid: str):
        self.version_number += 1
        self.effective_time = self._now()
        return {"document_id": self.document_id, "version": self.version_number}

@SagaClass(CMIConst.SPClassObject)
class ClassCDAHeader:
    """CDA Header elements: author, custodian, etc."""
    header_id = SagaField("str")
    author = SagaField("list[str]")  # LOIDs to assignedAuthor
    custodian = SagaField("str")  # LOID to representedCustodianOrganization
    legal_authenticator = SagaField("str")  # LOID to authenticator
    documentation_of = SagaField("list[str]")  # LOIDs to serviceEvent

    @SagaMethod()
    def __init__(self):
        self._cmi().__init__()
        self.author = []
        self.documentation_of = []

    @SagaMethod()
    def add_author(self, author_loid: str):
        self.author.append(author_loid)
```

```
        return {"header_id": self.header_id, "author_count": len(self.author)}


@SagaClass(CMIConst.SPClassObject)
class ClassCDABody:
    """CDA Body with sections (structured or unstructured)."""
    body_id = SagaField("str")
    structured_body = SagaField("bool", default=True)
    sections = SagaField("list[str]")  # LOIDs to ClassCDASection

    @SagaMethod()
    def __init__(self):
        self._cmi().__init__()
        self.sections = []

    @SagaMethod()
    def add_section(self, section_loid: str):
        self.sections.append(section_loid)
        return {"body_id": self.body_id, "section_count": len(self.sections)}


@SagaClass(CMIConst.SPClassObject)  # Multi-inherit from ClassIDMPProduct mixin if defined earlier for meds
class ClassCDASection:
    """CDA Section with coded entries (e.g., medications)."""
    section_id = SagaField("str")
    template_id = SagaField("list[dict]")  # e.g., {"root": "2.16.840.1.113883.10.20.22.2.1.1"} for meds
    code = SagaField("dict")  # CE: {"code": "10160-0", "codeSystem": "2.16.840.1.113883.6.1", "displayName": "History of medication use"}
    title = SagaField("str")
    text = SagaField("str")  # Human-readable narrative
    entry_refs = SagaField("list[str]")  # LOIDs to acts/observations (e.g., medication dispense)

    @SagaMethod()
    def __init__(self):
        self._cmi().__init__()
        self.template_id = []
        self.entry_refs = []

    @SagaMethod()
    def add_entry(self, entry_loid: str):
        self.entry_refs.append(entry_loid)
        return {"section_id": self.section_id, "entry_count": len(self.entry_refs)}
```

Interoperability Analysis

- With HL7 FHIR: CDA sections convert to FHIR Composition/Bundle

via LOID mappings (e.g., medication section to MedicationStatement); persistent versioning aids R2-to-R5 transitions.

- With ISO IDMP & SPL: Section.code binds CE to IDMP/RxNorm; text embeds SPL references for narrative consistency.
- With DSCSA/GS1: Entry acts link dispense observations to serialized units (ClassDSCSAUnit LOIDs); batch details in medication entries enforce traceability.
- With 21 CFR 312/FAERS: DocumentationOf serviceEvents tie to ClassClinicalInvestigation; adverse sections feed safety reports with confidentialityCode enforcement.
- With ISO 20022: Document effectiveTime conditions reimbursement proofs (e.g., discharge summary gates payment).
- Private Enclaves: PHI in header (e.g., recordTarget) encrypted; attestations for document integrity exposed for audits.
- SagaFeeds™: Composable feeds (e.g., "CDA sections for IDMP product X with confidentiality N in past 90 days") enable de-identified pharmacovigilance without full XML exposure.

Implications

- Regulators & Auditors: Persistent CDA objects provide immutable audit trails for clinical documents; automated template compliance reduces validation efforts.
- Providers & CROs: Structured sections streamline handoffs (e.g., medication narratives linked to FHIR Observations); pharma templates minimize reconciliation with IDMP/SPL.
- Payers & Donors: Verifiable document proofs (e.g., authenticated discharge) trigger milestone settlements; equitable access via sharded storage for LMIC exchanges.
- Patients: De-identified transparency into document lineage (header → sections → entries) supports informed care without PHI risks.
- Global Health: CDA R2 as executable ontology under SagaStandards bridges legacy documents to modern FHIR workflows, enhancing resilient pharmacovigilance and equity.

Section 7.4 illustrates SagaChain's embedding of HL7 CDA R2 into persistent clinical document classes, forging verifiable links from legacy XML to pharma proofs in a privacy-secured, standards-governed fabric.

7.5 Imaging Standards Integration (DICOM PS3.3-2024)

7.5.1 Background & Problem Statement

DICOM (Digital Imaging and Communications in Medicine) PS3.3-2024 defines the information interchange protocol for medical imaging and related workflows, encompassing file formats, network services, and information object definitions (IODs) for modalities like computed tomography (CT),

magnetic resonance (MR), ultrasound (US), and structured reporting. In pharmaceuticals, DICOM supports clinical trial imaging endpoints (e.g., tumor response assessment in oncology trials per RECIST criteria), radiotherapy planning, and pharmacovigilance through annotated images. Key elements include datasets with attributes (e.g., PatientID, StudyInstanceUID, Modality), pixel data (compressed via JPEG 2000 or RLE), and unique identifiers (SOPInstanceUID) ensuring traceability.

Challenges in current environments include:

- Data Silos: Imaging datasets are isolated in PACS (Picture Archiving and Communication Systems), disconnected from clinical records (FHIR/CDA) or trial protocols (21 CFR 312), delaying endpoint analysis.
- Privacy Risks: PHI-embedded attributes (e.g., PatientName, BirthDate) complicate secure sharing for multi-site trials without robust de-identification.
- Interoperability Gaps: No persistent linkage between DICOM instances and regulatory objects (e.g., DSCSA batches for radiopharmaceuticals); conversions to web formats (DICOMweb) are lossy.
- Scalability Barriers: Petabyte-scale imaging from global trials overwhelms centralized stores, lacking sharded persistence.
- Equity Issues: LMICs struggle with DICOM-compliant infrastructure, hindering participation in pharma-sponsored imaging studies.

This fragmentation slows drug development, inflates trial costs, and risks incomplete safety/efficacy proofs.

7.5.2 Technical Approach (SagaChain Implementation with Private Enclaves)

Under SagaStandards, SagaChain encodes DICOM PS3.3-2024 as multi-inheritable classes in the Global Pharmaceutical Universal Class Tree, converting transient datasets into persistent SagaPSA™ objects. DICOM attributes map to SagaField constraints (e.g., UID as str with pattern validation, pixel data as hashed URI for enclave storage), with LOID references for hierarchical linking (Study → Series → Instance). Private enclaves secure PHI (e.g., de-identified PatientID) and pixel data (encrypted blobs), emitting attestations for integrity.

Core mappings:

1. Study/Series/Instance Hierarchy: ClassDICOMStudy, ClassDICOMSeries, ClassDICOMInstance as chained classes, enforcing UID uniqueness and Modality enums.
2. Image IOD: ClassDICOMImage inheriting from Instance, with fields for ImageType and pixel data refs.
3. Structured Reporting: ClassDICOMSR for annotated reports, linking measurements to trial endpoints.
4. Pharma Extensions: Mixins for clinical trial IODs (e.g., RT Plan for

radiotherapy), binding to IDMP for radiotracers.

SagaOS enables DICOM services as methods (e.g., Store via message passing, Query/Retrieve via LOID resolution). The Global Class Registry stewards DICOM schema evolution under NEMA/DICOM Committee via SagaStandards.

Sample SagaPython Code

Classes adhere to strict SagaPython semantics: @SagaClass for persistence/multi-inheritance, SagaField for validated DICOM VRs (e.g., "str" for UID, "dict" for Code Sequence), @SagaMethod for services (e.g., store, de_identify). Enclave=True for PHI/pixel data; LOIDs for references.

```python
from saga import SagaClass, SagaMethod, SagaField, CMIConst


@SagaClass(enclave=True, CMIConst.SPClassObject)
class ClassDICOMStudy:
    """DICOM Study IOD root with PHI protection."""
    study_instance_uid = SagaField("str", pattern=r"^1\.2\.840\.\d+\.\d+\.\d+$") # Root UID
    study_id = SagaField("str")
    study_date = SagaField("datetime")
    patient_id = SagaField("str") # De-identified or enclave-protected
    series_refs = SagaField("list[str]") # LOIDs to ClassDICOMSeries
    modality = SagaField("str", enum={"CT", "MR", "US", "RT", "XA"}) # Partial list

    @SagaMethod()
    def __init__(self):
        self._cmi().__init__()
        self.series_refs = []
        self._enclave = {} # For PHI like full PatientName

    @SagaMethod()
    def enclaveSet_patient_info(self, key: str, value: str):
        self._enclave[key] = value # e.g., PatientName, BirthDate
        return self._enclave[key]

    @SagaMethod()
    def de_identify(self):
        self.patient_id = "DEID_" + self._enclave_hash(self.patient_id) # Mock hash
        self._enclave_attest("deid_proof")
        return {"study_instance_uid": self.study_instance_uid, "deid_status": True}

@SagaClass(CMIConst.SPClassObject)
class ClassDICOMSeries:
```

```
    """DICOM Series within Study."""
    series_instance_uid = SagaField("str", pattern=r"^1\.2\.840\.\d+\.\d+\.\d+$")
    series_number = SagaField("int")
    modality = SagaField("str", enum={"CT", "MR", "US", "RT", "XA"})
    instance_refs = SagaField("list[str]") # LOIDs to ClassDICOMInstance
    study_ref = SagaField("str") # LOID to ClassDICOMStudy

    @SagaMethod()
    def __init__(self, study_loid: str):
        self._cmi().__init__()
        self.instance_refs = []
        self.study_ref = study_loid

    @SagaMethod()
    def add_instance(self, instance_loid: str):

self.instance_refs.append(instance_loid)
        return {"series_instance_uid": self.series_instance_uid, "instance_count": len(self.instance_refs)}

@SagaClass(CMIConst.SPClassObject, ClassDICOMSeries) # Multi-inherit for hierarchy
class ClassDICOMInstance:
    """DICOM SOP Instance (e.g., Image)."""
    sop_instance_uid = SagaField("str", pattern=r"^1\.2\.840\.\d+\.\d+\.\d+$")
    sop_class_uid = SagaField("str") # e.g., "1.2.840.10008.5.1.4.1.1.2" for CT
    image_type = SagaField("list[str]", enum_values=["ORIGINAL", "PRIMARY", "AXIAL"]) # Partial CS
    rows = SagaField("int") # US for dimensions
    columns = SagaField("int")
    pixel_data_ref = SagaField("str") # Hashed URI to enclave blob
    series_ref = SagaField("str") # LOID to ClassDICOMSeries

    @SagaMethod()
    def __init__(self, series_loid: str):
        self._cmi().__init__()

self._cmi('ClassDICOMSeries').__init__(series_loid) # Multi-init
        self.image_type = []

    @SagaMethod()
    def store_image(self, pixel_uri: str, rows: int, columns: int):
        self.pixel_data_ref = self._enclave_hash(pixel_uri) # Enclave store
        self.rows = rows
        self.columns = columns
```

```
        return {"sop_instance_uid": self.sop_instance_uid, "stored": True}


@SagaClass(CMIConst.SPClassObject)
class ClassDICOMSR:
    """DICOM Structured Reporting for trial annotations."""
    sr_instance_uid = SagaField("str", pattern=r"^1\.2\.840\.\d+\.\d+\.\d+$")
    sop_class_uid = SagaField("str", default="1.2.840.10008.5.1.4.1.1.88.22") # Basic Text SR
    content_sequence = SagaField("list[dict]") # SQ: Measurements, e.g., {"ValueType": "NUMERIC", "NumericValue": 5.2}
    instance_ref = SagaField("str")  # LOID to ClassDICOMInstance

    @SagaMethod()
    def __init__(self, instance_loid: str):
        self._cmi().__init__()
        self.content_sequence = []
        self.instance_ref = instance_loid

    @SagaMethod()
    def add_measurement(self, value_type: str, numeric_value: float):
        self.content_sequence.append({"ValueType": value_type, "NumericValue": numeric_value})
        return {"sr_instance_uid": self.sr_instance_uid, "seq_count": len(self.content_sequence)}
```

Interoperability Analysis

- With HL7 FHIR: DICOM Study maps to ImagingStudy resource (e.g., StudyInstanceUID to id); Instance pixel_data_ref exposes via WADO-RS LOID queries; SR content feeds Observation.valueQuantity.
- With HL7 CDA: Encapsulate DICOM refs in CDA sections (e.g., ImagingReport template with entry Acts linking SOPInstanceUID); de_identify() aligns with CDA confidentialityCode.
- With ISO IDMP & SPL: Modality-specific IODs (e.g., PET for radiotracers) bind to IDMP codes; ImageType enforces SPL contraindications.
- With DSCSA/GS1: Radiopharmaceutical batches link via StudyDate to serialization expiry in trial workflows.
- With 21 CFR 312: ResearchStudy observations reference DICOM SR for endpoint proofs (e.g., tumor volume measurements).
- Private Enclaves: PHI attributes (PatientID) and pixel_data_ref encrypted; de_identify() emits proofs for trial sharing.
- SagaFeeds™: Composable feeds (e.g., "DICOM Instances for Modality=CT in Study UID X, de-identified") trial monitoring without PHI.

Implications

- Regulators & Sponsors: Persistent DICOM objects automate endpoint validation (e.g., RECIST compliance in oncology trials); real-time Query/Retrieve reduces FDA 21 CFR 312 imaging submission latency.
- CROs & Sites: Hierarchical LOID chaining streamlines multi-site uploads; pharma extensions minimize PACS integration costs.
- Payers & Donors: Verifiable imaging proofs (SR measurements) condition milestone payments for trial progress.
- Patients: De-identified transparency into imaging lineage (Study → Instance → SR) supports informed consent without PHI exposure.
- Global Health: DICOM PS3.3 as executable ontology under SagaStandards empowers LMIC trial participation, harmonizing imaging with FHIR/CDA for equitable drug development.

Section 7.5 demonstrates SagaChain's unification of DICOM PS3.3-2024 into programmable imaging classes, linking radiological proofs to clinical and regulatory workflows in a persistent, privacy-secured fabric governed by SagaStandards.

7.6 Workflow Standards Integration (IHE Profiles)

7.6.1 Background & Problem Statement

Integrating the Healthcare Enterprise (IHE) profiles, developed jointly by healthcare professionals and industry since 1998, define precise implementations of established standards (e.g., HL7, DICOM, SOAP) to address specific clinical workflows. In pharmaceuticals, relevant profiles span domains like Pharmacy (e.g., MPD for Medication Prescription and Dispense), Quality/Research/Public Health (QRPH; e.g., CRD for Clinical Research Document, RFD for Retrieve Form for Data Capture), and IT Infrastructure (e.g., XDS for Cross-Enterprise Document Sharing, XDS-I for Imaging). These profiles specify actors (e.g., Document Consumer, Prescription Filler), transactions (e.g., Provide and Register, Fill Prescription), and content modules (e.g., CDA-based documents for trials).

Challenges in current environments include:

- Workflow Fragmentation: IHE transactions (e.g., XDS Register) are point-to-point, lacking persistence across enterprises, which delays clinical trial data capture (RFD) or

medication reconciliation in pharma supply (MPD).

- Standard Overlaps: Profiles reuse HL7/CDA/DICOM but require manual orchestration, leading to errors in trial endpoints (CRD) or imaging sharing (XDS-I).

- Privacy and Traceability Gaps: No built-in persistence for audit trails in multi-actor flows; PHI in prescriptions (MPD) risks exposure without enclaves.

- Scalability Issues: High-volume transactions (e.g., millions of RFD queries in global trials) strain federated registries without sharded, canonical storage.

- Equity Barriers: LMICs adopt IHE selectively (e.g., basic XDS), but lack integration with pharma serialization (DSCSA) or global research (WHO VigiBase).

This leads to inefficient trial recruitment, medication errors, and siloed imaging/documents in pharma R&D.

7.6.2 Technical Approach (SagaChain Implementation with Private Enclaves)

Under SagaStandards, SagaChain encodes IHE profiles as multi-inheritable workflow classes in the Global Pharmaceutical Universal Class Tree, transforming episodic transactions into persistent SagaPSA™ orchestrators. Profiles map to SagaField for actor metadata (e.g., HomeCommunityID in XDS) and @SagaMethod for transactions (e.g., register_document in XDS), with LOID references for actor chaining (e.g., Consumer → Repository). Private enclaves secure sensitive payloads (e.g., prescription details in MPD) while attesting workflow integrity.

Core mappings:

1. XDS Profile: ClassIHExDSRepository as registry for document sharing, inheriting from CDA for content.
2. MPD Profile: ClassIHEMPDPrescription for prescription/dispense workflows, binding to IDMP for coded medications.
3. CRD/RFD Profiles: ClassIHECRDDocument for research docs, with @SagaMethod for form retrieval in trials.
4. XDS-I Extension: ClassIHExDSIImaging linking to DICOM instances for trial imaging.

SagaOS automates actor message passing (e.g., ITI-41 Provide and Register via deterministic bus). The Global Class Registry stewards profile evolution under IHE domains via SagaStandards.

Sample SagaPython Code

Classes adhere to strict SagaPython semantics: @SagaClass for persistence/multi-inheritance, SagaField for validated IHE elements (e.g., "str" for OID, "dict" for Slot), @SagaMethod for transactions (e.g., provide_and_register). Enclave=True for PHI; LOIDs for actor/transaction refs.

```python

from saga import SagaClass, SagaMethod, SagaField, CMIConst
```

```
@SagaClass(enclave=True, CMIConst.SPClassObject)
class ClassIHExDSRepository:
    """IHE XDS Repository actor for Cross-Enterprise Document Sharing."""
    repository_unique_id = SagaField("str") # OID
    home_community_id = SagaField("str") # OID for affinity domain
    document_refs = SagaField("list[str]") # LOIDs to ClassCDADocument or ClassIHECRDDocument
    submission_set_uuid = SagaField("str") # UUID for ITI-41 transaction
    slots = SagaField("dict") # e.g., {"creationTime": "2025-11-01T12:00:00Z", "sourceId": "PHARMA_TRIAL"}

    @SagaMethod()
    def __init__(self):
        self._cmi().__init__()
        self.document_refs = []
        self.slots = {}
        self._enclave = {}  # For document payloads

    @SagaMethod()
    def enclaveSet_document(self, key: str, payload: str):
        self._enclave[key] = payload  # e.g., CDA XML
        return self._enclave[key]

    @SagaMethod()
    def provide_and_register(self, doc_loid: str, slots: dict):
        self.document_refs.append(doc_loid)
        self.slots.update(slots)
        self.submission_set_uuid = self._generate_uuid()
        self._enclave_attest("xds_proof")
        return {"repository_unique_id": self.repository_unique_id, "ack_code": "Success"}

@SagaClass(CMIConst.SPClassObject) # Multi-inherit from ClassIDMPProduct for meds
class ClassIHEMPDPrescription:
    """IHE MPD Prescription actor for Medication Prescription and Dispense."""
    prescription_id = SagaField("str") # UUID
    author_organization = SagaField("str") # LOID to prescriber org
    subject_ref = SagaField("str") # LOID to ClassFHIRPatient
    medication_code = SagaField("dict") # CE: {"code": "RxNorm:12345", "display": "Aspirin 81mg"}
    dosage_instruction = SagaField("list[dict]") # Structured dosage
```

```
    dispense_ref = SagaField("str", default="")  # LOID to Dispense

    @SagaMethod()
    def __init__(self):
        self._cmi().__init__()
        self.dosage_instruction = []

    @SagaMethod()
    def fill_prescription(self, dispense_loid: str):
        self.dispense_ref = dispense_loid
        return {"prescription_id": self.prescription_id, "status": "dispensed"}

    @SagaMethod()
    def add_dosage(self, instruction: dict):
        self.dosage_instruction.append(instruction)
        return {"prescription_id": self.prescription_id, "dosage_count": len(self.dosage_instruction)}

@SagaClass(CMIConst.SPClassObject)
class ClassIHECRDDocument:
    """IHE QRPH CRD for Clinical Research Document in trials."""
    crd_id = SagaField("str")  # UUID
    trial_protocol_ref = SagaField("str")  # LOID to ClassFHIRResearchStudy
    form_data = SagaField("dict")  # eCRF fields
    submission_status = SagaField("str", enum={"draft", "submitted", "verified"})
    xds_ref = SagaField("str")  # LOID to XDS document

    @SagaMethod()
    def __init__(self, trial_loid: str):
        self._cmi().__init__()
        self.trial_protocol_ref = trial_loid
        self.form_data = {}

    @SagaMethod()
    def retrieve_form(self, form_key: str):
        # RFD transaction mock
        return {"crd_id": self.crd_id, "form_data": self.form_data.get(form_key, {})}

    @SagaMethod()
    def submit_to_xds(self, xds_loid: str):
        self.xds_ref = xds_loid
        self.submission_status = "submitted"
        return {"crd_id": self.crd_id, "status": self.submission_status}
```

Interoperability Analysis

- With HL7 CDA/FHIR: XDS documents bind CDA content (ClassCDADocument) or FHIR Bundles; MPD medication_code aligns with FHIR Medication.code via RxNorm/IDMP.
- With DICOM/XDS-I: CRD trial forms reference XDS-I imaging (ClassIHExDSIImaging LOID) for endpoint docs; Store Instance transaction chains to XDS Register.
- With ISO IDMP & SPL: Prescription dosage_instruction embeds IDMP strengths; CRD forms validate against SPL sections.
- With DSCSA/GS1: Dispense in MPD links to serialized units (ClassDSCSAUnit LOID) for traceability.
- With 21 CFR 312: RFD form retrieval feeds IND investigations; CRD submissions attest trial compliance.
- Private Enclaves: PHI in form_data or prescription details encrypted; workflow attestations exposed for audits.
- SagaFeeds™: Composable feeds (e.g., "XDS documents for Trial Protocol Y, submission_status=verified") research monitoring without payloads.

Implications

- Regulators & Sponsors: Persistent IHE orchestrators automate trial workflows (RFD/CRD), reducing 21 CFR 312 submission errors; XDS feeds provide real-time document oversight.
- CROs & Sites: Actor chaining streamlines multi-enterprise flows (e.g., MPD dispense across pharmacies); pharma profiles minimize custom integrations.
- Payers & Donors: Verifiable prescription proofs (MPD) condition reimbursements; equitable trial access via sharded XDS for LMICs.
- Patients: De-identified transparency into workflows (e.g., prescription → dispense lineage) supports adherence without PHI.
- Global Health: IHE profiles as executable orchestrators under SagaStandards harmonize standards-based workflows, accelerating pharma research while ensuring privacy and equity.

Section 7.6 demonstrates SagaChain's orchestration of IHE profiles into persistent workflow classes, linking actors and transactions to clinical/regulatory proofs in a canonical, privacy-secured fabric governed by SagaStandards.

7.7 Clinical Modeling Standards Integration (openEHR Release AM)

7.7.1 Background & Problem Statement

openEHR Release AM (Architecture Model, updated January 2025) provides an open, vendor-neutral specification for electronic health records (EHRs), emphasizing archetype-based data modeling to ensure

semantic interoperability. Core components include the Reference Model (RM) for foundational structures like EHR, Composition (clinical documents), Section, and Entry (e.g., Observation for lab results, Evaluation for assessments); Archetypes (reusable constraints on RM classes for domain concepts like "Blood Pressure"); Templates (binding multiple archetypes into operational data sets); and Operational Templates (OPTs) for runtime validation. In pharmaceuticals, openEHR supports archetype-driven modeling for clinical trial endpoints, pharmacovigilance signals (e.g., adverse event archetypes), and patient-centric data (e.g., medication administration archetypes linked to IDMP).

Challenges in current environments include:

- Semantic Drift: Archetypes evolve independently (e.g., via CKM governance), but lack persistent enforcement, leading to inconsistent trial data across sites.
- Interoperability Silos: RM Compositions do not natively link to FHIR resources or CDA documents, complicating pharma workflows like outcome reporting to FAERS.
- Privacy and Versioning Gaps: No canonical persistence for archetype revisions; PHI in Entries risks exposure without enclave controls.
- Scalability Barriers: Modeling millions of clinical observations in global trials overwhelms ad-hoc templates without sharded, multi-inheritable structures.
- Equity Issues: LMICs adopt openEHR archetypes for basic EHRs but struggle with pharma-specific bindings (e.g., to DSCSA serialization).

This results in fragmented clinical models, delayed trial analyses, and unreliable pharmacovigilance.

7.7.2 Technical Approach (SagaChain Implementation with Private Enclaves)

Under SagaStandards, SagaChain encodes openEHR Release AM as multi-inheritable modeling classes in the Global Pharmaceutical Universal Class Tree, transforming evolvable archetypes into persistent SagaPSA™ constraints. RM elements map to SagaField (e.g., archetype_id as str with pattern for ADL paths), with Templates as tree-like hierarchies via LOID references. Private enclaves secure Entry data (e.g., Observation values) while attesting archetype compliance.

Core mappings:

1. EHR Root: ClassOpenEHREHR as base record, containing Compositions.
2. Composition: ClassOpenEHRComposition inheriting from RM, with Sections and Entries.
3. Archetype: ClassOpenEHRArchetype for constraints, binding to domain concepts.
4. Template: ClassOpenEHRTemplate aggregating archetypes for pharma

use cases (e.g., trial endpoint template).

SagaOS enables archetype validation as @SagaMethod invariants (e.g., check OPT constraints). The Global Class Registry stewards archetype governance via openEHR Foundation under SagaStandards.

Sample SagaPython Code

Classes adhere to strict SagaPython semantics: @SagaClass for persistence/multi-inheritance, SagaField for validated RM elements (e.g., "dict" for DV types like DV_QUANTITY), @SagaMethod for operations (e.g., validate_archetype). Enclave=True for Entry data; LOIDs for hierarchy.

```python
from saga import SagaClass, SagaMethod, SagaField, CMIConst

@SagaClass(CMIConst.SPClassObject)
class ClassOpenEHREHR:
    """openEHR EHR root record."""
    ehr_id = SagaField("str")  # UUID
    ehr_status = SagaField("dict")  # e.g., {"modifiable": True}
    compositions = SagaField("list[str]")  # LOIDs to ClassOpenEHRComposition
    archetype_catalog = SagaField("dict")  # {"adl_path": "openEHR-EHR-OBSERVATION.blood_pressure.v1"}

    @SagaMethod()
    def __init__(self):
        self._cmi().__init__()
        self.compositions = []
        self.archetype_catalog = {}

    @SagaMethod()
    def add_composition(self, comp_loid: str):
        self.compositions.append(comp_loid)
        return {"ehr_id": self.ehr_id, "comp_count": len(self.compositions)}

@SagaClass(enclave=True, CMIConst.SPClassObject)
class ClassOpenEHRComposition:
    """openEHR Composition (clinical document)."""
    composition_id = SagaField("str")  # UID
    name = SagaField("dict")  # DV_TEXT: {"value": "Adverse Event Report"}
    archetype_id = SagaField("str", pattern=r"^openEHR-EHR-COMPOSITION\..+\.v\d+$")
    sections = SagaField("list[str]")  # LOIDs to Sections
    entries = SagaField("list[str]")  # LOIDs to Entries (Observation, etc.)
    ehr_ref = SagaField("str")  # LOID to ClassOpenEHREHR
```

```
    @SagaMethod()
    def __init__(self, ehr_loid: str):
        self._cmi().__init__()
        self.sections = []
        self.entries = []
        self.ehr_ref = ehr_loid
        self._enclave = {}  # For Entry values

    @SagaMethod()
    def enclaveSet_entry(self, key: str, value: dict):
        self._enclave[key] = value  # e.g., DV_OBSERVATION data
        return self._enclave[key]

@SagaClass(CMIConst.SPClassObject)
class ClassOpenEHRArchetype:
    """openEHR Archetype for domain constraints."""
    archetype_id = SagaField("str", pattern=r"^openEHR-EHR-ENTRY\..+\.v\d+$")  # e.g., blood_pressure.v2
    rm_origin = SagaField("str")  # RM class, e.g., "OBSERVATION"
    definition = SagaField("dict")  # ADL constraints: {"attributes": {"data": {"children": [...]}}}
    template_refs = SagaField("list[str]")  # LOIDs to ClassOpenEHRTemplate

    @SagaMethod()
    def __init__(self):
        self._cmi().__init__()
        self.template_refs = []

    @SagaMethod()
    def bind_template(self, temp_loid: str):
        self.template_refs.append(temp_loid)
        return {"archetype_id": self.archetype_id, "bound_count": len(self.template_refs)}

@SagaClass(CMIConst.SPClassObject)
class ClassOpenEHRTemplate:
    """openEHR Template binding archetypes."""
    template_id = SagaField("str")  # UID
    archetype_tree = SagaField("dict")  # {"root": "composition_archetype", "children": [{"id": "observation_bp", ...}]}
    opt_uid = SagaField("str")  # Operational Template UID
    pharma_use = SagaField("str", enum={"trial_endpoint", "pv_signal", "med_admin"})  # Pharma extension

    @SagaMethod()
    def __init__(self):
        self._cmi().__init__()
```

```
        self.archetype_tree = {}

    @SagaMethod()
    def validate_instance(self, instance_data: dict):
        # Mock OPT validation
        if "constraints_satisfied" in self.archetype_tree:
            return {"template_id": self.template_id, "valid": True}
        return {"template_id": self.template_id, "valid": False}
```

Interoperability Analysis

- With HL7 FHIR: Archetypes map to FHIR Profiles (e.g., Observation archetype to FHIR Observation); Compositions convert to FHIR Composition/Bundle via LOID resolution.
- With HL7 CDA: Template-bound sections align with CDA Sections; Entry DV types (e.g., DV_TEXT) embed in CDA narratives.
- With DICOM/IHE: Trial endpoint archetypes (e.g., imaging measurements) reference DICOM SR content; XDS documents encapsulate openEHR Compositions.
- With ISO IDMP & SPL: Evaluation archetypes bind coded values to IDMP/RxNorm; pharma_use enum enforces SPL dosage constraints.
- With DSCSA/GS1: Medication administration Entries link to serialized units (ClassDSCSAUnit LOID) for traceability.
- With 21 CFR 312: EHR Compositions feed IND trial data; archetype validation attests endpoint compliance.
- Private Enclaves: Entry values (e.g., patient observations) encrypted; archetype proofs exposed for audits.
- SagaFeeds™: Composable feeds (e.g., "Compositions using archetype blood_pressure.v2 in Trial X, de-identified") modeling oversight.

Implications

- Regulators & Sponsors: Persistent archetypes automate semantic validation for 21 CFR 312 trials; CKM-like governance via SagaStandards ensures evolvable models.
- CROs & Providers: Template hierarchies streamline data capture (e.g., pv_signal for FAERS); pharma extensions reduce modeling duplication.
- Payers & Donors: Verifiable EHR proofs (Composition validations) condition outcome-based payments; LMIC-friendly archetypes promote equity.
- Patients: De-identified transparency into modeled data (archetype → Entry lineage) supports personalized care without PHI.
- Global Health: openEHR Release AM as executable modeling ontology under SagaStandards unifies clinical

semantics, accelerating pharma innovation with interoperable, archetype-driven records.

Section 7.7 demonstrates SagaChain's embedding of openEHR Release AM into persistent modeling classes, constraining clinical data to pharma proofs in a semantically rich, privacy-secured fabric governed by SagaStandards.

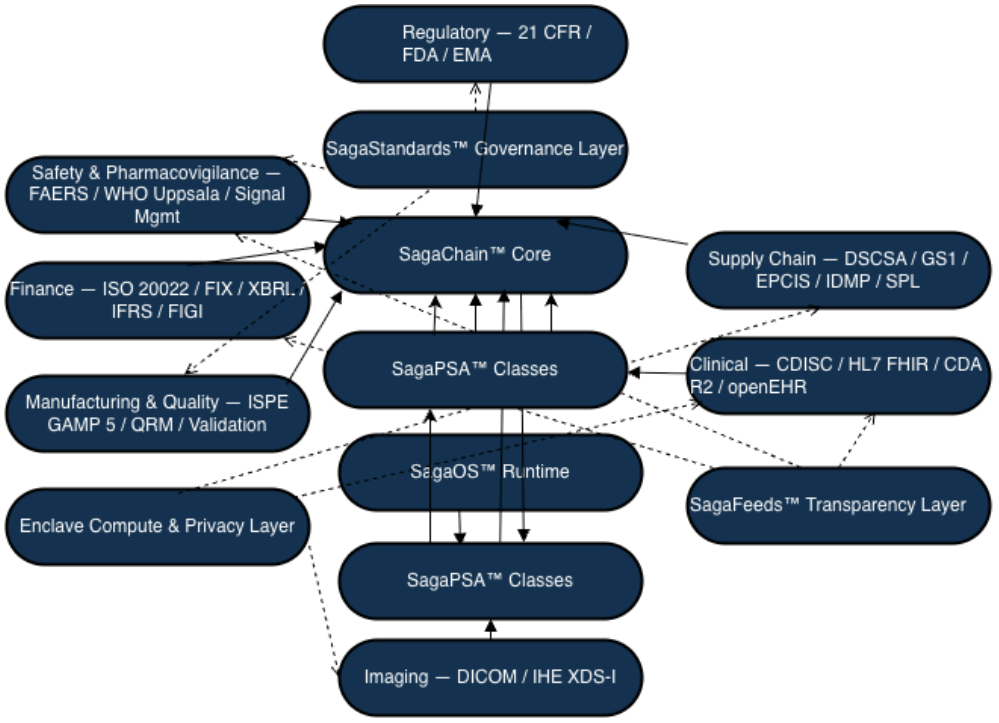


Diagram 6-A — Global Integration Map — Pharma Universe (SagaChain™ Core • SagaStandards™ • Enclaves • SagaFeeds™)

Section 8. Roadmap and Stakeholder Participation

Building the Global Pharmaceutical Universal Class Tree under SagaStandards

8.1 Overview: From Implementation to Institutional Participation

The success of the Global Pharmaceutical Universal Class Tree depends not only on its technical precision but on the breadth of **institutional participation**. To ensure global legitimacy, interoperability, and adoption, the PraSaga Foundation's **SagaStandards Division** serves as the formal coordination body for regulatory, industrial, academic, and public-sector engagement.

SagaStandards functions as a **multi-stakeholder governance and development**

framework, enabling regulators, payers, manufacturers, healthcare providers, standards organizations (SDOs), and civil society representatives to collaboratively extend the Class Tree under open governance. Each contribution—be it a new class definition, regulatory mapping, or dataset schema—passes through transparent review, versioning, and lineage preservation on-chain.

The roadmap is designed to evolve in **three primary phases**:

1. **Alpha (Proving the Model):** Complete foundational class trees, implement end-to-end lifecycle examples, and validate regulatory mappings with pilot partners.
2. **Beta (Institutional Integration):** Transition governance to cross-sector working groups under SagaStandards, harmonize with global SDOs (ISO, HL7, GS1, CDISC, WHO).
3. **Production (Sovereign Alignment):** Formalize governance under multi-jurisdictional custodianship, ensuring legal recognition, auditability, and operational readiness for public-sector adoption.

8.2 Role of SagaStandards in the Governance Model

SagaStandards is the **open governance and standards alignment division** of the PraSaga Foundation. It ensures that every schema, method, and interoperability rule within SagaChain™ meets the principles of:

- **Openness:** All class definitions and metadata are published under CABSL-1.0 (Community-Aligned Business Source License).
- **Interoperability:** Each class tree must align with at least one recognized global standard or regulatory framework (ISO, WHO, FDA, EMA, GS1, etc.).
- **Auditability:** Version lineage, authorship, and compliance rules are automatically logged and accessible via SagaFeeds™.
- **Neutrality:** The framework is politically and commercially neutral—its goal is to serve as a **public digital commons** for verifiable health, regulatory, and financial infrastructure.

Each working group operates within a **modular governance stack**:

- **Root Governance (PraSaga Foundation):** Maintains CABSL-1.0 license, ensures network neutrality and validator oversight.
- **Technical Councils (SagaStandards):** Steward domain-specific class trees—Finance, Pharma, Health, ESG, AI Governance, and others.
- **Regulatory & SDO Liaisons:** Provide validation that each class tree aligns with regional and international legal frameworks.
- **Public Conformance Registry:** All certified versions of classes and methods are published for industry and government reference.

8.3 Collaborative Participation Model

Stakeholders can join SagaStandards under defined roles:

Stakeholder Type	Role in SagaStandards	Primary Benefits
Regulators (FDA, EMA, WHO, MHRA, Swissmedic, etc.)	Submit compliance specifications and validation criteria as executable classes.	Audit automation, real-time policy enforcement, and instant lineage tracking.
SDOs (ISO, GS1, CDISC, HL7, USP, ICH)	Encode existing standards into executable SagaClass™ definitions and maintain version lineage.	Ensure standard conformance in live systems without loss of semantics.
Manufacturers & Pharma Supply Chain Actors	Implement persistent-state compliance workflows (GMP, DSCSA, EPCIS).	Reduced compliance cost, auditable traceability, and faster release cycles.
Healthcare Providers & Hospitals	Integrate FHIR/CDA and EHR interoperability via on-chain verified data streams.	Trusted, cross-border patient data exchange; de-identified analytics.
Payers & Donors	Define programmable reimbursement and funding models (ISO 20022/Equity Assets).	Instant proof of impact, transparent disbursements, and performance-based funding.
Academia & NGOs	Participate in ontology design, dataset alignment, and policy assurance.	Influence global frameworks, ensure ethical and equitable implementation.

8.4 Governance Workflow

Each proposal to the SagaStandards repository follows a **decentralized governance workflow**, ensuring that domain experts and regulators co-develop and approve modifications:

1. **Proposal Submission:** Contributors propose a new `@SagaClass()` or modification via a public pull request to the SagaStandards registry.
2. **Technical Review:** The **Technical Council** evaluates schema integrity, inheritance correctness, and adherence to the SagaPython™ Syntax & Semantics specification.
3. **Regulatory Validation:** The relevant **Regulatory Working Group** (e.g., Pharma, Finance, ESG) verifies that the implementation aligns with governing legal and standards frameworks (e.g., ISO IDMP, DSCSA, FDAAA 801).
4. **Stakeholder Comment Period:** Public review allows peer input and cross-domain feedback (industry, government, academia).

5. **Adoption & Publication:** Once approved, the class definition becomes a permanent, versioned artifact on-chain and is indexed via SagaFeeds™.
   All version histories are immutable and auditable through LOID lineage.
6. **Implementation & Monitoring:** Adopted standards can be instantiated by network participants, with compliance proofs automatically emitted through SagaFeeds dashboards.

8.5 Integration with Standards Development Organizations (SDOs)

SagaStandards is structured to complement, not compete with, existing standards organizations. The initiative works through **technical mappings** and **joint conformance agreements**, ensuring that executable on-chain versions remain true to their published definitions.

Integration channels include:

- **ISO Technical Committees:** Cross-mapping ISO 20022 (Finance), ISO IDMP (Pharma), ISO 11615/11616 (Medicinal Products).
- **GS1 & EPCIS:** Live linkage of serialized identifiers and event telemetry with programmable financial and compliance proofs.
- **HL7 & CDISC:** Direct embedding of healthcare data models (FHIR, CDA, SDTM, ADaM) into persistent SagaPSA™ objects.
- **USP & WHO:** Encoding compendial and international health standards as enforceable, multi-inheritable class trees.
- **OECD & UN SDG Alignment:** Integration of governance and equity metrics within the SagaFeeds™ transparency layer for ESG and global health impact tracking.

8.6 Regulatory and Institutional Custodianship

As the Class Tree matures, SagaStandards envisions **progressive decentralization of custodianship** toward multi-jurisdictional public governance:

- **Early Phase (Foundation Custody):** PraSaga Foundation curates and validates class evolution.
- **Intermediate Phase (Hybrid Oversight):** Joint governance by regulators and SDO representatives ensures trust across borders.
- **Steady-State (DAO Custodianship):** Public DAO under SagaStandards—governed by a cryptographically verifiable vote of member institutions—manages class evolution, deprecation, and extensions.

Each class carries embedded **RegulatoryCustodian()** metadata, enabling automated jurisdictional checks and attribution.

8.7 Roadmap to Adoption

Phase	Timeline	Objectives	Deliverables
Alpha	2025–2026	Core Pharma Class Tree (Finance, Supply Chain, Regulatory, Safety).	Public GitLab repository; validator tests; pilot integrations with GS1/HL7/FDA.
Beta	2026–2027	Establish SagaStandards Working Groups; implement cross-regulatory validation.	Formal Memoranda with WHO, EMA, USP, CDISC, ISO.
Production	2027–2029	Global governance transition to SagaStandards DAO; full regulatory interoperability.	Public audit dashboards (SagaFeeds™); live regulatory adoption by early jurisdictions.

8.8 How Stakeholders Can Participate

- **Regulators** can submit reference schemas or participate in the Regulatory Class Tree Validation Program.
- **Manufacturers** can onboard their GMP and DSCSA datasets into persistent-state compliance workflows.
- **Payers and Donors** can test programmable funding templates (ISO 20022 Milestone/Equity Payments).
- **Researchers and Academia** can define public datasets under CABSL-1.0 to support health equity research.
- **SDOs** can formalize dual publication (standard text + executable schema).

8.9 The Strategic Vision Ahead

SagaStandards transforms how humanity collaborates on shared public infrastructures. Instead of paper-based compliance and opaque data exchanges, **every regulatory standard becomes a living, executable class** that continuously verifies, reports, and harmonizes global health activity.

Through this participatory governance model, regulators, industries, and citizens co-create a universal trust fabric—one that redefines how medicines are discovered, manufactured, distributed, financed, and verified across borders.

SagaStandards is more than a technical consortium; it is a civic infrastructure project for programmable public trust.

8.10 Participation SagaStandards Governance Stack

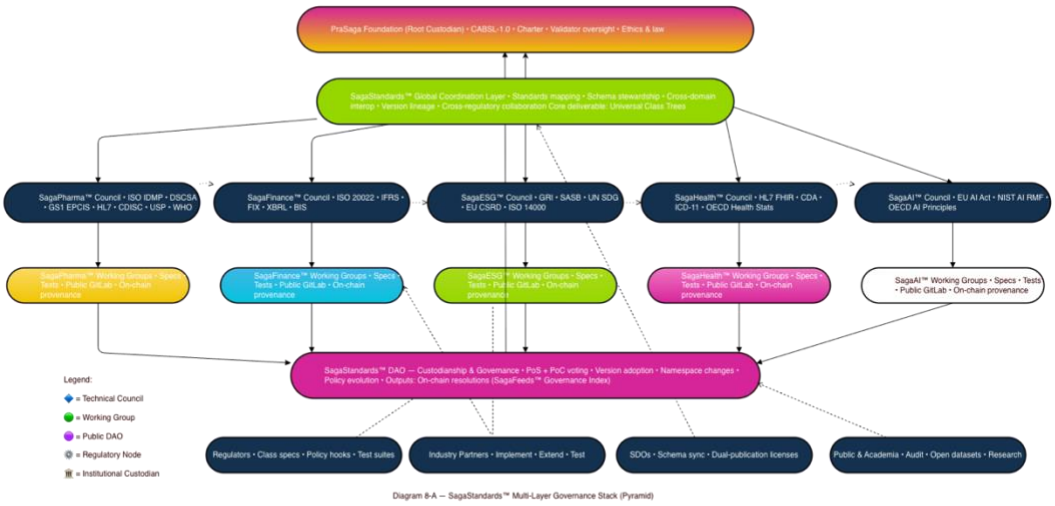


Diagram 8-A — SagaStandards™ Multi-Layer Governance Stack (Pyramid)

This diagram illustrates the participatory governance architecture under the PraSaga Foundation. SagaStandards coordinates open, interoperable class evolution through domain-specific Technical Councils and transparent Working Groups. All approved schemas and governance votes are versioned and auditable on SagaChain™, ensuring regulatory trust, institutional neutrality, and public accountability.

9. Implications for Global Health, Equity, and Innovation

Operationalizing Standards as Code for a Transparent, Equitable Pharmaceutical Ecosystem

9.1 Overview

The deployment of the Global Pharmaceutical Universal Class Tree on **SagaChain™**, under the custodianship of **SagaStandards**, marks a turning point in the governance of global health data and infrastructure.

Where traditional systems rely on fragmented compliance documents, delayed audits, and isolated digital registries, SagaChain enables **continuous, verifiable alignment** between innovation, access, and oversight.

The implications extend far beyond pharmaceutical operations: the framework lays the foundation for a **new global public infrastructure**—one where every dataset, process, and transaction is *programmable, auditable, and equitable* by design.

This section explores the broader societal, institutional, and ethical consequences of deploying SagaChain in the pharmaceutical domain and beyond.

9.2 Redefining Global Health Infrastructure

Traditional health systems operate within jurisdictional boundaries, governed by country-specific regulations and disconnected IT architectures. SagaChain's **persistent-state architecture** bridges these divides by providing a **neutral, standards-aligned substrate** where regulators, payers, manufacturers, and providers interact using the same canonical data models.

Key Transformations

- **From Data Silos to Shared Ontologies:** The Universal Class Tree integrates ISO IDMP, DSCSA, GS1 EPCIS, HL7 FHIR, CDISC, and WHO datasets into one executable object model—eliminating redundancy and improving continuity across the full product lifecycle.
- **From Static Compliance to Continuous Assurance:** Regulatory obligations (FDAAA 801, EMA EudraVigilance, WHO Prequalification) are encoded as persistent classes. Compliance is verified in real time through SagaPython™ decorators and enclave attestations.
- **From Proprietary Infrastructures to Public Commons:** Using the CABSL-1.0 license, SagaStandards ensures that critical schema and ontologies are available as **open digital public goods**, governed collaboratively by institutions and civil society.

By aligning data integrity, security, and interoperability within a single architecture, SagaChain establishes a **digital trust layer** capable of underpinning the entire global health supply chain—from molecule to medicine, from donor to patient.

9.3 Enhancing Global Equity and Access

Equity in health is not merely a moral imperative—it is a measurable, programmable condition. SagaChain transforms global access models by making **allocation, delivery, and affordability verifiable on-chain** through programmable equity classes (e.g., ClassEquityFundAsset, ClassISO20022PandemicSettlement).

Equity-Enabling Mechanisms

- **Programmable Donor Funds:** Donor commitments are executed as ISO 20022 escrowed assets that release only upon verified delivery of serialized medicines (GS1/DSCSA).
- **Transparency Dashboards:** SagaFeeds™ provides real-time public dashboards showing allocation fairness, carbon impact, and health outcomes across LMICs.
- **Treaty Enforcement by Code:** WHO Pandemic Treaty obligations (allocation percentages, response

times) become enforceable through smart, auditable classes.
- **Local Empowerment:** LMIC regulators gain direct access to verifiable manufacturing, labeling, and distribution data without dependence on foreign intermediaries.

Together, these mechanisms close the gap between **policy promise and delivery reality**, ensuring that every financed dose, treatment, or device can be tracked from donor fund to patient outcome—with mathematical proof of compliance.

9.4 Stimulating Innovation and Research Integrity

By uniting data provenance, regulatory lineage, and secure enclave analytics, SagaChain provides an ideal foundation for **research reproducibility and ethical AI in medicine**.

Key Innovations

- **Persistent Evidence Graphs:** Clinical trials, pharmacovigilance events, and real-world evidence (RWE) are linked through immutable Ledger Object IDs (LOIDs), creating machine-verifiable chains of evidence.
- **Privacy-Preserving Research:** Enclave-enabled computation allows researchers to analyze de-identified PHI, genomic data, and device telemetry while publishing only encrypted proofs.
- **AI Transparency:** Models for signal detection, adaptive trial design, and repurposing (ClassAISignalMonitor, ClassRepurposingMonitor) execute in enclaves with attestable reproducibility and explainability.
- **Cross-Domain Synergies:** Developers can extend SagaPython™ class trees to create interoperable AI agents that reason over public health, environmental, and financial data without breaching confidentiality.

This creates a **trustworthy research infrastructure**—a shared global knowledge base where innovation advances within the boundaries of compliance, ethics, and public accountability.

9.5 Regulatory Evolution and Continuous Oversight

SagaChain's multi-inheritable regulatory model eliminates the traditional lag between innovation and oversight. Regulators become **active participants** in the compliance process, not passive auditors of retrospective documentation.

For Regulators

- **Automated Policy Enforcement:** Regulatory rules are embedded as `@SagaMethod()` validations and policy hooks in each domain class.
- **Real-Time Oversight:** Dashboards powered by SagaFeeds™ provide continuous visibility into product status, recalls, trial postings, and safety signals.
- **Reduced Friction:** Instead of redundant filings, institutions maintain persistent compliance objects—automatically auditable by any authorized regulator.
- **Global Harmonization:** Multi-jurisdictional bodies (FDA, EMA, WHO) can co-govern overlapping

namespaces through SagaStandards DAO governance votes.

This ensures that **compliance and innovation advance together**, reducing regulatory backlog while maintaining public safety and trust.

9.6 Economic and Environmental Sustainability

SagaChain supports a **circular, performance-based economic model** that rewards compliance, sustainability, and transparency.
By linking ISO 20022 settlements to GMP quality, ESG metrics, and DSCSA verification, it embeds **accountability into financial flows**.

9.6.1 Sustainability Impacts

- **ESG-Linked Settlements:** Payments can be adjusted automatically based on carbon intensity, temperature excursion rates, or waste reduction performance.
- **Data Efficiency:** Persistent-state architecture removes duplication across systems, reducing data center emissions and operational waste.
- **Circular Incentives:** Stakeholders benefit financially from contributing to compliance proofs and data transparency—turning responsible governance into a measurable asset.

This approach transforms compliance from a **cost center into a performance engine**, incentivizing innovation that benefits both markets and society.

9.7 Ethical and Human-Centered Design

At the core of SagaChain's philosophy is the belief that **trust and dignity must be computationally encoded** into every transaction and dataset.
Through its CABSL-1.0 license and public governance via SagaStandards, the system enforces transparency, consent, and proportional disclosure at every level.

Ethical Safeguards:

- **Data Sovereignty:** Enclaves ensure patient and national control over sensitive datasets.
- **Human Oversight:** Every automated decision (financial, clinical, or regulatory) is linked to a human custodian with review privileges.
- **Public Accountability:** Open dashboards provide citizens with verifiable access to labeling, pricing, and safety outcomes.
- **Equity by Design:** Governance mechanisms explicitly encode equitable access for LMICs and underrepresented populations.

SagaChain thus becomes not only a **technical protocol** but also a **moral infrastructure**—one that encodes fairness, integrity, and collective responsibility into the fabric of global medicine.

9.8 The Path Forward: Global Collaboration Through SagaStandards

The Universal Pharma Class Tree is not an endpoint—it is the **foundation for continuous co-creation**.

Through the open governance model of **SagaStandards**, global stakeholders—regulators, SDOs, manufacturers, and researchers—collaborate to maintain a single, interoperable framework for trusted digital health operations.

Next Steps

1. **Institutional Participation:** Expand regulator and SDO representation within SagaStandards DAO.
2. **Interoperability Pilots:** Run coordinated pilots with WHO, EMA, FDA, and GS1 to validate persistent interoperability across jurisdictions.
3. **Public Education & Capacity Building:** Launch a training program for institutions to author and deploy @SagaClass() modules under CABSL-1.0.
4. **Global Expansion:** Extend the model to adjacent domains—AI Governance, ESG Finance, and Global Public Infrastructure.

9.9 Conclusion

SagaChain™, under the stewardship of SagaStandards, transforms pharmaceutical governance from a fragmented, reactive system into a **continuous, participatory trust network**.
By converting standards into executable, persistent code, it enables a future where **data integrity, equity, and innovation** coexist without compromise.

The promise of SagaStandards is simple yet profound:
Every standard, every law, and every ethical commitment—alive as code, transparent by design, and accessible to all humanity.

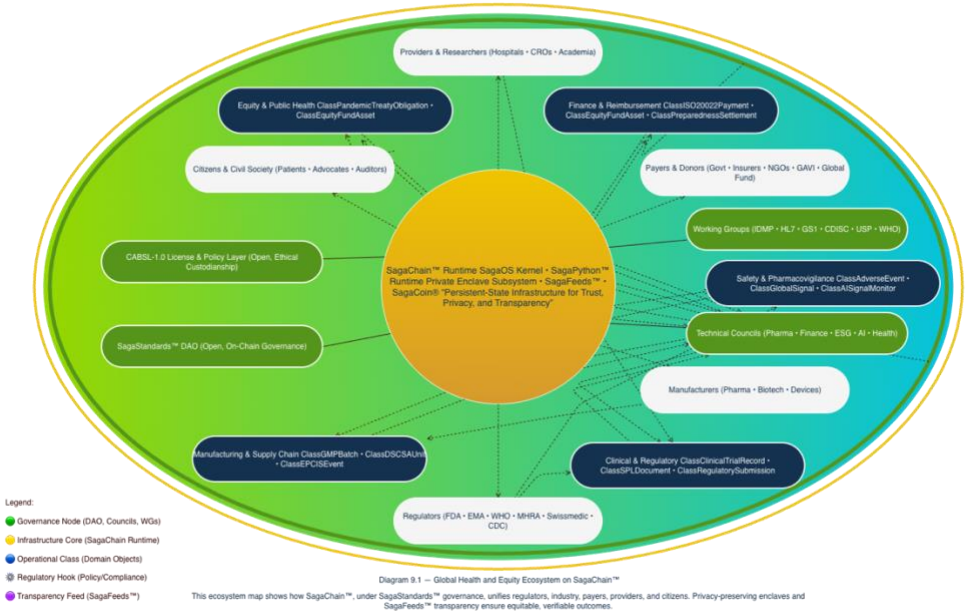


Diagram 9.1 — Global Health and Equity Ecosystem on SagaChain™
This ecosystem map shows how SagaChain™, under SagaStandards™ governance, unifies regulators, industry, payers, providers, and citizens. Privacy-preserving enclaves and SagaFeeds™ transparency ensure equitable, verifiable outcomes.

APPENDICES

APPENDIX A EXTENDED SagaPython™ CODE LISTINGS

The following examples illustrate how pharmaceutical, regulatory, and financial frameworks are encoded as persistent, inheritable **SagaPython™** classes using the **Class Manager Infrastructure (CMI)**. Each class follows the canonical decorator pattern:

```
@sagaclass()
<sagafield() declarations>
@sagamethod()
```

Base inheritance always begins with SPClassObject or its derived foundations: ClassFungibleAsset, ClassNonFungibleAsset, ClassConsumerAccount, ClassBusinessAccount.

A.1 ISO 20022 Payment Instruction

```
@sagaclass()
class ISO20022_Pacs008(SPClassObject):
    message_id = sagafield(max_length=35)
    creation_date_time = sagafield(pattern=r"\d{4}-
\d{2}-\d{2}T\d{2}:\d{2}:\d{2}")
    settlement_amount = sagafield()
    settlement_currency = sagafield(length=3)
    instructing_agent = sagafield()
    instructed_agent = sagafield()

    @sagamethod()
    def validate_payment(self):
        assert self.settlement_amount > 0
        assert len(self.settlement_currency) == 3
```

A.2 DSCSA Serialized Unit

```
@sagaclass()
class ClassDSCSAUnit(ClassNonFungibleAsset):
    sgtin = sagafield(type="str")
    lot_number = sagafield(type="str")
    expiry_date = sagafield(type="date")
    verification_status = sagafield(type="str",
default="unverified")
    epcis_events = sagafield(type="list[dict]")

    @sagamethod()
    def verify(self):
        self.verification_status = "verified"
        return {"verification_status":
self.verification_status}
```

A.3 CDISC Clinical Study

```
@sagaclass()
class ClassCDISCStudy(SPClassObject):
    study_id = sagafield(type="str")
    sponsor = sagafield(type="str")
    phase = sagafield(type="str")
    sdtm_datasets = sagafield(type="list[str]")
    adam_datasets = sagafield(type="list[str]")

    @sagamethod()
    def add_dataset(self, standard: str, ref: str):
        if standard.upper() == "SDTM":
            self.sdtm_datasets.append(ref)
        elif standard.upper() == "ADAM":
            self.adam_datasets.append(ref)
```

A.4 Lifecycle-Persistent Clinical Trial Record

```
@sagaclass()
class ClassClinicalTrialRecord(MixinFDAAA801,
MixinCDISC, MixinHL7FHIR,
MixinPrivacyEnclave):
    sponsor = sagafield(type="str")
    study_title = sagafield(type="str")
    phase = sagafield(type="str")
    start_date = sagafield(type="date")
    end_date = sagafield(type="date", default=None)

    @sagamethod()
    def bind_enclave_envelope(self, envelope_id: str):
        self.private_envelope_id = envelope_id
        return {"private_envelope_id":
self.private_envelope_id}
```

A.5 ISO 20022 Milestone Payment Linked to Clinical Trial

```
@sagaclass()
```

```
class
ClassISO20022MilestonePayment(SPClassObject):
    message_id = sagafield(type="str")
    debtor = sagafield(type="str")
    creditor = sagafield(type="str")
    amount_minor = sagafield(type="int")
    currency = sagafield(type="str")
    trial_ref =
sagafield(type="ref:ClassClinicalTrialRecord")
```

```
@sagamethod()
def release_if_trial_ready(self):
    trial = self.trial_ref
    if trial.registry_status == "posted":
        self.status = "released"
```

APPENDIX B — GLOSSARY OF TERMS

Term	Definition
SagaChain™	Layer-1 blockchain designed as a persistent object system where data, code, and accounts coexist within a unified execution layer.
SagaPython™	Python-based language for SagaChain, supporting @sagaclass, @sagamethod, and sagafield as blockchain-native semantics.
SagaPSA™	*Programmable Smart Assets* — persistent SagaPython™ objects representing standards or regulatory frameworks.
SagaOS™	The on-chain operating layer that governs namespaces, persistence, versioning, and message routing.
Global Class Tree	Canonical, single-instance hierarchy of all classes and objects on SagaChain™, harmonizing standards and regulatory logic.
SagaScale™	Dynamic sharding mechanism ensuring load balance, locality, and continuous availability.
SagaInterop™	Protocol suite for verifiable cross-chain interoperability with external blockchains.
SagaFeeds™	*Public verified source feeds available* for data transparency and regulated discovery. Supports free read access and controlled visibility for verified compliance data.
Private Enclaves	SagaChain™ mechanism for merging private and public blockchains on a single layer. An enclave is a combination of enclave shards, nodes, accounts, and code. Enclave transactions include the SagaPython™ transaction script and cryptographic hashes of output state and logs, while actual state data remains off-chain and private to the enclave nodes. Private classes may inherit from public classes but not vice versa. Enclave shards execute authorized

Term	Definition
	transactions in isolation, allowing private computation and code execution without leaving the SagaChain Layer-1 runtime. Hardware trusted execution (TEE) systems may be implemented optionally by users, but they are **outside the SagaChain specification**. SagaNode™ can operate on any authorized infrastructure, public or enclave.
SagaCoin™	Native medium of exchange for transaction fees, inter-chain settlement, and validator compensation.
SagaStandards	Governance division of the PraSaga Foundation responsible for open, executable standards and stewardship of the Global Class Trees.

APPENDIX C — DIAGRAM INDEX

#	Section	Diagram Title	Filename
1	1	Abstract	Global Pharma Standards Interoperability Map
2	2	2	Diagram_1-A_Global_Pharma_Interoperability_Map
3	3	2.2	Diagram_2-A_Universal_Pharma_Class_Tree_Conceptual_Architecture_v2
4	4	3	SagaPharma Technical Architecture Stack
5	5	3.2	SagaOS Core Components
6	6	3.4	Global Single-Instance Pharma Class Tree
7	7	3.5	SagaStandards™ Governance Workflow
8	8	3.6.1	SagaInterop Architecture Pharma
9	9	3.6.3	3-F Cross-Domain Data Handshake
10	10	4	Persistent-State Governance Model

#	Section	Diagram Title	Filename
5-A → 5-F	9	4.3	Enclave Compute Workflow
11-20	10	4.8	SagaFeeds™ Transparency Loop
21	1	Abstract	Global Pharma Standards Interoperability Map
22	2	2	Universal Pharma Class Tree Conceptual Architecture
23	3	2.2	Multi-Inheritance Example Graph

Appendix C Note
All diagram assets are managed through the SagaStandards registry. Provenance records are published through **SagaFeeds™ public verified source feeds available** for reproducibility and institutional validation.

APPENDIX D — SCENARIO COMPENDIUM

The following twenty-five scenarios expand Section 5 of the main paper, demonstrating how SagaChain™ encodes global standards into executable compliance lifecycles.

Scenario #	Title	Core Standards / Frameworks	Diagram Ref.
1	GMP Manufacturing Compliance	21 CFR 210/211 · USP · ISO 9001	5-A
2	DSCSA Serialization & Traceability	DSCSA · GS1 EPCIS · ISO IDMP	5-A
3	USP Monograph Integration	USP · ISO 11137 · ISO 20022	5-A
4	GMP Batch → DvP Finance Loop	ISO 20022 · USP · DSCSA	5-A
5	Clinical Research Lifecycle	FDAAA 801 · CDISC · HL7 FHIR · FAERS	5-B

Scenario #	Title	Core Standards / Frameworks	Diagram Ref.
6	Adaptive Trial Governance	CDISC · HL7 FHIR · WHO ICTRP	5-B
7	Pharmacovigilance Integration	FAERS · VAERS · MedDRA · USP	5-C
8	Real-World Evidence Validation	HL7 FHIR · OMOP · ISO IDMP	5-C
9	Safety Signal Automation	FAERS · EudraVigilance · WHO VigiBase	5-C
10	Global Label Synchronization	SPL · GS1 Digital Link · WHO Formulary	5-D
11	Equity & Donor Escrow Verification	WHO Treaty · ISO 20022 · LMIC Proofs	5-D
12	Pandemic Treaty Compliance	WHO · CDC · EMA · ISO IDMP	5-D
13	Digital Therapeutics Interoperability	FDA SaMD · EMA CE · HL7 FHIR	5-B
14	Supply Chain Finance & Risk Pooling	GS1 · DSCSA · ISO 20022 · ESG	5-E
15	ESG Accountability in Cold-Chain	ISO 14083 · GHG Protocol · ISO 20022	5-E
16	Emergency Recall Trigger	DSCSA · FAERS · FDA Recall Portal	5-F
17	AI-Driven Drug Repurposing	NIH · FDAAA 801 · ISO IDMP · HL7	5-B
18	Global Pharmacovigilance Network	FAERS · VigiBase · EudraVigilance	5-C
19	Donor-Funded Access Equity Model	WHO · GAVI · ISO 20022 · GS1	5-D
20	ESG-Linked Financing for Pharma	ISO 20022 · GHG Protocol · UN SDG 12	5-E

Scenario #	Title	Core Standards / Frameworks	Diagram Ref.
21	Crisis Response & Supply Resilience	DSCSA · WHO · FAERS · ISO 28000	5-F
22	Cross-Border Label Access	SPL · GS1 Digital Link · openEHR	5-D
23	Adaptive Regulatory Pathways (AI)	ISO IDMP · FDA AI Pilot · EMA AI Act	5-B
24	Global Health Equity Metrics	WHO · UNDP · SagaFeeds (public verified source feed)	5-D
25	Persistent Public-Private Oversight	SagaStandards · WHO · FDA · ISO	5-F

APPENDIX E — FUTURE WORK

1. **Regulatory Class Tree Expansion** — Integrate EMA, PMDA, MHRA, and Health Canada mappings.
2. **Domain Extensions** — Extend SagaPharma™ to diagnostics, medical devices, and clinical genomics.
3. **AI Governance Integration** — Embed algorithmic accountability via SagaPSA™ class lineage.
4. **Formal Governance under SagaStandards** — Create public RFC workflows and class-version stewardship boards.
5. **Public Data Governance** — Extend **SagaFeeds™ public verified source feeds available** to all regulatory and industry transparency data.

CONCLUSION OF APPENDICES

The appendices serve as the authoritative technical reference for the *SagaPharma™ Universal Class Tree White Paper*, providing reproducible SagaPython™ implementations, validated terminology, complete diagram and scenario indices, and accurate descriptions of **Private Enclaves** as defined in the *Merging Private and Public Blockchains on a Single Layer* framework by **David Beberman, CTO**. They ensure fidelity between academic documentation and the live SagaChain™ implementation, establishing a foundation for transparent, privacy-preserving, and standards-aligned digital health infrastructure.