PART II - CONTEXT MAPS AND RELATIONSHIPS

10. Anti-Corruption Layers

A model is not corrupted only by bad code. It is corrupted when another model's words enter it unexamined and begin to look native.

What you already know

You have integrated systems before. One system sends a field called status, another system expects state. One calls a machine asset, another calls it component. One timestamp is local time, another is UTC. One quality flag means the sensor value is good; another means only that the transport packet arrived. None of this is malicious. Each system is speaking honestly inside its own world. The danger begins when your code imports that foreign language directly and lets it become your model.

An anti-corruption layer is the architectural checkpoint that prevents this. It does not merely map fields because names differ. It translates meaning. It decides which external facts are acceptable, which must be converted, which need enrichment, and which must be refused because accepting them would make the internal model lie.

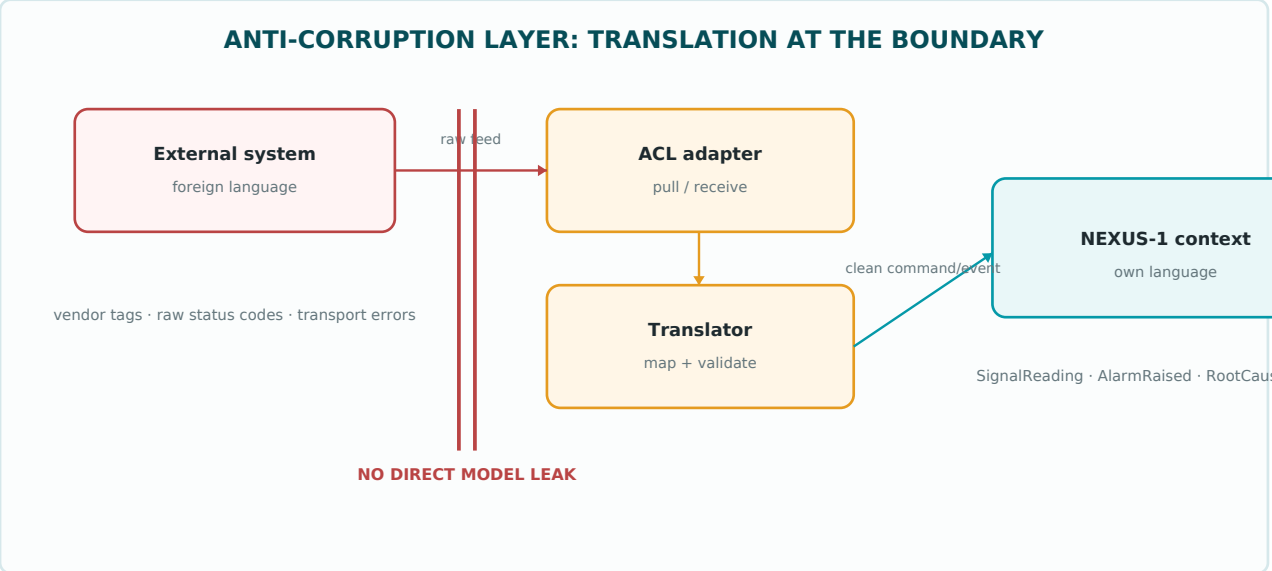

Figure 10.1 - The anti-corruption layer keeps raw external language from crossing directly into the NEXUS-1 model. The internal context receives clean commands, events, or value objects.

PLAIN EXPLANATION

An anti-corruption layer protects a bounded context from a foreign model. It receives external data, translates it into the context's own language, validates it, and refuses anything that cannot be translated honestly.

Why the word corruption matters

Corruption sounds dramatic, but in DDD it often begins quietly. A vendor historian has a field named Tag. NEXUS-1 has a SignalTag. They look similar, so a developer copies the vendor value straight into the domain. Later someone discovers that the vendor tag is sometimes an alias, sometimes a calculated point, and sometimes a temporary engineering name. The NEXUS-1 SignalTag was supposed to identify an instrumented signal inside the Instrumentation context. The imported field was not that thing. The code compiled, the

database accepted the row, and the model became less true.

The ACL exists to make that moment explicit. It asks: what does this external value mean in its own system, what does our model require, and can the first be translated into the second without lying? If the answer is no, the data stops at the boundary.

FOREIGN WORD	WHY IT IS UNSAFE	NEXUS-1 INTERNAL WORD
Vendor Tag	May be an alias, calculated point, or temporary engineering identifier.	SignalTag after lookup and validation.
Quality = 1	The code may mean transport success, not physical measurement quality.	SignalQuality with explicit semantic meaning.
Asset	A maintenance vendor may mean equipment, spare part, tool, or location.	MaintenanceAsset or ReactorFleet.Component, chosen deliberately.
User role	An identity provider role may be organizational, not authorization policy.	Security.Permission or PolicyAssignment.
Status	Status labels are often lifecycle-specific and cannot be reused blindly.	Context-owned state such as AlarmState or FindingStatus.

NEXUS-1 SENTENCE TO REMEMBER

The ACL does not protect NEXUS-1 from external systems because they are bad. It protects NEXUS-1 because their truth is not automatically the same truth.

Where NEXUS-1 needs anti-corruption layers

NEXUS-1 is a demonstrator, but it is deliberately shaped like a system that would one day meet external sources: a historian or telemetry provider, an identity system, a maintenance platform, a reporting/export engine, perhaps a document repository for evidence. Every one of those systems already has its own language. If NEXUS-1 simply imports that language, the contexts lose their boundaries before any microservice split begins.

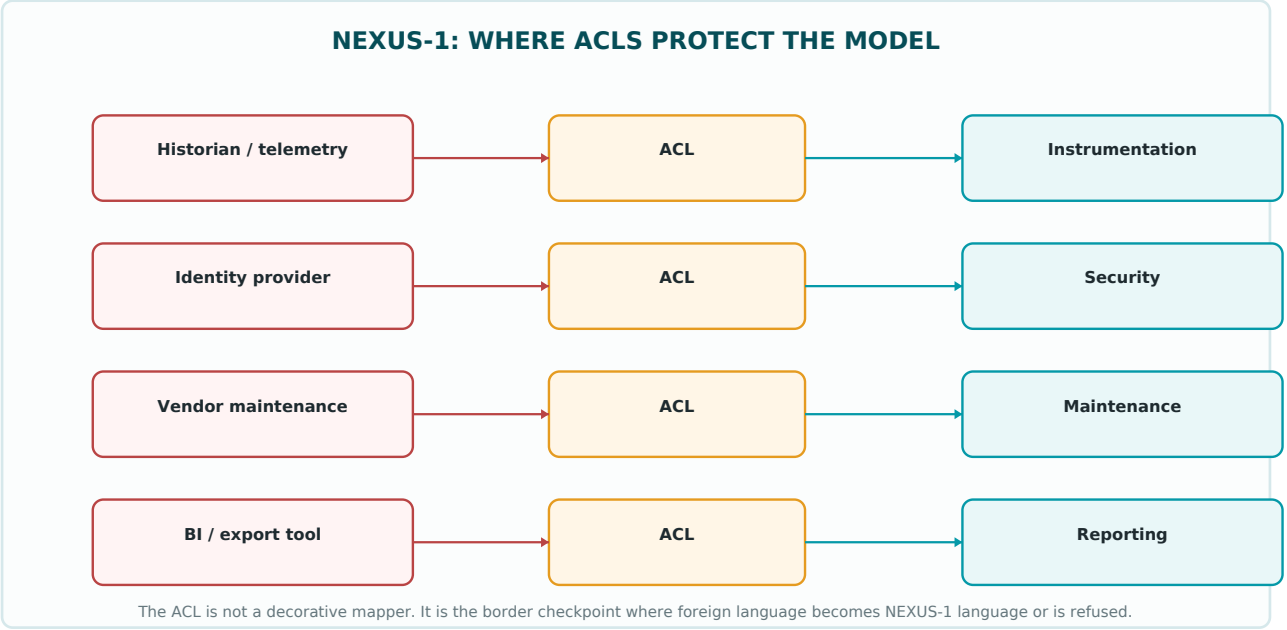

Figure 10.2 - Candidate ACLs around NEXUS-1. Each gate translates an external model into a context-owned language before the domain sees it.

EXTERNAL SOURCE	PROTECTED CONTEXT	ACL RESPONSIBILITY
Historian / telemetry feed	Instrumentation	Resolve external tags, validate timestamps, convert engineering units, translate quality codes.
Identity provider	Security	Translate external claims into NEXUS-1 permissions and policies without copying the provider role model.
Maintenance vendor platform	Maintenance	Translate vendor assets and work-order statuses into NEXUS-1 asset and work-order language.
BI / export tool	Reporting	Expose projections without letting the reporting tool define domain state.
Document repository	RootCause / Compliance	Translate file metadata into EvidenceReference and ComplianceEvidence without importing repository taxonomy.

The ACL pipeline

A useful ACL is not a single AutoMapper profile hidden in an infrastructure folder. It is a small pipeline with a clear sequence. First, receive the raw message and keep it outside the domain. Second, validate the message shape: required fields, timestamp window, duplicate key, version. Third, translate names and units. Fourth, enrich the message by resolving known identities such as UnitId or SignalId. Only then does the ACL produce an internal command or event.

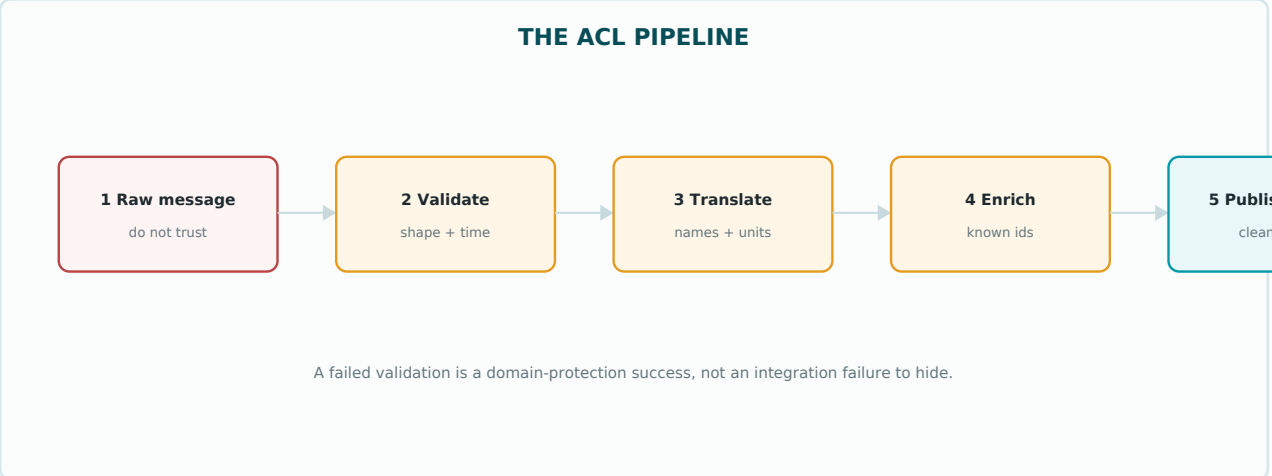

Figure 10.3 - The anti-corruption pipeline. Raw external data is not domain data until it survives validation, translation, and identity resolution.

```
// External contract: it belongs to the historian integration, not the domain.
public sealed record HistorianReadingDto(
    string ExternalTag,
    DateTime Timestamp,
    double Value,
    string UnitSymbol,
    int QualityCode);

// Internal command: it speaks Instrumentation language.
public sealed record ImportSignalReading(
    SignalTag SignalTag,
    DateTime TimestampUtc,
    decimal Value,
    EngineeringUnit Unit,
    SignalQuality Quality,
    CorrelationId CorrelationId);

public sealed class HistorianAcl
{
    public ImportSignalReading Translate(HistorianReadingDto dto)
    {
        var tag = ResolveSignalTag(dto.ExternalTag);      // alias -> owned tag
        var utc = ConvertToUtc(dto.Timestamp);            // time rule is explicit
        var unit = ResolveEngineeringUnit(dto.UnitSymbol); // no raw string leak
        var quality = TranslateQuality(dto.QualityCode);   // vendor code -> model

        var value = Convert.ToDecimal(dto.Value);
        var correlation = CorrelationId.New();

        return new ImportSignalReading(
            tag, utc, value, unit, quality, correlation);
    }
}
```

The important point is not the exact C# shape. The important point is the direction of dependency. The domain command does not know the historian DTO exists. The translator knows both languages and lives at the boundary.

Validation is part of translation

A weak ACL maps names and lets everything through. A serious ACL treats validation as part of translation. If an external reading arrives with an unknown tag, a future timestamp, a unit that cannot be converted, or a quality code the system cannot interpret, the ACL should not manufacture a false internal fact. It should reject the import, quarantine it, or emit an explicit integration failure event.

PROBLEM AT BOUNDARY	BAD RESPONSE	GOOD ACL RESPONSE
Unknown external tag	Create a new Signal automatically.	Reject or quarantine until the tag is mapped deliberately.
Local timestamp with no time zone	Assume server local time.	Require source time-zone rule or reject.
Quality code not recognized	Default to Good.	Map to Unknown / Invalid or refuse the reading.
Duplicate message	Write a second measurement.	Use idempotency key and ignore or reconcile.
Unit mismatch	Store the numeric value unchanged.	Convert units explicitly or reject the import.

COMMON MISTAKE

Do not hide failed translation by inventing a safe-looking default. A default that makes the domain believe something happened is not safe; it is silent corruption.

ACL is not the same as DTO mapping

A DTO mapper changes shape. An anti-corruption layer protects meaning. Sometimes the implementation uses DTOs, adapters, mappers, parsers, lookup tables, and validators. Those are techniques. The architectural responsibility is larger: keep the foreign model from becoming the internal model.

TECHNIQUE	CAN BE PART OF ACL?	BUT IT IS NOT ENOUGH BECAUSE
DTO class	Yes	A DTO only describes shape; it does not prove meaning was translated.
AutoMapper profile	Sometimes	Field-to-field mapping can copy corruption faster than hand code.
Adapter	Yes	An adapter can isolate transport but still leak language.
Validator	Yes	Validation must know the model rules, not only required fields.
Lookup table	Yes	A lookup table helps only when ownership and versioning are clear.

HONEST BOUNDARY

The ACL examples here are architectural sketches. This volume explains where the protection belongs and what it must protect. Full C# implementation patterns, package boundaries, integration tests, and adapters belong to the later implementation volumes.

The NEXUS-1 alarm flow through an ACL

Return to the recurring flow of this book: Alarm Raised -> Alarm Flood Detected -> Root-Cause Case Opened -> Evidence Attached -> Verdict Issued -> Recommendation Generated -> Audit Entry Written -> Compliance Review Triggered. If the first alarm comes from an external alarm system, NEXUS-1 should not treat the external alarm record as an AlarmEvent. The ACL translates it into NEXUS-1 alarm language first: known unit, known signal, known severity, known timestamp, known source, known confidence. Only then may AlarmManagement raise an internal event that RootCause can consume.

BEFORE ACL	AFTER ACL
ExternalAlarmRecord.SourceCode	AlarmSource.ExternalHistorian with mapped source id.
ExternalAlarmRecord.TagName	SignalTag resolved against Instrumentation.
ExternalAlarmRecord.Priority	AlarmSeverity translated by configured rule.
ExternalAlarmRecord.Time	RaisedAtUtc with explicit time-zone conversion.
ExternalAlarmRecord.Payload	Raw payload stored as provenance, not as domain truth.

This is why an ACL is a strategic DDD pattern. It is not local cleanliness. It decides which language is allowed to enter a context and in what form.

Microservice-readiness implication

An ACL becomes even more important after a microservice split. Without it, a service boundary only moves the corruption behind an HTTP call or a message broker topic. Before NEXUS-1 turns any context into an independently deployed service, every external dependency should have a named boundary strategy: conformist, published language, customer/supplier, shared kernel, or anti-corruption layer. The ACL is the right answer when the external model would damage the internal one if copied directly.

MICROSERVICE-READINESS CHECK

A context is not ready to become a microservice while it still imports external DTOs, vendor enums, foreign status codes, or another context's aggregate types directly into its domain. Those are signs that the anti-corruption layer has not been drawn yet.

Chapter 10 checkpoint

QUESTION	A GOOD ANSWER SHOULD INCLUDE
What is an anti-corruption layer?	A boundary component that translates and validates a foreign model before the internal domain sees it.
What does it protect?	Language, rules, identities, units, timestamps, states, and domain meaning.
Why is DTO mapping not enough?	Because changing shape does not guarantee semantic translation.
Where does NEXUS-1 need ACLs?	Telemetry/historian feeds, identity providers, maintenance platforms, document repositories, and external reporting tools.
What should happen when translation fails?	Reject, quarantine, or emit an explicit integration failure; do not invent false domain facts.
Why does this matter before microservices?	Because runtime boundaries do not fix model leakage; they can make it harder to see.

END OF STEP 10

Step 10 adds Chapter 10: Anti-Corruption Layers. The next step I would define is Chapter 11: Published Language.