Pipeline demo: the architecture in five views

The entity of interest is the demo: the SysML adapter, the example stack behind it and the image they ship in. The pipeline the example models is a system the demo describes, never the demo itself. Each view answers the concerns of named stakeholders and has a text form in the architecture description, which is the record.

V1 Context

the visitor

The demo among the things it touches: a browser with two tabs, the Docker host, the registry the image is pulled from, the maintainer's machine. Nothing leaves the container while it runs.

What is inside, what is outside, what crosses the line?

V2 Composition

the model owner, the document owner, the maintainer, an organisation deciding

Three subgraphs, their schemas, the entity keys they join on, and the merged graph the router serves. Who owns which field, and what the capacity service has to agree with the adapter.

What do the services agree on?

V3 Runtime

the visitor, the model owner, the document owner

What happens on an edit: mutation, version event, fan-out, refetch, rollup on read, render in both apps. The case where nothing moves, a document edit, a reset, and startup.

How does a change in one tab reach the other?

V4 Deployment

the maintainer, an organisation deciding

One container, one process tree, one published port. The image layers, the router copied in and its licence fetched beside it, the environment that keeps it offline, the build and the tag-triggered publish.

What runs where, and how does it get built?

V5 Adapter

the model owner, the maintainer, a contributor

Inside the one reusable component: lexer, parser, AST with source spans, resolver, projection, server. How an edit patches the text, how unsupported syntax is refused, how the example is validated.

How does a model file become a subgraph?

a component of the demo, named with its type and technology

a component of the demo owned by the example rather than the adapter

the router, and what is shared through it

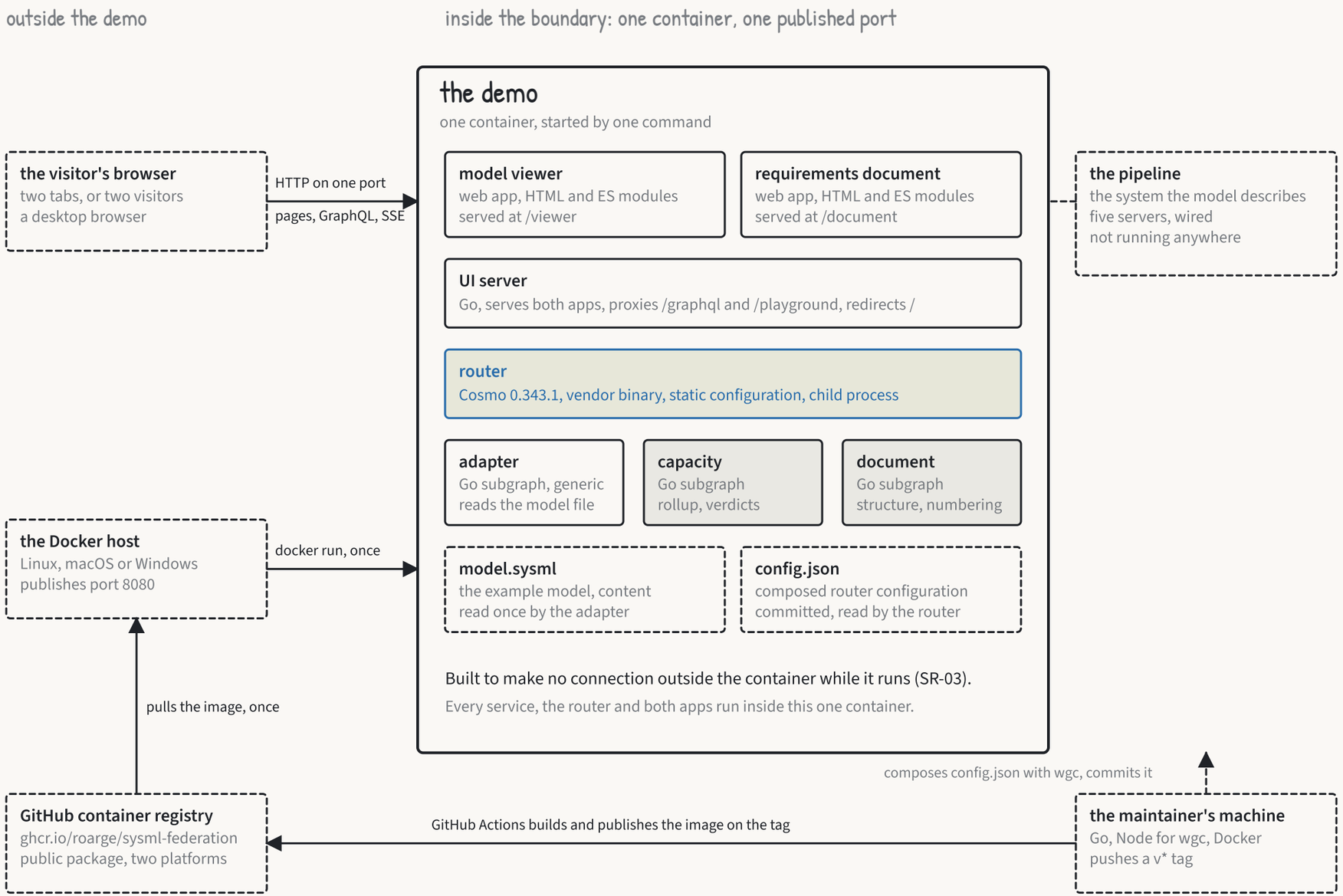
something outside the demo

a failing requirement or a bottleneck, the only red

a labelled, one-directional relationship

V1 Context

the demo among the things it touches, at the level of one container and its neighbours



What crosses the line

The browser makes HTTP requests to one published port: the two pages, GraphQL queries and mutations, and the SSE streams that carry version events. The Docker host pulls the image once. The maintainer's machine, never the container, runs the composition step and pushes the tag that publishes the image.

Two things easy to mistake for the inside

The example model is content the adapter reads, authored in the repository and validated with the OMG pilot before it ships (SR-45). The pipeline it describes does not run anywhere. The router is a vendor binary copied into the image and driven only by configuration and environment (AD-0010).

Concerns addressed

The visitor: can I launch it with one command, and where do I look (US-01, SR-01, SR-04). An organisation deciding looks past this view: V2 for the agreement between services, V4 for what to replace.

a component of the demo, with its type and technology

owned by the example rather than the adapter

the router, the only place the services meet

outside the demo, or content the demo reads

arrows are one-directional and labelled with their intent

V2 Composition

three subgraphs, one merged graph, and what the services agree on

two clients, both of the router only (SR-40)

model viewer
web app, HTML and ES modules
SSE: modelChanged

requirements document
web app, HTML and ES modules
SSE: modelChanged, documentChanged

queries, mutations, SSE subscriptions

queries, mutations, SSE subscriptions

router
Cosmo router, merged graph from config.json, plans every query across the three subgraphs

fetch, first

fetch, entity representations carrying the @requires fields

fetch by key

model (adapter)
gqlgen subgraph, :3011

Model
version, text, roots, parts, requirements, verificationCases

Part @key id
id, shortName, name, definition, doc, attributes, ports, parts, connections, satisfies

Attribute
name, value, unit, editable, expression

Requirement @key id
id, shortName, name, text, subject, quantity, comparison, limit, limitUnit, limitEditable, derivedFrom, derives, satisfiedBy, verifiedBy

VerificationCase @key id
id, shortName, name, verifies

Mutation
setAttribute, setLimit, resetModel

Subscription
modelChanged

no example names (SR-17)

capacity
gqlgen subgraph, :3012

Part @key id
capacity, bottleneck

capacity @requires
attributes { name value }
parts { id name attributes { name value } }
connections { from to }

bottleneck @requires
parts { id name attributes { name value } }
connections { from to }

Requirement @key id
verdict, verdictReason

verdict, verdictReason @requires
quantity comparison limit
verifiedBy { shortName }
subject { id name attributes { name value }
parts { id name attributes { name value } }
connections { from to } }

configured
capacity: the quantity it computes
throughput: the attribute it reads

agreement with the adapter:
the entity key,
this field set, two names

document
gqlgen subgraph, :3013

Document
version, nodes

Node
id, kind, number, text, requirement, children

Requirement @key id
documentNumber, included

Mutation
moveNode, insertHeading, addProse, editText, excludeRequirement, includeRequirement, resetDocument

Subscription
documentChanged

never reads the model: its shipped tree names the example's requirement ids, the one place the example identifiers enter a service

numbering is recomputed on every mutation and served on the nodes and, through the entity, on Requirement.documentNumber (SR-33)

the merged graph

Requirement, as the router serves it
one entity, three subgraphs, keyed on id

id	adapter
shortName	adapter
name	adapter
text	adapter
subject	adapter
quantity	adapter
comparison	adapter
limit	adapter
limitUnit	adapter
limitEditable	adapter
derivedFrom	adapter
derives	adapter
satisfiedBy	adapter
verifiedBy	adapter
verdict	capacity
verdictReason	capacity
documentNumber	document
included	document

the playground query (SR-43)

```
{
  requirement(id: "PIPE-R1") {
    text
    verdict
    verdictReason
    documentNumber
  }
}
```

the router resolves it with three fetches

1 adapter: requirement plus @requires fields
2 capacity: entity resolver with those fields
3 document: entity resolver with the key

config.json
composed router configuration, committed graph.yaml with wgc, drift test (SR-42)

Who owns what

No service resolves a field another service resolves. The capacity service adds four fields to Part and Requirement, the document service two to Requirement, and every other field on those types is the adapter's. The entity types are Part, Requirement and VerificationCase, keyed on id, the SysML short name or, failing that, the qualified name (SR-21, AD-0018).

What the capacity service agrees to

It carries two configured names, capacity and throughput, and no other word of the model (SR-31). The rest of the agreement is the entity key and the field set its @requires declares, which the router fills from the adapter's answer and hands over as the representation. Its four fields are computed on read from the fields its @requires names (SR-28 to SR-32).

The first spike

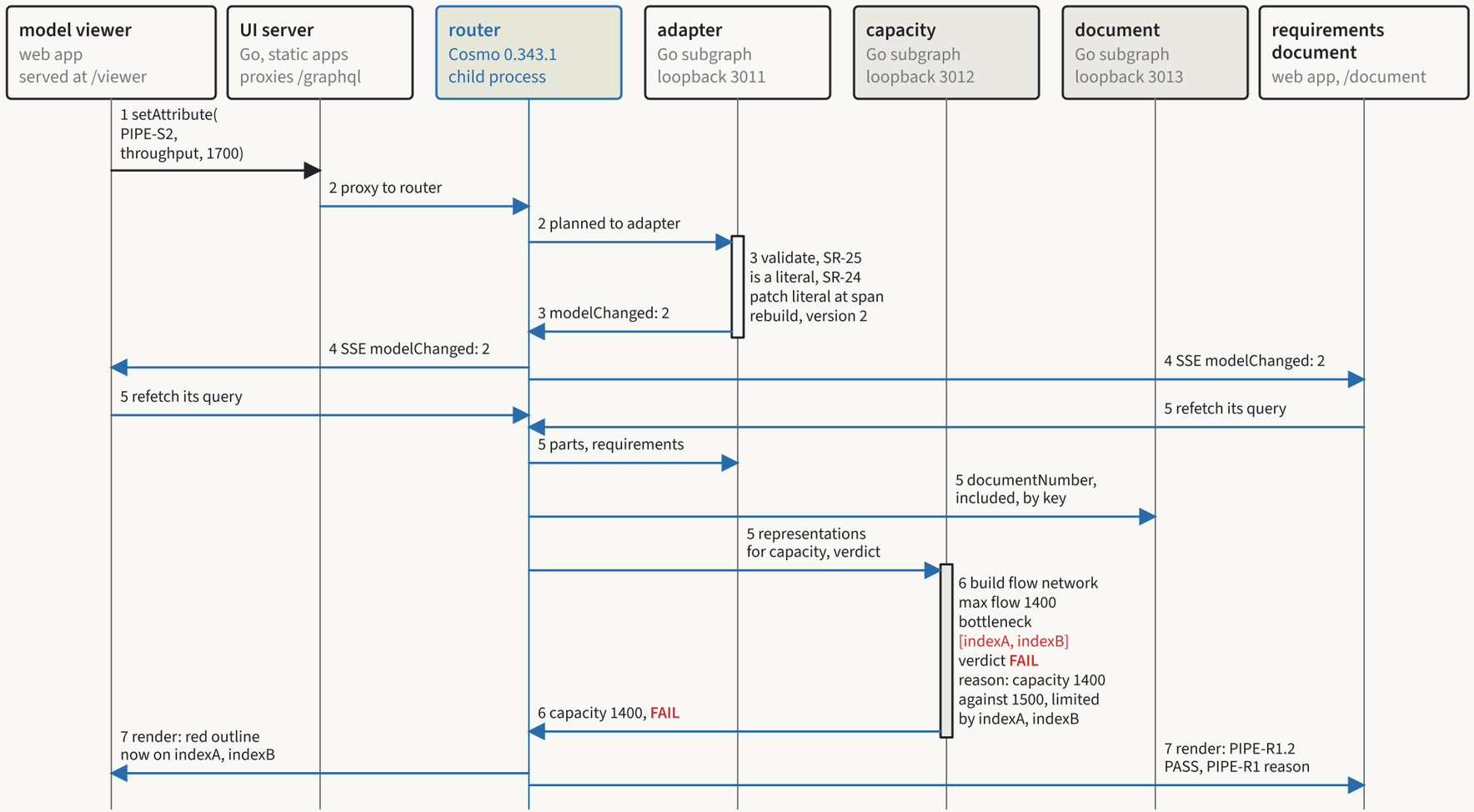
Whether Cosmo composition and gqlgen accept a nested-list @requires, and where @external and @shareable go on the value types inside it, is the first implementation spike (C-15). The fallback keeps the same semantics behind one scalar: the adapter adds Part.wiring, a JSON document of the children and connections, and the capacity service requires subject { id name wiring } in place of the nested lists.

a component of the demo, with its type and technology
owned by the example rather than the adapter
the router, and the merged graph as it serves it
content the demo reads
@key id marks an entity type, resolved across subgraphs by id
arrows are one-directional and labelled with their intent

V3 Runtime

what happens on an edit, from the viewer to both apps

an edit in the viewer: parse raised from 1200 to 1700, and the seven steps until both apps show it



Nothing moves

ingest raised to 3000. Steps 1 to 5 are identical. In step 6 the cut is still parse and every number and verdict is unchanged. In step 7 only the number on ingest changes. The event still fires and the apps still refetch. Nothing detects a no-op.

bounded by SR-39: two seconds from step 3 to step 7

A document edit

PIPE-R2 moved under PIPE-R1: moveNode(id, parentId, index) reaches the document service. It reorders, renumbers (SR-33), increments its version, emits documentChanged. The viewer, not subscribed, does nothing. The model version is unchanged (SR-36).

Reset

One mutation from either app carries both root fields, resetModel and resetDocument. Each service restores its shipped state, increments its version (C-92) and emits its event, and both apps refetch (SR-44). If one field fails the other is still applied. Both are sent because neither service knows about the other (SR-41).

The round trip

One mutation leaves the viewer, passes the UI server's reverse proxy and reaches the router, which plans it to the adapter. The adapter patches the literal, rebuilds, moves the version to 2 and emits the event on the WebSocket subscription the router holds. The router fans the event out over SSE, each app refetches, and each answer comes from the adapter first, then the entity resolvers. Both apps' traffic passes the proxy, which flushes on text/event-stream (C-62), drawn once at step 2.

What the demo does not do

Nothing detects that nothing changed. Raise ingest to 3000 and the event still fires, both apps refetch and only the number on ingest differs. No service calls another. The capacity service computes from the representation the router hands it, and the document service never reads the model. Capacity, bottleneck, verdict and reason are computed on every read from the fields the @requires names (SR-28 to SR-32).

Startup

serve starts the three subgraphs as goroutines on loopback ports, the adapter reading /app/model.sysml by default, and waits for each health endpoint. It then starts the router as a child process with SR-03's environment and EXECUTION_CONFIG_FILE_PATH pointing at the copied configuration, waits for /health/ready on the router's loopback port and only then opens the published port. What /health/ready reports before the subgraphs answer is C-57's spike, and the order does not depend on it.

---

a component of the demo, with its type and technology

owned by the example rather than the adapter

the router, and every message that passes through it

an activation, where a service computes (steps 3 and 6)

FAIL red marks the failing verdict and the bottleneck, nothing else

arrows are one-directional, numbered by step and labelled with what they carry

V4 Deployment

one container, one process tree, one port, and how the image is built and published

one container, one process tree, one port

container
ghcr.io/roarge/sysml-federation, one process tree, one published port

sysml-federation serve
PID 1, Go supervisor

- **adapter**
  goroutine, 127.0.0.1:3011, gqlgen subgraph "model"
- **capacity**
  goroutine, 127.0.0.1:3012, gqlgen subgraph "capacity"
- **document**
  goroutine, 127.0.0.1:3013, gqlgen subgraph "document"
- **/router**
  child process, 127.0.0.1:3002
  Cosmo router 0.343.1
- **UI server**
  goroutine, 0.0.0.0:8080
  embed.FS apps, reverse proxy

environment ★ C-20
set by the supervisor
LISTEN_ADDR=127.0.0.1:3002
PLAYGROUND_PATH=/playground
EXECUTION_CONFIG_FILE_PATH=
/app/config.json
DO_NOT_TRACK=1
COSMO_TELEMETRY_DISABLED=true
TRACING_ENABLED=false
METRICS_OTLP_ENABLED=false

-p 8080:8080

Docker host
Linux, macOS
or Windows
publishes 8080

the image, layer by layer, base at the bottom

/app/model.sysml	example model, read once by the adapter
/app/config.json	composed router configuration, committed
/sysml-federation	Go, static, CGO_ENABLED=0, a few MB
/router, /router.LICENSE	Cosmo router 0.343.1, about 40 MB ★ C-53
distroless static-debian13	nonroot base image, 2 MB

HEALTHCHECK healthcheck subcommand, interval 10 s, start period 15 s
GETs /health/ready on the router's loopback, /viewer on 8080
exits non-zero if either fails (C-55)

the image, built and published

compose step
wgc router compose graph.yaml, Node
output config.json committed
drift test in Go (SR-42)

commits config.json

repository
v* tag pushed by the maintainer (AD-0020)

tag push starts the workflow

GitHub Actions publish.yml
workflow, runs on push of a v* tag
contents: read, packages: write

cross-compiles both platforms

docker build
examples/pipeline/Dockerfile, root context
linux/amd64, linux/arm64, no QEMU (C-49)

pushes with the workflow token

GHCR
ghcr.io/roarge/sysml-federation, public
tags {{version}} and latest
provenance attestations off (C-52)

read back after the push

manifest read back
two platforms, each under 80 MB
or the job fails (SR-05, SR-06)

The process tree

One binary is PID 1 and everything else runs under it: the three gqlgen subgraphs as goroutines on loopback ports, the vendor router as a child process, and the UI server on the one published port (AD-0010, AD-0011). The supervisor is the router's direct parent and reaps it through os/exec without tini (C-58). Subgraphs start first, each waited for on its health endpoint, then the router until /health/ready answers on its loopback port, then port 8080 opens.

Offline by construction

No connection leaves the container while it runs (SR-03). The four telemetry variables are baked into the image with ENV (AD-0013), and the supervisor adds the listen address, the playground path and the path of the copied configuration, so the router needs no YAML file of its own. Whether it starts from environment alone in a distroless image is C-20's spike. Its licence text is fetched at build time from the vendor's repository at the pinned tag, with a pinned checksum, so it cannot drift from the binary (SR-07).

Subcommands

serve is the default and runs everything in the tree. adapter, capacity, document and ui each run one component on a given address, for anyone who wants to see the services apart. healthcheck GETs the router's /health/ready on its loopback port and /viewer on the published port, and exits non-zero if either fails. That is what the image's HEALTHCHECK calls, since a distroless image has no shell to run anything else (C-55).

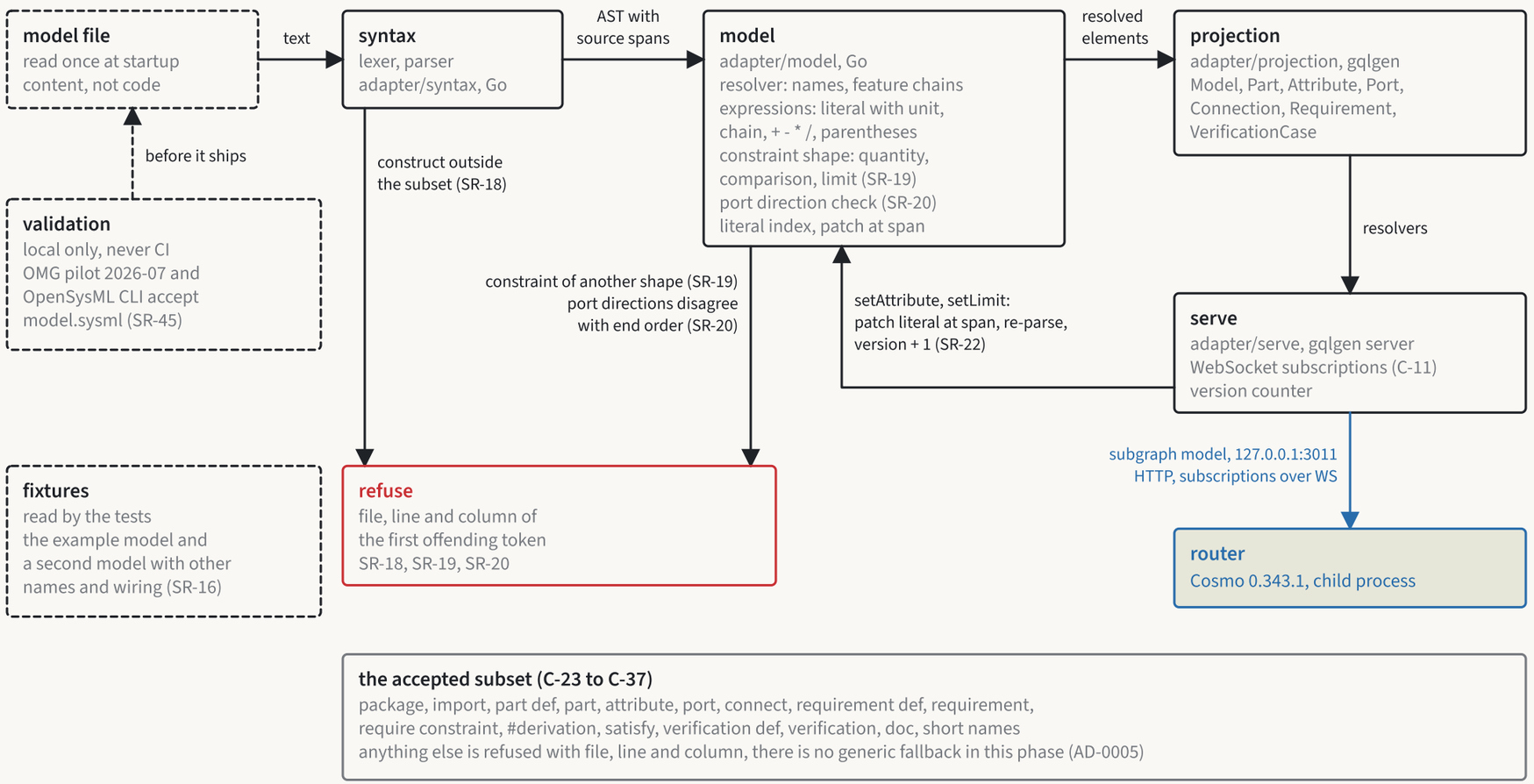
- a component of the demo, with its type and technology
- owned by the example rather than the adapter
- the router, a vendor binary run as a child process
- outside the demo, or content the demo reads

★ a spike before the first tag: C-20 the router from environment alone, C-53 COPY --from on the arm64 layer

arrows are one-directional and labelled with their intent, the trunk and ticks are the process tree

V5 Adapter

inside the reusable component, from model file to subgraph



The served text is the source with edited literals patched in (SR-22). The file is read once and never written.

The adapter has no example identifiers (SR-17): every name in this view is generic, the values in the other views come from the example.

Spans are the mechanism

Every node of the AST carries its byte range in the source, and the model package keeps an index from every numeric literal to its span. That is what makes an edit cheap and honest: setAttribute and setLimit replace one literal at its recorded span and re-parse, so the served text and the projection are rebuilt together and the version counter moves once (SR-22). A value bound by an expression has no literal to patch and is refused (SR-24), which is why editable is a fact about the source rather than a setting.

No opinion about the example

The types are Part, Attribute, Port, Connection, Requirement and VerificationCase, and nothing in adapter/ names a server or a pipeline (SR-17). A second fixture model with other names and wiring runs through the same tests (SR-16). A construct outside the subset is refused with its position rather than served through a generic element type. That escape hatch stays open in AD-0005, and this phase takes the smaller cut.

What it will not do

It does not cover the language. The subset of the syntax card (C-23 to C-37) is parsed by hand and everything else stops at the first offending token (AD-0015). It does not evaluate the rollup: expressions go as far as literals with a unit, feature chains, the four operators and parentheses, enough to bind a derived limit (SR-23), and capacity belongs to the service that computes it. It never writes to disk. Edits live in the served text and its version until reset.

- a package of the adapter, with its technology
- the router, where the subgraph is shared
- the refusal path, with file, line and column
- outside the adapter, or content it reads
- a note, not a component

arrows are one-directional and labelled with their intent, the loop is the edit path