LINX127

Third-party maintenance tool

Who woke the on-call engineer at 3am? or how we automated maintenance ingestion, from manual input to LLM.

Mariano Juliá, Enterprise Architect

18th November 2025

linx

THE PROBLEM

The middle-of-the-night loop

Alarm fires

A circuit or port goes down. The NOC sees the alert.

NOC pages on-call

The NOC engineer pages the on-call engineer.

Checks the calendar

They find a third-party maintenance already in the shared calendar.

Back to bed

Nothing to fix. It was on the calendar all along.



linx

THE PROBLEM

The calendar wasn’t connected to anything

Provider notifications

Arrive by email. A person reads the mailbox and types each one into a shared calendar by hand.

Manual, slow, inconsistent



Not connected

Alarms

Monitoring raises an alert. The NOC engineer decides whether to page the on-call engineer.

Paged by the NOC

linx

THE GOAL

Every provider maintenance in our systems, accurate and on time.

So the NOC can decide what to do with it. Feeding it into monitoring comes next.

linx

Three stages so far

01

Manual

A person reads the mailbox and adds each maintenance to a shared calendar that isn't integrated with anything.

02

Rules-based

The NANOG BCOP and an open-source parser create maintenances in TSI, for providers the rules understand.

03

AI-based

An LLM reads every email. The NOC reviews each draft before it counts.

→

Next

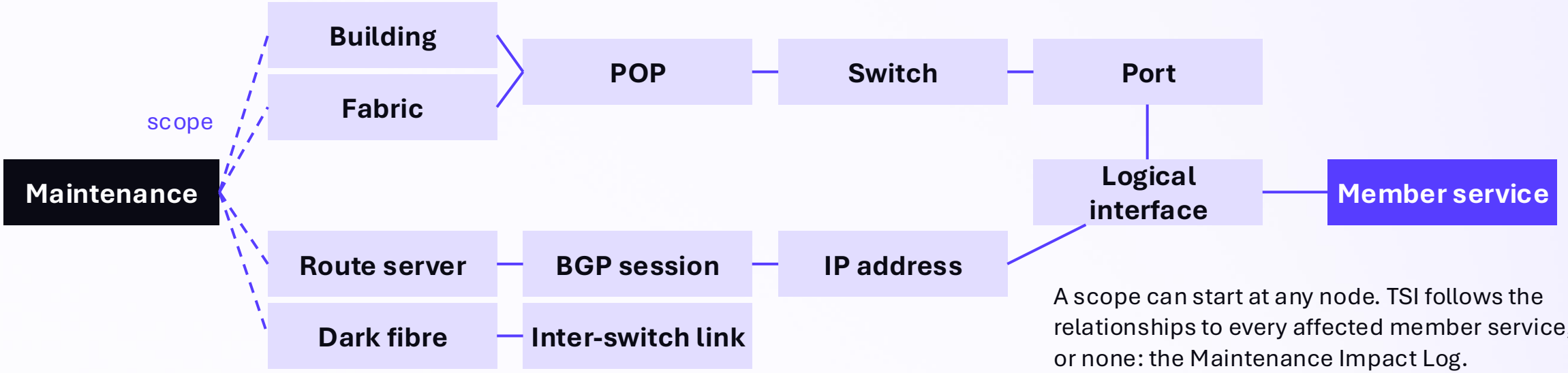
Connect maintenance data to monitoring. Not started yet.

linx

STAGE 2 · THE FOUNDATION

TSI: the data model behind every service

TSI is our model of how every member service connects to the network. We already use it to work out who our own maintenances affect. Now third-party maintenances get the same answer.



STAGE 2 · RULES-BASED

We started with the standard: the NANOG BCOP

Each notice is an **iCalendar event** (RFC 5545) with extra **X-MAINTNOTE-** fields: provider, account, circuit, impact, status.

UID: one ID for the maintenance, kept on every email about it. That's how a reminder or cancellation finds the original.

SEQUENCE: a counter that goes up with each update. If emails arrive out of order, the highest number wins.

IMPACT: no impact, reduced redundancy, degraded or outage (the default).

```
BEGIN:VEVENT
UID:WO-31415@example.net
SEQUENCE:2
X-MAINTNOTE-PROVIDER:example.net
X-MAINTNOTE-ACCOUNT:12345
X-MAINTNOTE-OBJECT-ID:circuit-001
X-MAINTNOTE-IMPACT:OUTAGE
X-MAINTNOTE-STATUS:CONFIRMED
END:VEVENT
```

Source: Draft NANOG BCOP, "A machine parseable standard for formatting maintenance notifications" (via Euro-IX)

linx

STAGE 2 · RULES-BASED

We built on open source

We built on **circuit-maintenance-parser**, Network to Code's open-source Python library that turns provider notifications into one normalised model.

BCOP iCal

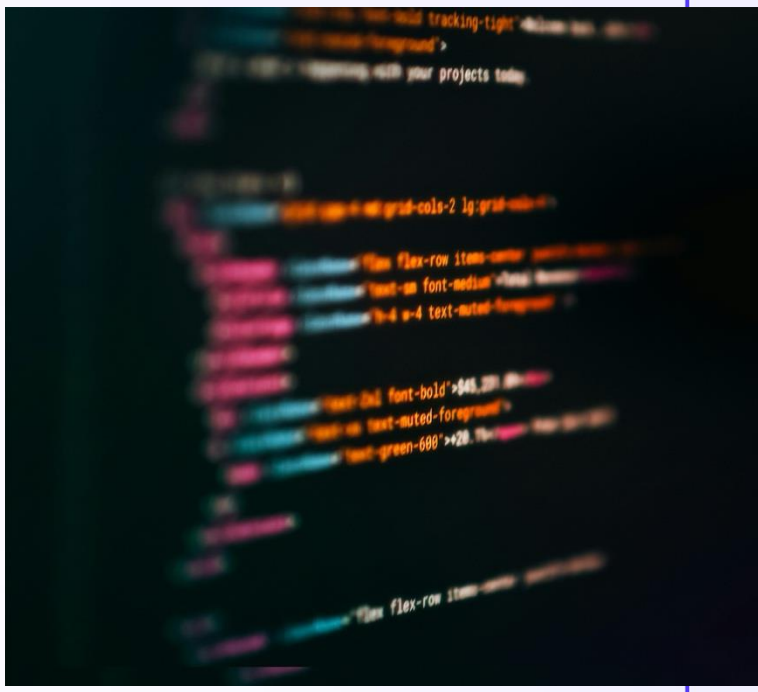
Providers that follow the standard go through one generic parser.

HTML & text

Everyone else needs a hand-written parser per provider.

One output

Provider, maintenance ID, circuit IDs, impact, start, end, status.



github.com/networktocode/circuit-maintenance-parser

linx

STAGE 2 · RULES-BASED

What we delivered with rules

Mailbox

Shared notifications inbox

→

Cron job

Python app reads it

→

Parser

Rules per provider

→

TSI-API

Creates the maintenance

→

Calendar

NOC sees it in ENG-UI

Parsed

Digital Realty and Equinix notices became maintenances automatically.

Not parsed

Sent back to the NOC to enter by hand, as before.

Deployed with an Ansible role; built and tested by a Jenkins pipeline.

linx

STAGE 2 · RULES-BASED

It worked, for the emails it understood

Most of our mailbox, it didn't:

Not just circuits

Many of our third parties are data centres: power, cooling, building work. The library covered dark fibre only.

Free text

Many notices are prose written by a person, not a template.

"Supported" isn't fully supported

The library listed Equinix, but didn't fully parse its emails.

Every provider is new code

Each one means another parser, more tests and more upkeep.

linx

STAGE 3 · AI-BASED

The pipeline worked. The weak link was the reading.

So we kept the mailbox, TSI and the NOC workflow, and replaced the rules with an LLM. Reading messy human text and returning structured data is what LLMs are good at.

linx

STAGE 3 · AI-BASED

Same pipeline, new reader

Mailbox

Shared notifications inbox

→

Filter

Known senders only

→

LLM

Claude today; any provider in config

→

Deskpro

A ticket for every email

→

TSI draft

NOC reviews in ENG-UI

Prompt in config

Kept outside the code, so it can be tuned without a release.

Same schema out

ID, status, UTC windows, impact, circuits, provider, summary.

Follow-ups linked

Reminders and cancellations matched by the provider's ID.

linx

STAGE 3 · AI-BASED

Risks we identified

Letting a model read provider emails brings risks. Providers sometimes send the wrong site or fibre reference, and we've had the wrong dark fibre taken down before.

RISK	HOW WE HANDLED IT
The LLM gets it wrong	Output is an automated draft, hidden until the NOC reviews it
An email slips through	Every email gets a ticket: no skipping NOC review
Unknown or internal sender	Sender whitelist; our own addresses excluded
Follow-ups duplicate work	Matched to the original by the provider's ID
The job fails silently	Failed runs alert the NOC, with the message ID

Scope: dark fibre first. Other work is drafted with the impact fields left for the NOC.

linx

STAGE 3 · AI-BASED

Then we took it to the people who’d own it

Threat model

Mapped against the OWASP Top 10 for LLM applications.

Sign-off

Architecture, the AI council and support managers approved it, on conditions.

Conditions for go-live

Hard spending limit, MFA on the API account, NOC owns the sender whitelist and is alerted on rejects.

Still frontier-dependent

Our tests with small language models didn’t parse well enough. The design is portable; today we rely on frontier models.

Where it missed

28 drafts reviewed by the NOC: 15 with correct data. The misses that matter are about structure, not language.

One email, eight windows

A UPS programme with 8 dates, some service-affecting and some not. Only 3 came out right, and one was duplicated.

One provider, multiple services

Providers like Equinix provide a multitude of service all with different notification formats.

Extensions confuse durations

On “extended” notices the duration didn’t match the ticket, and details filled with stray characters.

Source: NOC qualitative review of AI-generated drafts

linx

What we’re doing about it

DONE

Fixed format, no randomness

Same email, same answer: follow-ups match.

Regression tests on real emails

A fix for one provider can’t break another.

Metrics

Parsing success, spend, circuit matches.

NEXT

Learn from NOC decisions

Suggest verdicts from past cases.

Maintenance groups

One email, many windows, one group.

Check circuits against TSI

Catch wrong references early.

Four AI terms, in plain words

Temperature

Randomness in word choice. At 0 the model always picks its likeliest answer, so the same email gives the same result.

Structured output

Instead of free text, the model must fill in a fixed form (a JSON schema). If it can’t, we know straight away.

RAG (retrieval)

Before answering, look up similar past cases and show them to the model. It uses our history without retraining.

Frontier vs small models

The largest general models. Small ones can run locally, but didn’t parse well enough in our tests.

CREDITS

The team behind it

First solution draft
Jan kayser

Built & documented
Marek skalicky
Charlotte mulvey
Vinay dahiya
Lukas oržekauskas

Product owner
Viktorija martinkienė

Scrum master
Edward weld

Noc testing & feedback
Stephen ellis
Kamil lusio
Dimitri loizou
Daniel zarracayo
John cory-wright
Henry edwards
Ben bloomfield
Dominykas valys

Workflow review
Anne roberts-mulvey
Mark lloyd
Moyaze shivji
Sion hopes

Maintenance api & tsi updates
Lukas oržekauskas
Dan thorpe
Vytautas bauras

linx

What we learned

The standard is right; adoption isn’t

Few providers send BCOP iCal, and data centres almost never do.

“Supported” isn’t “works for us”

Test a parser library on your own mailbox before you build on it.

The model outlived the parsers

The maintenance API stayed: TSI knows what each service is built from.

Building was the easy part

Follow-ups, multi-window notices, security and ownership took the real effort.

Recommendations for implementers

01 Data model first

Build on a model that knows what every service is made of (your CMDB).

02 A person decides

The LLM drafts, the NOC confirms. Link to monitoring once trusted.

03 Fix the output

Structured output, temperature 0: same email, same answer.

04 Prompt outside code

Tune without a release; test every change on real emails.

05 Plan for follow-ups

Match on the provider's ID; expect reschedules.

06 Guard the edges

Sender whitelist, spend cap, failure alerts, any model.

linx