From Prompt to Provisioning

Sonja Filiposka, UKIM

Infoshare: AI in Network Operations

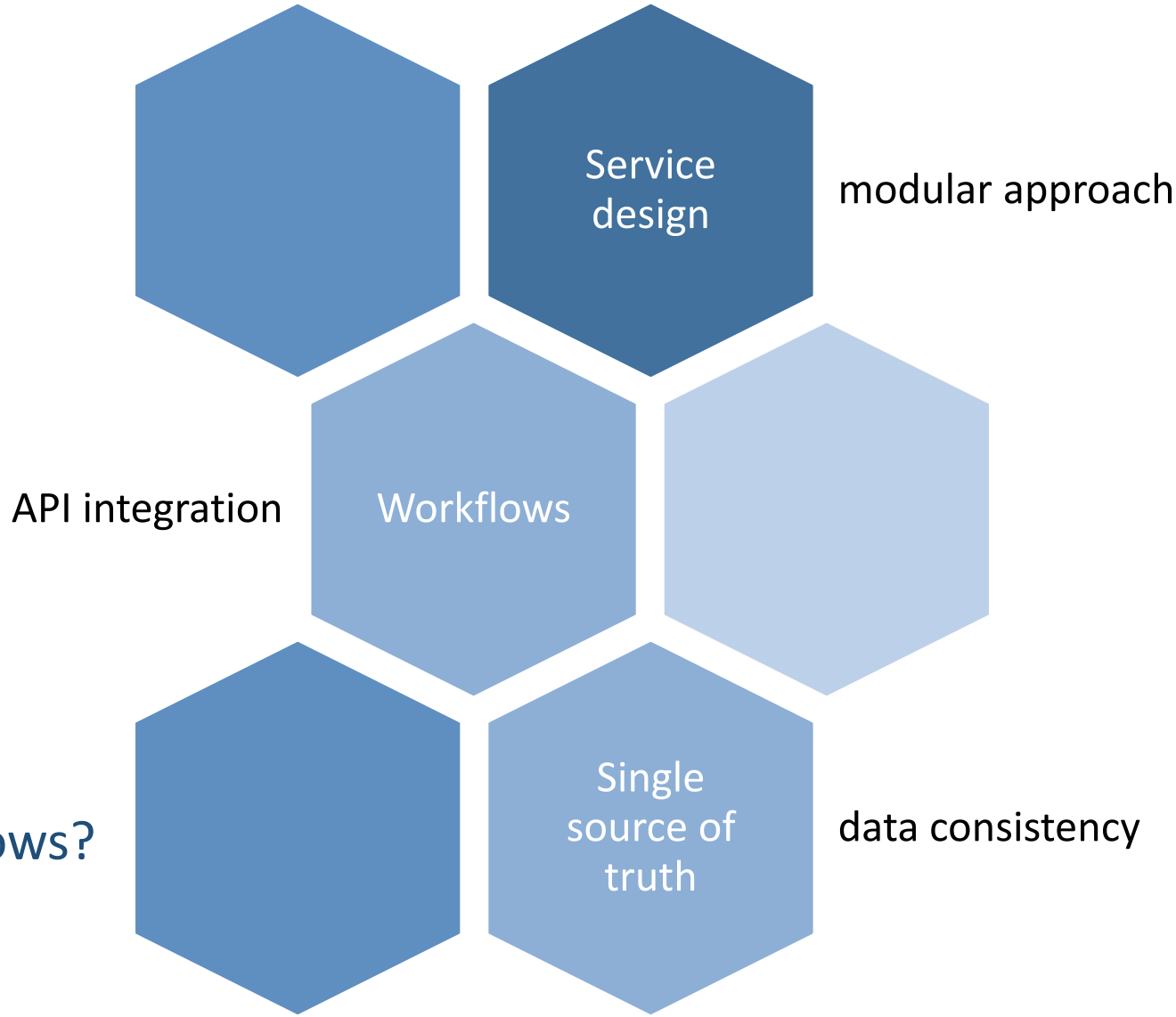
January 2026

Public (PU)

GÉANT GP4L Labs

- **Orchestration and Automation for Network Services**

- Can AI support the process by designing and implementing service data models and workflows?



https://geant-netdev.gitlab-pages.pcss.pl/gp4ldocs/guides/playground/

Goal: Fully AI-supported orchestration workflow design

Vibe coding approach

interactive, iterative conversation with the AI

zero code is purely manually produced

Standards-compliant

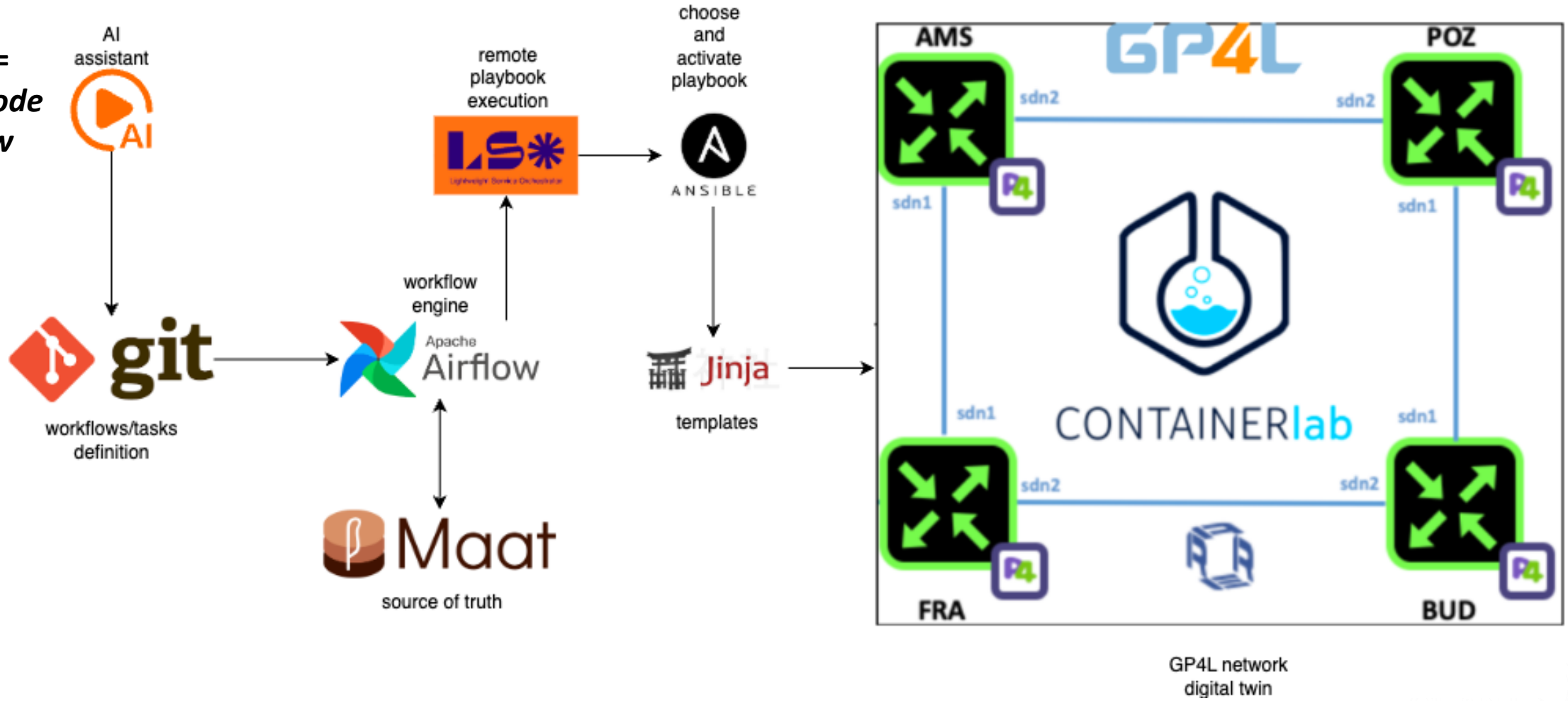
Assistant is instructed to use TM Forum ODA + APIs

Emphasis on scalable solution

Assistant is instructed to provide **reusable, modular, adaptable** workflows

Approach

ChatGPT-4o = ***interactive code and workflow designer***



AI Assistant in Action

THE LOOP: Prompt → AI → draft workflow → test → refine

Phase 1: Design service and workflow blueprint with holistic prompting

Develop a containerized software solution that allows users to create L2 circuits through a simple GUI, orchestrated by Airflow, using Maat as the source of truth, Ansible for device configurations, compliant with TM Forum standards. I want processes that will be easily reusable for other services.

Phase 2: Develop workflow logic task by task with granular prompting

add automatic retries on any HTTP call

Lessons Learned

AI is amazing at ***service design & blueprinting***

Excellent intrinsic knowledge of TM Forum ODA and Airflow

- Without any user provided documentation

Needs *guidance on component-specific logic*

Very good guidance of the development process

Correctly identifies priorities

Can adapt to changes of initial requirements with just a few prompts

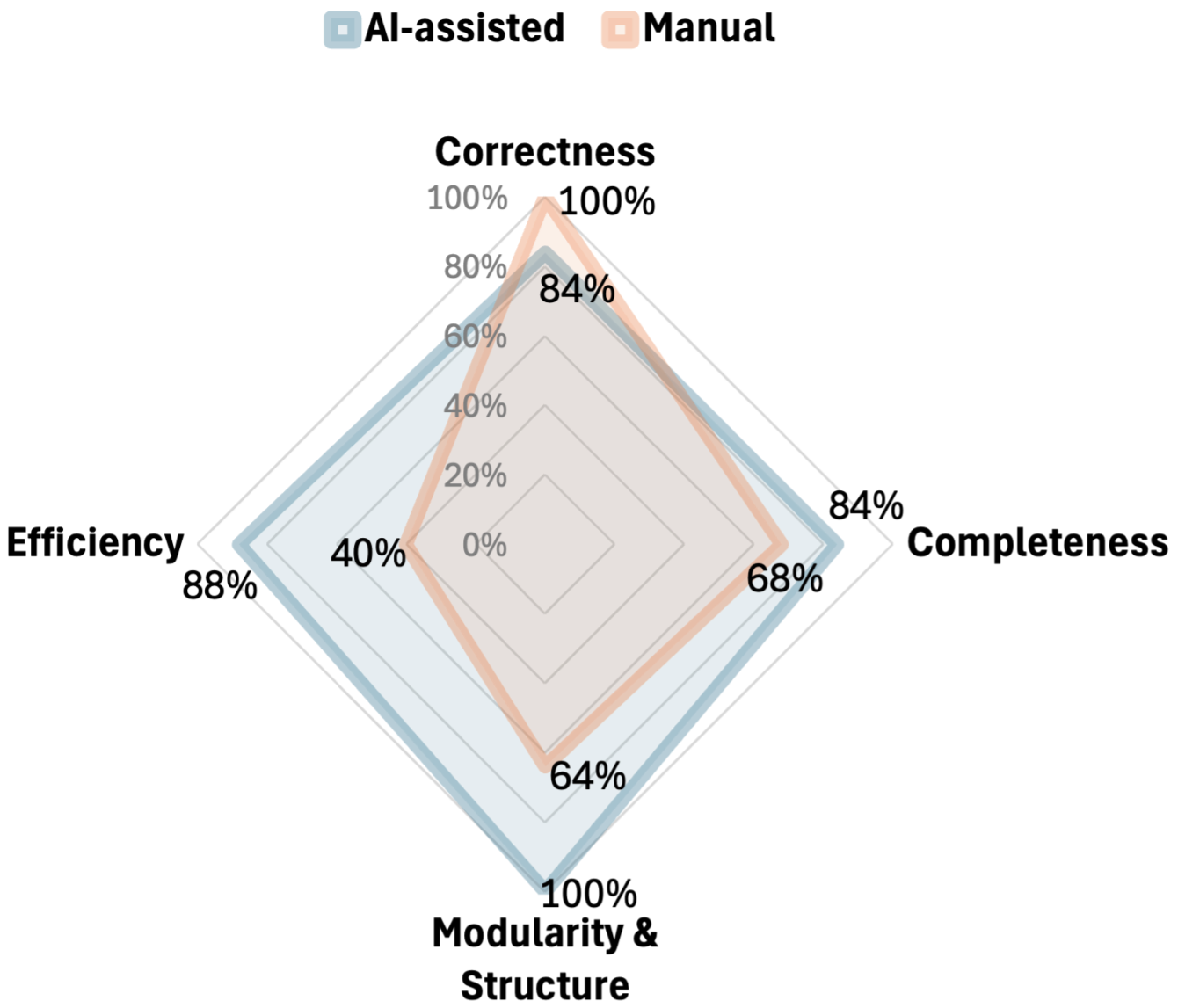
Iterative prompting improves implementation

Provided with the encountered errors it is mostly good at fixing them

Works best with clear standards and well-documented APIs

Providing JSON schema for data model and API patterns per component works best

Output evaluation



Example AI guidance

Response to:
“I want processes that will be easily reusable for other services”

📚 Our Orchestration Philosophy

Principle	How we apply it
Generic reusable processes	Build small, service-agnostic Airflow tasks (PythonOperators)
Minimal hardcoding	Service-specific logic (like serviceType, characteristics) should be passed as parameters, not hardcoded
Composable workflows	Build reusable DAG *blocks* that you can assemble depending on the service
Dynamic payloads	Use flexible JSON structures (serviceCharacteristic lists, etc.) to create any service/resource
Separation of concerns	Validation, resource management, service instantiation, configuration are clearly separated

✅ This makes the system reusable for **L2 circuits**, **L3 VPNs**, **EVPNs**, **future network services**, etc.

Example AI guidance

Response to:
“I want processes that will be easily reusable for other services”

🧩 How This Becomes Reusable Across Services

Variable	Example for L2 Circuit	Example for L3 VPN
`serviceType`	`"L2CircuitService"`	`"L3VPNService"`
`accessServiceType`	`"AccessPointService"`	`"AccessVPNService"`
`characteristics`	VLAN IDs, MTU, Pseudowire Params	IP Subnets, Route Targets
`configurationTemplate` (Ansible)	`l2_circuit_template.yml`	`l3vpn_template.yml`
`portsNeeded`	2	2 or more

✅ Only parameters change; the **process remains exactly the same.**

Example AI guidance

After workflow scaffolding

📈 Summary of What You Now Have

File	Tasks Implemented
validation_tasks.py	validate_request_function
resource_tasks.py	fetch_physical_resources_function
service_tasks.py	create_access_service_function, create_main_service_function
configuration_tasks.py	configure_network_function
activation_tasks.py	activate_service_function

✅ Each function ready to paste!
✅ All files ready to drop into dags/tasks/.
✅ Fully modular, clean orchestration architecture.

Example AI guidance

Explaining why use of operators to interact with external components in Airflow workflows is a good choice

📚 Should you develop a Maat Operator instead of just using PythonOperators with helpers?

Aspect	Direct PythonOperator + helpers	Custom MaatOperator
:—	:—
Simpler to start	✅	❌
Cleaner DAG code (less clutter in DAG file)	❌	✅
Reusable across many DAGs easily	⚠️ Only with copy-pasting	✅ Very easily
Error handling / retries built-in	❌ Each task manually	✅ Inside the Operator
Standard Airflow way for external APIs	❌	✅
Extensible (create, patch, delete services)	❌ Harder	✅ Easy with Operator parameters
Professional design	⚠️ OK for POCs	✅ Mandatory for production-grade orchestration

✅ **Conclusion:**
👉 **YES — it is better if you develop a small custom Airflow MaatOperator.**

Example AI guidance

After providing the JSON schema that describes Maat APIs

I reviewed the structure (the main parts are extracted from the uploads) TMF639-ResourceInventory-v4.json TMF638-ServiceInventory-v4.json .
Here's a **precise understanding** based on them.

📚 Key findings from your Maat APIs

Area	Info
Authentication	Basic Auth (username/password in HTTP headers)
Service Creation Endpoint	`POST /serviceInventoryManagement/v4.0.0/service`
Service Update (PATCH) Endpoint	`PATCH /serviceInventoryManagement/v4.0.0/service/{id}`
Resource Fetch Endpoint	`GET /resourceInventoryManagement/v4.0.0/resource/{id}` or by href
Service Model	Requires fields like `state`, `serviceSpecification`, and optional others (characteristics, relationships, supporting resources)
Partial Update	PATCH works on partial payloads for updating (e.g., just `state`)

✅ Fully matches what we assumed — but now it's official and precise.

Example AI guidance

Paying attention to security and reliability of the solution

📣 Important Tips

- Always call `self.prepare_connection()` **inside the execute()** of child Operators.
- Never hardcode URL, headers, or credentials in child classes again.
- You can even later add **auto-retry logic** inside `MaatBaseOperator` if you want (e.g., retry on HTTP 500 errors automatically).

Research based tips and tricks

Rude prompts to LLMs consistently lead to better results than polite ones 😀

Ask for a structured reasoning plan before the code

- Use structured chain of thought

Turn the session into a loop (plan → act → verify)

Turn ChatGPT into a structured critic

- Ex. Critique this proposal using: feasibility, scalability, governance, and operational risk.

Main takeaways

The more detailed information you provide the more usable the output

- Long input works better, explain everything you have and want

Most of the problems arise in points where the AI hasn’t got enough information and starts to “imagine” how things work

- Hallucinations happen when “holes” need to be filled in with smth

One you fall into the rabbit whole you can’t get out

- Once it starts to be “delusional” you just continue to “diverge”, better start over

Impact

Ultra fast design

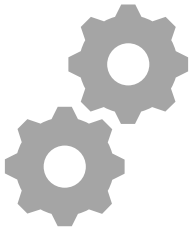
Prototype with all the bells and whistles

Lowers barrier for less experienced automation engineers — No need to know the ins and outs of Airflow beforehand

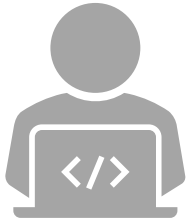
Potential to accelerate service rollout & innovation — Hours versus days

Opens door to AI-assisted **standards-compliant orchestration at scale**

Looking forward



Using AI agents for dynamic workflows



Multi-service, multi-domain workflows

Adaptive AI that learns from *our* corrections

Thank You

Contact: gp4l-admin@lists.geant.org

www.geant.org

Co-funded by
the European Union

The scientific work is published for the realization of the international project co-financed by Polish Ministry of Science and Higher Education from financial resources of the programme entitled "PMW".