Digital Twin as a Service

Testbed 2

Model Orchestration

31-10-2025

Contents

Introduction

Digital Twins provide critical insights for designing built environments, offering a data-driven approach to tackle complex challenges such as housing shortages, climate change, and network congestion. They empower policy makers to create smart and well-considered plans for the future of cities.

Because of the shared nature of the challenges in the built environment, it makes sense to collaborate on the definition and the implementation of what these Digital Twins could look like. This collaborative effort is exemplified by Testbed 2, a collaboration of twelve organisations and initiated by Geonovum to further develop an interconnected architecture for interoperable Digital Twin applications in the Netherlands. Testbed 2 aims to adopt existing components, integrate new ones, and consolidate various initiatives to address complex urban challenges. Building on Testbed 1, it engages new stakeholders and explores technical questions through use cases to clarify user needs.

Authored by Avineon Tensing, this report details a study conducted for Testbed 2. One of the topics of Testbed 2 investigates how calculation models in Digital Twins are instrumental in defining the built environment by enabling simulations and scenario testing. This report focusses on model orchestration, as seamless orchestration between various calculation models is the next step in the maturity of Digital Twins.

While a single model can be highly effective, the real value emerges from linking multiple calculation models together in a chain. An example is integrating a weather model that influences a traffic model, which in return supports traffic rerouting strategies. The first step in tying together these calculation models, accomplished by manual integration, is known as 'glue'. The term used for connecting and managing these different models is 'orchestration'. Currently, the state of model orchestration is evolving. This involves techniques and standards that facilitate effective chaining and integration. In this context, it is important to describe the current landscape of model orchestration, highlighting what is achievable today, the methods or standards that are actively used, and the challenges or limitations faced in implementing seamless model workflows.

Reading guide

This report entails a description, observations and advice regarding research sub-topic 8.2 "Model orchestration, status quo", as set out by Geonovum. The report opens in chapter 1 with providing a list of key concepts and definitions. Next, the report will elaborate on the use case/metaphor that had a central role in the study in the second chapter. The third chapter explores the current technical landscape for model orchestration. In chapter four, the missing elements in model orchestration are explained. Chapter five offers a maturity model for model orchestration and finally, the report closes with future work and recommendations in chapter 6.

Chapter 1: Definitions

Taking into account the research topic, a list of definitions including the OGC API standards and some JSON terms is provided in this chapter. These definitions are drawn up to provide clarity in use. The definitions, in some cases, are according to the interpretation of the researchers and could be up to discussion in different settings outside this report.

Definitions

Digital twins

A Digital Twin is a virtual representation of the real world, including physical objects, processes, relationships, and behaviours. A Digital Twin is inherently a simplified representation or model of the real world. In this report, Digital Twins are discussed exclusively within a spatial context.

Glue code

Glue code is code that connects different software components or modules, enabling them to work together, even if they weren't originally designed to be compatible or to communicate directly. It acts as a bridge or intermediary layer, ensuring that the various parts of a system or software function correctly as a whole. In the context of this research, glue code is inherently not something to strive for, as it requires developer to build and manage it. However, glue code can be of value by allowing various calculation models to be used in sequence.

DCAT

DCAT, which stands for Data Catalog Vocabulary, is a standard developed by W3C to facilitate the description of datasets and data services in data catalogues. It's essentially a way to make data more discoverable and accessible on the web by providing a consistent vocabulary for describing datasets, services, and catalogues, and their relationships. All API standards are extensions of the W3C standards for geospatial data.

Orchestration

Orchestration can have an internal and an external component. The aim of this research is to elaborate on external, or "model orchestration". Internal orchestration is used *within* calculation modules, as it describes the handling of requests within a model or system. It is critical to note the importance of internal orchestration, as it highlights which 'glue' is used internally by different model providers.

External or model orchestration

Model orchestration is the process of automating, coordinating, and managing the flow of data across different systems and applications. The process includes gathering data, transforming it, and passing it on for analysis and decision-making. Model orchestration helps automate the flow of data between tools and systems to ensure organizations are working with complete, accurate, and up-to-date information (Twilio Segment, n.d.).

An objective of this research is to examine whether dataflow can be exchanged between models when one model session is ready, possibly through ETL (extract, transform, load) processes or flowing going back through an orchestration layer.

A probable assumption for this research is that the communication layer between the orchestration repository and designated data models will be through a standardized approach such as API processes.

Internal or job orchestration

Job orchestration, or job 'chaining', is the process of linking multiple jobs together so that they execute in a specific sequence and calculate a specific situation. The output of one job generally serves as the input for the next.

In relation to this research, we define the management of tasks inside a model session as job orchestration.

OGC API standards

OGC API - features

OGC API - Features, also known as OAPIF, is a standard developed by the Open Geospatial Consortium (OGC) for accessing and managing geospatial data on the web through a standardized API.

The standard consists of multiple parts. Part 1, "Core" defines the fundamental API capabilities for retrieving features. It supports geometries expressed in the WGS 84 coordinate reference system. Part 2 extends support to features using other coordinate reference systems, allowing broader spatial representation.

OGC API - processes

OGC API - processes is a standard developed by the Open Geospatial Consortium (OGC) that allows computational tasks to be turned into executable processes accessible through a Web API and be invoked by a client application either synchronously or asynchronously. The standard specifies a processing interface to communicate over a RESTful protocol using JavaScript Object Notation (JSON) encodings.

OGC API - records

OGC API - Records is a multi-part standard that provides a consistent way to create, modify, and search metadata about geospatial resources on the Web. It standardizes how descriptive information about datasets and services is shared and discovered. Part 1 offers basic access and simple search functions, while future parts will add support for more advanced queries and record management features.

JSON

JSON-LD = JSON-Linked Data

JSON-LD is a lightweight Linked Data format. It is a way to create a network of standards-based, machine-readable data across Web sites. It allows an application to start at one piece of Linked Data and follow embedded links to other pieces of Linked Data that are hosted on different sites across the Web.

JSON – schema

JSON Schema is a vocabulary that provides a format for defining the structure and constraints of JSON data. It allows you to specify what a JSON document should look like, including the data types of its properties, whether certain properties are required, and even more complex rules like data ranges or regular expressions for string values. Essentially, it acts as a blueprint or contract for your JSON data, ensuring consistency and enabling validation.

Chapter 2: Orchestration in a Digital Twin context

In order to make the concept of model orchestration in relation to Digital Twins more understandable, an analogy is used to describe the current situation as well as the desired situation. Model orchestration is like asking a librarian what books to read to get information on a certain topic.

The library: what books are in it and where to find them

Let's start with the library. At the heart of the library is a box of catalogue cards, describing all the items available in the library. This box of catalogue cards is the metadata catalogue, described for instance using DCAT. Each card tells you about a book: what it is about, who wrote it, and where to find it. In data terms, each card is a metadata record about a dataset or service. If you're looking for specific items in the library, you can ask a librarian. You don't need to flip through all the cards by hand. Asking the librarian is like using OGC API Records to query the catalogue. Once you've found the right card, it tells you in which row to find the book. Or, in the context of this study, where to find the dataset.

A book: what information is in it?

From here you may proceed with the dataset as you wish. If you want part of the dataset (*"pages 1–50 of a book"*), you use OGC API Features or if you want to do something with the dataset (e.g. *"summarize this book"*), you use OGC API Processes.

Anthology: what books are related to each other?

When you query using OGC API Records, you usually only get pieces of the puzzle. Asking *"Show me everything about science fiction"* won't give you the full picture in one go. You often have to query multiple times, knowing what to query, stitch the results together and figure out the order yourself.

Some calculation models of Digital Twins can already act like a local librarian, helping you navigate within their own collection (or *"book"*) by internally combining different jobs, for instance to highlight the differences between different model runs.

What is currently missing, is a system or catalogue, that contains information about the different calculation models and how to combine them (or *"what books do I need to read to know everything about Roman history?"*). If you want to combine knowledge from different models, you still have to do the hard work manually, together with the experts of the calculation model: run models separately, interpret the results, rebuilding model output to fit the next models' input and hard-coding orchestration workflows.

External model orchestration, meaning a system or catalogue that can manage different calculation models, would be like a librarian saying *"To understand Roman history, read these four books, in the following order: [..]."* Model orchestration, in any way or form, would provide a complete, structured workflow across calculation models (from different organizations) without glue code.

Chapter 3: Current situation

This chapter examines the current technological landscape of model orchestration for Digital Twins. It analyses available open- and closed-source technologies and standards, and concludes with a comparative suitability matrix.

What technologies are available?

The goal of model orchestration is to let the APIs of different organizations collaborate with each other. The following paragraphs discuss different technologies available in the market and their suitability for orchestration and implementation. Descriptions of the technologies used in this chapter are available in Appendix I.

For a comprehensive overview, the technologies that are available (both open and closed source) will be placed in one of the following categories:

- Workflow management systems: these are systems designed to manage and automate processes.
- Messaging queues: these are trigger-based systems where platforms can subscribe to.
- Custom solutions: these are methods designed with a specific goal in mind.

Please keep in mind that the lists in this research are not complete. They provide insight into common technological frameworks. For a more extensive technological overview, additional research must be conducted.

Workflow management systems

Platforms that manage, automate, and orchestrate workflows, sequential or conditional chains of tasks.

Technology	Type	Short Description	Source
FME	Closed source	Visual ETL platform, strong in geospatial transformations and workflow automation	(Safe Software, n.d.)
Apache Airflow	Open source	Python-based DAG scheduler for batch workflows and pipelines	(Apache Airflow, n.d.)
Camunda	Open source / enterprise	BPMN-based workflow engine focused on business process modelling and governance	(Camunda, n.d.)
Temporal	Open source	Code-driven, distributed workflow engine for reliable, long-running processes	(Temporal Technologies, n.d.)
Node-RED	Open source	Visual, low-code, flow editor for lightweight integrations and data tasks	(Node-RED, n.d.)
n8n	Open source	Low-code workflow automation tool for connecting APIs and services	(n8n, n.d.)
Zapier / Make	Closed source	No-code integration platforms, good for simple automations	(Zapier, n.d.; Make, n.d.)

Messaging queues

Trigger-based or event-driven systems enabling asynchronous communication between services.

Technology	Type	Short Description	Source
Apache Kafka	Open source	Distributed event streaming platform, highly scalable and fault-tolerant	(Apache Kafka, n.d.)
RabbitMQ	Open source	Lightweight message broker, supports multiple messaging protocols	(RabbitMQ, n.d.)
Azure Service Bus / AWS SNS/SQS	Closed source	Cloud-based messaging queues for service communication	(Microsoft, n.d.; Amazon Web Services, n.d.)
MQTT	Open standard	Lightweight protocol used in IoT; useful for device-data streaming	(MQTT, n.d.)

Custom solutions

Code-based approaches tailored to specific needs that offer flexibility but require more technical effort.

Technology	Type	Short Description	Source
Python	Open source	General-purpose language with rich libraries (e.g. Pandas, Geopandas, Flask) for custom integrations	(Python Software Foundation, n.d.)
NestJS	Open source	Scalable Node.js framework (TypeScript) used to build custom APIs and orchestration layers	(NestJS, n.d.)
Java/Go	Open source	Used in platforms like Temporal; powerful for building robust backend services	(Oracle, n.d.; Google, n.d.)
Jupyter Notebook	Open source	Useful for prototyping custom data processing or model interaction logic	(Project Jupyter, n.d.)

What standards are supported?

This section compares key tools used for data workflows, orchestration and automation, focusing on the standards they support. Each tool varies in its alignment with industry standards such as API protocols, spatial data formats, process modelling languages, and data serialization formats. The table below provides a concise overview of how each tool aligns with these categories, highlighting their strengths and limitations in relation to standards.

Tool	Workflow Standards	API/Data Standards	Spatial Standards	Authentication	Comments
FME	–	REST, JSON, XML, CSV	OGC (WMS, WFS, GML, GeoJSON, KML), ISO 19115	–	Leading in geospatial data and transformation standards

Node-RED	–	REST, JSON, XML, OpenAPI	GeoJSON	–	Great for lightweight API integration and visual flows
NestJS	–	REST, OpenAPI, JSON Schema	Optional (OGC if added)	OAuth (via libraries)	Strong for custom API logic, scalable backends
Airbyte	–	REST, JSON, CSV, Parquet, Avro	–	–	Focused on data ingestion into warehouses
Qlik	–	SQL, REST, JSON	–	–	Data access and visualization only
Temporal	Internal state protocols	REST, gRPC	–	–	Built for reliability; no external standards required
Camunda	BPMN 2.0, DMN, CMMN	REST, JSON	–	–	Best for formal business process modelling and compliance
Apache Airflow	Python-based DAGs (custom)	JSON, CSV, Avro, Parquet	–	–	Strong for batch jobs and ETL; no formal workflow standards
iPaaS (Zapier, etc.)	–	REST, JSON, Webhooks	Limited (GeoJSON in some cases)	OAuth 2.0, OpenID Connect	Great for SaaS integration and event-driven automation

Model orchestration suitability matrix

To determine the suitability of the different technologies in model orchestration, a qualitative analysis on their usability, scalability, manageability, and openness is conducted. In the matrix below shows the analysis on the four qualities for each of the mentioned technologies.

- Usability: what is the required technical level to build an orchestrator?
- Scalability: is the platform able to cope with a varying request rate?
- Manageability: in what way does the platform allow for edits on the different integrations?
- Openness: in what way does the platform support various standards and techniques, aligning with the key principles of the digital twin architecture designed by Geonovum?

Tool	Usability	Scalability	Manageability	Openness
FME	High – Low-code, little coding needed	High – FME Server scales well	High – Visual workflows easily editable	High – Supports OGC, ISO 19115, WMS/WFS
Node-RED	High – Very accessible, drag-and-drop, no coding needed	Medium – Good for prototyping, not heavy load	Medium – Easy to update flows, version control limited	Medium – REST, JSON, OpenAPI, GeoJSON

NestJS	Low – Developer-level, strong TypeScript/NodeJS needed	High – Built for scalable microservices	High – Modular, maintainable codebase	Medium – REST, OpenAPI, can support OGC manually
Airbyte	Medium – UI-based setup, but some technical setup required	Medium – Good for regular ingestion tasks	Medium – Connector management via UI or config files	Medium – Supports API, JSON, CSV, Parquet, Avro
Qlik	High – Business-friendly UI, no coding required	High – Scales well with enterprise setups	High – Strong GUI-based model and dashboard editing	Low – Visualization only; not built for standards exchange
Temporal	Low – Requires developer expertise (Java/Go)	High – Built for fault tolerance and scale	Medium – Code-driven; debugging needs skill	Low – No formal external standard support
Camunda	Medium – Visual modeling (BPMN) but setup/config needs skill	Medium – Depends on infrastructure	High – BPMN diagrams are maintainable, versionable	High – BPMN, DMN, CMMN compliance
Airflow	Low – Python skills required	High – Excellent for batch pipelines	Medium – DAGs manageable in code; UI for scheduling	Medium – Data formats (CSV, JSON), no BPMN
iPaaS (Zapier, n8n, Make)	High – No-code or low-code visual interfaces	Medium – Scales for small/medium tasks	High – GUI-based, editable flows	Medium – REST, Webhooks, OAuth, OpenAPI

Chapter 4: What is missing

Chapter 3 provided a current state of technologies and standards. Clearly, the technology field for working with API's or other types of services is already quite vast. With various difficulty levels, a technology platform can be used that connects with one's level of expertise. With the current technological state, orchestration between different calculation models is something that can take place, albeit by custom integrations.

However, technological implementation of model orchestration is only part of the equation. Every calculation model is designed to cater different needs, has different inputs, and 'calculates' in different ways. This means that the understanding of a model's workings and output is another critical component in the orchestration-field. Especially when the output from for a calculation model (e.g. model A) is the input for another calculation model (e.g. model B), it is crucial to know how the output from model A can be interpreted and how it is used in model B.

At this point, how calculation models work is only available by involving model developers. When combining different calculation models (thus orchestrating), these model developers have an active role to guide the orchestration process. In order to move towards automated orchestration, an additional information layer is needed that is currently missing.

Service catalogues tell us what services are available and how to access them. They do not tell us how to correctly understand and implement them or how they can be used in another model. To move towards automated orchestration, this gap in information needs to be resolved.

In addition, calculation models can require various steps to be taken before a model can actually calculate something. For instance, when running a model to check the effect of a new speed limit of roads, first all road items in the model need to be updated with the new speed limit. This use of models is quite common, and it requires additional knowledge of how to set up a model before it is actually run. This information, which is also required when chaining various models together, is not available in a common system or knowledge base.

So technically a model can be run and with human intervention, model output can be downloaded, extracted and uploaded in a new model. This is orchestration, but not the type of orchestration that allows for automatic and mature Digital Twins. To remove manual actions, a common knowledge base is required containing information on how to set up and run a model and how to interpret the results.

In other words, service catalogues now only answer: "What is available?", but not yet "Given my question, what do I actually need, and how do I put it together?" That's where the knowledge catalogue comes in.

The knowledge catalogue could act like a recipe book: a central space where organisations publish information about model components, how they can be operated and used in different models – also published to the knowledge catalogue. Even though OGC APIs and DCAT already provide a strong foundation, several pieces are still missing before a mature knowledge catalogue and orchestration can be realized:

Knowledge catalogue (central space for data publication)

- The service catalogue (information on what exists) is already in place in many cases: it describes what datasets, features, and processes exist and how to access them.
- Missing: a knowledge catalogue, a central space where organisations can publish models and information on how to use them (*"the recipes"*). It's the librarian's brain, connecting prompts to datasets, features, and processes, and suggesting the right sequence.
- It answers:
    - *How do I know what is inside a model?*
    - *How do I know what to use form that model?*
    - *How do I use it (via API Processes, etc.)?*

External orchestration (between organisations)

- Some internal orchestration already exists; certain models guide users within the context of that specific model.
- Missing: overarching orchestration across organisations. The ability to take a user's question and coordinate across multiple knowledge catalogues, stitching together results into one coherent workflow.
- Without this, users still need to manually glue results from different organisations.

In other words, knowledge catalogues can provide information on what model components to use in order to calculate a specific, dynamic scenario. Or to use the terms of the analogy: to understand Roman history, first read chapters 1 and 2 from book A, then chapters 3 and 6 from book B.

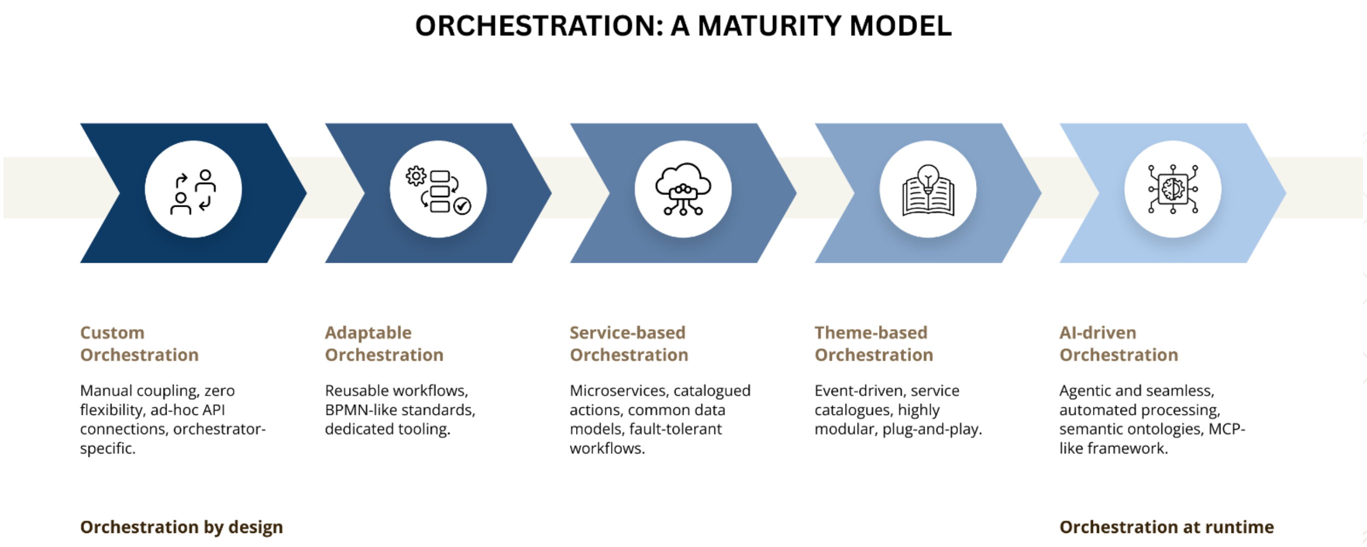
Chapter 5: Model orchestration maturity matrix

Based on the findings of what's missing, the technical capabilities of the tools discussed in this report (FME, Temporal, Camunda, Airflow, etc.) and various sessions with technology providers, this chapters elaborates on different levels of maturity of orchestration. Maturity of orchestration is measured using the following four indicators:

- Orchestration type: *How does the integration happen?* Focuses on the method of orchestration, moving from scheduled tasks to real-time, event-driven, and intelligent systems.
- Governance & structure: *How is the orchestration formally defined, managed, and shared?* Focuses on the adoption of formal standards (like BPMN) and mechanisms for sharing (like DCAT/Catalogues) to achieve interoperability.
- Reliability & resilience: *How trustworthy and robust is the workflow execution, especially under failure?* Focuses on the system's ability to handle long-running processes, failures, and maintain state, moving away from custom code.
- (Geospatial) integration: *How natively, precisely, and intelligently is the specialized (geospatial) data handled?* The specific ability to handle complex spatial data (or domain data in general) and transformations with required precision and standards (e.g., OGC, W3C, FME capabilities).

Maturity model for model orchestration

Using these indicators, 5 levels of orchestration maturity can be determined. The maturity model ranges from custom glue code to a fully agentic orchestrator. Each of the maturity levels is ranked based on the 4 indicators. The maturity model is depicted in in the image below. In the remainder of this chapter, the individual steps of the maturity model are illustrated.



ORCHESTRATION: A MATURITY MODEL
Custom Orchestration
Manual coupling, zero flexibility, ad-hoc API connections, orchestrator-specific.
Adaptable Orchestration
Reusable workflows, BPMN-like standards, dedicated tooling.
Service-based Orchestration
Microservices, catalogued actions, common data models, fault-tolerant workflows.
Theme-based Orchestration
Event-driven, service catalogues, highly modular, plug-and-play.
AI-driven Orchestration
Agentic and seamless, automated processing, semantic ontologies, MCP-like framework.
Orchestration by design
Orchestration at runtime

Level 1: Custom orchestration

Integration is a necessary evil, implemented reactively using manual coding or simple, point-to-point connections. Workflows are tightly coupled to specific technologies, making them expensive to maintain. Highly dependent on manual intervention.

Orchestration is a human-to-human activity that needs to be developed at the design stage of a Digital Twin. It has zero dynamic capabilities.

Focus: Solving immediate, localized integration issues.

Orchestration type	Manual coupling / glue code. Ad-hoc API connections with no intelligence; often point-to-point.
Governance & structure	Zero flexibility. Orchestrator-specific.
Reliability & resilience	High maintenance. Manual coding. Failures require manual intervention.
Geospatial integration	Minimal / external. Spatial tasks are executed manually or in silos; custom code must be written for all spatial format conversions.

Level 2: Adaptable Orchestration

First steps into standardization. Workflows are reusable and are governed by a defined schedule (e.g., nightly batch jobs using tools like Airflow). The introduction of open standards (like OGC APIs) and visual modelling (like BPMN) make the process transparent, but execution remains manual or scheduled.

Focus: Standardization and reusability for planned data processing.

Orchestration type	Reusable workflows (scheduled). Python DAGs (Airflow) or simple iPaaS flows for batch/scheduled jobs. Focus on a reusable workflow.
Governance & structure	Open standards and BPMN. Open standards (e.g., OGC APIs, JSON) are adopted. Workflows often use BPMN for visual design (Camunda).
Reliability & resilience	Basic retries/state. Individual tasks contain basic retry logic, but state management is simple.
Geospatial integration	Specialized tooling. FME is introduced when precision and transformation are key. Geodata is exchanged as files or basic GeoJSON.

Level 3: Service-Based Orchestration

The focus shifts from workflows to services and reliability. Processes are broken down into microservices with catalogued actions and defined common data models. Dedicated engines like Temporal are adopted to ensure fault-tolerant, long-running workflows with guaranteed state, providing a robust, enterprise-grade backbone. Interoperability allows the systems on a network to share, exchange, combine and use data with minimal human interaction.

Focus: Reliability, service composition, and process management.

Orchestration type	Workflow-based (microservices). Processes are modelled visually (BPMN/Camunda) or built as microservices for defined sequences.
Governance & structure	Common data models and catalogued actions are defined. OGC API Records/DCAT used to discover the 'what'.
Reliability & resilience	Fault-tolerant workflows. Use of dedicated engines like Temporal for reliable, long-running workflows with retries and failure handling.
Geospatial integration	Spatial services are exposed as interoperable services using OGC API - Features and OGC API - Processes.

Level 4: Theme-Based Orchestration

Orchestration is treated as a strategic, systemic capability. The architecture is highly modular and event-driven, enabling plug-and-play integration. Services are published in a comprehensive service catalogue, and data is enriched with linked data principles, allowing processes to be discovered and composed based on a thematic understanding of the geospatial environment.

Focus: Interoperability, discoverability, and data context.

Orchestration type	Event-driven (plug-and-play). Real-time, asynchronous communication (e.g., Messaging Queues) with plug-and-play and modularity.
Governance & structure	Service catalogue and linked data are used for full discoverability. Focus shifts to a 'knowledge catalogue' to combine recipes.
Reliability & resilience	Built-in system-level fault tolerance across services, with high modularity minimizing single points of failure leads to high availability of the system.
Geospatial integration	Data-driven and thematic. Geospatial processes are part of broader themes. Location data is treated as linked data for contextual queries.

Level 5: AI-Driven Orchestration

The ultimate state, where orchestration is fully automated, intelligent, and self-optimizing. The system uses semantic ontologies to understand the purpose of models and utilizes the Model Context Protocol (MCP) to seamlessly integrate AI. A user can request a high-level outcome ("chat-to-orchestrate"), and the system intelligently orchestrates the necessary sequence of services and models across domains.

Orchestration is a machine-to-machine activity that is fully dynamic.

Focus: Full autonomy, proactive execution, and strategic value creation.

Orchestration type	Agentic and seamless. Orchestration is AI-driven and seamless, automatically sequencing processes across organizations to answer a user prompt.
Governance & structure	Semantic ontologies allow the system to understand model relationships. This stage utilizes the Model Context Protocol (MCP) to enable the necessary integrations and uses a comprehensive knowledge catalogue.
Reliability & resilience	Self-healing. Workflows are orchestrator independent and self-adapting; system can reroute or compensate for service failures automatically.
Geospatial integration	Automated geo-selection. The system automatically identifies and orchestrates the correct OGC API Processes and feature layers required to answer a geospatial query (e.g., "Find suitability for foundations").

Chapter 6: Conclusion

The proposed maturity matrix is the first step to benchmark model orchestration and to have a common understanding to assess tools and solutions within a shared framework. In order to achieve the highest level of model orchestration there are missing elements: a knowledge catalogue and an overarching orchestration platforms across organisations. Improving the maturity level of model orchestration in the use of Digital Twins is an essential step from manually controlled systems to machine automated systems, making their use more efficient and usable. Future work is recommended to iterate on this maturity model and refine it into an adopted model.

Appendix I

1. FME (Feature Manipulation Engine)

FME is a powerful tool specifically designed for workflows involving spatial or geospatial data. It is particularly effective when tasks include spatial transformations, integrating different geodata sources, and creating visual workflows. The platform offers a user-friendly, visual ETL environment through FME Workbench, where workflows are built by linking various Transformer components. It supports integration with external systems via its HTTPCaller transformer, which can connect to any REST API and directly process JSON or XML responses within the workflow.

With FME Server, workflows can be automated to run on predefined schedules or triggered by specific events, making it a good solution for continuous or large-scale data processing. A drawback is the cost. As a commercial product, both FME Desktop and FME Server can be expensive to license and maintain.

2. Node-RED

Node-RED is a flexible and user-friendly option for building visual API workflows and developing proof-of-concept solutions, particularly when dealing with simpler use cases. It is a flow-based visual programming tool built on NodeJS, where APIs and services can be connected using intuitive drag-and-drop nodes where no coding is required.

It supports formats like JSON and GeoJSON, it is not specifically designed for advanced geospatial processing. However, it performs well for basic spatial tasks. The platform also allows automation of workflows based on schedules or triggered events, making it suited for lightweight automation and testing in small to medium-sized scenarios. There are a few limitations. It lacks advanced spatial analysis capabilities and is not built to support highly complex or large-scale systems that require robust scalability or fault tolerance.

3. NestJS

This is a developer-oriented framework designed for building scalable and maintainable backend services. Built on NodeJS and TypeScript, it provides a foundation for developing server-side applications, APIs, and orchestration layers.

An advantage is the architecture, which makes it easier to build reliable and modular APIs. It excels in scenarios where multiple API calls need to be integrated or when orchestration logic must be encapsulated within a custom service.

Typical use cases include creating custom orchestration APIs, backends for data processing, or microservices that manage complex, long-running workflows. However, using this framework does require programming expertise in TypeScript and NodeJS.

4. **Airbyte**

This tool is focused on large-scale data ingestion, rather than real-time workflow orchestration. It is an open-source ETL platform designed to replicate data from APIs and other sources into data warehouses or similar storage destinations. Its strength lies in its ability to continuously and reliably sync data from external APIs into centralized storage systems. This makes it particularly effective for populating data lakes or warehouses where the data will be used later for analysis or reporting. However, it is not built for handling complex orchestration logic, chaining API calls, or managing event-driven, real-time workflows. Its functionality is limited to data extraction and loading, with little support for conditional logic or flow control.

5. **Qlik**

This platform is primarily focused on business intelligence and data visualization. It is designed for building dashboards and exploring data through visual analytics. Its main strength lies in presenting and interpreting results, making it ideal for reviewing the outcomes of data processing or orchestration workflows. However, it is not intended for managing workflows, integrating APIs, or executing automation logic.

6. **Temporal**

This is a fault-tolerant, distributed workflow engine designed for managing complex, long-running processes. It has a code-centric approach and enables the development of workflows with built-in support for retries, error handling, and distributed execution.

The platform ensures that critical workflows can be completed successfully, even in the face of system failures or restarts. This makes it particularly well-suited for orchestrating complex API interactions that require robust state management and precise recovery mechanisms.

However, it does require significant programming knowledge and is more appropriate for engineering teams with development expertise.

7. **Camunda**

Camunda is an enterprise-grade platform built around BPMN (Business Process Model and Notation), designed for modelling and managing formal business workflows. It enables users to visually design processes using BPMN diagrams, which makes it useful for scenarios requiring clear process documentation, governance, and human task management.

Its strengths lie in visual clarity, support for manual approvals, and robust process monitoring. Camunda is well-suited for workflows that need to comply with formal business rules or governance frameworks. However, it's less effective for high-frequency API orchestration or highly distributed, technical workflows.

8. **Apache Airflow**

Apache Airflow is a widely adopted open-source tool for scheduling and orchestrating complex data pipelines. Written in Python, it is suited for managing batch processes with clear dependencies, retry logic, and monitoring features.

It is effective when workflows need to run at scheduled intervals, such as daily ETL jobs or data syncs. However, it's not well-suited for real-time or event-driven API orchestration. It also requires technical setup and maintenance, making it more appropriate for teams with development expertise.

9. **iPaaS (Zapier, Make, n8n)**

These integration platforms (Integration Platform as a Service) offer low-code or no-code environments for connecting APIs and automating simple workflows. With drag-and-drop interfaces and prebuilt connectors, tools like Zapier, Make, and n8n make automation accessible to non-developers.

Their key advantage is speed and ease of use. Suited building MVPs, small-scale automations, or quick integrations. However, they're limited in handling complex logic, and costs can increase quickly at scale. Reliability and fault tolerance may also fall short in larger, enterprise-grade scenarios.

AVINEON TENSING

www.avineon-tensing.com