texolution

i2ai / i2store Walkthrough

Dr.-Ing. Boris Horner

Overview

What i2ai / i2store does

- Extensible middleware for AI models (LLM, speech-to-text, image processing, ...).
- Extensible middleware for arbitrary transforms and I/O with other systems (RDBMS, CMS, CRM, ERP, ...).
- Sophisticated API with login, exposing **all** functions of the server.
- Integrate and expose the AI-optimized storage engine **i2store** combining metadata, graph and full-text search with vector ranking.
- Provide a web GUI for administration and custom plugins.
- User / group / permission management.
- Generic prompt editor and process editor and runtime, enabling power users to build new use cases from reusable, elementary functions.
- Using the metadata, relation, text and vector management, powerful Content Delivery Portals can be easily built by adding a GUI.

What i2ai / i2store does not

- **i2ai** neither provides its own AI models nor does it offer training or fine-tuning infrastructure; local models can be hosted in the network and used by **i2ai**.
- **i2store** is not meant as a replacement for authoring environments, even though it could be used. It is perfect for managing AI knowledge, CDP data etc. with deep semantics.

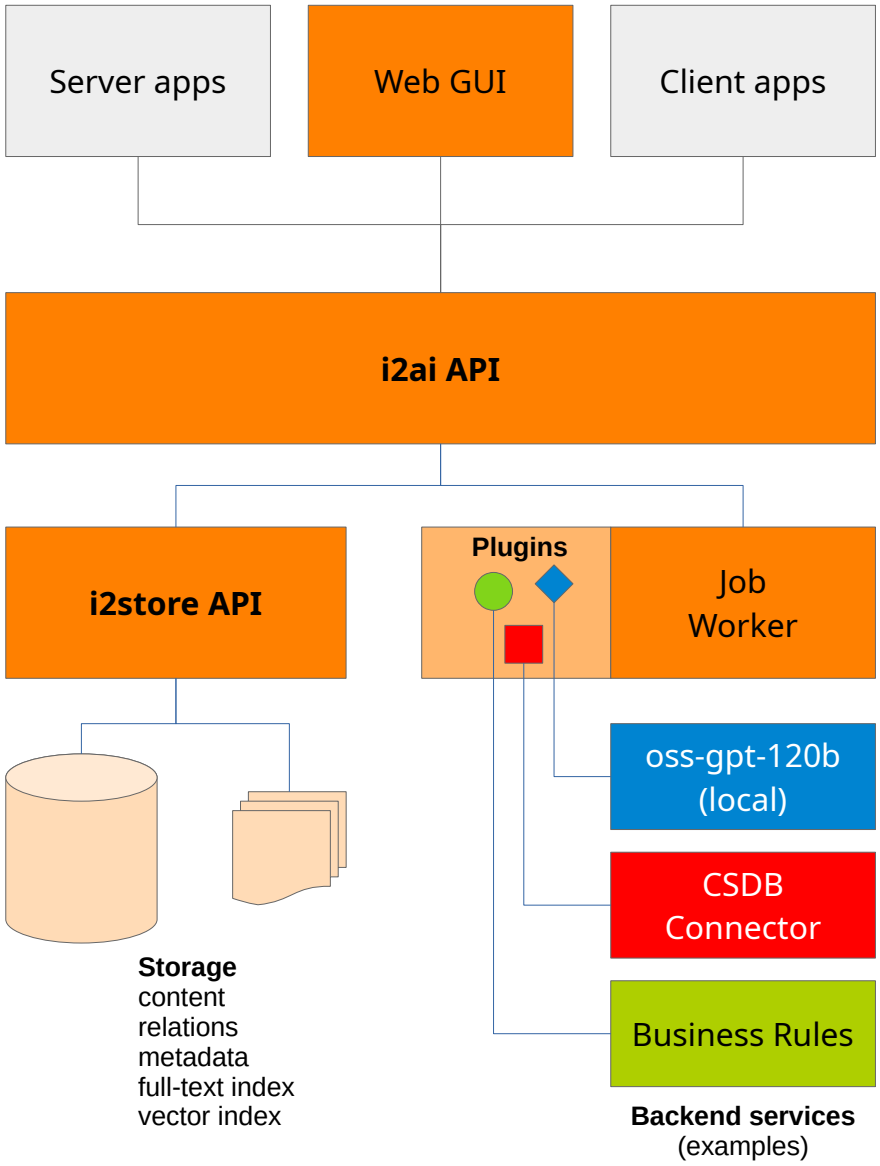
Architecture

i2ai API

- The core part of the system is the API that can be addressed by standard HTTP.
- The API exposes commands as needed, to enqueue requests and fetch results etc.
- Storage is used by the API and exclusively accessible through the API.
- API use is restricted by a token; optionally user management and permissions.

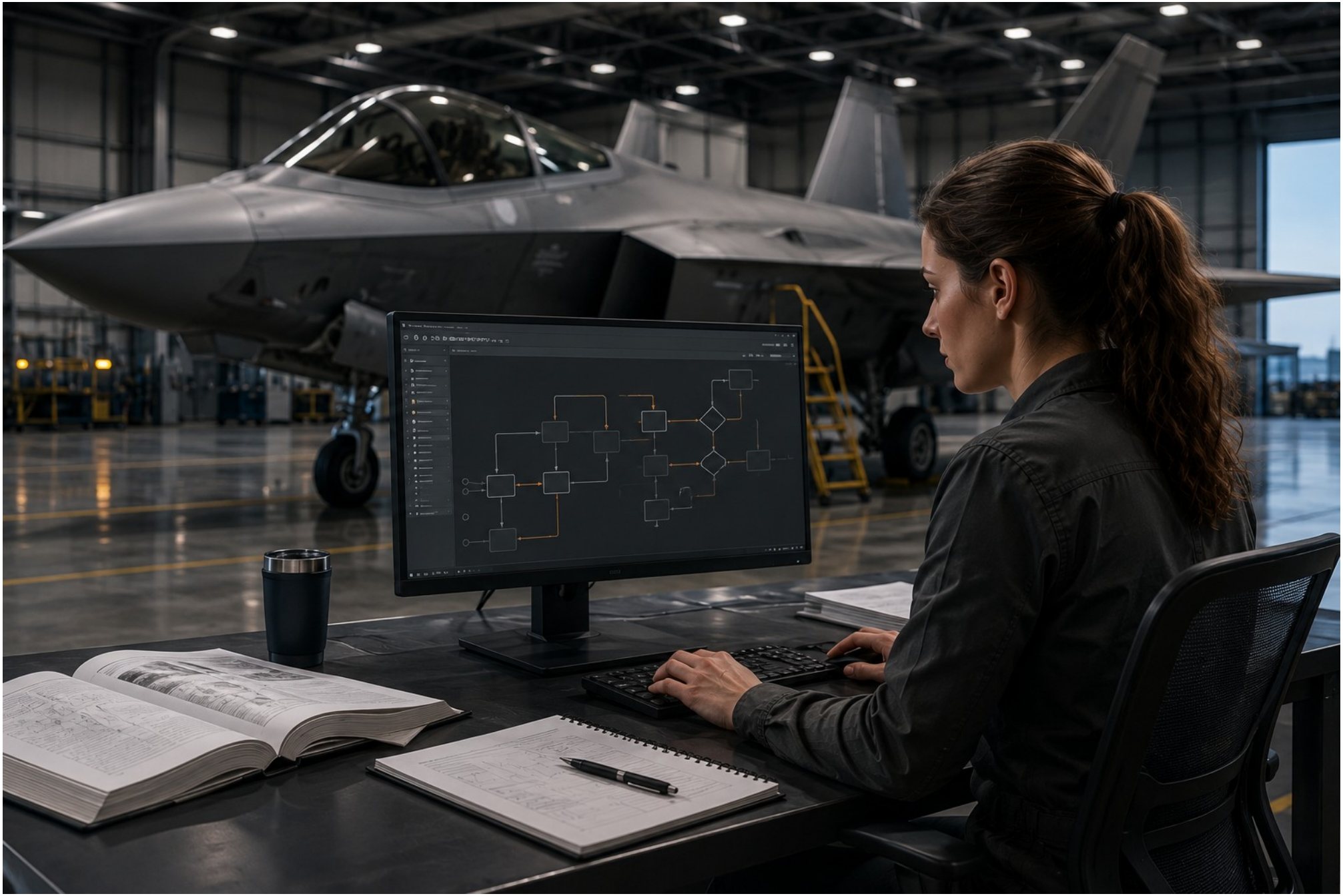
Job Worker

- The Job Worker executes enqueued tasks and sets the status to ready (or failed) when done.
- The various atomic functions are provided as plugins; new plugins can be added.
- Plugins can have dependencies like local or remote AI models or server apps.
- Concurrency behaviour and capabilites can be configured per resource.



texolution

User perspective



What users see

i2ai / i2store expose their entire functionality under a complete API. There is a standard web application implementing GUI for all administrative tasks and a plugin interface for custom GUI, but the frontend app only takes input and displays results.

Thus, all functions are available in the standard GUI, but generally will be integrated elsewhere, in addition to the web GUI or exclusively:

- New, custom web interface (with or without login)
- Extension of an existing web application
- Client tools like XML editors, review frontends, ERP clients, …
- Server processes

What the user sees in detail is determined by the integration.

In our example, we'll show a simple RAG application based on documentation for the medical imaging standard DICOM (Digital Imaging and Communications in Medicine). We'll use the standard web application with a specific RAG viewer plugin.

Other applications (like business rules validation in defense, technical authoring support, data analysis and anything else) can be built using the same techniques.

texolution

What does RAG mean?

RAG stands for „Retrieval Augmented Generation“.

That means, the Large Language Model (LLM) does not respond (**generate**) directly to the user’s question. Before it answers, the process **retrieves** matching content and uses it for a fact-based, improved (**augmented**) response.

Typical RAG process (AI-based steps: ■)

- Translate user input to search terms for matching data ■
- Execute the search and retrieve matches with URLs and summaries
- Rank search results according to summaries / remove irrelevant ones ■
- For all remaining results:
  - Fetch content
  - Filter content to leave only the sections relevant to the user input in place ■
  - Join all results in a JSON structure
- Answer the user input based on the retrieved content ■

Later, we’ll see how such a process is modeled.

RAG is only one example – any kind of functionality can be built.

texolution

What are the alternatives?

Most people use AI in their daily life and know how helpful it can be, and wish to use it for their professional tasks, too.

So decision-makers basically have three options with respect to AI use:
1. Tolerate employees using AI services they are familiar with.
2. Restrict 1.
3. Analyze the internal processes and provide controlled AI infrastructure.

Option 1 is undesirable, because every work day, confidential information and intellectual property go where they shouldn't go („Shadow AI"), and the models and prompts used are totally uncontrollable.

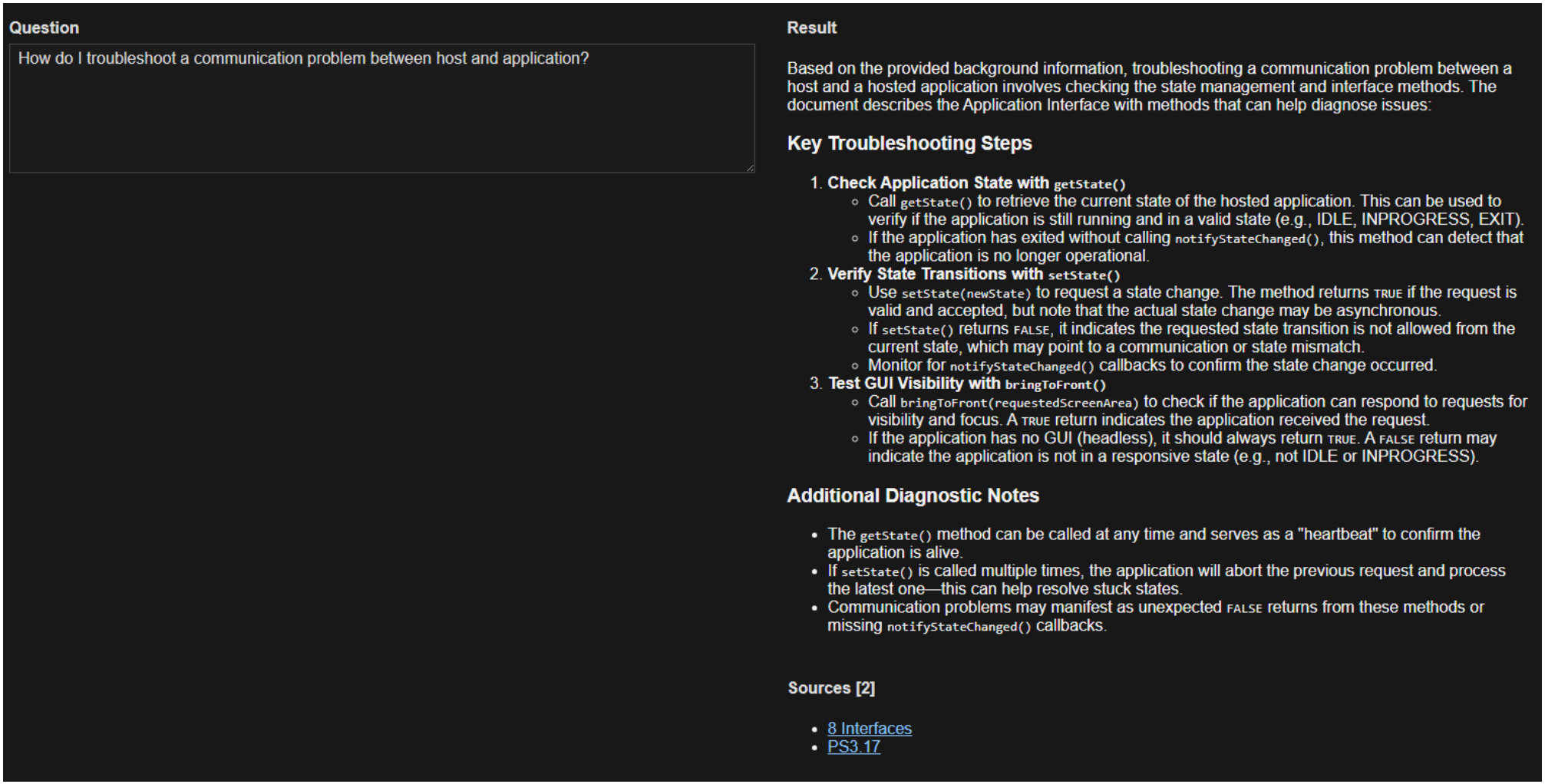
Option 2 (if it can be 100% implemented) solves the Shadow AI issue at the cost of efficiency.

Option 3 fulfills the need for AI based automation, but data protection and standardization of prompts and other process steps are **by design**.

Again, all this is not only true for RAG, but for any AI process.

texolution

Simple RAG GUI example



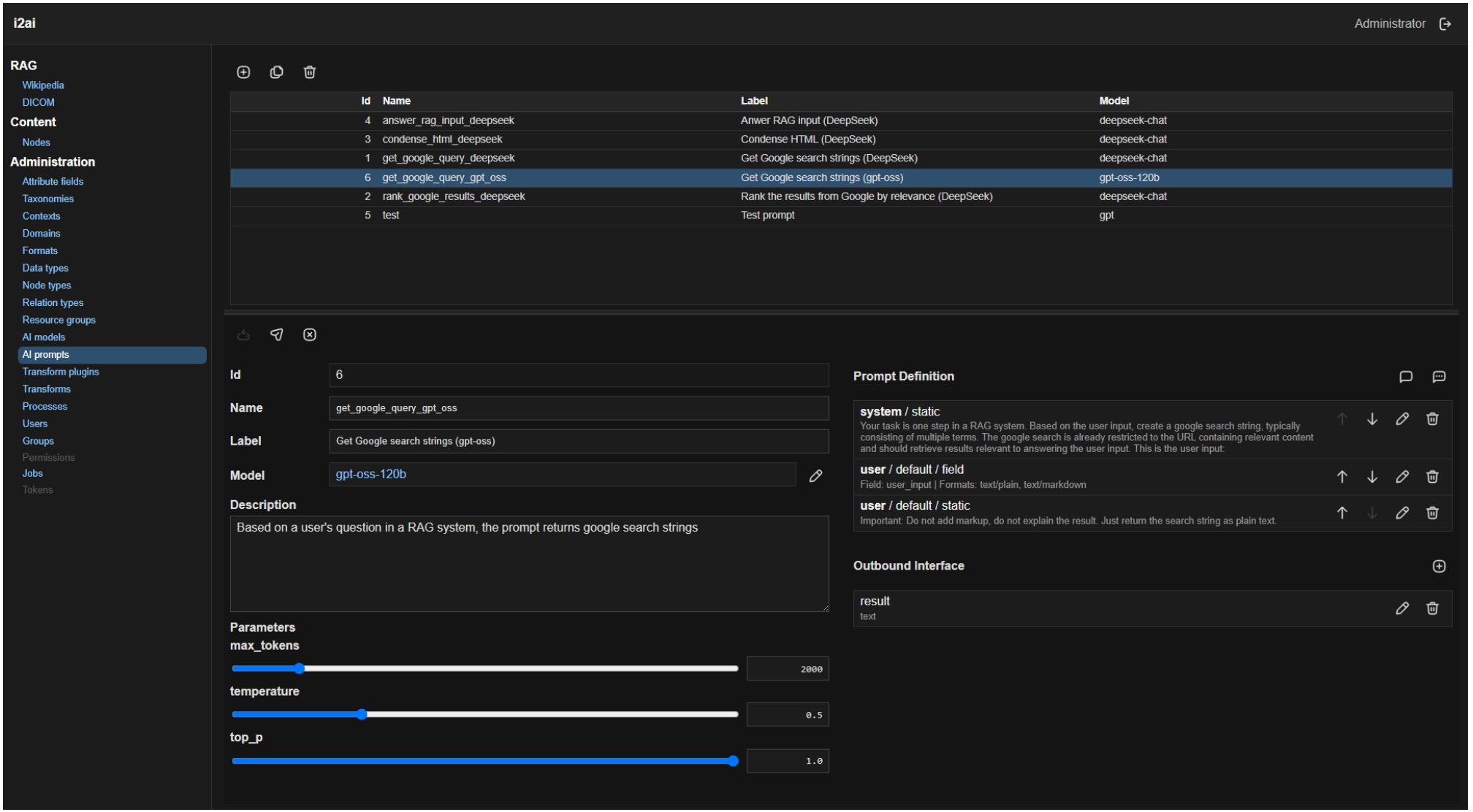
- **Question:** What the user has typed in.
- **Result:** User input answered by the LLM.
- **Sources:** Clickable links to the data modules used.

texolution

Builder perspective

texolution

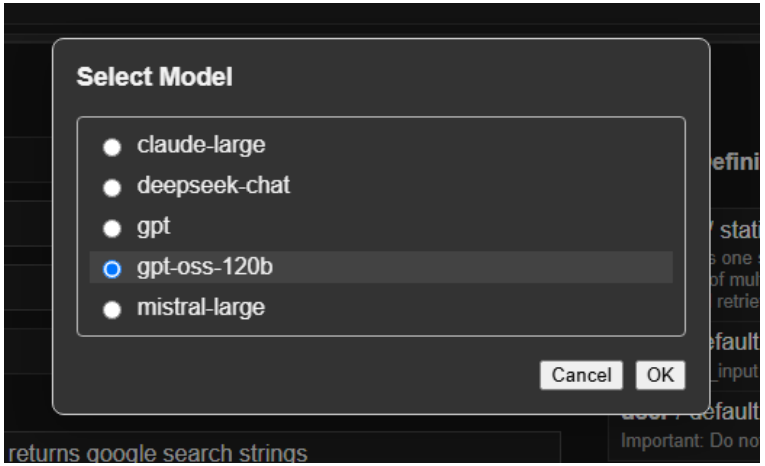
RAG example: prompts



Start with building prompts. A prompt consists of static and field messages that can be added and edited as needed. Parameters can be changed with sliders.

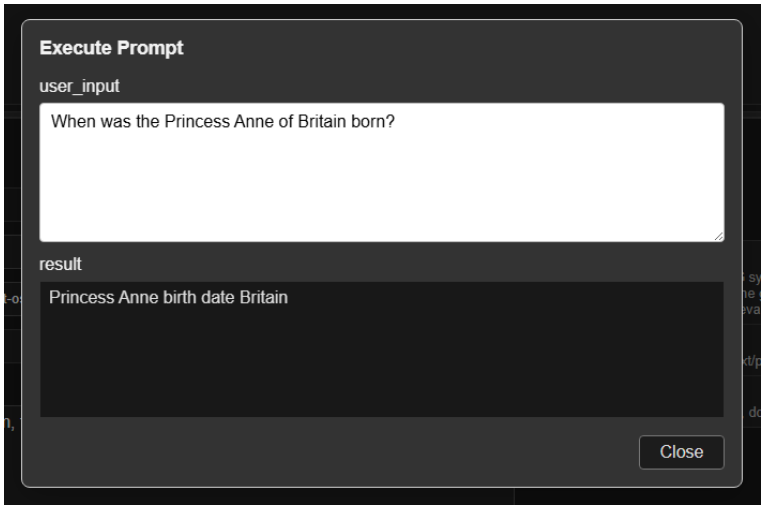
texolution

RAG example: prompts



Model support

- Models are configurable and extensible.
- Custom models can be integrated with custom model adaptors.
- Choosing or changing the model a prompt uses is just one mouse click.

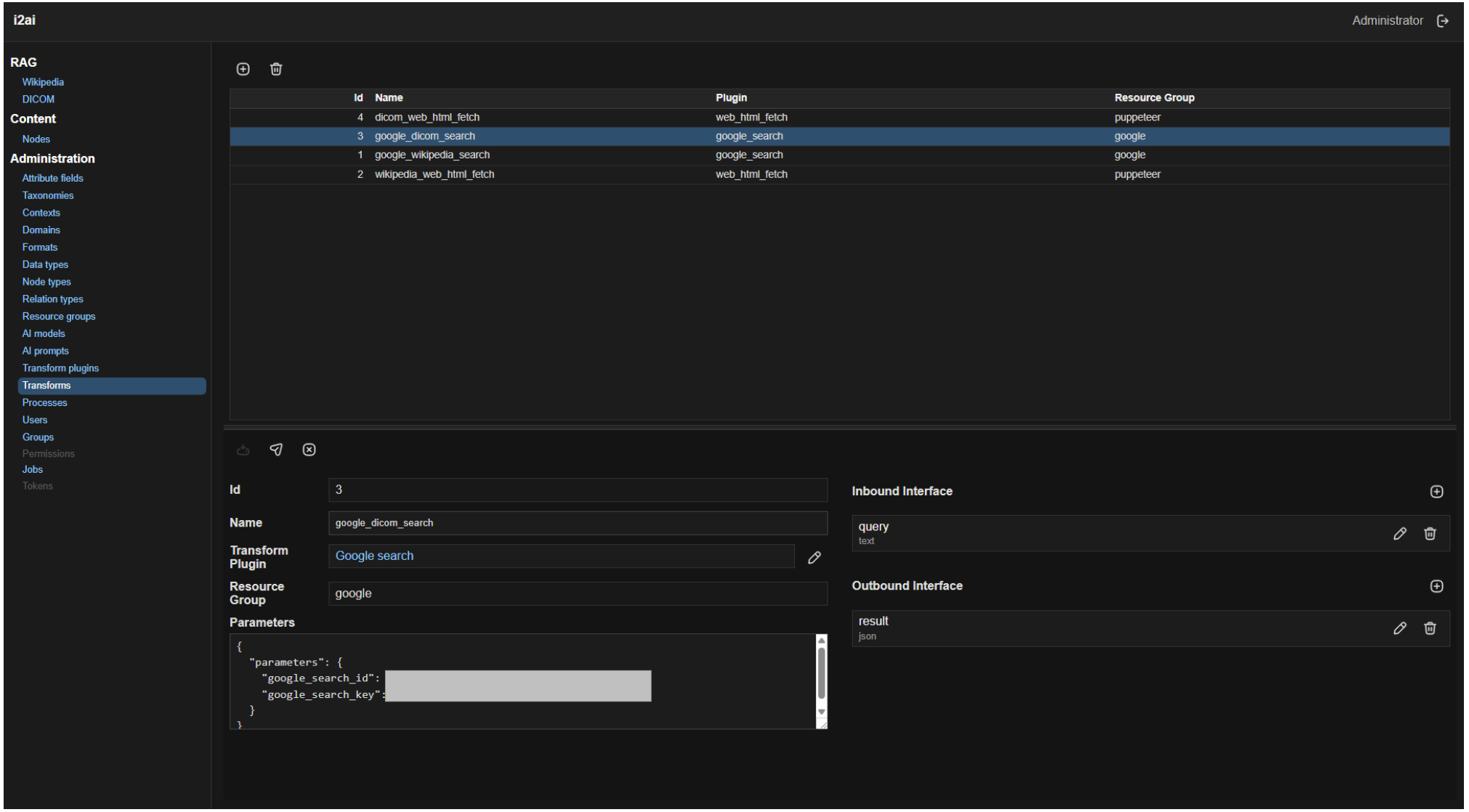


Immediate testing

- Directly from the prompt editor, prompts can be tested.
- Prompts with multiple field messages can also have multiple inputs.
- This prompt generates a Google search string from a user input.

texolution

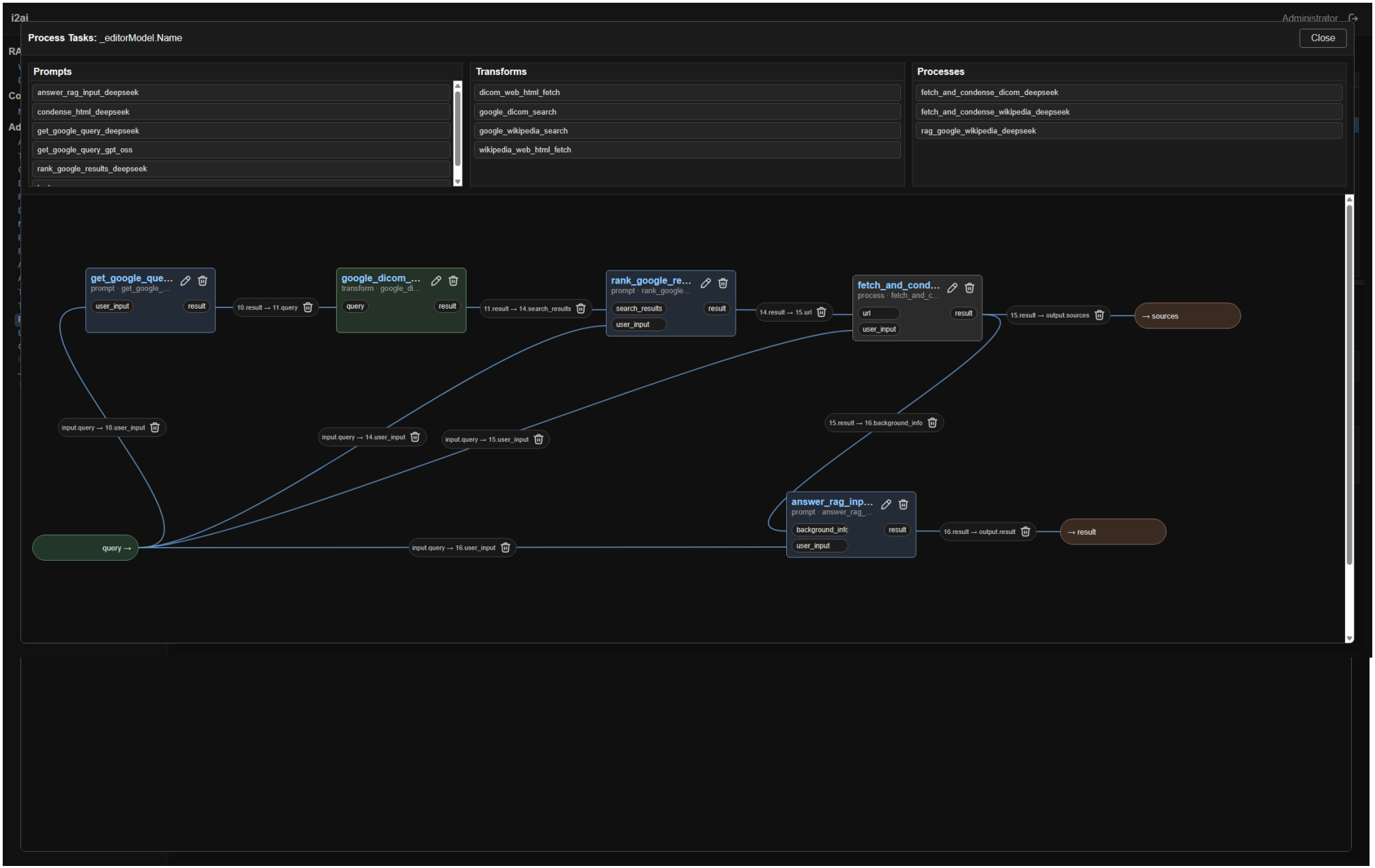
RAG example: transforms



Transforms work much like prompts, have inputs and outputs, but do not employ an AI model. A transform can, e. g., read or write content, convert formats or access an API. This transform performs a search with Google Search API.

texolution

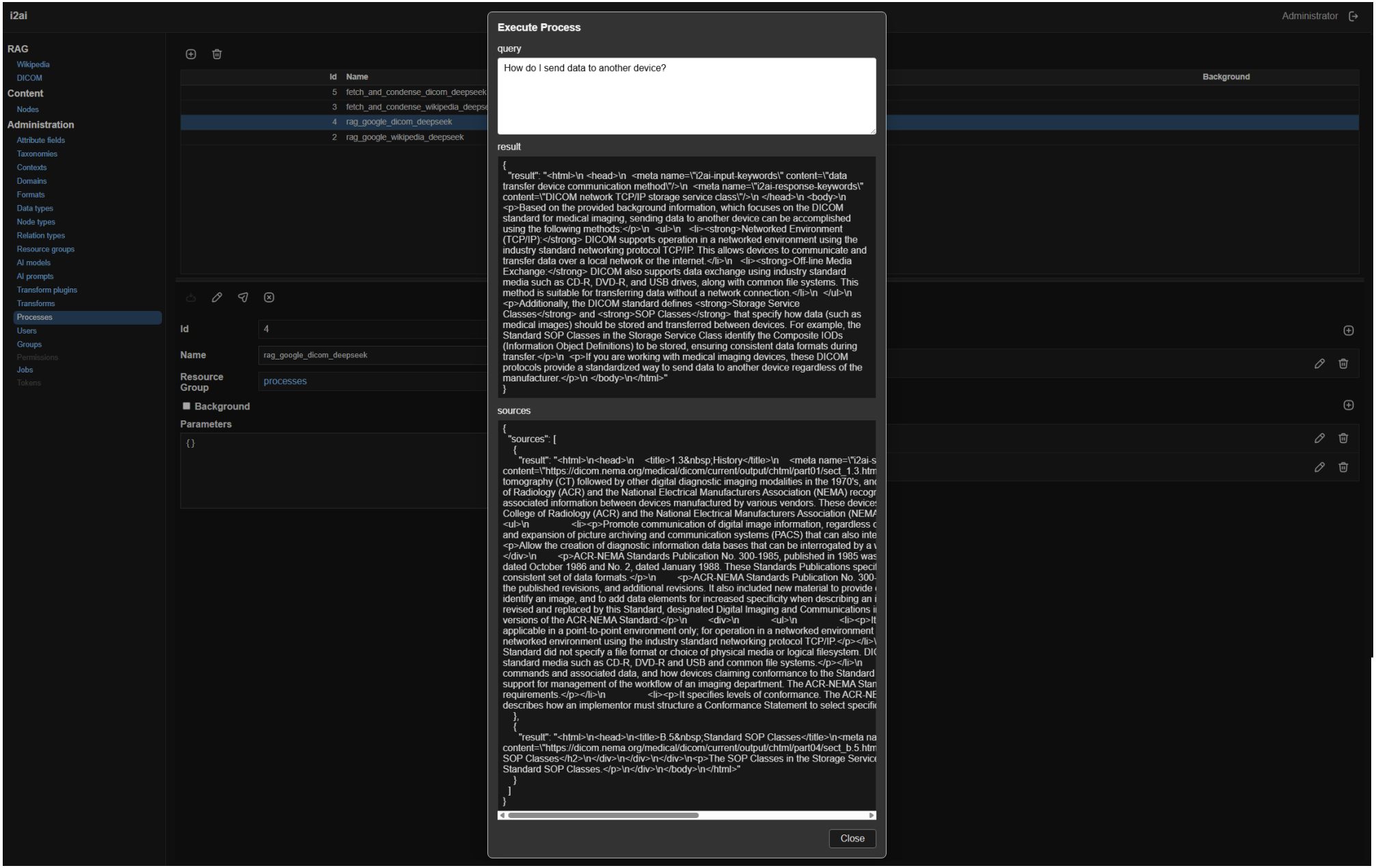
RAG example: processes



Prompts, transforms and subprocesses can be plugged together to processes.

texolution

RAG example: processes



Processes can be tested from the editor, too. That's all, just hook it under the GUI.

texolution

Storage

Storage

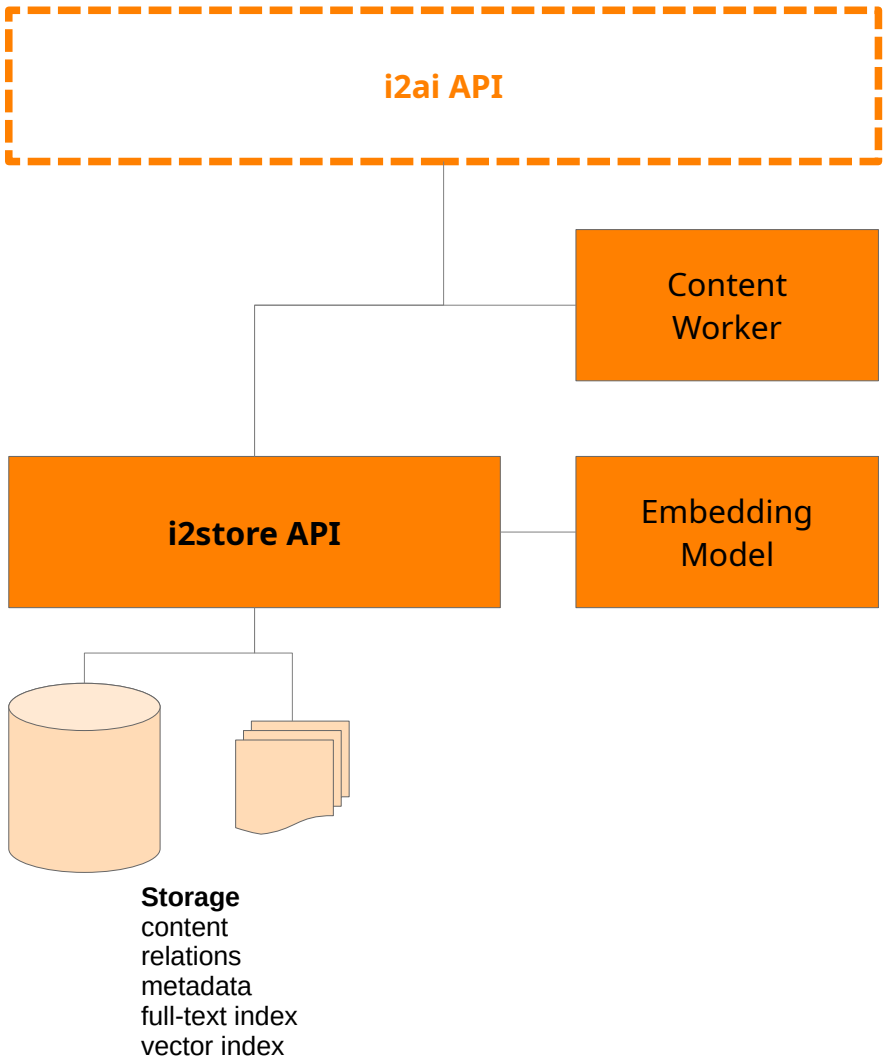
The RAG example we've shown does not use local storage at all, rather uses Google Search API and retrieves content from public web.

i2store API

- i2store API provides access to all content operations.
- It is wrapped by i2ai API and not directly accessed by clients.
- For vector indexing and search, an embedding AI model is running as an internal service.
- Content worker extracts text for indexing from PDF, XML, JSON and other formats (plugins).

Search

- Attributes have types and can use taxonomy.
- Fulltext search on all text content.
- Semantic relations can be followed.
- Vector ranking.
- All can be combined in one query.



texolution

Nodes and relations

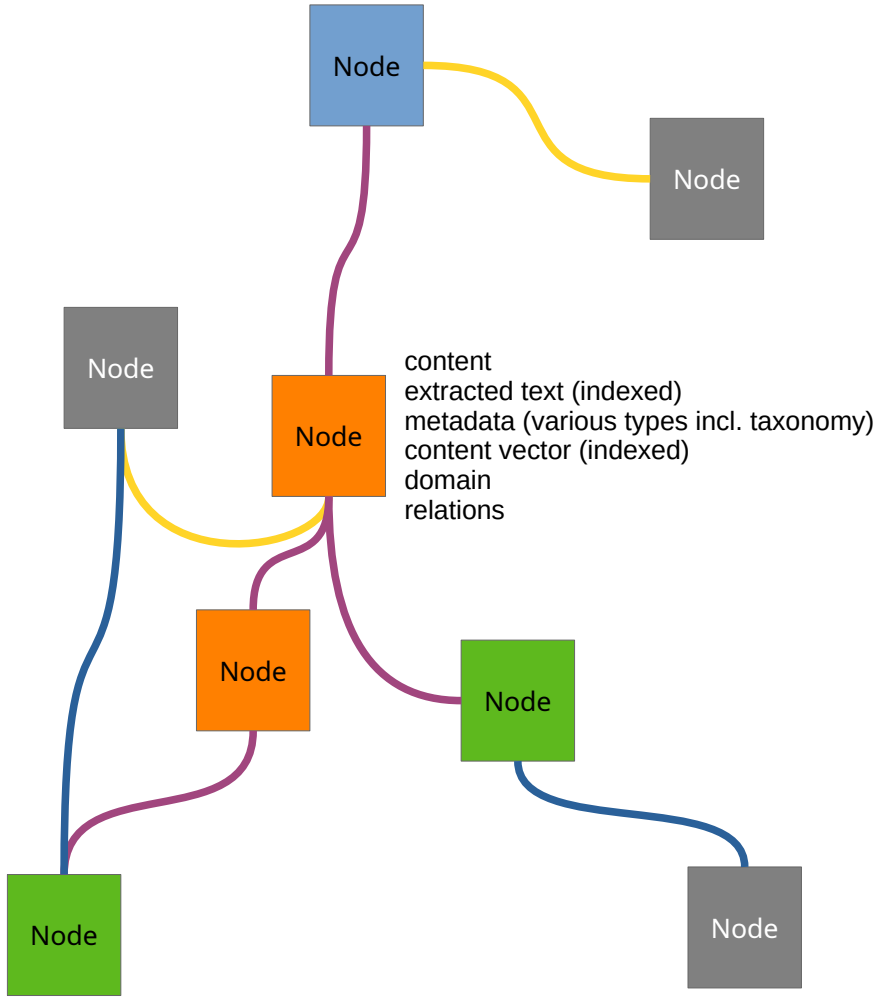
Each data object is represented as a **node**:

- Searchable **metadata**
- **Full-text** and **vector indexed** content
- Typed **relations** connecting nodes
- Relations can also have searchable metadata
- Every node belongs to exactly one **domain** for grouping / packaging
- All nodes in a domain have the same permissions (but read-only flag available)

Thus, the data model combines these axes:

- Metadata search in RDBMS
- Full-text index
- Vector index based on embedding model
- Relation tracking filtered by relation type and metadata

One search API call can query all four axes simultaneously.



texolution

Contexts and domains

Domains can serve as all types of containers on very different granularity, as needed. A domain could be used for:

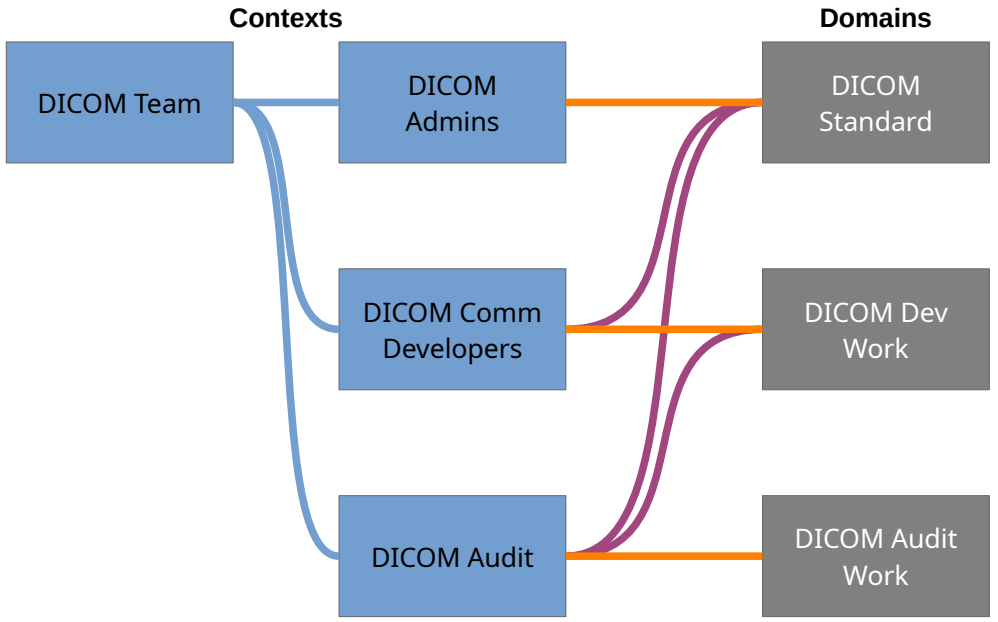
- Sets of LLM background data
- Upload packages for one product in a CDP
- Resources that have common access rights in general

Domains have an owner with full access. Otherwise, domains have no permission sets.

Contexts are the layer that (implicitly) assign permissions to domains.

Each context has a **root domain** with r/w (restricted by permissions) and optionally one or more read-only **secondary domains**.

Contexts can be **nested**.



Other node attributes

Nodes have more properties helping organize the data:

- Read-only flag, so that certain nodes can be protected from overwriting in r/w domains.
- Format from an extensible list with MIME types and extensions.
- Node type as classification of the node purpose (image, topic, …).
- Data type for mapping generic data properties that help modules decide whether they can work with the data or not.
- Several content fields:
  - Core content is stored in the DB (JSON, XML …) or as a file (PDF, images, …).
  - On write operations, full text is extracted for supported formats. More formats can be added as plugins. The full text gets indexed for search.
  - Once full text is available, a vector is generated by an embedding model and stored in the DB. The vectors are used for similarity ranking.
  - Preview field for optional HTML representation.
  - Summary field for optional, AI-generated abstract (can be used for ranking etc.).
  - Nodes have a grouping mechanism within the domain, so versions and variants (for example, languages) can be systematically managed.

texolution

Use Cases

texolution

Use case: Knowledge based chat

We've discussed a RAG application in this presentation. In this basic form, RAG is just a one question – one answer pattern. This is particularly useful in no-login setups, since no data is stored and no user context needs to be known.

An obvious extension is a chat application that enables a dialogue between a user (or multiple users that can be invited) and the system (or multiple agents). Chats are persistent and can be picked up later – thus, the message history is saved.

Apart from that, uploaded files are saved in **i2store** as well as topics with gained knowledge. They can also be categorized with metadata, so they can be found and reused by the system more easily.

In a knowledge based chat, users can discuss technical or other questions with the system and reuse the knowledge gained from the dialogue.

texolution

Use case: Data migration

Moving data like technical documentation from one infrastructure to another can often be formally automized to some level, but many content-related tasks are usually manual.

Here are some examples for steps that can be partially or fully robotized:

- Taxonomy extraction
- Semantic specialization (e.g., topic → concept, reference, task)
- Metadata assignment
- Summary creation
- Re-modularization according to rules
- Authoring rules (how to write instructions, warnings, ...)
- Cross-comparison and link creation

This is useful for:

- CCMS introduction and modularization of monolithic documents
- Preparation for Content Delivery of content with little or no metadata
- Merging technical documentation after an acquisition

Apart from efficiency benefits, enforcing metadata models and authoring rules by AI pushes the customer towards cleaning up grown structures.

texolution

Use case: Research

Working into a complex field of knowledge the traditional way requires lots of reading and flipping over content that's irrelevant for the desired information.

The basic idea is similar to RAG, but in a broader scope.

- In the orange box, the user tells the process what to do research about, provides files and URLs as the base and optionally a style sample (e.g., some text from a scientific paper, if the result should be this way).
- The system first extracts all information about the scope and its perimeter from the content and stores it as modules with some axes of keywords, then creates a TOC.
- From the modules created, it builds modules filling the TOC positions.
- Finally, for each result module, the knowledge modules are searched for similar content using vector search, and the output topic is rounded out and lingually adjusted.

Use case: Vibe coding

This is another well-known AI application, and there are numerous solutions, like CoPilot.

Since we explictly control the background data archive and (using the API) can integrate with Integrated Development Environments (IDE) like VisualStudio (Code), intelliJ and others, **i2ai / i2store** can deliver where other approaches can't.

- With an IDE integration, the AI processes have access to the last version of the code base.
- Multiple solutions (backend service and frontend GUI) can be managed at the same time.
- History can be used for extracting knowledge (approaches that worked well or not so...).
- Most models are good at coding Python, C# etc. directly from the training. Historic or exotic languages are weaker. **i2ai / i2store** can manage valid code samples as reference for generation.
- Likewise, in higher languages, coding styles (how to name variables) and patterns (how to issue a Web API call the same way in the entire app) can be defined and applied.

Two examples are: retro computer games in for C64 & Co., and legacy mainframe languages still in production use (like COBOL or PL/1)

texolution

THANKS for your attention