TWIN DIGITAL TWINS

Connecting a DITA Knowledge Model to iiRDS

A practical architecture for DITA RDF, semantic mapping, and intelligent delivery

DITA	RDF	iiRDS
Detailed source structure	Connected knowledge graph	Discovery and delivery

Michael Iantosca
Senior Director of Knowledge Platforms and Engineering
Avalara Inc.

Harald Stadlbauer
Managing Director
NINEFEB Technical Documentation GmbH

Executive Summary

Organizations that already have a DITA 1.3 OWL ontology are well positioned to build a documentation knowledge graph. The ontology supplies the vocabulary for describing authored content. It can define topics, tasks, steps, commands, maps, keys, reuse relationships, and specialized structures. What it does not do by itself is turn the organization's actual DITA files into RDF data.

A ready-to-use DITA-to-RDF transformation is available as a plug-in for the DITA Open Toolkit. It extracts RDF triples from DITA maps and topics and provides a practical starting point for creating RDF instance data. Organizations should evaluate and reuse this capability before building a completely new conversion pipeline.

iiRDS serves a different purpose. DITA describes how technical content is authored and assembled. iiRDS describes how technical information is classified, found, exchanged, and delivered. The strongest architecture keeps both views and connects them.

> **The central idea**
>
> Think of the DITA graph as the detailed engineering drawing of the documentation, while the iiRDS graph is the catalog that helps people and systems find the right piece of information. One explains how the content is built. The other explains what it is for and when it should be delivered.

A practical implementation therefore uses DITA Open Toolkit processing to generate a detailed DITA RDF graph, maps or projects selected information into an iiRDS-oriented delivery graph, and builds native vector and lexical indexes directly from the DITA source for semantic and exact-match candidate discovery. The graph is not a substitute for those indexes. Its distinctive role is to determine applicability, authoritative source, reuse, dependencies, provenance, and downstream consequences. Product vocabularies, deterministic rules, validation, and stable identifiers complete the architecture.

Recommended architecture at a glance

```
 Incoming product-change collateral
    |
    v
LLM normalization into a compact change record
    |
    v
Controlled-vocabulary resolution
    |
    v
Initial graph query for explicit facts
    |
    +--> Native DITA vector search for semantic candidates
    |
    +--> Lexical and terminology search for exact candidates
    |
    v
Merged stable DITA element identifiers
    |
    v
Graph verification, applicability filtering, reuse tracing, and dependency expansion
    |
    v
Deterministic impact rules and explainable affected-element set
    |
    v
Bounded DITA fragments retrieved for LLM-assisted revision
```

1. Two Semantic Views of the Same Documentation

DITA and iiRDS overlap, but they were designed to answer different questions. Treating them as interchangeable usually removes useful detail from one side or forces unnecessary detail into the other.

The DITA view

DITA is concerned with authoring structure, reuse, assembly, and publishing. A DITA-oriented knowledge model can describe:

- Topics, maps, and topic references
- Task bodies, steps, commands, information, and results
- Keys, key scopes, and indirect references
- Content references and reused content
- Profiling attributes and conditional content
- Map membership and publication structure
- DITA specializations and their inheritance

There is the inherent semantics of DITA. Topics are of a distinct topic type, already pointing to what is described like e.g. a task description etc.
Moreover, the DITAMap

The iiRDS view

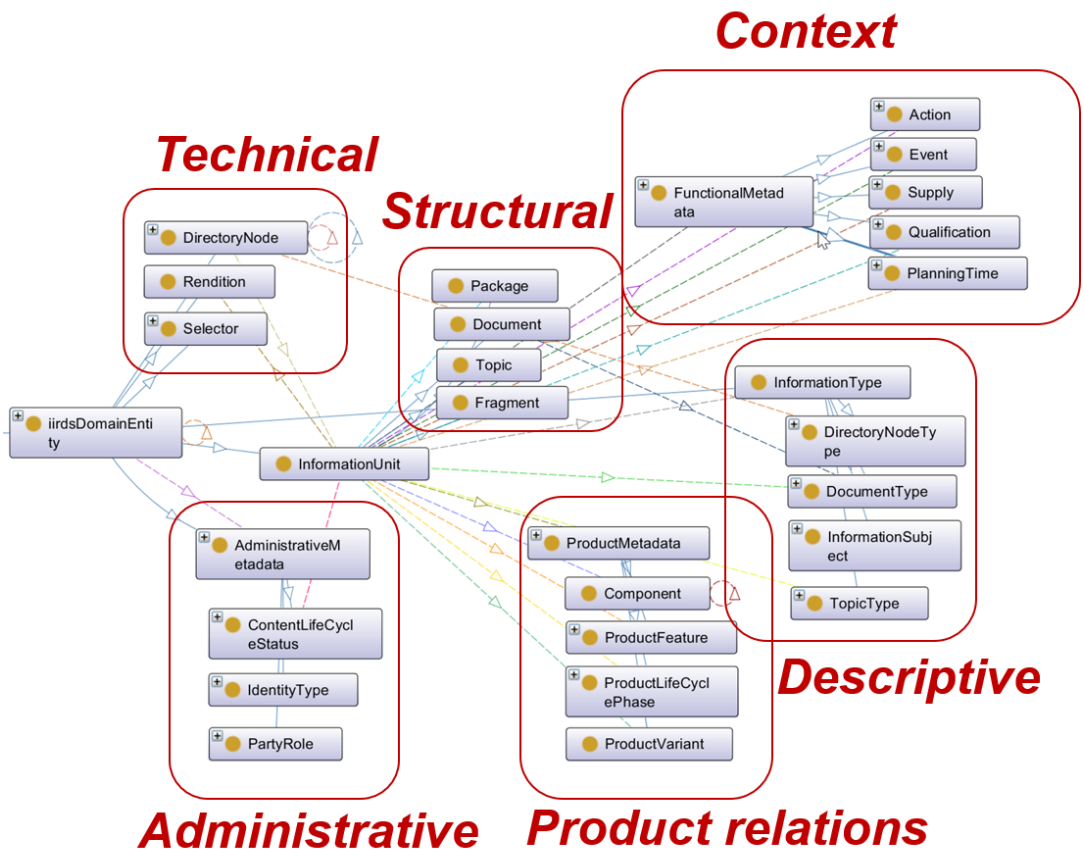
iiRDS is concerned with standardized metadata for finding, exchanging, and delivering technical information.

iiRDS is the abbreviation of **I**ntelligent **I**nformation **R**equest and **D**elivery **S**tandard.

An iiRDS-oriented graph can describe:

- Information units and information types
- Products, product variants, and components
- User roles, audiences, activities, and events
- Lifecycle status and administrative metadata
- Relationships among information units
- Navigation and retrieval metadata

iiRDS is consisting of the following main categories / classes:



And, iiRDS is schema agnostic, able to connect DITA topics with videos in mp4 format, and PDFs etc.

> **Warehouse analogy**
>
> DITA is like the warehouse blueprint and packing specification. It records which item sits in which box, how boxes are grouped, and which parts are reused. iiRDS is like the searchable inventory catalog. It helps someone find the correct item by product, purpose, user role, or lifecycle status.
> And, iiRDS is schema agnostic, this is finding the right ink to the printer, too in that analogy.

Why both views matter

A service portal may only need to locate maintenance procedures for a certain product and technician role. It usually does not need to know whether a sentence came from a DITA <cmd> element or a DITA <info> element. An authoring analytics system may need exactly that distinction. Keeping connected views allows

each system to use the level of detail it needs. For change-impact analysis, however, the detailed DITA graph becomes valuable only when it encodes actionable relationships that a text index cannot reliably infer, including map and publication membership, conref and keyref reuse, product and version applicability, structural roles, and dependencies. Semantic similarity should be handled by a native DITA vector index, while the graph should validate and expand the candidate set.

2. The Role of the DITA Open Toolkit RDF Plug-in

An OWL ontology defines the terms that may appear in a graph, but it does not populate the graph with the content of actual DITA files. A transformation is still required to read the source and generate RDF resources and relationships.

The DITA RDF project provides a DITA Open Toolkit plug-in that extracts RDF triples from DITA maps and topics. The project reports support for metadata such as identifiers, titles, languages, authors, topic reference targets, keys, RDF types, and DITA specialization types.

> **Plain-language distinction**
>
> The ontology is the dictionary and grammar. The RDF transformation is the reader that goes through the documentation and creates actual statements using that dictionary. Without the transformation, the ontology describes what could be said, but the graph contains no documentation facts.

What “ready to use” means

The plug-in provides an existing transformation that can be installed into DITA Open Toolkit and run against DITA content. This is a significant advantage because the organization does not have to begin with an empty XSLT, Java, or Python project.

Ready to use does not mean that its output will automatically match every organization’s target ontology, URI policy, product model, provenance requirements, or iiRDS implementation. It means there is a functioning base transformation that should be evaluated and extended where necessary.

Usually, there are no URIs, IRIs in DITA.

What it should not be called

The RDF transformation should be described as an installable DITA Open Toolkit plug-in, not as one of the standard output formats bundled in every DITA-OT installation. This distinction prevents readers from assuming that RDF output will appear automatically alongside the standard HTML, PDF, Markdown, or normalized DITA transformations.

Why DITA-OT is a sensible processing layer

DITA files rarely stand alone. Their meaning can depend on maps, inherited metadata, keys, key scopes, content references, profiling, subject schemes, and specialization rules. DITA Open Toolkit already understands these mechanisms and provides preprocessing services for resolving or normalizing them.

> **Analogy: a recipe and its pantry**
>
> Reading a DITA topic without its map context can be like reading one page of a recipe while ignoring the pantry list, substitutions, and serving instructions. DITA-OT supplies the wider context before the RDF transformation records the result.
> The map is somehow the context supplied by the structure of the publication, which shall be maintained.

For this reason, an RDF transformation within the DITA-OT environment is usually more reliable than a generic XML script that reads each file independently. A custom pipeline may still be appropriate, but it should be chosen after the existing plug-in has been tested against real requirements.

3. From DITA XML to a Detailed RDF Graph

The detailed graph represents actual documentation resources using classes and properties from the DITA ontology. The ontology defines concepts. The generated RDF creates instances of those concepts.

A simple example

```
dita:TaskTopic
    a owl:Class .

dita:Step
    a owl:Class .

dita:hasStep
    a owl:ObjectProperty .
```

These statements define vocabulary. They say that task topics and steps are classes and that hasStep is a relationship.

```
ex:reset-controller
    a dita:TaskTopic ;
    dcterms:title "Reset the controller" ;
    dita:hasStep ex:reset-controller-step-1 .

ex:reset-controller-step-1
    a dita:Step .
```

These statements describe actual documentation. They identify a particular task and a particular step.

> **Blueprint analogy**
>
> The ontology is the blueprint legend that defines symbols such as door, wall, and window. The RDF instance data is the completed floor plan showing where each specific door, wall, and window appears.

What the detailed graph may preserve

- Topic type and specialization
- Title, language, author, and identifiers
- Map and publication membership
- Steps, commands, notes, and results
- Keys and resolved targets
- Content reference sources and uses
- Related links and topic relationships
- Profiling and applicability metadata
- Repository and transformation provenance

What the detailed graph is missing

- Reference to the Product or variant

4. Connecting DITA to iiRDS

Mapping tells a graph system how a DITA concept relates to an iiRDS concept. Some relationships can be expressed directly in an ontology. Others require transformation rules or a separate iiRDS projection.

Subclass mappings

```
dita:TaskTopic
    rdfs:subClassOf iirds:Topic .
```

In plain language, this says that every DITA task topic may also be treated as an iiRDS information unit. It is similar to saying that every service manual is a publication, while not every publication is a service manual.

Subproperty mappings

```
dita:hasProduct
    rdfs:subPropertyOf iirds:relates-to-product-metadata .
```

This says that a DITA-specific product relationship is a more precise form of the broader iiRDS subject relationship. The detailed relationship remains available, while iiRDS-aware applications gain a standard interpretation.

Equivalent-class mappings

```
dita:TaskTopic
    owl:equivalentClass ex:DITATaskInformationUnit .
```

Equivalence is much stronger. It says that the two classes have exactly the same meaning and membership. This should be used only when the concepts truly match.

> **Analogy: “dog” and “pet”**
>
> A dog can be modeled as a subclass of pet in a household vocabulary, but dog and pet are not equivalent because some pets are cats, birds, or fish. In the same way, a DITA task may be a kind of information unit without being exactly interchangeable with every iiRDS information-unit concept.

When rules are required

An iiRDS classification may depend on more than the DITA topic type. It may depend on specialization, map position, subject-scheme values, workflow state, product attributes, or organizational metadata. In these cases, simple subclass statements are not enough.

The organization can use XSLT, SPARQL CONSTRUCT, SHACL rules, Java, Python, or another deterministic transformation stage to create the iiRDS view.

5. One Resource or Two Connected Resources?

A topic can participate in both the DITA and iiRDS models as one RDF resource, or the architecture can create a separate iiRDS information-unit resource linked to the DITA source resource.

One shared resource

```
ex:reset-controller
    a dita:TaskTopic ;
    a iirds:Topic;
    dita:hasStep ex:step-1 ;
    iirds:has-topic-type ex:task ;
    iirds:relates-to-component ex:controller-x.
```

This is simple and useful when the DITA topic boundary and the iiRDS information-unit boundary are the same.

Two linked resources

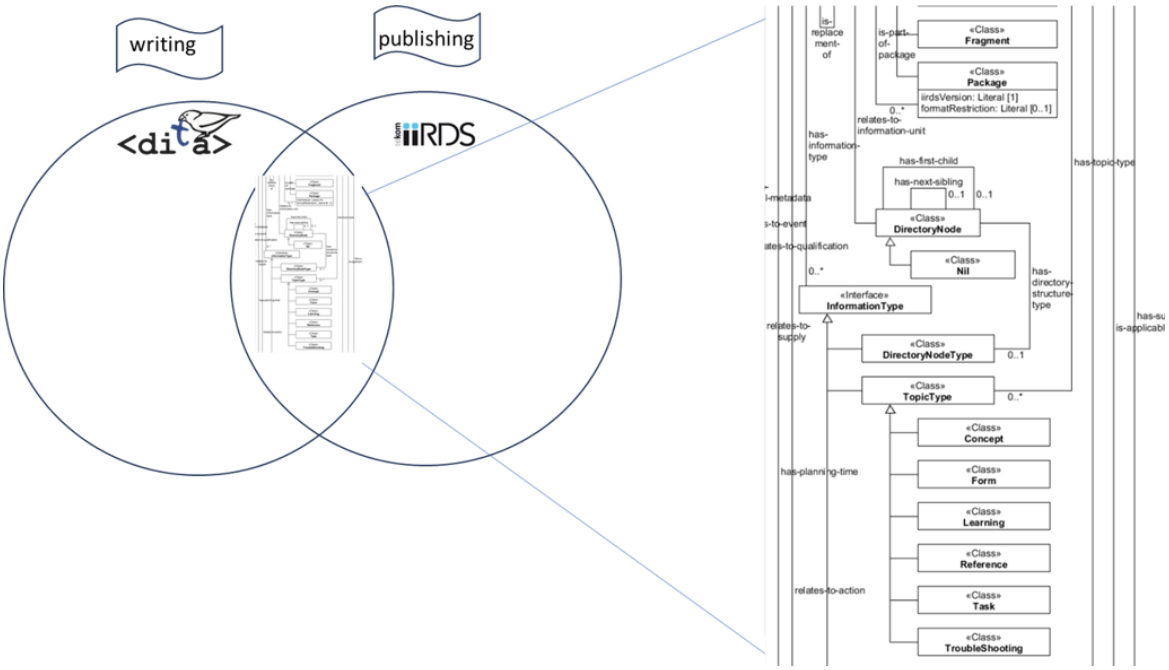
```
ex:iirds-unit-reset-controller
    a iirds:Topic;
    foaf:primaryTopic ex:dita-topic-reset-controller .

ex:dita-topic-reset-controller
    a dita:TaskTopic ;
    dcterms:isPartOf ex:service-manual-map .
```

This approach is useful when the source topic and the delivered information unit are not identical. One information unit might combine several topics, or one topic might produce several context-specific delivery units.

> **Analogy: master file and published edition**
>
> A source topic is like the master artwork. An iiRDS information unit is like a published edition prepared for a particular product, market, or audience, semantically “dressed up”. They are closely related, but they may not be the same object.

This illustration shows a little bit where they interrelate:



6. Why Two RDF Views Are Often Better

Many production systems benefit from generating two connected RDF views from the same DITA source. For impact analysis, these views should be complemented by native DITA vector and lexical indexes. The indexes discover textually and semantically related candidates; the graph determines scope, context, applicability, reuse, and consequences.

```
DITA source
   |
   +--> Detailed DITA RDF graph
   |
   +--> iiRDS delivery metadata graph
```

The DITA graph can preserve fine-grained authoring detail and the direct context supplied by DITA maps. Its impact-analysis value comes from explicit, queryable relationships, not from duplicating semantic search. It should identify where a topic or element belongs, what it applies to, whether it is authoritative or reused, and which dependent content must also be reviewed.

The iiRDS graph can remain focused on standardized discovery and delivery. This avoids making service portals understand every DITA element while still allowing specialist applications to inspect detailed structures.

Detailed DITA graph

- Topics, maps, topic references, and publication context
- Task bodies, steps, commands, information, and results
- Notes, warnings, examples, and related links
- Keys, key scopes, and content references
- Profiling attributes and specializations

iiRDS delivery graph

The iiRDS graph is a further contextualization of the DITA graph.

- Information units and information types
- Products, variants, components, and versions
- Roles, skill levels, activities, and events
- Lifecycle and administrative metadata
- Navigation and relationships among information units (even of different formats)
- Renditions, i.e.

```
DITA source
   |
   +--> Detailed DITA RDF graph --> bringing metadata outside in a graph
   |                            --> mirroring the topic type & the place of the topic in the manual
   |
   +--> iiRDS delivery metadata graph
   |                            --> connect the iiRDS metadata to the DITA graph
   |                            --> contextualize the information which product, feature, variant
   |                                      it belongs to
   |                            --> contextualize in explaining which information type the info is
   |                            --> specify explicitly what it deals with as information subject
```

```
|                              --> contextualize to connect it with events, roles, skill levels
|                              --> express that different formats are the same content
|                                   (like a topic and a video)
```

DITA and iiRDS – both are standards forming together the basis for a Digital Twin of a product, because they allow supplier information to integrate and connect automatically when using these 2 standards.

7. The Graph's Role in DITA Impact Analysis

Recommended Retrieval Sequence for DITA Impact Analysis

I want to clarify the intended architecture for identifying which DITA topics and elements must be updated when new product-change collateral is received. The central requirement is that the LLM must not search, read, or compare the DITA corpus at runtime. Our earlier test, which required roughly eight hours to analyze a single user guide and consumed significant tokens, demonstrates why that approach is not viable at scale.

The LLM may be used only to interpret the relatively small set of incoming collateral and convert it into a compact, normalized change description. All searching of the DITA corpus must be performed by indexed, non-LLM systems: graph queries, vector retrieval, lexical search, and deterministic rules.

The recommended order of operations is as follows.

1. **Process the incoming collateral.** Extract the product, component, feature, old behavior, new behavior, applicable version, user role, error condition, and other relevant change signals. This step creates a structured change record; it does not search the DITA corpus.
2. **Resolve terms against controlled vocabularies.** Map product names, aliases, component names, versions, features, and terminology to canonical graph identifiers. This prevents variants such as product codes, abbreviations, and display names from being treated as unrelated entities.
3. **Query the graph for explicit matches.** Use SPARQL or equivalent graph queries to find DITA topics and elements already known to be connected to the affected product, component, feature, version, role, information type, or lifecycle state. This is the high-precision retrieval pass.
4. **Query the DITA vector index.** Search a vector index built directly from the DITA source at the topic, section, step, command, warning, note, troubleshooting condition, and UI-label level. This finds semantically related content that may use different wording or lack complete metadata. No LLM reads the DITA content during this search.
5. **Run lexical and terminology searches.** Search exact product names, error codes, field names, API names, command names, UI labels, deprecated terms, and other exact strings. This can run in parallel with vector retrieval.
6. **Merge all candidate identifiers.** Combine the explicit graph matches, vector results, and lexical results into one candidate set. Every vector and lexical record must contain a stable DITA element identifier and its corresponding graph URI.
7. **Query the graph again to verify and expand.** For each candidate, use the graph to confirm product, component, version, audience, lifecycle, language, and publication applicability. Traverse reuse, conref, keyref, map membership, dependency, prerequisite, warning, and related-topic relationships. Remove false positives and identify the authoritative source element.
8. **Apply deterministic impact rules.** Use explicit rules to include related warnings, prerequisites, results, reused fragments, dependent procedures, screenshots, localization targets, publication occurrences, and other content that must change as a consequence of the original update.

9. **Produce an explainable final impact set.** Classify results as definitely affected, likely affected, related dependency, or rejected. Record the evidence for each decision, including semantic match, graph relationships, applicability, reuse, version, and source provenance.
10. **Retrieve only the approved DITA fragments.** After the impact set is finalized, retrieve only the selected elements and the minimum surrounding context needed for revision. The LLM may then edit these bounded fragments; it must not search the corpus or decide which topics to retrieve.

Value of the Twin Digital Twin Graph Compared with a Native DITA Vector Store

The Twin Digital Twin Graph is not primarily valuable because it performs better semantic search. A native vector store built directly from the DITA source can probably perform semantic candidate discovery just as well, and may do so more simply. The graph provides its distinctive value after and around retrieval by determining what a textual match means within the documentation system.

What the vector store can do

A DITA-native vector index can efficiently find topics and elements that discuss the same concept in different words, procedures that mention an affected feature, troubleshooting content related to changed behavior, and warnings, notes, commands, and UI labels with semantic similarity. For candidate discovery, the engineering team is likely correct that the DITA elements could be embedded directly without first converting the graph into vectors.

What the graph should add

Applicability filtering. The graph can distinguish content for different products, models, releases, configurations, audiences, markets, and languages. A vector result may be semantically similar but apply to the wrong version or product variant.

Reuse tracing. A matched paragraph may be a conref target, keyref source, reused warning, or shared procedure. The graph can identify the authoritative source and every place where it appears. A vector store may return several occurrences without recognizing that they are manifestations of one source element.

Dependency expansion. A changed procedure may affect prerequisites, warnings, expected results, troubleshooting topics, related tasks, screenshots, API references, and downstream publications. These consequences may not be textually similar enough to appear in vector results.

Structural context. The graph can represent that an element is a step in a procedure, a warning associated with a command, a result of an action, or a prerequisite for another task. Embeddings generally flatten these distinctions unless substantial structural information is included in every vector record.

Canonical identity and terminology. The graph can resolve aliases, product codes, old names, display names, abbreviations, and component hierarchies to the same canonical entity.

Provenance and explainability. The graph can explain why an item was selected, such as a warning being affected because it is reused by several procedures that operate on the changed component in specified versions. A vector store usually explains only that the text was semantically similar and received a particular score.

Deterministic completeness. Once one affected source element is found, graph traversal can deterministically identify connected reuse and publication consequences. Vector similarity alone cannot guarantee that all required dependents have been discovered.

The graph provides these benefits only when the relevant relationships actually exist and are reliable. If the Twin Digital Twin Graph contains little more than vectorized topic content, basic metadata, and broad related-to links, it adds limited value over a well-designed DITA vector store. To justify the graph, it must encode

actionable relationships such as element-to-topic membership, map and publication membership, conref and keyref relationships, product and version applicability, procedure-to-component relationships, warnings governing procedures or commands, localization targets, and dependencies that require related content to be reviewed.

Practical test of the graph's value

For each result returned by the DITA vector store, the Twin Digital Twin Graph should be able to answer the following questions without examining the full text again:

- Is this result applicable to the affected product and version?
- Is this the authoritative source or merely a reused occurrence?
- Where else is this element reused or published?
- Which dependent content must also be reviewed?
- Which similar results should be rejected because of applicability?
- Why was each final element included?
- Can the system demonstrate that all known structural consequences were followed?

If the graph cannot materially improve those answers, it is not yet contributing enough to warrant being in the retrieval path.

Bottom line

A direct DITA vector store can replace the graph for semantic discovery. It cannot replace a well-constructed graph for scope determination, relationship traversal, applicability, reuse analysis, dependency expansion, provenance, and explainability. The vector store finds content that might be affected. The Twin Digital Twin Graph determines what that content represents, whether it actually applies, and what else must change because of it.

The graph should therefore be evaluated not by whether it returns better similarity matches than the native vector store, but by whether it converts those matches into a more accurate, complete, nonduplicative, and explainable impact set.

In summary: the graph should be queried first for known facts, the vector and lexical indexes should then improve recall, and the graph should be queried again to validate applicability and expand the impact analysis. The final sequence is:

This design keeps runtime search fast and economical while preserving the graph's unique value: deterministic context, applicability, dependency analysis, reuse tracing, provenance, and explainability. A vector database alone can identify similar text, but it cannot reliably determine all downstream consequences of a change unless those relationships are represented and traversed in the graph.

The guiding principle should be: the indexes discover candidates, the graph determines scope and consequences, and the LLM edits only the final bounded content.

8. Evaluate Before You Extend

The existing DITA-OT RDF plug-in should be tested against a representative documentation set before an organization designs custom extensions. The purpose of the evaluation is to separate capabilities that already exist from genuine gaps.

Representative test content

- Tasks, concepts, references, maps, and bookmaps
- Nested topic references and reused topics
- Keys, key scopes, and content references
- Conditional processing and subject schemes
- Custom specializations and multilingual content
- Topics used in several maps or publications

Questions the evaluation should answer

1. Which metadata and relationships does the plug-in already generate?
2. How are RDF resources and URIs constructed?
3. Does the output align with the organization's DITA OWL ontology?
4. Are identifiers stable across builds and environments?
5. How are maps, keys, and specializations represented?
6. What structural details are missing for the intended applications?
7. Which additions belong in the DITA transformation and which belong in the iiRDS projection?

> **Practical mindset**
>
> Treat the plug-in as an existing foundation that must pass an architectural review, not as either a complete final solution or an irrelevant prototype. Reuse what works, document what does not, and customize only the gaps.

The existing DITA-OT iiRDS plug-in then harvests metadata and maps them to iiRDS ones, building up a knowledge graph of the DITA topics and allows to connect them with context.
There, it is essential to have use cases and determine, which metadata are needed for them.
These metadata can be stated using iiRDS specific subject scheme possibilities. These metadata connect automatically with the standard properties of the iiRDS standardized knowledge graph.

9. Extending and Post-Processing the Output

A production pipeline may extend the DITA-OT plug-in or apply additional RDF processing after the initial extraction.

```
DITA source
   |
DITA-OT preprocessing
   |
DITA RDF plug-in
   |
RDF normalization and URI assignment
   |
DITA ontology alignment
   |
DITA OT iiRDS plug-in
   |
additional iiRDS projection and mapping (what could not be retrieved from the plug-in)
   |
SHACL validation
   |
GraphDB load
```

Common extension requirements

- Generate stable organizational URIs
- Translate plug-in predicates to organizational ontology terms
- Represent steps, commands, warnings, results, and specialized structures
- Resolve product and component identifiers to controlled vocabularies
- Create iiRDS information types and subject relationships
- Record repository, commit, map context, and transformation versions
- Separate schema, domain, content, and provenance data into named graphs

Deterministic processing

The same source, configuration, ontology version, and vocabulary versions should produce the same identifiers and semantic relationships. Determinism makes graph comparisons, incremental updates, testing, and troubleshooting practical.

> **Analogy: a reliable compiler**
>
> A documentation transformation should behave like a compiler. Given the same source and settings, it should produce the same result. If identifiers change randomly between builds, links and comparisons become unreliable.

10. Stable Identifiers

Every important topic, map, information unit, product, and component should have a stable URI. A stable identifier should not change merely because a file moves to another folder or a build runs on another server.

```
https://docs.example.com/topic/reset-controller
https://docs.example.com/map/service-manual
https://products.example.com/product/controller-x
https://products.example.com/component/power-module
```

The source path can still be recorded for traceability, but it should not be the only identity of the resource.

Issues the URI policy must address

- Unique identifier pointing to the resource
- Duplicate topic IDs in different files
- Topics moved or renamed without a change in meaning
- One topic used in several publications
- Multiple languages and product versions
- Information units assembled from several topics
- Context-specific uses of one source topic

11. Product and Component Vocabularies

A topic may state that it applies to controller-x, but the graph also needs a resource that defines Controller X. Otherwise the graph contains a label without a dependable identity.

```
ex:controller-x
    a iirds:ProductVariant ;
    skos:prefLabel "Controller X"@en .

ex:power-module
    a iirds:Component ;
    skos:prefLabel "Power module"@en .
```

These resources may come from a subject-scheme map, product information management system, product lifecycle management system, terminology database, enterprise ontology, or controlled data export.

Why controlled values matter

The labels Controller X, controller-x, CX100, and CX-100 may all refer to one product. Without a canonical URI, a graph may treat them as four different things.

> **Analogy: contact records**
>
> A contact list becomes unreliable when “Robert Smith,” “Bob Smith,” and “R. Smith” are stored as unrelated people. Controlled product identifiers solve the same problem for technical information.

12. Handling DITA Features That Depend on Context

Maps

Maps define organization, navigation, publication membership, and inherited metadata. A topic can appear in several maps and receive different context in each one. The graph must decide whether to represent one topic with several contexts, separate topic-use resources, or separate delivery information units.

Keys

Keys are indirect references whose targets may vary by map or key scope. A robust graph can preserve the authored key reference, the key definition, the scope, and the resolved target. This supports both authoring analysis and delivery interpretation.

Content references

Content references reuse content from another location. The graph should preserve the authoritative source and the reuse relationship. It may also represent the resolved occurrence in a particular delivery context.

Profiling and applicability

Profiling attributes can express product, platform, audience, market, channel, or version applicability. The architecture should distinguish the conditions in unfiltered source from the applicability of a particular filtered delivery.

Specializations

Specializations add precise business meaning to DITA structures. The detailed graph should preserve that meaning. The iiRDS view should map it to a broader standardized concept only when the mapping is valid.

> **Analogy: specialist and general catalogs**

> A hospital may record a detailed diagnosis in a clinical system while using a broader category in an appointment portal. The detailed term is preserved for specialists, while the broader category supports common workflows. DITA specializations and iiRDS classifications can work in a similar way.

13. Provenance and Traceability

Every generated RDF resource should be traceable to its source and processing context. Provenance supports audits, change analysis, troubleshooting, and controlled deployment.

Useful provenance data

- Repository name and URI
- Branch, tag, and commit identifier
- DITA file path and topic ID
- Map context and key scope
- Language and publication identifier
- DITA-OT and RDF plug-in versions
- Custom transformation and ontology versions
- Transformation date and build identifier

```
ex:dita-topic-reset-controller
    prov:wasDerivedFrom
        <https://git.example.com/docs/reset-controller.dita> ;
    ex:sourceCommit "8bf3a91" ;
    ex:sourceTopicId "reset-controller" ;
    ex:transformationVersion "2.1.0" .
```

> **Analogy: a manufacturing lot number**
>
> Provenance is the lot number and production record for a graph resource. It tells you where it came from, which process produced it, and which version should be trusted.

14. Validation

A graph can be valid RDF and still be wrong or incomplete. RDF syntax only proves that the statements are formatted correctly. It does not prove that required titles, product links, identifiers, or lifecycle values are present.

SHACL can define graph-quality rules such as:

- Every topic must have a stable URI and title
- Every product reference must resolve to a known product
- Every iiRDS unit must link to its DITA source
- Every released unit must have lifecycle metadata
- Every generated resource must record transformation provenance
- Every required information type must be assigned

> **Analogy: grammar check versus inspection**

> DITA schema validation is like checking that a form was filled out in the correct format. SHACL validation is like checking that the required business information is present and sensible. Both are needed, but they answer different questions.

DITA grammar and Schematron rules validate the authored XML. SHACL validates the RDF generated from that XML.

15. Organizing the Data in GraphDB

Named graphs separate different responsibilities and make updates, troubleshooting, and version management easier.

```
/graph/schema/dita
/graph/schema/iirds
/graph/schema/dita-iirds-mapping
/graph/domain/products
/graph/domain/components
/graph/content/dita
/graph/content/iirds-view
/graph/provenance
/graph/validation/shapes
```

Purpose of each graph

Graph	Purpose
/graph/schema/dita	The DITA OWL ontology and its classes and properties.
/graph/schema/iirds	The separately managed iiRDS semantic model.
/graph/schema/dita-iirds-mapping	Mappings and rule references that connect DITA and iiRDS.
/graph/domain/products	Products, variants, product versions, and related concepts.
/graph/domain/components	Components, assemblies, and subsystems.
/graph/content/dita	Detailed RDF generated from DITA source.
/graph/content/iirds-view	The iiRDS-oriented discovery and delivery representation.
/graph/provenance	Repository, commit, build, and transformation traceability.
/graph/validation/shapes	SHACL shapes used to validate graph quality.

16. Practical End-to-End Process

1. Validate the DITA topics and maps against the applicable DITA grammar and organizational authoring rules.
2. Generate the detailed DITA RDF graph with stable identifiers, map and publication context, reuse relationships, applicability, structural roles, and provenance.

3. Generate the connected iiRDS view and link it to controlled product, component, role, activity, version, lifecycle, and information-type vocabularies.
4. Build vector and lexical indexes directly from the DITA source at the topic and element levels. Every index record must carry a stable DITA element identifier and corresponding graph URI.
5. When collateral arrives, use the LLM only to normalize the small incoming package into a structured change record. The LLM must not search or compare the DITA corpus at runtime.
6. Resolve the change record against controlled vocabularies and canonical graph identifiers.
7. Query the graph first for explicit product, component, feature, version, role, information-type, and lifecycle matches.
8. Run native DITA vector retrieval and lexical or terminology searches to improve recall and discover content that uses different wording or incomplete metadata.
9. Merge candidate identifiers and query the graph again to verify applicability, reject false positives, identify authoritative sources, trace reuse, and expand dependencies.
10. Apply deterministic impact rules for prerequisites, warnings, results, related tasks, screenshots, API references, localization targets, and publication occurrences.
11. Produce an explainable affected-element set classified as definitely affected, likely affected, related dependency, or rejected, with evidence for each decision.
12. Retrieve only the approved DITA fragments and minimum surrounding context for revision. The LLM may edit these bounded fragments but must not decide which corpus content to retrieve.
13. Validate the resulting DITA and RDF changes, load or replace named graphs through a controlled deployment process, and test representative business queries before release.

Representative business queries

- Find all maintenance tasks for Controller X.
- Find information intended for service technicians.
- Find all topics containing electrical safety warnings.
- Find all publications that include a particular topic.
- Find the DITA source behind an iiRDS search result.
- Find all information affected by a product or version change, including authoritative source elements, reused occurrences, publication contexts, and dependent content.
- Explain why each element was included or rejected and demonstrate that all known structural consequences were followed.
- Find resources generated from a specific repository commit.
- Find unresolved product or component references.

17. Do You Need Another Ontology?

Usually, an organization that already has a DITA 1.3 OWL ontology does not need another generic document ontology. The DITA ontology can remain the detailed source model, while iiRDS supplies the standardized retrieval and delivery model.

A small extension ontology may still be needed for business concepts that neither model covers well, such as error codes, firmware ranges, serial-number applicability, failure modes, required tools, certifications, maintenance intervals, diagnostic symptoms, and regulatory classifications.

Or it is intended to use a special purpose ontology like for industrial data processing, the IDO ontology to connect then iiRDS and DITA RDF with maintenance data.

Design principle

> Add concepts because applications and queries require them, not because it is theoretically possible to model everything. A useful ontology is like a well-organized toolbox. It contains the tools needed for real work without becoming a warehouse of unused equipment.

Conclusion

A DITA OWL ontology and iiRDS are complementary semantic layers. The DITA ontology preserves the structure and meaning of the authored source. iiRDS supplies a standardized way to classify, retrieve, exchange, and deliver technical information.

The available DITA Open Toolkit RDF plug-in provides the practical starting point for generating RDF from DITA maps and topics. It should be evaluated and reused before a new conversion pipeline is built. Its output may require extension, normalization, URI assignment, product linking, provenance enrichment, or an iiRDS projection, but those needs do not erase the value of the existing transformation. The iiRDS projection can be supported by the DITA OT iiRDS plug-in. If more metadata can be already supplied at the time of drafting the DITA topics, these can be supplied with iiRDS specific subject scheme maps, which will then be processed by the plug-in.

The recommended architecture is to generate a detailed DITA RDF graph, create a connected iiRDS delivery view, add controlled domain vocabularies and provenance, validate the result, and load the resulting named graphs into GraphDB. For impact analysis, native DITA vector and lexical indexes should perform candidate discovery, while the graph should determine applicability, authoritative identity, reuse, dependencies, provenance, and downstream consequences. The LLM should interpret only incoming collateral and edit only the final bounded fragments selected by the indexed retrieval and graph-validation pipeline.

> **Final analogy**
>
> The result is a pair of connected digital twins for technical information. One twin shows how the documentation is constructed. The other shows how the information should be found and delivered. Because they share identifiers and links, users and systems can move from the catalog view to the detailed source view without losing context.

The guiding principle is: the indexes discover candidates, the graph determines scope and consequences, and the LLM edits only the final bounded content.

References and Further Reading

DITA Open Toolkit. Official project site

The DITA RDF Project. DITA ontology and DITA-OT RDF plug-in overview

Note: The RDF capability discussed in this article is an installable DITA Open Toolkit plug-in. It is not presented as a standard RDF output bundled with every DITA-OT installation.