Introduction to the Darwin Information Typing Architecture

Toward portable technical information

Don R. Day (dond@us.ibm.com), IBM Corporation
Michael Priestley (mpriestl@ca.ibm.com), IBM Corporation
David A. Schell (dschell@us.ibm.com), IBM Corporation

March 2001

Updated October 2001

The Darwin Information Typing Architecture (DITA) is an XML-based architecture for authoring, producing, and delivering technical information. This article introduces the architecture, which sets forth a set of design principles for creating information-typed modules at a topic level, and for using that content in delivery modes such as online help and product support portals on the Web. This article serves as a roadmap to the Darwin Information Typing Architecture: what it is and how it applies to technical documentation. The article links to representative source code.

The XML-based Darwin Information Typing Architecture (DITA) is an end-to-end architecture for creating and delivering modular technical information. The architecture consists of a set of design principles for creating information-typed topic modules, and for using that content in various ways, such as online help and product support portals on the Web. At the heart, the DITA is an XML document type definition (DTD) that expresses many of these design principles. The architecture, however, is the defining part of this proposal for technical information; the DTD, or any schema based on it, is just an instantiation of the design principles of the architecture.

Background

This architecture and DTD were designed by a cross-company workgroup representing user assistance teams from IBM, Lotus, and Tivoli. After an initial investigation in late 1999, the workgroup developed the architecture collaboratively during 2000 through postings to a database and weekly teleconferences. We're offering the architecture on IBM's *developerWorks* Web site as an alternative XML-based documentation system, designed to exploit XML as its encoding format.

Information interchange, tools management, and extensibility

IBM, with millions of pages of documentation for its products, has its own very complex SGML DTD, IBMIDDoc, which has supported this documentation since the early 1990s. The workgroup had to consider from the outset, "Why not just convert IBMIDDoc, or use an existing XML DTD such as DocBook, or TEI, or XHTML?" The answer requires some reflection about the nature of technical information.

First, both SGML and XML are recognized as meta languages that allow communities of data owners to

describe their information assets in ways that reflect how they develop, store, and process that information. Because knowledge representation is so strongly related to corporate cultures and community jargon, most attempts to define a universal DTD have ended up either unused or unfinished. The ideal for information interchange is to share the semantics and the transformational rules for this information with other data-owning communities.

Second, most companies rely on many delivery systems; the ways that they process their information differ widely from company to company. Therefore any attempt at a universal toolset also proves futile. The ideal for tools management is to base a processing architecture on standards and to leverage the contributed experience of many others, solving common problems in a broad community.

Third, most attempts to formalize a document description vocabulary (DTD, or schema) have been done as information modeling exercises to capture the current business practices of data owners. This approach tends to encode *legacy* practices into the resulting DTDs or vocabularies. The ideal for future extensibility in DTDs for technical information (or any information that is continually exploited at the leading edge of technology) is to build the fewest possible presumptions about the top-down processing system into the design of the DTD.

In the beginning, the workgroup tried to understand XML's role in this leading edge of information technology. As the work progressed, the team became aware that any DTD design effort would have to account for a plurality of vocabularies, a tools-agnostic processing paradigm, and a legacy-free view of information structures. Many current DTDs incorporate ways to deal with some of these issues, but the breadth of the issues leads to more than just a DTD. To support many products, brands, companies, styles, and delivery methods, we had to consider the entire authoring-to-delivery process. We ended up with a range of recommendations that required us to represent our design as an information architecture, not just as a DTD.

The topic as the basic architectural unit

A topic is a unit of information that describes a single task or concept or reference item. The information's category (concept, task, reference) is its information type. Typed topics are easily managed within content management systems as reusable, stand-alone units of information. For example, selected topics can be gathered, arranged, and processed within a delivery context to provide a variety of deliverables, such as groups of recently updated topics for review, helpsets for building into a user-assistance application -- or even chapters or sections in a booklet, printed from user-selected search results or "shopping lists."

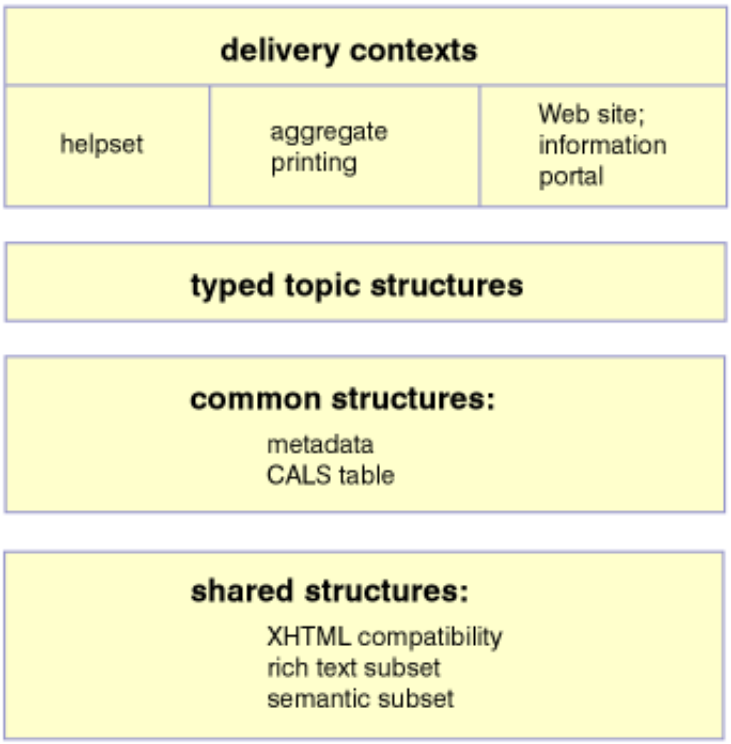
Authors can introduce new information type by specialization from the structures in the base topic DTD (explained in detail in the companion article, Specialization in the Darwin Information Typing Architecture).

DITA overview

The Darwin Information Typing Architecture defines a set of relationships between the document parts, processors, and communities of users of the information.

As shown in Figure 1, the Darwin Information Typing Architecture has four layers that relate to specific design points expressed in the reference DTD, `ditabase`.

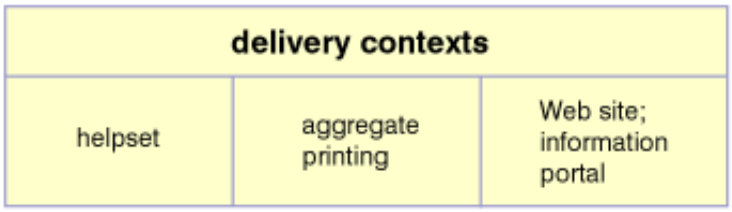
Figure 1. Layers in the Darwin Information Typing Architecture



A typed topic -- whether concept, task, or reference -- is a stand-alone unit of publishable information. Above the typed-topic layer are any processing applications that may be driven by a superset DTD; below it are the two types of content models that form the basis of all specialized DTDs within the architecture. Next, we'll look at each of these layers in more detail.

DITA delivery contexts
The delivery-contexts domain represents the processing layer for topical information. Topics can be processed singly or within a delivery context that relates multiple topics to a defined deliverable. Delivery contexts also include document management systems, authoring units, packages for translation, and more.

Figure 2. The delivery-contexts layer



DITA typed topic structures
The typed topics represent the fundamental structuring layer for DITA topic-oriented content. The basis of the architecture is the topic structure, from which the concept, task, and reference structures are specialized. Extensibility to other typed topics is possible by specialization.

Figure 3. The typed-topic-structures layer

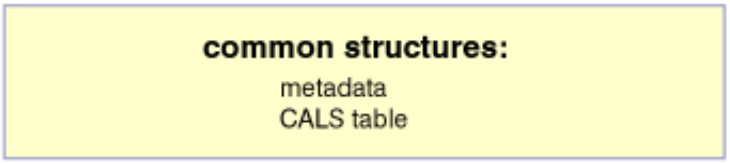
The four information types (topic, concept, task, and reference, which we call *reftopic*) represent the primary content categories used in the product documentation community. Moreover, specialized information types, based on the original four, can be defined as required.

As a notable feature of this architecture, other communities can extend or define additional information types that represent their own data by using the common and shared structures. Examples of such content

include product support information, programming message descriptions, and GUI definitions.

DITA common structures
The metadata and table structures are unchanging structures that can be used within any topic, even in other XML vocabularies.

Figure 4. The common-structures layer

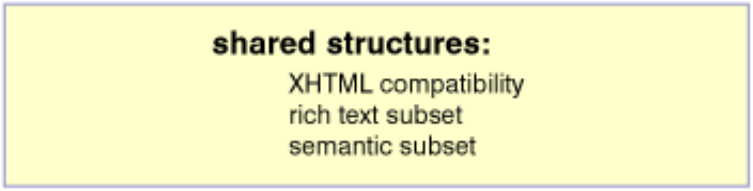


The metadata structure defines document control information for individual topics, higher level processing DTDs, or even for HTML documents that are associated to the metadata as sidefiles or as database records.

The table structure provides presentational semantics for body-level content. The CALS display model is supported in many popular XML editors.

DITA shared structures
The shared structures provide elements and content models that can be used in many types of technical documentation. These include basic document structures (word processor equivalence for emphasis and layout), a copy-and-paste compatible subset of XHTML 1.0, and semantically significant phrases and structures for content.

Figure 5. The shared-structures layer



The workgroup made an effort to select element names that are popular or common with HTML. And some semantic names have been borrowed from industry DTDs that support large SGML libraries, such as IBMIDDoc and DocBook.

Specialization
A company that has specific information needs can define specialized topic types. For example, a product group might identify three main types of reference topic: messages, utilities, and APIs. By creating a specialized topic type for each type of content, the product architect can ensure that each type of topic has the appropriate content. In addition, the specialized topics make XML-aware search more useful, because users can make fine-grained distinctions. For example, a user could limit a search for *xyz* to messages only or to APIs only. A user could also search for *xyz* across reference topics in general.

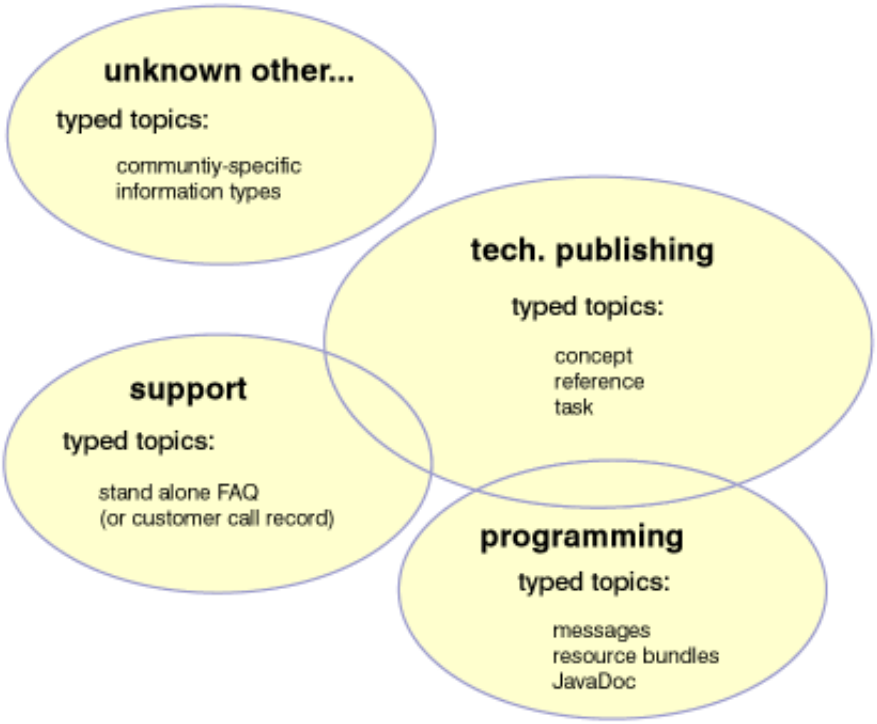
Rules govern how to specialize safely: each new information type must map to an existing one, and must be more restrictive in the content that it allows. With such specialization, new information types can use generic processing streams for translation, print, and Web publishing. Although a product group can override or extend these processes, they get the full range of existing processes by default, without any extra work or maintenance.

A corporation can have a series of DTDs that represent a consistent set of information descriptions, each of which emphasizes the value of specialization for those new information types.

Role of content communities in the DITA
The technical documentation community that designed this architecture defined the basic architecture and shared resources. The content owned by specified communities (within or outside of the defining community) can reuse processors, styles, and other features already defined, but those communities are responsible for defining their unique business processes based on the data that they manage. They can do so by creating a further specialization off of one of the base types.

Figure 6 represents how communities, as content owners at the topic level, can specialize their content based on the core architecture.

Figure 6. Relationship of specialized communities to the base architecture



In Figure 6, the overlap represents the common architecture and tools shared between content-owning communities that use this information architecture. New communities that define typed documents according to the architecture can then use the same tools at the outset, and refine their content-specific tools as needed.

Resources

- Find out more about DITA in a companion article, Specialization in the Darwin Information Typing Architecture, which outlines how to implement DITA.
- Find out how to join the discussion in the DITA forum, moderated by Don Day and Michael Priestley.
- Go directly to the DITA forum, moderated by Don Day and Michael Priestley.
- Download the latest DITA DTDs, style sheets, and sample documents, or download the original version.
- Refer to the DITA FAQ set.
- Get some background on the topic of information architecture at the Argus Center for Information Architecture or the 10 Questions about Information Architecture site.

About the authors

Besides his main work as husband, father, and cat lover, Don designs and supports publishing tools for IBM's Information Development community and has represented IBM on the W3C XSL and CSS Working Groups. He has B.A.s in English and Journalism and an M.A. in Technical and Professional Communication from New Mexico State University. You can contact Don at dond@us.ibm.com.

Michael Priestley is an information developer for the IBM Toronto Software Development Laboratory. He has written numerous papers on subjects such as hypertext navigation, singlesourcing, and interfaces to dynamic documents. He is currently working on XML and XSL for help and documentation management. You can reach Michael at mpriestl@ca.ibm.com.

Dave Schell is IBM's chief strategist and tools lead in support of its technical writing (User Technology) community. You can reach Dave at dschell@us.ibm.com.