CTN Test Report
91-012

AFTB-ID-91-013

How to VALIDATE DOCUMENT TYPE DEFINITIONS (DTDs)

April 30, 1991

DTIC QUALITY INSPECTED

Prepared for
Air Force Logistics Command
Air Force CALS Test Bed (LMSC/SBC)
Wright-Patterson AFB, OH 45433-5000

19960826 108

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

CTN Test Report
91-012

AFTB-ID-91-013

How to VALIDATE DOCUMENT TYPE DEFINITIONS (DTDs)

April 30, 1991

Prepared By
SOFTWARE EXOTERICA CORPORATION/CENTECH for
Air Force CALS Test Bed
Wright-Patterson AFB, OH 45433

CTN Test Report
91-012

AFTB-ID-91-013

How to VALIDATE DOCUMENT TYPE DEFINITIONS (DTDs)

April 30, 1991

Prepared By
SOFTWARE EXOTERICA CORPORATION/CENTECH for
Air Force CALS Test Bed
Wright-Patterson AFB, OH 45433

AFTB Contact
Gary Lammers
(513) 257-3085

CTN Contact
Mel Lammers
(513) 257-8882

Prepared for

How To Do
Independent Verification And Validation
On An
SGML-Defined
Markup Language

Table of Contents

1. INTRODUCTION 1

1.1 Background 1
1.1.1 CALS 2
1.1.2 SGML 3
1.1.3 MIL-M-28001 and MIL-M-28001A 4

1.2 What Is A Markup Language? 6
1.2.1 Types of Markup Languages 7
1.2.2 Format and Information in a Markup Language 8
1.2.3 CALS Markup Languages 8
1.2.3.1 Markup Minimization 9
1.2.3.2 OMITTAG 10
1.2.3.3 Short References 10

1.3 Definitions 11

2. THE I V & V PROCESS 12

2.1 Initial Considerations 12
2.1.1 Reference Documents 12
2.1.2 Context 13
2.1.3 Audience 13

2.2 Procedure 14
2.2.1 Background Analysis 16
2.2.2 Problem Analysis 17
2.2.2.1 Purpose of the Markup Language 17
2.2.2.2 Goals of the Markup Language 18
2.2.2.3 Constraints on the Markup Language 18
2.2.3 Design Analysis 19
2.2.3.1 Structure Outline 19
2.2.3.2 Structure Diagram 20
2.2.3.3 Data Dictionary 24
2.2.4 Markup Language Definition Analysis and Testing 24
2.2.4.1 Links from DTD to Design 25
2.2.4.2 Testing the DTD 25
2.2.4.3 Testing the Document Instance 26
2.2.5 Testing Sample Documents 28
2.2.5.1 Markup Errors 29
2.2.5.2 Cross-References 29
2.2.5.3 Proper Functional Identification 29
2.2.5.4 Markup Inconsistencies 30

2.3 Completing the I V & V Task 30

3. SUMMARY 32

1. INTRODUCTION

Document Type Definition Independent Verification and Validation is the final step in the process of creating and testing an SGML Document Type Definition prepared in compliance with the CALS standard MIL-M-28001A. The sequence of tasks of which Independent Verification and Validation is the final step includes:

- examination and analysis of a selected class of related documents, and of the use that is expected to be made of those documents;
- design of a markup language which effectively and efficiently captures the information in documents of the selected type;
- formal description of the markup language using a Document Type Definition (DTD) conforming to the Standard Generalized Markup Language (SGML); and
- testing the markup language and its formal description by marking up sample documents of the selected class and processing them in ways that exemplify their expected uses.

Independent Verification and Validation (I V & V) examines these tasks, determines if these tasks have been completed, how accurately and effectively each one was accomplished, and how well each task is documented. I V & V is crucial to confirming the usability of a DTD produced by a DTD Design Project.

This report describes:

1. the background and principles on which I V & V is based,
2. what a I V & V project is looking for in a DTD and its accompanying documentation,
3. how to analyze a DTD and its documentation, and
4. how to prepare a Document Type Definition Independent Verification and Validation Final Report.

The DTD I V & V Final Report provides the documentation of the task of Independent Verification and Validation. It is the deliverable from a I V & V project.

1.1 Background

The military specification MIL-M-28001 provides a basis for developing text markup languages that form a key component of the Computer-aided Acquisition and Logistics Support (CALS) initiative of the U.S. Department of

Defense (DoD) and of industry. The basic markup language provided in MIL-M-28001 does not in all cases adequately support the content requirements of technical manuals. During the transition from a paper to a digital environment, other markup languages have had to be developed, based on MIL-M-28001, to capture the critically important information content of these documents. Study of various types of technical manuals has revealed that some function very differently from others, and it has become evident that one general style of markup language would be insufficient for all requirements. Some technical manuals are not even best suited to print applications, but would be represented more effectively by hypermedia applications with a heavy reliance on graphics. A markup language designed primarily for a print-oriented application would not be robust enough to support the functionality required in such manuals.

The markup languages developed in addition to the basic one provided in MIL-M-28001 incorporate new concepts, it is critical that they be tested, verified, and validated. This report describes the process of independent verification and validation of markup languages, their formal descriptions, their documentation, and their use.

1.1.1 CALS

CALS is designed to achieve the transition from paper-based acquisition and logistics processes to automated and integrated modes of operation. In more concrete terms, CALS is a set of standards and specifications which dictate the automation requirements for a set of technical data including engineering drawings, product definition and logistics support analysis data, technical manuals, training materials, technical plans, and reports.

The scale of the problem addressed by CALS is exemplified by the subset of technical manuals called technical orders. The technical orders include publications that contain instructions for the installation, operation, maintenance, training, and support of weapon systems, weapon system components and support equipment. The U.S. Air Force requires approximately 150,000 technical orders to operate and maintain their systems and equipment.

The present Air Force Technical Order system is paper-based, with documents printed, distributed, and stored at enormous expense, monetarily and environmentally. The time spent accessing the information in these documents and the time spent producing documents and updating them also represent an enormous expenditure. With the current print-based system, there can be a substantial time-lag (measured in years) between the date on which a change is implemented physically in the field and the date on which the documentation reaches those who actually use and maintain the changed equipment.

The implementation of CALS will help alleviate all types of expenses involved in writing and maintaining text and drawings. CALS will begin to replace the current expensive, unwieldy system by a state-of-the-art ". . . 'system of systems' that can create, transform, store, reproduce, change, distribute, and use information as it evolves through the design, manufacture, maintenance, and logistics support [processes]."[1]

1.1.2 SGML

The Standard Generalized Markup Language (SGML) is a machine-processable language which is used to describe the syntax and structure of markup languages. A markup language is described in two parts:

1. The lexical structure of a markup language is described by an SGML Declaration. The lexical structure determines the form of the marks that are used to identify different parts of text. This includes what characters are used as delimiters and, where names are used in markup, what characters can appear in names, whether or not the distinction between upper- and lower-case letters is significant in names, and limits on the length of names.

   The lexical structure of a markup language also determines how characters are coded. Different computer character sets use different numbers to represent the upper-case letter "A", for example. Which number is used to represent the upper-case letter "A" is determined by the SGML Declaration. Character sets are usually invariant over a large class of computers and applications, so the encoding of the character set is usually not of concern to the user. However, when documents are prepared on one system and transfered to another, some provision has to be made for describing the character set used in a document. The SGML Declaration allows for this.

2. The syntactic structure of a markup language is described by a Document Type Definition (DTD). The syntactic structure determines how the marks and characters defined by the SGML Declaration are used in combination to represent information with complex structure.

What SGML does not provide for is the definition of how marked-up text is to be interpreted. In other words the "meaning" of a document's markup cannot be determined without documentation in addition to an SGML Declaration and a DTD. In addition, a DTD is not a suitable form of documentation either for personnel preparing marked-up documents or for personnel creating computer software that processes marked-up documents.

---

[1]Kurt N. Molholm, "The DoD Computer-aided Acquisition and Logistics Support (CALS) Initiative",*Electronic Transfer of Information and its Impact on Aerospace and Defence Research and Development*, AGARD-CP-466 (AGARD 1990), p. 14-1.

As a consequence, it is as important that the DTD I V & V task determine that a markup language is documented in ways usable by both these groups, as it is that the markup language perform its function of capturing the information in the selected class of documents.

1.1.3 MIL-M-28001 and MIL-M-28001A

MIL-M-28001, "Markup Requirements and Generic Style Specification for Electronic Printed Output and Exchange of Text", "establishes the requirements for the digital data form of page-oriented technical publications. Data prepared in conformance to these requirements will facilitate the automated storage, retrieval, interchange, and processing of technical documents from heterogeneous data sources. The requirements set forth by this military specification include:

a. procedures and symbology for markup of unformatted text in accordance with this specific application of the Standard Generalized Markup Language (SGML),

b. SGML compatible codes that will support encoding of a technical publication to specific format requirements applicable to technical manuals,

c. output processing requirements that will format a conforming SGML source file to the style and format requirements of the appropriate Formatting Output Specification Instance (FOSI) based on the Output Specification (OS)."[2]

In other words, MIL-M-28001 provides a markup language for technical manuals, the formal description of that markup language by an SGML DTD, and a formal specification of how a document so marked up is to be formatted.

MIL-M-28001, published 26 February 1988, contained a "conforming" DTD and a "nonconforming" DTD. The conforming DTD defined a markup language that was intended to apply to all documents whose published format conformed to MIL-M-38784B, "Manuals, Technical: General Style and Format Requirements". The nonconforming DTD was intended to provide a starting point for documents whose requirements are not accommodated by the conforming markup language, either because the published documents varied significantly from the specifications in MIL-M-38784B or because the intended uses of the documents varied significantly from those envisioned in MIL-M-28001.

[2]MIL-M-28001A, Military Specification —Markup Requirements and Generic Style Specification for Electronic Printed Output and Exchange of Text, 20 July 1990, p. 1.

MIL-M-28001A was published 20 July 1990. The major change from MIL-M-28001 was the recognition that the requirements of few, if any, documents could be satisfied by MIL-M-38784B, especially with the increasing need to support the information storage requirements of database applications. MIL-M-28001A therefore concentrates on what in MIL-M-28001 were called "nonconforming" documents and provides a "template" document type definition which defines a "baseline tag set", on which markup languages are expected to be based. The "conforming" DTD is relegated to a later annex, primarily to serve as an example of using the template DTD. MIL-M-28001A can be thought of as describing a class of markup languages, of which the conforming one is an example.

Existing markup languages developed under the CALS umbrella are based on MIL-M-28001, MIL-M-28001A, and on intermediate drafts of MIL-M-28001A. MIL-M-28001A incorporates changes to MIL-M-28001, especially with regard to the markup of tables. There is a tendency, therefore, for markup languages based on different versions of MIL-M-28001 to vary: the same thing may be done in different ways. MIL-M-28001A is now the approved standard, and markup languages based on earlier drafts should be considered for being brought into line with its specifications.

MIL-M-28001A also introduces a standard way of describing formatting requirements for marked-up documents. Where formatting is a significant requirement of documents in the selected class, a FOSI specification of how they are to be formatted, based on that in MIL-M-28001A, should be provided.

As more interest is developed in electronic, non-print applications of technical manuals it is to be expected that even greater variations will be required in the markup languages that support such applications. Examples from existing CALS document types for which variant markup languages have been proposed are the Fault Isolation and Fault Reporting Manual types defined by the publishing specification MIL-M-83495A, "Manuals, Technical, On Equipment Set, Organizational Manuals: Detailed Requirements For Preparation Of". The production of automated aids to the detection and repair of faults in equipment presents requirements far different from those of a print-oriented environment. As a consequence, manuals such as these need to be marked up in different ways from those intended primarily for print.

These variant markup languages introduce the requirement for Independent Verification and Validation. The correctness, applicability and utility of the variant markup languages and of their formal description cannot be assumed without thorough checking and confirmation.

MIL-M-28001A itself embodies new concepts, and any examination of its application necessarily consists in part of examining the applicability of the

class of markup languages based on MIL-M-28001A, and the utility of the form of documentation presented in MIL-M-28001A.

1.2 What Is A Markup Language?[3]

Although a project that results in the development of an SGML DTD based on MIL-M-28001 is usually described as "DTD Design", the primary deliverable from such a project is a markup language for a selected class of documents. This primary deliverable is documented in two forms:

1. The DTD itself is a description of the markup language in a machine-processable form. Using the DTD, computer software can "parse" text that uses the markup language, and perform further processing on its data content. Examples of further processing include formatting the text for the printed page, analyzing text for readability or technical content, and loading the text into a database.
2. Human-readable documentation is provided both to describe, in natural language terms, how to use the markup language, and to describe, in more formal terms, what information is captured by the markup language.

Additional documentation describes the criteria which form the basis of the development of the markup language and the DTD , and describes the process of testing the markup language and its definition.

A markup language provides the basis for entering and storing text interspersed with marks (or markup). The markup both identifies the components of text and distinguishes these components. Once marked up, these components can be assembled into a structure that makes explicit their interrelationships.

A markup language is similar to a human language in that it "packages" components of information in a structure. This structure is determined by the method of packaging; it does not necessarily directly represent the structure of the information itself. The syntactic structure of a sentence of speech is the structure of the sentence, not the structure of what is being talked about. The syntactic structure of marked-up text is likewise the structure of the markup. This will only be the same as, or even similar to, the structure of the information represented by the markup when the latter structure is linear and relatively simple.

---

[3]The discussion of markup languages has been copied and adapted, with permission, from Exoterica Complex Tables (EUM09-0291-2), Software Exoterica Corporation, 18 February 1991, pp. 1-2.

The structure of a markup language, like that of a human language, is defined by its grammar. Like utterances in a human language, documents marked up in a markup language have to be processed to determine their "meaning". Determining the "meaning" of a complex marked-up document typically involves repackaging the information in the document into the form (structure) required by a text formatting language or a database system. This repackaging can require:

- making copies of text,
- moving text around,
- creating "boilerplate" text, and
- transforming text (the simplest form of which is exemplified by uppercasing titles).

Using a markup language to enter text has long been found to be the most effective method of creating complex text documents. When used in conjunction with generic markup techniques, markup languages allow precise identification of the components of documents while at the same time avoiding limiting the documents to specific representations: multiple uses can be made of a single generically marked-up document. Documents stored as plain-text files with clear text markup are amenable both to efficient entry and editing, and to processing by computer programs.

1.2.1 Types of Markup Languages

There are many different kinds of markup languages. A markup language can represent a very abstract view of information. This is the ideal of generic markup languages. A markup language can, on the other hand, directly represent a physically-realizable structure, such as that of a formatted document. Most markup languages are somewhere between these two extremes, although the growing need to use data for more than one purpose means that there is an increasing tendency to aim for the generic ideal.

When designing a markup language, care must be taken to distinguish between the marks and the structure of the markup language. The markup must allow the text components and their structure to be identified. It must also be easy to enter, and easy to read, both to ensure reliable markup on entry, and to facilitate editing and revision. On the other hand, there is no need to make the structure of the markup language explicit. The marks themselves are important to the input operators and editors. The structure serves to assemble the marked-up text for later processing and transformation into the structure of what is being represented.

The kinds of markup languages that can be designed are limited by the tools that are available for implementing them. The adoption of SGML, a standard

for describing the grammar and marks of markup languages, and the more recent introduction of sophisticated tools for processing marked-up documents, means that powerful markup languages can now be designed and implemented.

1.2.2 Format and Information in a Markup Language

Information captured by a markup language can either be print-oriented, data-structure-oriented, or a combination of both. The usual arrangement of text into various levels of headings and paragraphs corresponds both to the manner in which it is usually formatted, and to a logical, hierarchical structure of information: paragraphs grouped under a heading usually deal with the same topic. Headings and paragraphs, therefore, represent both print-oriented information and an underlying structure of data content. The marks that identify topical keywords or codes in text that have meaning in another context, such as references to other material or part numbers, illustrate the capture of non-print-oriented information. Tables, on the other hand, usually represent a particular presentation of the information they contain. This presentation is primarily print-oriented.

Although data-content-oriented markup is the ideal, this is sometimes impractical, as tables illustrate. A markup language for the information in tables would ideally provide markup for each kind of information in each table, in such a way that all the interrelationships could be found. A presentation or storage format for each type of table would then be determined by formatting, presentation or database software However, tables are ubiquitous and varied, and creating and supporting what would in effect be a separate markup language for each table is generally impractical. The alternative is a generalized presentation-oriented markup language for tables as presented in MIL-M-28001A.

1.2.3 CALS Markup Languages

SGML was established as the standard for defining markup languages used by Federal Organizations (and therefore CALS) by Federal Information Processing Standard FIPS PUB 152. FIPS PUB 152 specifies that the OMITTAG and FORMAL optional features of SGML be supported, as well as the "Core" concrete syntax. The OMITTAG feature provides additional functionality for the definition of markup languages. The FORMAL feature constrains the form of system-independent file identifiers to that described in ISO 8879. The Core concrete syntax differs from the "Reference" concrete syntax usually used by SGML-defined markup languages in not allowing any "short references" to be defined. The FIPS allows, but does not require, the support and use of other optional features of SGML.

MIL-M-28001 and MIL-M-28001A use only the OMITTAG and FORMAL optional features. MIL-M-28001 uses the Core concrete syntax. MIL-M-28001A

uses the Reference concrete syntax, although short references are used only in limited contexts.

1.2.3.1 Markup Minimization

The OMITTAG optional feature and short references provide two ways of determining which "marks" are used in a markup language. They are called "minimization" features because, among other things, they reduce the number of keystrokes required to enter markup.

The reduction of keystrokes and the consequent improvement in the efficiency of entering marked-up text is an important benefit of using SGML-defined markup languages. It is not, however, the major reason for using SGML's minimization features. The major benefit of minimization is in the improved clarity of documents that use appropriate markup languages defined using these features.

Appropriately designed markup languages improve the clarity of a document by:

1. reducing the amount of extraneous markup,
2. making the size of the "marks" appropriate to what they are marking up, and
3. using "marks" of a form that suites the surrounding material.

The OMITTAG optional feature and short references support all these ways of improving the clarity of markup. OMITTAG reduces the number of items of markup required. Short references allow the use of forms of markup other than "tags" (i.e. start- and end-tags). Short references are especially useful in markup-intensive contexts where tags would tend to overwhelm small fragments of text.

Clearly understandable document markup benefits both data entry operators, and editing staff. For data entry operators, having no unnecessarily redundant markup and having markup that is meaningful in terms of the text around it reduces the incidence of errors. For editors, this same markup makes documents easier to read and interpret.

These SGML features have been designed in a manner to allow these benefits to be realized with a wide range of applications, and on a large number of systems. They are not limited to one particular application or hardware base.

While consistency between markup languages is important, a markup language based on MIL-M-28001A should consider using both OMITTAG and short references in a more extensive manner that does MIL-M-28001A where doing so would produce a functionally efficient markup language.

1.2.3.2 OMITTAG

OMITTAG allows a markup language definition to specify that certain items of markup can be implied by the presence of other markup (using well-defined rules provided in ISO 8879). For example, the end of a paragraph can be implied by the start of a new paragraph or chapter, if the formal definition of the markup language is appropriately constructed. Similarly the start of a chapter can be made to imply the start of a chapter title.

OMITTAG is a powerful accessory in markup language definition. It substantially reduces the amount of markup required — usually by close to 50%. OMITTAG is easy to use. Markup operators can be provided with simple rules for its use, such as "starting a paragraph ends the previous paragraph". (The general rules for OMITTAG can be summarized as: the start of something can be implied by that something being the only thing allowed at that point, and the end of something can be implied by some text or markup being present in the document that is not allowed without the something being ended prior to what is present.)

MIL-M-28001 and MIL-M-28001A do not use the full power of OMITTAG. These specification take advantage of OMITTAG in most cases where an end can be readily implied. The ability to omit redundant markup at the start of an element, on the other hand, is used only in a few of the possible cases.

1.2.3.3 Short References

MIL-M-28001A makes use of the mathematics markup defined in ISO/IEC TR 9573.[4] The math package makes more extensive use of OMITTAG than the rest of the MIL-M-28001A markup language. As well, it makes use of the short references allowed by the Reference concrete syntax. Short references allow the use of context-specific marks in place of the basic named tags and entity references whose syntax is defined by ISO 8879. As their name implies, short references are (usually) shorter than the equivalent tags and entity references, and are especially appropriate for contexts where intensive markup is required.

The math package is illustrative of text that requires more detailed markup than that needed to mark a simple view of chapters and paragraphs. In such a context, tags and entity references would tend to overpower the small amounts of text that intervene between them, and make the content very difficult to enter and edit accurately. Although the need for consistency amongst markup languages used in a given environment dictates that the

[4]ISO/IEC 9573, Information processing — SGML Support Facilities — Techniques for Using SGML, International Organization for Standardization (ISO), 1989, pp. 83-98.

CALS style of using OMITTAG and short references be maintained for marking up the kinds of textual strictures dealt with in MIL-M-28001A, the increasing use intensive markup to capture detailed, non-print-oriented information will require increased use of the minimization features of SGML when marking up new forms of material.

1.3 Definitions

Document Type Definition (DTD) — "rules, determined by an application, that apply SGML to the markup of documents of a particular type. A document type definition includes a formal specification, expressed in a document type declaration, of the element types, element relationships and attributes, and references that can be represented by markup. It thereby defines the vocabulary for the markup for which SGML defines the syntax."[5]

Markup Language — a language whose expression consists of placing marks in text for the purpose of capturing its information content.

Standard Generalized Markup Language — "a language for document representation that formalizes markup and frees it of system and processing dependencies."[6]

Validation — the process by which soundness, reasonableness, and adequacy are confirmed according to user requirements and accepted principles. Applied in the context of CALS, the principles include the rules of SGML, MIL-M-28001, and all relevant publishing specifications.

Verification — the process by which correctness and accuracy are confirmed by means of examination or demonstration; testing the usability of what was built. Applied to a CALS-based markup language and DTD, examination will include parsing by an SGML parser and demonstration will include the use of the markup language described by the DTD to mark up a sample document with acceptable and accurate results.

---

[5]ISO 8879-1986(E), Information processing — Text and office systems — Standard Generalized markup Language (SGML), International Organization for Standardization (ISO), 1986, p. 10.

[6]Ibid. p. 19.

2. THE I V & V PROCESS

This chapter describes the I V & V process itself. In its simplest terms, the process consists of determining whether a CALS markup language conforms to the principles described in the previous chapter.

2.1 Initial Considerations

At the start of the process of independent verification and validation, reference documents must be acquired, and the context and audience of the I V & V Final Report determined.

2.1.1 Reference Documents

A list of all relevant standards, specifications and directives used during the development of a markup language must be delivered as part of the documentation of the markup language. The documentation available to the I V & V task must include:

- MIL-M-28001A: Markup Requirements and General Style (dated 20 July 1990).

  If the markup language was based on MIL-M-28001 (dated 26 February 1988) or a draft of MIL-M-28001A dated prior to 20 July 1990, that draft must also be available.

- ISO 8879-1986: Information Processing — Text and Office Systems — Standard Generalized Markup Language (SGML)

  ISO 8879 must be used only in conjunction with Amendment 1 (published in 1988). This amendment incorporates vital corrections and modifications to ISO 8879.

- Standards and specifications which have historically been used to describe the marked-up material. For Air Force technical manuals, for example, the major document of historical importance is MIL-M-38784B: General Style and Format Requirements.

  Some markup languages have been developed based on MIL-M-38784C. If this is the case, MIL-M-38784C must also be available.

  If additional or (for other services) alternative publication standards have been used in the development of the markup language, they must be consulted, either in addition, or instead of MIL-M-38784.

- All directives which form part of the project's background and which may have influenced design decisions.

- The SGML DTD which formally describes the markup language. The DTD must be available to the I V & V task in machine-readable form so that its correctness can be confirmed by computer processing.
- The sample marked-up documents used to test the markup language. The marked-up documents must also be available to the I V & V task in machine-readable form, to allow machine analysis.

  The sample marked-up documents must contain examples of each type of markup allowed by the markup language. This means that the sample documents must contain examples of each type of text allowed by the original publishing specifications.

  Where they exist, copies of the original documents from which the sample marked-up documents were created must be provided, together with the results of processing the sample marked-up documents.
- The documentation produced during the design and testing of the markup language.

2.1.2 Context

The context of the I V & V process must be well understood. The most important consideration is the rationale for developing the markup language in the first place. This rationale, the history of the project, and all other background considerations on which project decisions have been based must be documented, either in the directives that initiated the project, or in additional documentation provided to the I V & V task. The correctness and appropriateness of a markup language cannot be fully evaluated without considering all the decisions which determined its design.

The context of a project must be fully documented prior to the I V & V task. Personnel performing I V & V have to be able, if necessary, to recreate any part of the process that resulted in the creation of the markup language, the DTD that formally describes it, and their documentation.

2.1.3 Audience

Another consideration is the audience for the documentation of the markup language development project and the audience of the I V & V Final Report.

The terminology of the standards, specifications, etc. being followed should have been used throughout the documentation for the markup language development project. Any new terminology must have been introduced only with adequate definitions. The markup language documentation is addressed to three audiences, each of which should be expected to comprehend only the terminology appropriate to its use of the markup language:

1. The markup language itself should be documented in terms comprehensible to personnel doing markup, or at least to those who direct such personnel. There should be no unnecessary references to the details of SGML concepts, and details of the applications to which the marked-up documents are to be put should be avoided.
2. Documentation should be available that allows application developers to make use of marked-up documents without being concerned with the details of the markup language. This audience is only interested in the information content of the documents and the structural interrelationships between components of information. If the documentation discusses the use of documents by a database application, for example, then terms from database technology can be used in the report without jeopardizing comprehension.
3. The SGML DTD that formally describes the markup language should be described in terms of SGML concepts, in terms of markup language requirements, and in terms of what information is made available to applications. The DTD is the link between the markup language and applications that make use of captured information, and its design must take both of these subject domains into consideration.

   It should be noted that the details of the design of the DTD are only of interest to its designers and those, such as the I V & V personnel, who are tasked with examining it. Users of the DTD are no more concerned with its details than the users of a computer program are concerned with the details of the program's implementation.

The findings of the I V & V task are of concern to those who are in a position to act on its findings and recommendations, as well as to members of the above three audiences. The I V & V Final Report must use the same terminology as is used in the documentation of the markup language. In addition, general findings must be reported without using the technical language appropriate to discussing the details of using a markup language.

2.2 Procedure

The I V & V task consists of subtasks that parallel the tasks that preceded it: markup language design, formal definition, and testing. Personnel doing I V & V follow a set of procedures that match those followed by the personnel who developed the markup language. At each point in each procedure there must be a document, on paper or in machine-readable form, which provides the link between the original task and the I V & V subtask.

The following table demonstrates the parallelism between the markup language development process and the I V & V process, and shows the

documents that provide the links between each pair of corresponding processes.

DTD Development Process	Link	I V & V Process
Background analysis.	Standards and specifications	Background analysis.
Problem analysis: define purpose, goals, and constraints.	Supporting documentation	Evaluate analysis.
Markup Language Design.	Structure description and Data Dictionary	Examine the documentation for completeness.
Formal Definition — define the markup language using an SGML DTD.	DTD	(a) Evaluate correctness of DTD by machine parsing. (b) Evaluate accuracy of DTD by comparing it to the markup language design documentation.
Markup Testing: mark up sample documents.	Sample marked-up documents	Confirm accuracy of the sample documents.
Information Capture Testing: use the marked-up document samples. For example, produce formatted output.	Output samples FOSI	Compare output samples with expectation (represented by previously published documents, where available).

Table 1 — Relationship of Markup Language Development and I V & V Tasks

The documents used by the I V & V task are evidence that original tasks were performed. The following subsections describe the subtasks of the I V & V Tasks, and draw more detailed parallels with the original markup language development task.

If the markup language project has not produced all the documentation required to do the analysis required by I V & V, the I V & V personnel may have to create the missing documentation in order to enable them to complete the I V & V task. Such documentation should be incorporated in the DTD I V & V Final Report. It is then available for incorporation into the documentation of the markup language design documentation.

2.2.1 Background Analysis

Before the design of the markup language can begin, personnel must be thoroughly conversant with the standards, specifications and directives relevant to the task at hand. Personnel doing I V & V must be equally familiar with these documents, and so must repeat in full this first subtask of the personnel who designed the markup language.

The major standards and specifications to be analyzed provide the following information:

1. ISO 8879 "provides a coherent and unambiguous syntax for describing whatever a user chooses to identify within a document."[7] This is the general authority on SGML and is the standard on which MIL-M-28001, MIL-M-28001A and all other CALS formal markup language definitions are based.

2. MIL-M-28001 and MIL-M-28001A provide tools for the design of markup languages for documents which conform to publishing specifications based on MIL-M-38784. A Baseline Tag Set is provided as a template for a markup language. A formal definition for a markup language for the class of documents described by MIL-M-38784B is included, together with markup language components that can be used to assemble new markup languages.

   MIL-M-28001A contains an Output Specification. An Output Specification is a tool for defining a Formatting Output Specification Instance (FOSI) which, in a similar manner to an SGML DTD, provides a formal definition of how documents marked up using a particular markup language are to be formatted. New markup languages should be accompanied by a FOSI.

3. Content and processing specifications provide the historical basis for understanding the content of existing documents. Traditionally, the primary technology used for disseminating the information found in technical manuals was print, and as a consequence, the print formatting specifications for the different types of manuals provide the largest single source of information about the documents other than MIL-M-28001A.

   MIL-M-38784 and other publishing specifications are the primary authority on the components information present in each type of document and the interrelationship between these components. It must be clearly understand that the goal of these publishing specifications is not to provide this information; they exist primarily to describe the format in which already existing information is to be presented and have a print-medium orientation. It is generally not possible to design a

[7] Ibid. p. 2.

markup language (other than a text formatting language) based entirely on these specifications.

The limitations of a publishing specification becomes clear when an attempt is made to produce a formal description of a markup language for the class of documents based on the publishing specification. Often there is not enough information explicitly stated in the specification to determine what combinations of structural elements are allowed in conjunction with others, and what elements exclude the presence of others.

Nonetheless, publishing specifications do contain substantial information, and generally form the largest single source of information on the potential content of a document.

When publishing specifications are provided as part of the information on what a document must contain, they should be accompanied by sample print documents that conform to the publishing specifications. The sample documents serve to make explicit much of the information that is implicit in the publishing specifications. The sample print documents should exemplify everything described in the publishing specifications. This usually requires more than one document.

These standards and specifications do not provide all the information needed at this point in the task. Directives must also be provided indicating the goal or goals of the markup language being designed. At its simplest, the goal could be to reproduce the print forms of the documents. At its most complex, the goals could be to provide enough information to enable an interactive multimedia presentation of the documents to be created. The driving goals of the CALS initiative are to take operational documentation away from a solely print-oriented technology, so there must be stated goals for any markup language project that take it beyond simply reproducing print documents.

2.2.2 Problem Analysis

2.2.2.1 Purpose of the Markup Language

This corresponds to what is actually the first step in the process of DTD design. One must be able to state simply and clearly the "why" of the project. One or two sentences should convey in general terms why the DTD is being written and provide some information about the scope of the DTD. A general statement describing the potential uses of the marked-up data is appropriate here. At the end of the I V & V process, one should be able to refer to the statement of purpose and ascertain that the DTD reflects the reasons it was written.

Since those performing I V & V on a DTD are cognizant of all of the same information available to those designing and writing the DTD, fair

evaluations can be made concerning the correctness and appropriateness of the statement of purpose. This objective look at the purpose envisioned by the DTD developers may suggest that the purpose as stated has been, for example, too wide, or too narrow, or perhaps inappropriate. Comments on the purpose should be compiled and included in the Final Report.

2.2.2.2 Goals of the Markup Language

Simply stated, the goals list "what" the DTD should accomplish. At this point one should have in mind the products of the DTD design and should write the goals so that each product will be given appropriate attention in the design process.

Goals should be stated in a logical sequence so that the design will proceed according to that sequence and will not attempt, for example, to do specific things before the general framework is designed.

In more specific terms, when a class of documents must comply with a publishing specification, goals for a DTD for that class of documents must make clear the fact that all of the relevant information in the publishing specification is reflected in the DTD, given the purpose of the DTD.

However, if the publishing specification dictates print formatting only, and the DTD is to be used for other purposes, the goals must explicitly state that the information captured in the DTD must be detailed and robust enough to serve all other purposes as well. For example, if a DTD will be used for a class of documents which must comply with a formatting specification, but which also will be included in a text database, the structure of the DTD must provide enough information for both cases, and the goals must be explicit about naming the specifications, databases, etc.

It must be explicitly stated that the DTD must be parsable by machine. If DTDs are written according to ISO 8879, then they must be parsable by an SGML parser, and that should be stated.

2.2.2.3 Constraints on the Markup Language

Constraints on a DTD design limit its use for specific reasons. The application which will use a DTD may impose constraints on the structure because it is able to use only a small set of elements. Perhaps the nature of the discipline of the marked-up documents is such that one would require a less "verbose" markup language. For example, when a document consists largely of lists of part numbers, which themselves are made up of a number of structural elements, one would not want lengthy tags to occur between the sections of the part number; shorter, more symbolic language would be desirable here. This type of constraint on the markup language should be stated clearly in the list of constraints.

In the CALS context, the elements' tag names are taken from the generic DTD provided in MIL-M-28001 and MIL-M-28001A. However, new elements which are defined during the DTD design exercise should exhibit a structure and language which follow the style of the MIL standards mentioned. This is a constraint specific to this context, but the same type of situation could occur frequently and must be documented.

2.2.3 Design Analysis

The markup language designed to conform to the established standards and to satisfy the stated goals of the project must be documented from two points of view:

1. from the point of view of the process of marking up text and
2. from the point of view of the form and content of information captured by markup.

Once the markup language development project is completed, these two forms of documentation serve as documentation not only for the markup language but also as the key components of the documentation of the DTD that formally describes it.

The documentation for the process of marking up text is usually represented by a "Structure Outline" or "Structure Diagram". Both of these techniques describe the "marks" placed in marked-up text, where these marks are allowed, and what they signify. The meaning of markup depends, in general, on where it is in a document. The structure presented with the markup indicates where markup is allowed (and required).

The documentation for the form and content of information captured by markup is usually represented by a "Data Dictionary". A data dictionary simply lists each component of the content together with its characteristics.

2.2.3.1 Structure Outline

A structure outline itemizes the allowed components of a marked-up document in order, as a list. Subcomponents of other components are indicated by indented lists. Each item, corresponding to a component or subcomponent, should be accompanied by information about:

- what the component represents, usually described in terms of the original publishing specifications and project directives on which the markup language is based;
- where in the original specifications and directives the information used to define the component was found;

- the markup that accompanies the text or subcomponents and indicates the presence of the component;
- how many occurrences of the component are allowed, or if it is optional; and
- any notes required to explain why the component exists, if it does not correspond one-to-one with some component of text described in the original publishing specifications or project directives.

For documents as structurally complex as most of those of concern to the CALS initiative, the structure outline has been found to be an inadequate description technique. It is very hard to find one's way around a structure outline. When deeply nested structures occur, textual indentation does not provide a strong enough visual clue as to which components are associated with each other, and which are subsidiary to others.

The underlying problem with structure outlines is that they present markup from a "hierarchical" point of view, whereas markup is primarily a linear task. This is even though the data structures represented by markup are hierarchical. For example, conceptually, a chapter consist of a title, paragraphs and subsections as subcomponents, and the subsection consists of a title, paragraphs, and other material. However, from the point of view of the markup language, the chapter title is followed by one or more paragraphs, a subsection title and paragraphs of the subsection.

2.2.3.2 Structure Diagram

The structure diagram is an alternative method of describing markup. It uses a graphical technique that focuses attention on the linear sequence of text and markup in a document.

As elements of structure are pulled from the document in question, they can be arranged graphically in terms of their sequence and occurrence. Their names should be kept as descriptive as possible at this point; in fact they should be assigned names in the same terminology as that in the data from which they were pulled. There should also be explicit links between the paragraph(s) in the publishing specification, for example, and the structural element as described in the data dictionary and in the graphic representation of the structure. Figure 1 is such a structure diagram.

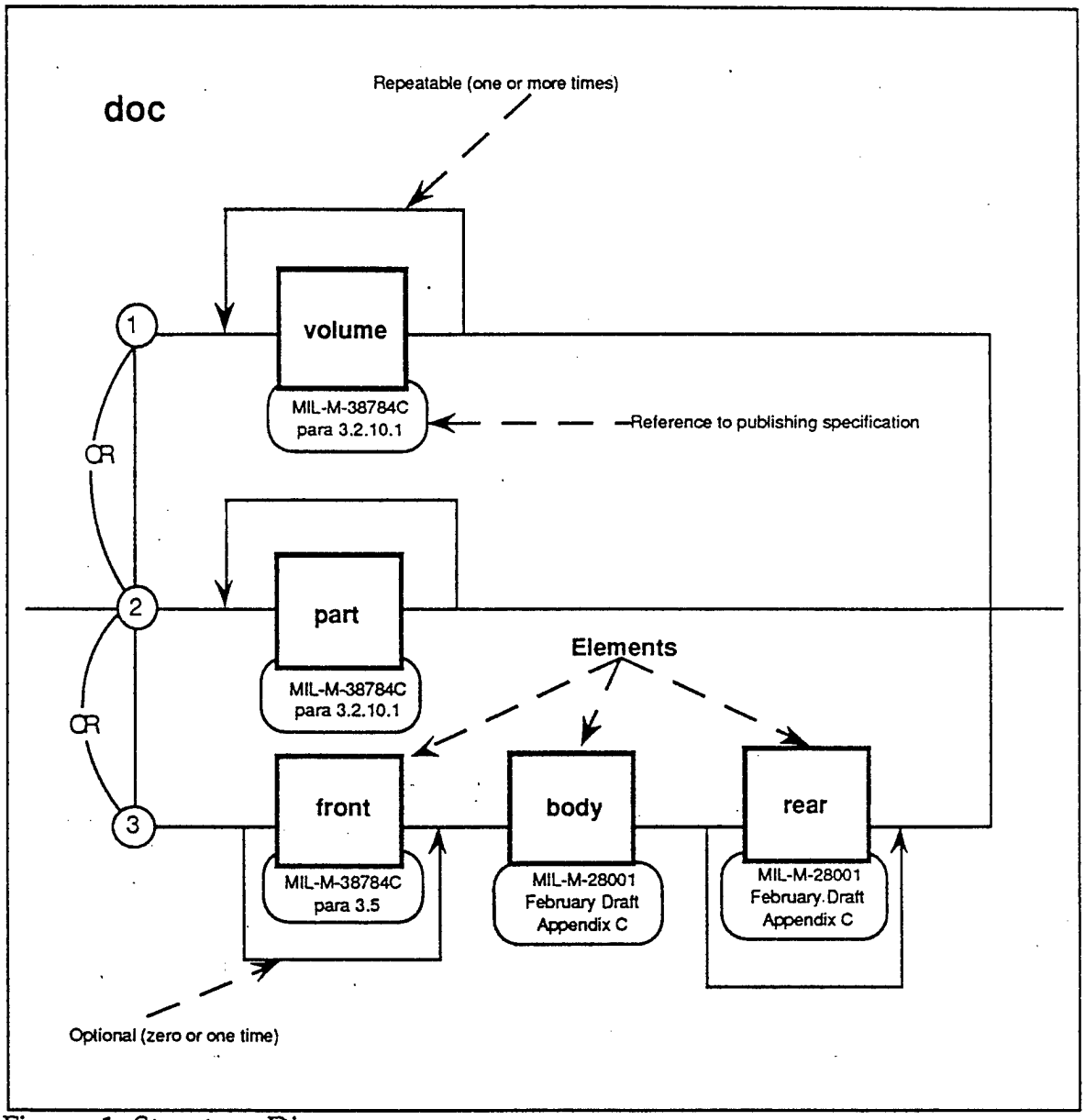

Figure 1: Structure Diagram

Figure 1 is an example of the type of graphic structure diagram which is the most useful way of portraying the structure of a document.

- Each structural element is assigned a descriptive name and a box is drawn around it. Large units of structure are identified first in this top-down approach.
- Attached to each box is a clear link to the paragraph in the publishing specification which explains the rules for that particular portion of the text.
- The way the structure is drawn reveals the sequence of elements.

- One element may simply follow the other. This is indicated by horizontal lines which run between the boxes, e.g. between the "front", "body", and "rear" boxes in Figure 1.

- A choice may exist as to which element may appear after a given element. In this case the boxes are organized one above another with lines leading into each from a central path. From that central point one could enter any one of the boxes and proceed through the diagram. Note that in Figure 1 there are three choices at the outset: one can begin by entering the "volume" box, the "part" box, or the "front" box.

- Other lines on the diagram indicate occurrence:

  - An element can occur once and lead directly to the next element (e.g. from "front" to "body" to "rear").

  - An element can occur optionally (zero or one time). This optionality is shown by lines which loop down and under an element box with an arrow at the end of the line indicating the direction in which that line flows. The flow is generally from left to right (e.g. the lines under the "front" box and the "rear" box).

  - An element can occur repeatedly (one or more times). This repeatability is shown by lines which loop up and back over an element box with an arrow at the end of the line indicating the direction in which that line flows. Optionality generally is indicated from right to left (e.g. the lines over the "volume" box and the "part" box).

  - An element can occur optionally or repeatedly (zero or more times). This is a combination of the two previous examples and is shown in Figure 2. The element can be missing completely, or it can occur many times over at that point.

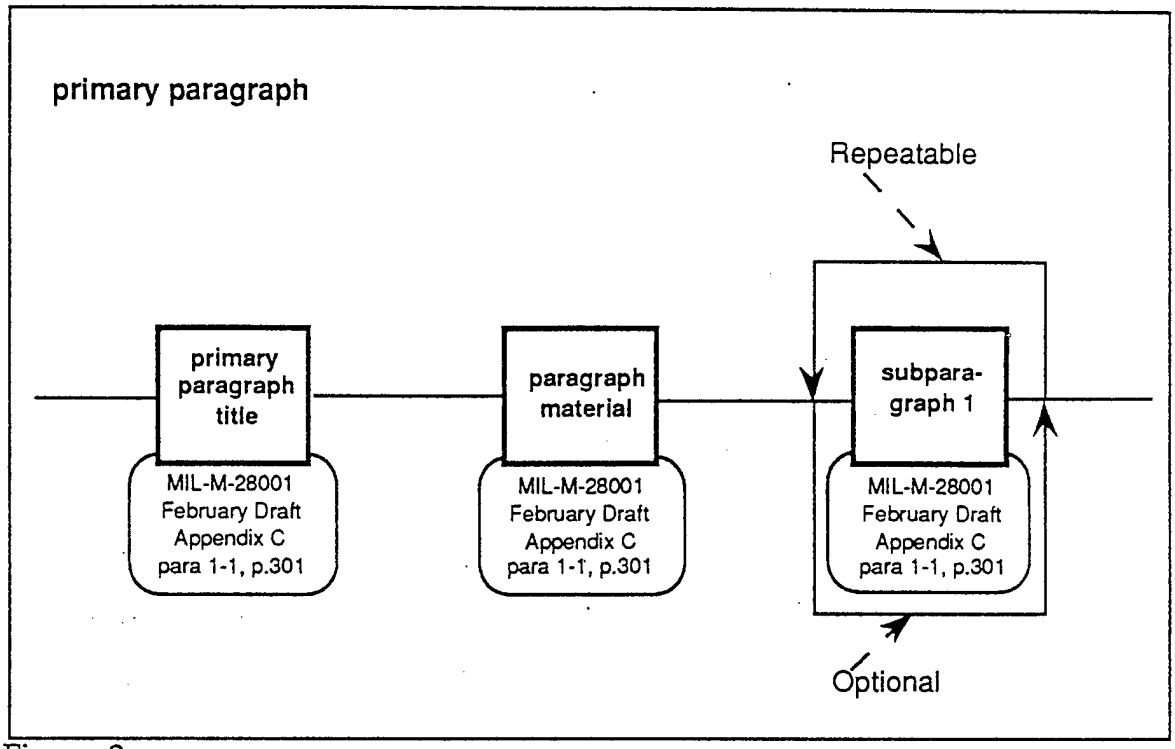

Figure 2

Up to this point, the structure diagrams have been used to describe the sequence of components of a marked-up document. Once this structure has been determined, the actual markup used in data entry can be added to the diagram. The following illustrates adding markup in the form of SGML start- and end-tags:

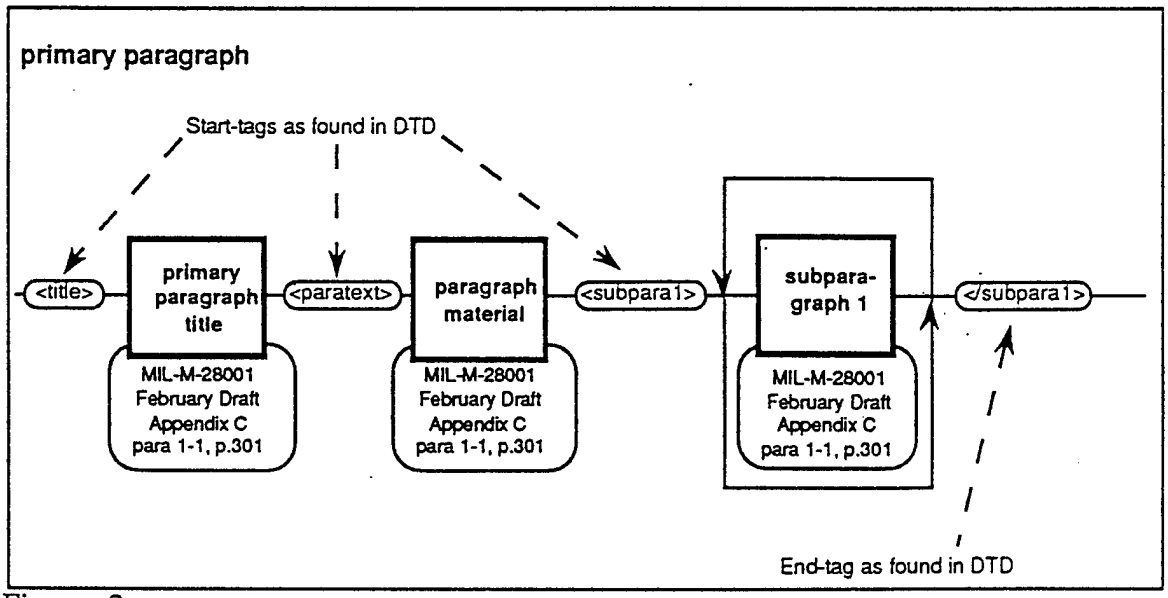

Figure 3

The structure outline has been found to be very useful in describing a markup language to data entry operators.

2.2.3.3 Data Dictionary

A data dictionary documents information captured by a markup language.

The data dictionary is one of the parts of markup language design which is labelled supporting documentation. This is where the more "technical" and detailed parts of markup language design begin. The type of manual or technical order whose content and structure must be described is reviewed from the beginning and the pieces of relevant information are "pulled out" . Each piece of information which one may want to talk about as a unique piece of structure should be described and documented in a number of ways.

- Elements — Elements which are pulled out as meaningful pieces of structure must be defined and described. The information content of each element must be described in "plain English" and everything about that element should be documented.
- Attributes — If the element can be qualified by a number of attributes, these should be listed as well with their possible values.
- Element content — Elements will often contain other elements and so the structure of an element is described as a content model in the same place as the rest of the description.
- Terminology — The terminology used in the data dictionary description should be the terminology of the relevant publishing specification or other standard. This is the place to which a person new to the project would refer to obtain a full definition and description of any given piece of structure.
- Order — Eventually, the data dictionary can be sorted into alphabetical order and so it can serve as a helpful index with which to navigate between manual, specifications/standards, DTD, and marked-up document.

2.2.4 Markup Language Definition Analysis and Testing

After verifying the correctness of the DTD design and validating the reasonableness of that design, one can proceed to look at how well the DTD reflects the design. A comparison of the DTD and the markup language with the DTD design analysis will reveal if the DTD correctly reflects the design and if the DTD reflects the design in valid ways.

2.2.4.1 Links from DTD to Design

There must be documentation available which demonstrates that the DTD is indeed the one that was designed. The DTD itself, with element and entity declarations, attribute lists, content models, etc. is not easy to read by anyone other than a person directly involved in the writing of the DTD. One essential part of supporting documentation is that which explicitly links the DTD to its design and analysis.

When the supporting documentation includes something like the structural diagrams described previously, it is a relatively easy task to link the structure to the DTD in terms of tag names. The DTD includes generic identifiers which are the actual tag names included in the markup language. It may not be immediately obvious what the tag refers to; there must be a clear method to link what is seen in the DTD with the documents which have been analyzed to produce that DTD. One way of providing that link is to produce another version of the structure diagram, this one with tag names actually enclosing the structural features which they mark up.

With this type of arrangement one can travel both ways between a DTD and its design analysis. In the documentation provided as part of the analysis, one can see the markup language which one expects to find in the DTD. Alternatively, if one is reading the DTD and needs more information about a specific structural feature, the tag name in question may be found in the structure diagram. Further, the structure diagram in turn will provide a reference to the paragraph number in the publishing specification (or other standard) which describes the rules for that particular portion of text.

There should be a one-to-one correlation between the parts of structure one finds in the DTD and the types of structure which one finds in the design analysis (in the data dictionary and the structure diagrams for example). If that one-to-one correlation is found to be untrue, then the DTD must be assumed to be faulty, given that the analysis has already been verified and validated. All of the information identified in the analysis process must be captured in the DTD. For I V & V purposes, if that fact cannot be verified, then all areas of discrepancy must be reported and corrections suggested.

2.2.4.2 Testing the DTD

At this point some of the verification and analysis process can be automated. It is hoped that the DTD will be provided for I V & V testing in machine-readable format. If that is not the case, however, then the print version of the DTD should allow for optical scanning. The physical format of the DTD is important in this case. For instance, if the DTD is printed in a tabular format, errors will occur wherever there is any slight irregularity in the format. Proof-reading and correction will be required in this case and it will be

difficult to ascertain absolutely that the resulting machine-readable form is identical to the original printed version. The fact remains however, that an electronic version of the DTD is the only one that can be parsed by computer, and that is the only way to accurately parse a DTD.

The most efficient and most accurate way to discover errors in a DTD is to use a parsing tool. The results of the parsing exercise will clearly reveal errors which have broken the rules of SGML. Such things as syntax errors, ambiguous content model errors, mixed content models, and errors in cross-referencing will be reported by the parser and these types of errors must be corrected before the DTD can be used. Problems which have to do with style only need not be corrected, but suggestions for improvements should be made.

There are other design details which should be reviewed when looking at the DTD specifically. DTDs are technical documents in their own right, and there is a style associated with writing a DTD. That style can be commented on as having an impact on the readability of the DTD, for example. One practice that would impair the readability of a DTD is that of using expanded entity references where they occur in the DTD. Sometimes an entity reference is used many times over, and when it is lengthy and provided in its expanded format each time, it makes the DTD visually and intellectually confusing. This also a dangerous practice, because there may be two lengthy, but very similar content models used in the DTD and it would be very easy to mistake one for the other.

Other points of style should be considered. There are some things which would not be technically wrong, but which would cause the document marked up according to that DTD to be exceeding unwieldy. An example of this is lack of tag minimization. If the DTD insists that all end-tags, for example, must be included, then the resulting document instance would look excessively cluttered. Not only that, the person marking up the document would have a longer and more tedious job.

2.2.4.3 Testing the Document Instance

Another automated portion of the I V & V process is that of parsing the document instance, or the marked-up document. Assuming that the DTD has been parsed successfully, one can test its application and evaluate the markup language for the following qualities:

- technical accuracy
- convenience
- consistency
- completeness.

Technical Accuracy. When a computer parses a document instance it will report on each error in markup according to the DTD it is parsing against. As in the case of the DTD, any serious technical errors must be corrected and the corrections reported as part of the I V & V process.

Convenience. This is the point at which the markup language can be evaluated, and here validation takes on a more prominent role. The markup language must be considered from the point of view of the person who must mark up the document and the reasonableness and convenience of the language take on prime importance. Those coding the input text must be able to identify quickly and easily what part of structure is at hand and how to code that part of structure. If short references are used, they must be easy to enter on the keyboard in terms of the sequence of keystrokes. They should also be characters which are available on most keyboards, because this type of work is intended to be device independent. There should also be a judgement made as to the type of tag used in what context. It should be determined whether verbose language and short tags are used in appropriate and convenient places from the point of view of the person marking up the document.

As mentioned earlier, it is more convenient for the person coding the document if such things as end-tags are allowed to be omitted. If end-tags may be omitted in logical places there is less likelihood of error than if all elements required end-tags. There would be places where a number of things end concurrently, and it would be easy to miss one of them when working quickly through a document.

Consistency. The markup language must be consistent in two ways. It must be used consistently:

- inside one DTD, and
- between related DTDs.

Some examples will show how the language is to be used consistently. If there is a particular type of structure, e.g. notes, cautions, and warnings, that can occur at various places in the document, and if these types of structure look the same wherever they do occur, the language which marks up these types of structure should exhibit similar characteristics. The markup language for a note, caution, and warning should be allowed in the same places in the structure consistently. The way these type of structures are marked up should be consistent as well. If the tags are verbose for some, they should all be verbose; if short, then they should all be short.

The second area of consistency is inter-DTD. In the specific context of the Air Force DTDs, elements which have the same structural function in a number of DTDs should have the same names in each DTD. Some elements may be

optional in some DTDs and not in others, but the names should be the same so as not to obscure the actual function of that element.

Completeness. All of the language necessary to mark up the document must be available. It would be frustrating and inefficient to find a piece of structure in a document for which no markup existed. There would be considerable time wasted as well, because the person marking up the document would have to contact the person who wrote the DTD and would have to wait for an amendment to the DTD. Sometimes this can be a lengthy process.

2.2.5 Testing Sample Documents

Two types of documents are tested in this process:

- marked-up document instances, or input samples, and
- final formatted documents, or output samples.

The purpose of the input and output samples is to provide the user with a practical indication of the relationship between a marked-up document and the kind of formatted document he is accustomed to seeing. The input samples, SGML marked-up documents, should contain a wide range of cases, and include as many commonly occurring cases as possible. The output samples, the formatted documents, should represent the marked-up documents in the form that most clearly indicates the function of the markup.

The input samples should use the optimum markup. To mark up a document for output style alone is a very limited use of SGML, so the input samples should not be limited by the requirements of the output samples. Using markup to identify all the information in the input samples should be the goal.

Sample documents should be examined for :

- information markup
- markup errors
- treatment of cross-references
- proper functional identification
- markup inconsistencies.

Information Markup. All sample documents should exhibit information markup, as opposed to formatting markup. Tags to identify such things as tools, test equipment, and material, i.e. <tool>, <testeq>, and <material>, do not affect formatting, but they do capture bits of information which would be very useful when included in a database, for example.

When information markup is used, it is vitally important that it be used consistently and completely. For example, when a part number, <partno>, is defined as an element, it should be used whenever part numbers occur. The sample marked-up document should be reviewed carefully to reveal that markup is complete and notes should be made documenting all cases of incomplete markup.

2.2.5.1 Markup Errors

Sample marked-up documents must be "edited" simply for accuracy. It is easy to tag a piece of information incorrectly when tag names are similar and one is working quickly. A quality assurance process should be performed on all markup to make sure that outright markup errors are caught, reported, and corrected.

This step can be automated by passing the document instance through a parser. All errors which represent violations of syntax and markup as prescribed in the DTD will be reported so that corrections can be made.

2.2.5.2 Cross-References

The markup of existing reference and cross-reference elements should be consistent, thorough and correct. For example, the publishing specification may specify that references to paragraphs, steps, tables, charts, figures, etc. should be tagged with a reference tag. The I V & V process should ascertain that all references are complete and accurate, and report on any that are not.

References are frequently made to other document numbers and titles and these should be consistently marked up inside the document in which they are referenced.

2.2.5.3 Proper Functional Identification

It is not sufficient for SGML markup to produce the correct output format in just one application. Markup must represent the function of what is being marked up, so that correct processing is assured on any system. It is important to identify cases where samples have been marked up to produce a particular typographical effect rather that to identify the function of the data they contain. For example, titles which are marked up with tags to make them print in a boldface font may actually be bold because they function as titles. The boldness should not be marked up, but rather the function of being titles.

Another example of proper functional identification is when samples of such things as warnings, cautions, and notes are provided in a document. These must be marked up as samples, not as actual warnings, cautions, or notes.

2.2.5.4 Markup Inconsistencies

Various inconsistencies can exist between the output document and the marked-up document. It is important to verify that a formatted output sample has indeed been produced by the markup in the input sample.

- Anything that is missing in the output sample, but is clearly marked up in the input sample should be identified and reported on.
- Similarly, things that occur on the pages of the output sample must have been marked up in the input document, and any discrepancies of that type must be reported.
- An even more serious problem can occur; that is when something appears in the output document for which there is no markup in the input document, and no provision in the DTD. In such a case it is obvious that the output document could not have been produced from the input document.

2.3 Completing the I V & V Task

Simply put the Document Type Definition Independent Verification and Validation Final Report must document the results of each of the I V & V subtasks. Each subtask will require different forms of documentation. Some will be narrative descriptions of the process and its results. Some will be the output of an SGML parser used to process the machine-readable DTD and sample input documents. Some will be sample formatted documents.

Where shortcomings are found in the delivered DTD, the markup language it describes, the accompanying documentation or the sample documents provided, the Final Report must indicate this and recommend action to be taken. Such recommendations fall into four categories:

1. Recommendations for correcting or improving the markup language, the DTD and the documentation can be simple enumerated.
2. Where the markup language, documentation or testing have been found to be incomplete, and where the I V & V task found it necessary to complete these tasks to enable I V & V to progress, a recommendation can be made to incorporate parts of the DTD I V & V Final Report in the documentation for the DTD Design project.
3. Where it was found to be impractical or inappropriate for I V & V personnel to complete an incomplete DTD Design task, suggestions should be made as to the form of what needs to be added.
4. Where the design of the markup language has been found to be inappropriate for its function, a recommendation must be made to restart the DTD Design project. Such a recommendation should be

accompanied by suggestions as to how the new project should differ from the old.

Finally, the DTD I V & V Final Report must summarize its findings, particularly with regard to whether or not use of the DTD and the markup language it describes can commence, or whether further design and documentation is required.

3. SUMMARY

Document Type Definition Independent Verification and Validation is the final, and a crucial, step in the process of creating and testing an SGML Document Type Definition prepared in compliance with the CALS standard MIL-M-28001A. Following the principles and steps described in the preceding sections will result in a Document Type Definition Independent Verification and Validation Final Report. The Final Report can then be used to determine whether the markup language documented by a Document Type Definition Design Project can be put into production, or if not what further work is required to do so.