2000

Web-enabled COTS-supported alternative to Interactive Electronic Technical Manual Database (IETMDB) specification

Dina Mahfouz Ghobashy
West Virginia University

Web-enabled COTS-supported Alternative to Interactive Electronic Technical Manual Database (IETMDB) Specification

Dina M. Ghobashy

Thesis Submitted to
The College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements
for the degree of

Master of Science
in
Electrical and Computer Engineering

Hany H. Ammar, Ph.D., Chair
Bojan Cukic, Ph.D.
Cris Fuhrman, Ph.D.

Department of Computer Science and Electrical Engineering

Morgantown, West Virginia
2000

Keywords: IETM, Web application, Object Model, Component based development

ABSTRACT

Web-enabled COTS-supported Alternative to Interactive Electronic Technical Manual Database (IETMDB) Specification

Dina M. Ghobashy

An Interactive Electronic Technical Manual (IETM), is a digital package of information required for diagnosis and maintenance of complex weapon systems and, both, military and commercial equipment. The lack of interoperability within and among IETM systems has become a major challenge to the U.S. Department of Defense (DoD) IETM community. The initial phase of user-level interoperability support has been undertaken by the development of a Web-based Joint IETM Architecture (JIA). Within this architecture, there is a need to develop a standardized Web-enabled alternative to the IETM database specification. Object oriented modeling provides the conceptual foundation for assembling the Web-enabled alternative out of Web components using Web technology such as Java and XML.

This thesis introduces an original IETM Object Model. This is a UML-based model that captures the dynamic aspects of the specification. We define UML extensions to capture the semantics of the IETM database specification. We propose a COTS-supported IETM display system architecture based on the developed IETM Object Model. We also demonstrate the applicability and merit of the approach by implementing a prototype Web-based IETM display system.

ACKNOWLEDGMENTS

I wish to express my deep gratitude to my research advisor, Dr. Hany Ammar for helping me define my research goals and for providing valuable guidance during this research.

I would also like to thank all members of my committee, Dr. Bojan Cukic and Dr. Cris Fuhrman for their support and review and for their time in serving on my committee.

I would like also to express gratitude to the project team at ManTech Inc., especially to Mr. Don Reynolds for his directions, support and encouragement.

Many thanks to my patient and loving husband Dr. Sherif Yacoub who gave graciously of his time and knowledge in this work. He has been an unfailing source of support and encouragement to me during the research period.

I also thank my father for always motivating me to pursue higher education and to expand my scientific knowledge. I offer my mother, sister, brother and friends heartfelt thanks for their invaluable consideration and moral support.

I am also grateful to all my colleagues in the research lab. Thank you for your help and for creating a very positive atmosphere that makes it easier to withstand the difficulties that sometimes arise.

This work was funded by ManTech Advanced Systems International, Inc. research grant No. 6982/091/2001 to West Virginia University through the Software Engineering Research Center (SERC).

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

1.1 What's an IETM?

An Interactive Electronic Technical Manual (IETM) [1] is a Technical Manual, meaning its primary purpose is to support the diagnostics, maintenance, and repair of complex technical systems. The information package supporting these operations includes descriptive, procedural, troubleshooting and parts data.

An IETM is prepared in a digital form, on a suitable media with the aid of an automated authoring system and designed for electronic screen display to an end user. An IETM is interactive; it dynamically presents information to the user according to current conditions and user's inputs. IETM information sources define the conditions under which certain data should be presented.

In 1992, the DoD issued military specifications for service-wide use in the acquisition of IETMs. These specifications have been successful in their original objective of guiding the development of IETMs, which are now being acquired for many of the DoD's new major weapon systems. However, as individual systems have matured, issues in the area of interoperability between the differing IETM systems, as well as, incompatibility between these IETM systems and the growing inventory of legacy data Electronic Technical Manual (ETM) systems, have arisen.

1.2 IETM Interoperability Issues

The separation of the IETM database content from the presentation attributes has found favor in IETM system design and development. However, the standardization focus has been on the database structures and the *targeted "look and feel"* and not on the run-time and presentation version of the IETM. *In attempting to maintain a flexible approach to the definition of the IETM database, the specifications nearly guarantee non-portability across different vendor products* [25].

In the context of Interactive Electronic Technical Manuals (IETMs), "Interactive" means that a software application reacts to input from users as it is received. This reaction is often to tailor the content and presentation of subsequent display of information. The IETM specifications for document delivery (MIL-PRF-87269A) standardize structures for IETM content, and introduce logical conditions for information rendering. To create an IETM, authors and developers must consider a philosophy different from that used to create page-based documents; they must implement the behavior aspects into the document. The encoding of logic to control document behavior is one facet of electronic delivery that can cause incompatibility between IETMs, and between an IETM and another information system.

The DoD specifications provide a basis for standardization, however, they are, by design, general specifications intended to include a variety of implementations. IETM vendors leading efforts has emerged independently of each other and used their own differing interpretation of the specifications. Consequently,

- A variety of proprietary and COTS products are developed and used for IETM authoring and presentation systems
- Delivery formats are not interoperable between viewers, i.e., presentation systems could only display information specifically authored for that particular system
- Delivery formats and viewers are interdependent
- IETM documents are authored for a specific platform and application

Additionally, since a common standard for the structure of the delivered IETM is lacking, it has been exceptionally challenging to define the requirements for, or to make the initial design of, a common electronic infrastructure.

In this environment, the fact that differing IETMs cannot interoperate has become a major impediment. As the use of IETMs as information assets became more widespread and as they incorporated more functionality, it became important to achieve full interoperability between delivered IETMs.

The IETM community is faced with a requirement to enable interoperability within and among IETM systems. This requirement applies to the entire community constituencies:

- IETM authoring organization
- Government infrastructure activity managing the IETMs
- User operational sites
  - Site electronic library
  - End-user

IETM data is needed to sustain joint and multi-unit operations, and a uniform approach is required for building, acquiring, managing, and viewing this data. Interoperability among IETMs stems from the need to provide:

- IETM access and distribution via electronic technical libraries as sharable technical data
- Infrastructure for acquisition, management and deployment of existing and new IETMs
- Life-cycle support for numerous IETMs independent of specific authoring and presentation systems to manage and distribute IETM updates to multiple field sites
- Functional equivalence across platforms independent of vendor specific GUI appearance or functionality
- Interoperability between IETMs that refer to information in each other
- Integration, reusability and maintainability of IETM source data from different authoring environment

The initial phase of interoperability support has been undertaken by the development of a Joint IETM Architecture (JIA) [4]. The JIA effort is lead by the US Navy out of the Naval Surface Warfare Center Carderock Division with ManTech Advanced Systems as the principal technical support team. The JIA enables user-level interoperability and is based on

Commercial Off The Shelf (COTS) Web technologies. Other levels of interoperability are still unresolved and need further study.

Several approaches are available for standardizing many of the development issues such as server extensions and a variety of third-party middle-ware products; but they are all proprietary and not universally accepted. Commercial server extensions have several risks associated with introducing a high infrastructure cost and may create a situation involving proliferation of non-standard servers that is not in the interest of interoperability.

The technology and state-of-COTS are not sufficiently mature at this time to propose any one of them as a DoD standard so that all IETMs can operate on a single server. To achieve operational interoperability with a particular server, the various IETM providers must put their own physical server(s) plus the IETM source data on the shared user Intranet.

The specification for IETM deployment and manipulation on DoD Intranets is the missing component of the Web-enabled COTS-supported interoperability for IETMs.

1.3 Problem Statement

We identify the following problems:

1. *Problems dealing with the IETMDB specification:* The study of the IETM database specification revealed a set of major problems also encountered by other developmental efforts using the same specification [5]:

    - *Complexity*: The IETM SGML data model resembles a programming language more than a document or database definition.
    - *Ambiguity*: The SGML-centered specifications in MIL-PRF-87269A are precise in their prescriptions for the data element contents, but ambiguous with respect to the semantics for handling these elements in an IETM system.

- *Obscurity*: The specifications do not realize a technical architecture that is easily recognized and implemented using common practice techniques and client-server technologies.
- *Source data includes information on document behavior*: Use of variables, expressions, conditional branching constructs, and context filtering elements introduce dynamic behavior into the IETM data model.

2. *How can we provide data source interoperability within the JIA architecture?* The JIA architecture achieves interoperability at the user end, however, issues with the deployment and manipulation of IETM source data on DoD Intranets in an interoperable environment still remains.

1.4 Research Objectives

Our main research objective is to:

1. Develop a standardized Web-enabled COTS-supported alternative to the IETMDB specification to conceal its inherent difficulties: complexity, obscurity, ambiguity and dynamic behavior, which we described above.
2. Address interoperability issues in handling IETM source data within the JIA architecture. Our primary focus is to allow IETM presentation systems to display IETM source data from varying authoring systems, with a minimum amount of human intervention.

To solve the IETMDB specification problems, the alternative to the IETMDB specification should:

- Provide compatibility with current standards and specs without modification.
- Handle the logic and behavior in IETMs
- Add the missing layer: standardization of the methods for encoding document behavior, and connecting the content to presentation in an unambiguous way.

- Can be implemented by existing IETM programs with minimum impact

To achieve the required interoperability level, the alternative to the IETMDB specification should:

- Provide consistency across dissimilar authoring and presentation systems
- Address the issue of IETM deployment and manipulation within the JIA architecture.
- Improve compatibility and reuse of IETM presentation software.

1.5 Technical Approach

1. Develop an IETM Object Model. Our core approach to arriving at a COTS-supported, Web-enabled alternative to the IETM database specification began with a comprehensive modeling effort. The Unified Modeling Language (UML) [6] was employed in this work. Using UML, we described our IETM object model static design view as well as its interactions with an IETM display system. The model developed covers the entire IETM database specification hierarchy. The resulting UML model could be thought of as the IETM database specification presented in a visual thus simplified alternative.

2. Develop a component-based IETM system architecture. We propose an architecture built from components with well-defined interfaces and centered around the developed IETM Object Model.

3. Develop a prototype. A file-based system to demonstrate the approach applicability and compatibility with widely-used Commercial Off The Shelf (COTS) Web browser and Web server.

1.6 Thesis Structure

This thesis is organized as follows:

The following chapter provides the basic background related to our research. It describes the Interactive Electronic Technical Manuals (IETMs) in more detail. It also summarizes the object-oriented technology and discusses the latest modeling language in the object-oriented paradigm, the Unified Modeling Language (UML). We also summarize the Standard Generalized Markup Language (SGML) standard and the Web technologies used in the approach.

Chapter 3 discusses the previous efforts in the IETM domain to standardize the interchange of IETM documents.

We introduce our approach to solve the problems identified above in chapter 4.

The IETM Object Model developed using UML is described in chapter 5.

We dedicate chapter 6 to discuss the proposed IETM display system architecture.

In chapter 7 we discuss the case study and the design of the file-based demonstration system implemented, we also discuss issues involved in implementing the system using a database.

Finally, we conclude the thesis and enumerate our findings and resolved issues. we also identify areas of further development and research.

CHAPTER 2: BACKGROUND

2.1 Interactive Electronic Technical Manual (IETM)

2.1.1 Overview

An Interactive Electronic Technical Manual (IETM) [1], is a package of information and documentation required for diagnosis and maintenance of complex weapon systems and both military and commercial equipment. An IETM possesses the following characteristics:

- *Electronic:* To enhance comprehension, an IETM contains information that is designed and formatted specifically for screen presentation.
- *Interactive*: The IETM display system operates interactively as a result of user requests and the associated input information.
- *Navigable*: The information is hyper-linked; the end user can access the same data via a variety of paths. Information elements can reference other elements within the IETM as well as information sources outside the IETM.

IETMs are authored in a way that allows a user to retrieve and comprehend the required information faster and easier than is possible with a paper technical manual. Maintenance and troubleshooting procedures in an IETM represent a database of the knowledge of authors with experience in this area. The IETM display system can examine and decide which unit of information is presented next based on the values of conditions entered by the user and evaluated according to the operations entered by the authors. These intelligent features of IETM systems provide powerful interactive maintenance and troubleshooting procedures for users in the field.

2.1.2 IETM Specifications

To set the stage for standardization of IETMs, the DoD issued three military performance specifications [1, 2] that define the process and content for authoring and displaying IETMs. The roles of these specifications are:

- MIL-PRF-87268A defines how the IETM should look and behave to the end user.
- MIL-PRF-87269A SGML-based [3], which defines the IETM Data Base (IETMDB) forms, structure, and key controlling mechanisms.

The engineering and preparation of the content data is the most costly part of any IETM and as such it should be in a form that will outlast technology improvements in presentation systems.

2.1.3 DoD Classes of Electronic Technical Manuals

The various approaches to technical manuals automation have been placed into categories or classes as follows [26]:

Class 0: Non-Electronically-Indexed Page Images

Systems of digitized page images that are intended for electronic archival filing or print-on-demand. These allow pages to be viewed on an electronic display but have no detailed index for navigation through the document for purpose of on-line usage. This system is not an ETM.

Class 1: Electronically Indexed Page Images

Systems of digitized page images intended for full-page display and use allowing navigation by means of an automated intelligent index to the page images for user access. These systems can be used in a library or reference setting for reading and research use.

Class 2: Electronic Scrolling Documents

Systems for interactive display of ASCII encoded documents using an intelligent index and Hypertext tags inserted into a tagged document file. In general, the document is the result of a

simple conversion from a page-oriented document but with little reauthoring with the exception of adding hypertext tags. These allow a user to navigate through the document, but have very limited, if any, author inserted navigation aids or a content driven functions.

Class 3: Linearly Structured IETMs

Interactive display of technical information which is SGML tagged using MIL-PRF-87269A tags to the maximum extent possible and using a Hypertext presentation system for display in accordance with MIL-PRF-87268A. It is based on a linear SGML document file and not a hierarchically based database.

Navigation is based on author developed constructs employing prompted dialog boxes and content driven logical navigation functions.

Class 4: Hierarchically Structured IETMs

Interactive electronic display of technical information specifically authored into and maintained in a relational or object-oriented hierarchical database. These source data are subsequently extracted and formatted for interactive presentation in accordance with the DoD IETM specifications (MIL-PRF-87268A and MIL-PRF-87269A). This class provides same functionality as class 3.

Class 5: Integrated Database IETIS

Integrated Electronic Technical Information System (IETIS). Interactive presentation of class 4 IETMs integrated in with other processes including expert-system rules for the display of information and other user-applications such as diagnostics or computer-managed training.

Each of these classes has benefits over the current paper technical manual systems and the degree of benefit increases with each higher class.

IETMs span classes 3 through 5, they all show almost identical features at the user-presentation level. Systems of the Class 5 type exist but, in general, are proprietary in implementation and,

typically, involve state-of-the-art technology. They are designed to be a natural extension of an IETM class 4 database. In addition, class 4 data base systems have been designed to easily evolve into or usefully coexist with class 5 integrated data systems.

This work addresses issues related to IETM classes 3 and 4.

2.2 SGML (Standard Generalized Markup Language)

SGML (Standard Generalized Markup Language) is a standard for how to specify a document markup language or tag set. Such a specification is itself a document type definition (DTD). SGML is not in itself a document language, but a description of how to specify one. It is a metalanguage.

SGML is based on the idea that documents have structural and other semantic elements that can be described without reference to how such elements should be displayed. The actual display of such a document may vary, depending on the output medium and style preferences. Hypertext Markup Language (HTML) is an example of an SGML-based language.

SGML was created to represent information in a neutral, self-describing format. These properties of an SGML document make it ideal for ensuring that information content remains accessible and reusable during a document's useful life. Document maintenance can be aided by automated processes using SGML tags that describe its structure.

Interactive hypermedia documents stand to gain even greater benefit when dynamics of its presentation can also be embedded within it. As an application of SGML that was specifically designed for hypermedia applications, HyTime provides the model for embedding interactive (e.g., dynamic, conditional) content in a document.

SGML and HyTime form the basis for the IETMDB specification.

2.3 Object Oriented Technology

Object-oriented technology is a decomposition technique that came out in the eighties. It is one credible approach for large system development. Object-oriented models have been envisioned to encourage reuse of software components, reduce development risks, and yield systems that are more resilient to change. Object oriented analysis and design are techniques that matured to deploy the object technology and develop reliable, reusable and modular programs using object oriented programming languages such as C++ and Java [13].

2.3.1 Object Oriented Analysis and Design

Object Oriented Analysis (OOA) is the methodology that examines the requirements from the perspective of classes and objects, explicitly or inherently stated in the problem domain. Several object-oriented analysis methods have been proposed. Their main focus is to identify candidate classes and objects in the application under construction. Shlaer and Mellor [17] suggest a method for identifying objects and classes based on tangible things, roles, events, and interactions. Coad and Yourdon [18] suggest another set of sources for identifying objects that is based on structures, devices, locations, and organizational units. Wirfs-Brock *et. al.* [19] emphasize on identifying objects from the behavioral description of the problem where the object's responsibility is the service it provides to all of its contacts. Rubin and Goldberg [20] propose another approach to identify classes and objects from the system functions: ".. *the approach we use emphasizes first on understanding what takes place in the system. We next assign these behaviors to parts of the system and try to understand who initiates and who participates in these behavior*s". Jacobson [21] propose the use-case analysis method in which patterns of system usage are defined. Each system usage is then elaborated upon with one or more scenarios. Scenarios are powerful tool of object oriented analysis and hence they are adopted in the Unified Modeling Language (UML) models [6]. Generally, analysis addresses the problem of identifying the desired system behavior and the roles and responsibilities of the objects that carry out this behavior.

Object Oriented Design (OOD) is the design methodology that defines the process of deriving logical as well as physical models and static as well as dynamic models of the system under development. During design, the analysis classes are further detailed, objects' behavior is further elaborated, and more implementation details of objects and classes and their relationships are derived.

Several analysis and design views are required to capture the user's requirements. Each view represents a way to model specific aspect of the application. Booch [22] identifies several views and models that are important for object oriented development. These models include static, dynamic, logical, as well as physical models. Logical models describe classes and objects structure, while physical models describe the module and process architecture. The result is a set of analysis and design diagrams representing various views of the system. The most widely used diagrams are class diagram, object (collaboration and interaction) diagrams, state transition diagrams, deployment diagrams, and component diagrams.

Wirfs-Brock *et. al.* [19] define two techniques in object-oriented analysis and design: Responsibility-Driven Design (RDD)/ Class-Responsibility-Collaboration (CRC). In RDD, a model is developed from the requirement specification by the extraction of nouns and verbs. For each class, different responsibilities are defined which specify the roles of objects. In order to fulfill these responsibilities, classes need to collaborate with each other and thus CRCs are developed. The Object Oriented Software Engineering (OOSE) [21] method has its origin from the telecommunication field. The initial ideas of this development methodology are presented early by Ivar Jacobson in 1987. Lately, it is scaled-down to a reduced version called Objectory. The famous aspect of this approach is the reliance on use cases as glue for tying together all models.

The Object Management Technology (OMT) approach defines three sets of concepts that provide three different views of a system. The approach defines a method that leads to three models of the system corresponding to these views. The three models are the object models, functional models, and dynamic models.

Recently, several object-oriented approaches have evolved to suit particular needs in specific system domains. The Hierarchical Object Oriented Design (HOOD) method addresses a design step that identifies the child objects of a given parent object and their individual relationships to other existing objects. The main additional features of this process are the hierarchy and connectivity.

2.3.2 The Standard Object Modeling Language: Unified Modeling Language (UML)

UML Scope

Prior to UML, several modeling languages have been available, most of which share a set of commonly accepted concepts that are expressed differently. This lack of agreement discourages users from using OO technology. UML is the unified language for specifying, visualizing, and documenting the various artifacts of a system to be developed using object oriented methods. It is not only a descriptive visual language, but it has both syntax and semantics capabilities as well. UML fuses the concepts of Booch [22], OMT, Object Oriented Structure Analysis OOSA, and Object Oriented Software Engineering OOSE [21] to develop a single pervasive modeling language. UML also targets the modeling of concurrent and distributed systems. UML does not mandate a specific development process. It provides syntax and semantics for modeling OO analysis and design. UML authors promote a development process that is use-case driven, architecture centric and iterative and incremental. UML doesn't include programming language compilers, but it has support for automatic code generation (code frames around which development proceeds). It defines techniques and methods that can be used and it is up to the application analyst to utilize these techniques in his development process.

UML Models

No single model is sufficient to capture the application's static and dynamic properties. The choice of models and diagrams has a deep influence on how a problem is attacked and how the corresponding solution is proposed.

Therefore, every complex application has to be approached using a set of independent views. The models supported by UML can be summarized as follows:

- *Use Case diagrams.* Use cases are run-types of an application. A use case defines a situation in which the system is used, its inputs and possible outcomes. Use cases are further analyzed to produce possible scenarios. The collection of scenarios is one aspect of the requirement specifications of the application.
- *Class diagram.* Class diagrams describe one structure aspect of the application. It is a representation of possible classes and their relationships. Detailed class diagrams reveal details about the design, which include the attributes and their types, methods and their signatures.
- *Behavior diagrams.* These diagrams explain behavioral aspects of the applications.
  - *Interaction diagrams:* Interaction diagrams are either sequence or collaboration diagrams:
    - *Sequence diagrams.* Object scenario diagrams are means to record design decisions by examining the interaction between objects in the system in a timely ordered manners. Normally, there are one or more scenarios for each use-case.
    - *Collaboration diagrams.* Collaboration diagrams are another view of scenarios without the timely sequence ordering. Their objective is to show how objects interact and send messages to each other. Concurrent threads of execution can also be represented.

- *Statechart diagra*m: Statecharts are techniques to model the behavior of large complex systems. The behavior of objects in an object-oriented design can be studied using statecharts support in UML

• *Implementation diagrams.*

- *Component diagram.* A component diagram shows the application components and how they interact. It describes a static view of the system. *"Component diagrams are related to class diagrams in that a component typically maps to one or more classes, interfaces, or collaborations*"[6]

- *Deployment diagram.* The deployment diagram shows the system physical architecture in terms of nodes and their relationships. Deployment diagrams are related to component diagrams in that a node typically encompasses one or more components.

These diagrams provide multiple viewpoints of the system under development. The underlying model integrates these perspectives so that a self-consistent system can be analyzed and built. Models used for object-oriented analysis and design can be generally classified as static and dynamic models.

Static models describe the structure of the application, the high level functional and data organization, as well as the broad information content. Static views include class diagrams, package diagrams, component diagrams, and deployment diagrams. Dynamic views include use-cases, scenario and collaboration diagrams, and the behavioral aspects of individual object using statecharts. Dynamic models contain events, responses, messages, and invocations. Static models are the bases upon which application code is developed. So we start with several dynamic models, produce the static model and then continuously iterate on both models. This iterative process continues until complete application models are developed.

2.4 Internet and Web Technologies

2.4.1 Extensible Markup Language (XML)

The World Wide Web Consortium (W3C) has created an SGML Working Group to build a set of specifications to make it easy and straightforward to use the beneficial features of SGML on the Web. The goal of the W3C SGML activity is to enable the delivery of self-describing data structures of arbitrary depth and complexity to applications that require such structures. The first phase of this effort is the specification of a simplified subset of SGML specially designed for Web applications. This subset, called XML (Extensible Markup Language) [10], retains the key SGML advantages of extensibility, structure, and validation in a language that is designed to be vastly easier to learn, use, and implement than full SGML.

XML differs from HTML [11] in three major respects:

1. Information providers can define new tag and attribute names at will.
2. Document structures can be nested to any level of complexity.
3. Any XML document can contain an optional description of its grammar for use by applications that need to perform structural validation.

XML has been designed for maximum expressive power, maximum teachability, and maximum ease of implementation. The language is not backward-compatible with existing HTML documents, but documents conforming to the W3C HTML specification can easily be converted to XML, as can generic SGML documents and documents generated from databases.

2.4.2 Extensible Style Language (XSL)

XSL is a language for expressing stylesheets. Given a class of arbitrarily structured XML documents or data files, designers use an XSL stylesheet to express their intentions about how that structured content should be presented; that is, how the source content should be styled, laid out, and paginated onto some presentation medium, such as a window in a Web browser or a hand-held device, or a set of physical pages in a catalog, report, pamphlet, or book.

Unlike the case of HTML, element names in XML have no intrinsic presentation semantics. Absent a stylesheet, a processor could not possibly know how to render the content of an XML document other than as an undifferentiated string of characters. From the very beginning of the XML effort, it was recognized that in order to successfully send XML documents over the Web, it would be necessary to have a standard mechanism for describing how they were to be presented. That's why we need stylesheets. XSL provides a comprehensive model and a vocabulary for writing such stylesheets using XML syntax.

The Extensible Style Language (XSL) is the style language for XML. On August 18, 1998, the XSL Working Group (WC) released its first Working Draft.

An XSL stylesheet processor accepts a document or data in XML and an XSL stylesheet and produces the presentation of that XML source content that was intended by the designer of that stylesheet. There are two aspects of this presentation process: first, constructing a result tree from the XML source tree and second, interpreting the result tree to produce formatted results suitable for presentation on a display, on paper, in speech, or onto other media. The first aspect is called tree transformation and the second is called formatting. The process of formatting is performed by the formatter. This formatter may simply be a rendering engine inside a browser.

Tree transformation allows the structure of the result tree to be significantly different from the structure of the source tree. For example, one could add a table-of-contents as a filtered selection of an original source document, or one could rearrange source data into a sorted tabular presentation. In constructing the result tree, the tree transformation process also adds the information necessary to format that result tree.

Formatting is enabled by including formatting semantics in the result tree. Formatting semantics are expressed in terms of a catalog of classes of formatting objects. The nodes of the result tree are formatting objects. The classes of formatting objects denote typographic abstractions such as page, paragraph, table, and so forth. Finer control over the presentation of these abstractions is provided by a set of formatting properties, such as those controlling

indents, word- and letter-spacing, and widow, orphan, and hyphenation control. In XSL, the classes of formatting objects and formatting properties provide the vocabulary for expressing presentation intent.

2.4.3 Java

Java is a programming language expressly designed for use in the distributed environment of the Internet. It was designed to have the "look and feel" of the C++ language, but it is simpler to use than C++ and enforces a completely object-oriented view of programming. Java can be used to create complete applications that may run on a single computer or be distributed among servers and clients in a network.

The major characteristics of Java are:

- The programs you create are portable in a network. Your program is compiled into Java bytecode that can be run anywhere in a network on a server or client that has a Java virtual machine. The Java virtual machine interprets the bytecode into code that will run on the real computer hardware. This means that individual computer platform differences such as instruction lengths can be recognized and accommodated locally just as the program is being executed. Platform-specific versions of your program are no longer needed.
- The code is "robust," meaning that, unlike programs written in C++ and perhaps some other languages, the Java objects can contain no references to data external to themselves or other known objects. This ensures that an instruction can not contain the address of data storage in another application or in the operating system itself, either of which would cause the program and perhaps the operating system itself to terminate or "crash." The Java virtual machine makes a number of checks on each object to ensure integrity.
- Java is object-oriented, which means that, among other characteristics, similar objects can take advantage of being part of the same class and inherit common code. Objects are thought of as "nouns" that a user might relate to rather than the traditional procedural "verbs." A method can be thought of as one of the object's capabilities or behaviors.

- In addition to being executed at the client rather than the server, a Java applet has other characteristics designed to make it run fast.
- Relative to C++, Java is easier to learn.

Java was introduced by Sun Microsystems in 1995 and instantly created a new sense of the interactive possibilities of the Web. Both of the major Web browsers include a Java virtual machine. Almost all major operating system developers (IBM, Microsoft, and others) have added Java compilers as part of their product offerings.

The Java virtual machine includes an optional just-in-time (JIT) compiler that dynamically compiles bytecode into executable code as an alternative to interpreting one bytecode instruction at a time. In many cases, the dynamic JIT compilation is faster than the virtual machine interpretation.

Applets

Applets are small application modules built in Java that can be sent along with a Web page to a user. Applets make it possible for a Web page user to interact with the page. They can perform interactive animations, immediate calculations, or other simple tasks without having to send a user request back to the server.

JavaScript

JavaScript should not be confused with Java. JavaScript, which originated at Netscape, is interpreted at a higher level, is easier to learn than Java, but lacks some of the portability of Java and the speed of bytecode. Because Java applets will run on almost any operating system without requiring recompilation and because Java has no operating system-unique extensions or variations, Java is generally regarded as the most strategic language in which to develop applications for the Web. However, JavaScript can be useful for very small applications that run on the Web client or server.

Servlet

Servlet is a small program that runs on a server. The term was coined in the context of the Java applet, a small program that is sent as a separate file along with a Web (HTML) page. Java applets, usually intended for running on a client, can perform services such as performing a calculation for a user or positioning an image based on user interaction.

Some programs, often those that access databases based on user input and those that require large system resources need to be on the server. Typically, these have been implemented using a Common Gateway Interface (CGI) application. However, with a Java virtual machine running in the server, such programs can be implemented with the Java programming language. The advantage of a Java servlet on servers with lots of traffic is that they can execute more quickly than CGI applications. Rather than causing a separate program process to be created as in CGI programs, each user request is invoked as a thread in a single daemon process, meaning that the amount of system overhead for each request is slight.

The Java servlet support continues to become available on more and more servers, either through add-on modules or by providing direct support.

2.4.4 Document Object Model (DOM)

Overview

The Document Object Model (DOM) is an application programming interface (API) for HTML and XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated in object oriented programs. In the DOM specification, the term "document" is used in the broad sense. Increasingly, XML is being used as a way of representing many different kinds of information that may be stored in diverse systems, and much of this would traditionally be seen as data rather than as documents. Nevertheless, XML presents this data as documents, and the DOM may be used to manage this data.

The Document Object Model is a platform and language neutral interfaces that allows programs and scripts to dynamically access and update the content, structure and style of documents. The Document Object Model provides a standard set of objects for representing HTML and XML documents, a standard model of how these objects can be combined, and a standard interface for accessing and manipulating them. Vendors can support the DOM as an interface to their proprietary data structures and APIs, and content authors can write to the standard DOM interfaces rather than product-specific APIs, thus increasing interoperability on the Web.

The Document Object Model currently consists of two parts, DOM Core and DOM HTML. The DOM Core represents the functionality used for XML documents, and also serves as the basis for DOM HTML. The HTML section provides additional, higher-level interfaces that are used with the fundamental interfaces to provide a more convenient view of an HTML document.

With the Document Object Model, programmers can build documents, navigate their structure, and add, modify, or delete elements and content. Anything found in an HTML or XML document can be accessed, changed, deleted, or added using the Document Object Model, with a few exceptions.

The DOM originated as a specification to allow JavaScript scripts and Java programs to be portable among Web browsers. "Dynamic HTML" was the immediate ancestor of the Document Object Model, and it was originally thought of largely in terms of browsers. However, when the DOM Working Group was formed at W3C, it was also joined by vendors in other domains, including HTML or XML editors and document repositories. Several of these vendors had worked with SGML before XML was developed; therefore, the DOM has been influenced by SGML and the HyTime standard.

As a W3C specification, one important objective for the Document Object Model is to provide a standard programming interface that can be used in a wide variety of environments and applications. The DOM is designed for usage with any programming language. In order to

provide a precise, language-independent specification of the DOM interfaces, the specification is defined in OMG Interface Definition Language (IDL) [24], as defined in the CORBA 2.2 specification [12]. In addition, the DOM specification provides language bindings for Java and ECMAScript (an industry-standard scripting language based on JavaScript) [14]. The Document Object Model can be implemented in any computing environment. It is important to realize that these interfaces are an abstraction, much like "abstract base classes" in C++. They are a means of specifying a way to access and manipulate an application's internal representation of a document. Interfaces do not imply a particular concrete implementation. Each DOM application is free to maintain documents in any convenient representation, as long as the interfaces shown in the specification are supported. Some DOM implementations will be existing programs that use the DOM interfaces to access software written long before the DOM specification existed. Therefore, the DOM is designed to avoid implementation dependencies

The DOM Structure Model

The DOM model closely resembles the structure of the documents it models. In the DOM, documents have a logical structure, which is very much like a tree; to be more precise, it is like a "forest" or "grove", which can contain more than one tree. However, the DOM does not specify that documents must be implemented as a tree or a grove, nor does it specify how the relationships among objects be implemented. The DOM is a logical model that may be implemented in any convenient manner. In this specification, we use the term structure model to describe the tree-like representation of a document; we specifically avoid terms like "tree" or "grove" in order to avoid implying a particular implementation. The name "Document Object Model" was chosen because it is an "object model" in the traditional object oriented design sense. As an object model, the DOM identifies:

- the interfaces and objects used to represent and manipulate a document
- the semantics of these interfaces and objects including both behavior and attributes

- the relationships and collaborations among these interfaces and objects

The DOM presents documents as a hierarchy of "Node" objects that also implement other, more specialized interfaces. The Node interface is the primary datatype for the entire Document Object Model. It represents a single node in the document tree. All objects implementing the Node interface expose methods for dealing with children. Some types of nodes may have child nodes of various types, and others are leaf nodes that cannot have anything below them in the document structure. The node types are: `Document`, `DocumentFragment`, `DocumentType`, `EntityReference`, `Element`, `Attribute`, `ProcessingInstruction`, `Comment`, `Text`, `CDATASection`, `Entity`, `Notation`. The `Document` interface represents the entire HTML or XML document. Conceptually, it is the root of the document tree, and provides the primary access to the document's data.

2.4.5 Common Object Request Broker Architecture (CORBA)

In a distributed environment like the DoD Intranets, objects representing application components and/or IETM data need to work together across multivendor platforms and machine and network boundaries. Objects interfaces can be used to wrap existing distributed applications allowing them to be easily integrated in the JIA architecture. CORBA is an architecture and specification for creating, distributing, and managing distributed program objects in a network. It allows programs at different locations and developed by different vendors to communicate in a network through an "interface broker." CORBA was developed by a consortium of vendors through the Object Management Group (OMG), which currently includes over 500 member companies. Both ISO and X/Open have sanctioned CORBA as the standard architecture for distributed objects (which are also known as components). CORBA 3.0 is the latest release.

The essential concept in CORBA is the Object Request Broker (ORB). ORB support in a network of clients and servers on different computers means that a client program (which may itself be an object) can request services from a server program or object without having to understand where the server is in a distributed network or what the interface to the server

program looks like. To make requests or return replies between the ORBs, programs use the General Inter-ORB Protocol (GIOP) and, for the Internet, its Internet Inter-ORB Protocol (IIOP). IIOP maps GIOP requests and replies to the Internet's Transmission Control Protocol (TCP) layer in each computer.

A notable hold-out from CORBA is Microsoft, which has its own distributed object architecture, the Distributed Component Object Model (DCOM). However, CORBA and Microsoft have agreed on a gateway approach so that a client object developed with the Component Object Model will be able to communicate with a CORBA server (and vice versa).

CHAPTER 3: RELATED WORK

3.1 Metafile for Interactive Documents (MID)

The Metafile for Interactive Documents (MID) is a common interchange structure, based on the international standards for SGML and HyTime, that takes neutral data from varying authoring systems and structures it for display on dissimilar presentation systems, with minimal human intervention. MID draft specification dates November 1994. Alternative instructions for rendering of the data may be specified by MID authors, along with the conditions determining which instructions are executed at runtime in any presentation software.

It envisions that a MID Instance (the actual MID document) is a "hub" document, containing references to various internal or external source data components. The MID Instance is created by an interactive, automated process (i.e., a "MID Writer") and is interpreted for viewing by off-the-shelf software incorporating a "MID Reader."

MID does not contain format or style information. MID Reader software is responsible for determining the positioning and appearance of MID elements, based on the structural relationships between information containers. This is done to maintain separation of data content and logical flow from the specifics of how it is rendered.

The MID team clearly knew that their work was related to the ISO document presentation standard DSSSL even though they were not able to use it and referred to *adopting standard tools when available* in their MID guide.

3.2 Interchange Standard for Multimedia Interactive Documents (ISMID)

Because an increasing number of technical manuals are being developed and delivered on computers, it is important that these manuals be able to dynamically adjust what information is retrieved and presented, based on previous and current inputs. There is no standardized

language for describing this dynamic behavior. A standardized architecture for such languages is important to enable interactive documents to retain their device independence, and still be processed by a variety of commercial products.

More recently, there has been a move to bring IETM interoperability issues into the ISO domain, by means of a standard for representing the interactive behavior of an interactive document, for interchange purposes (ISMID).

Interchange Standard for Modifiable Interactive Documents (ISMID) is a new addition to the SGML family of standards, currently under development. It is a descendant of the MID specification discussed above, intended to meet the same objectives within a wider context.

ISMID, a proposed international standard, describes an architecture for creating transportable interactive documents. It replaces the former Standard Multimedia/ Hypermedia Scripting Language (SMSL).

The ISO project proposal for ISMID describes the new standard as follows:

"This International Standard, known as the Interchange Standard for Modifiable Interactive Documents or ISMID, facilitates the interchange of interactive multimedia documents among heterogeneous document development and delivery systems by providing the architecture from which common interchange languages can be created."

There are circumstances where it is desirable for users to have control of a document presentation; other times it is desirable for authors to have control, or for some mix of user and author control. ISMID encompasses these scenarios with one standard.

However, ISMID is not intended to act as an all-purpose scripting language, or even as a standard Application Programming Interface (API) to scripting languages. Either of these approaches would require constant revisions and additions to keep up with the development of scripting languages. Therefore, ISMID endeavors instead to represent the relationship types that exist between information content and control processes. If the specific relationships that

exist between a given document and a given script conform to the ISMID architecture, the application will be considered a conforming ISMID application.

3.3 Hypermedia Information Systems (HIS) Committee

With the shift from formally managed and maintained military specifications to industry and international standards has come uncertainty as to the availability and appropriate use of standards for development of hypermedia systems. Such standards are needed as guidance in their application to IETM developments. The Hypermedia Information Systems (HIS) Committee was formed to identify the elements of interactive hypermedia systems that are common with IETMs and propose an architecture for application of hypermedia standards that will ensure the long-term value of IETM data.

The consideration of hypermedia development standards and issues in the context of an IETM architecture illustrates where guidance on the application of hypermedia standards is needed and indicates that continued support of the standards process is required to facilitate the movement of information to hypermedia systems. The approach outlined in the HIS White Paper will lead to IETM programs that meet the military requirements for interoperability, portability, common "look and feel," and control of the development and acquisition process for hypermedia information.

3.4 Joint IETM Architecture (JIA)

In response to directives from the Office of the Secretary of Defense, all of the Military Services have ongoing efforts to convert paper-based technical documentation into digital format. They are replacing existing maintenance and logistic-support technical manuals with legacy-data-conversion products in the form of Interactive Electronic Technical Manuals (IETMs).

The primary goal for the JIA [4] is to establish a technical framework for acquisition and deployment of the whole spectrum of electronic technical manuals. Thus, when the sharable and interoperable technical information is distributed to the work location of an end-user, he

will be able to view and utilize that data through a common user interface, no matter what the authoring source or data format. In so doing, the DoD will be able to establish a unified approach to the acquisition, management, and use of existing IETMs and newly procured IETMs. To meet this goal, the overall approach will be based on the use of existing Commercial Off The Shelf (COTS) and Internet and World Wide Web technology. An overall end goal is to achieve end-user-level interoperability of the IETMs delivered to and used by the entire DoD Operational Community.

The overall concept of this effort capitalizes on the technology and resources that are emerging as a result of the growth of the Internet and the World Wide Web in the commercial marketplace. Such an approach is becoming the de facto standard for corporate information-distribution systems worldwide. Figure 3.1 illustrates the components of the Joint IETM Architecture (JIA)

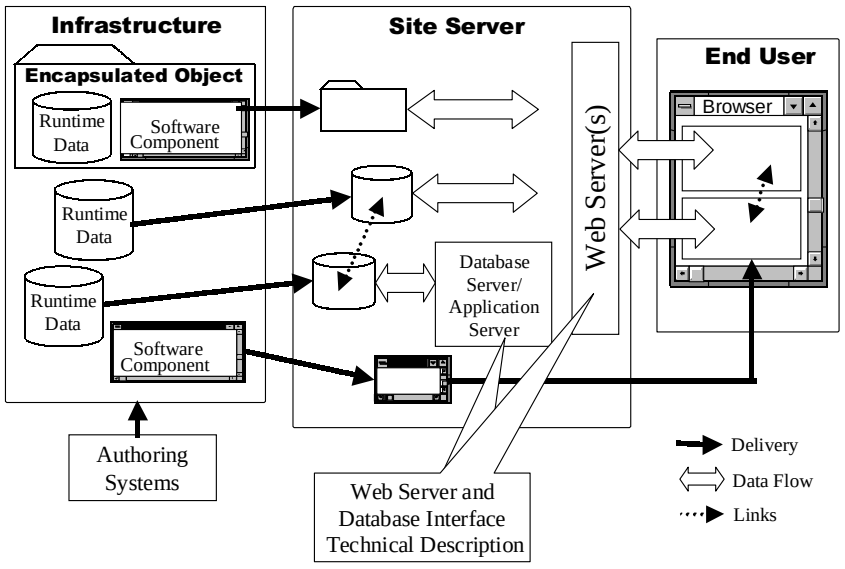

Figure 3.1 Joint IETM Architecture (JIA) (Adopted from [4])

A JIA-compliant IETM product will apply the vendor software and standards being developed for the World Wide Web and the Internet in a dedicated and private intranet environment.

Following the rapid change trend in Internet technology, the JIA has been designed to be extensible, flexible, and able to accommodate the predictable rapid growth in technology for all aspects of the Internet, the Web, and the emerging electronic documentation applications being developed to operate on the Web.

In addition to requiring the HTTP and TCP/IP networking protocols utilized by the Internet and utilized by virtually all commercial Web-based Intranet products and COTS systems, the JIA is be specified in the following areas:

1. *Object Encapsulation and Component Interface*: This specification is needed for definition of the delivery, transport, and structure of the integrated collection of software components and data contained in the IETM View Packages. This specification includes the interface between multiple components, when they exist, and the automated mechanisms for placing the IETM on the targeted Intranet. It also includes requirements for the capability to automatically install these components on a user device in a manner sufficiently simple so that no professional system administrator is needed. It is the primary specification to tell the IETM developers in what form they are to deliver the IETM View Package. View Packages can be delivered in a variety of formats that comply with the proposed architecture. However, the specification recommends that program managers specify non-proprietary data formats.

2. *Intranet Server and Database Interface:* For those IETMs that do require the services of an application server and/or a database server, the IETM supplier must provide the proper software extensions to the basic JIA Intranet Web server if they are not already in place. This specification outlines needed cooperation between the constructors of the end-user Intranet infrastructure and the IETM provider, and the interfaces and protocols involved.

3. *Common Browser:* The goal of this specification is to ensure that the commonly used browsers are functionally equivalent in any JIA Intranet. The specification allows for extensions to Web browsers via specified plug-ins and controls.

4. *Electronic Addressing and Library Model:* An addressing model specification based on Universal Resource Locators (URLs) for mapping logical to physical locations of information in a distributed, client-server environment. This specification holds the enterprise collection of IETM information together by means of digitally encoded and executable-link references.

The interface requirements recommended for the JIA are constructed so as to encourage innovative and effective solutions, especially in light of the constantly expanding technology base. The JIA design adheres to open standards and de facto Internet standards widely implemented by multiple vendors, with the clear intent to maximize the use of commercially available software products.

The JIA final recommendations on the use and encapsulation of server extensions is under technical investigations, since the technology and marketplace needs to mature before a full tradeoff and the development of specific recommendations can be accomplished.

CHAPTER 4: APPROACH

Extensible Markup Language (XML) being a simplified subset of SGML designed for Web applications may seem like a solution for the problem at hand. Yet XML is not enough, it provides the ability to capture and transmit IETM self-describing data structures on the Web, but it doesn't provide a standard mean to interpret behavior information required for document presentation. IETM functionality includes a number of features which together enable and control the user's traversal of the IETM structure. Since these functions are performed at presentation time, and are written into the IETM, they can be regarded as software programs (in a high-level language) which control the presentation of the technical information [7].

The following sections describe the approach that we take to achieve the required objectives.

4.1 Develop an IETM Object Model

Our core approach to arriving at a COTS-supported, Web-enabled alternative to the IETM database specification began with a comprehensive modeling effort.

4.1.1 Why Modeling?

Modeling is a proven and well-accepted software engineering technique. Good modeling is essential to assure architectural soundness and to ease communication among our project team. We need to build a model of the IETM complex system because we cannot comprehend it in its entirety and as the complexity of systems increases, modeling techniques become more effective. Moreover, the resultant model enables us to explore multiple solutions effectively.

We choose to view the IETM database specification in an object-oriented fashion for the following main reasons:

1. SGML information structures look quite object-like

2. The SGML specification represents the IETM documents by an abstract data model that is centered around the data. An object-oriented model encompasses not only the structure of a document, but also the behavior of a document and the objects of which it is composed
3. In the Web environment driven by object technology, to achieve open exchange of information, there is a need to provide a standardized way for object-based software to access and manipulate a document
4. Object oriented modeling provides the conceptual foundation for assembling our system out of Web components using Web technology such as XML and Java Beans.

A rigorous modeling language standard is essential for the success of an application development methodology. A modeling language should include definitions of the possible model elements and notation that visually render the model elements.

4.1.2 Why UML?

The SGML specification is purely textual which makes it hard to grasp, understand, and translate to formats that web browsers understand. To capture the objects specified in the specifications and their relationships in a form easier to understand and operate on, we need a modeling language that has powerful notational capabilities and at the same time formal enough to model the specifications (has an underlying semantics) but not too mathematical that we make things worse.

UML is the standard visual modeling language for software-intensive systems [6]. It is the latest modeling language in object-oriented paradigm that is the result of integrating several other modeling efforts (OMT, Booch, etc.). UML is standardized, many companies have invested in standardizing UML to improve the way object-oriented modeling, design, and implementation is done.

This thesis will adopt UML modeling techniques whenever applicable. The analysis methods and models are used to develop our new IETM Object Model and the proposed architecture. The reasons for adopting the UML are:

- It provides an expressive visual modeling language to develop and exchange meaningful models.
- It facilities the extensibility and specialization mechanisms to extend the core concepts.
- It is independent of programming languages.
- It provides a formal basis for understanding the modeling language (syntax and semantics capabilities).
- It is the integration of lots of efforts over the years and the fusion of many models developed through out the research in the latest decades

Another advantage of using UML is that it has extension mechanism that we can use to extend UML metamodel whenever UML constructs are not totally adequate to use. We used these extension mechanisms to give business meanings to UML artifacts.

4.2 Design a Component-based IETM Display System Architecture

Software component development provides for reusability, modularity, avoidance of duplication, reliability, and flexibility for expansion. These are essential issues to achieve interoperability in a complex environment.

The benefits of a component-based architecture include:

- Flexible upgrade of existing applications
- Application customization
- Component libraries for fast development
- Distributed applications

We use the IETM Object Model, that we develop in the first step of our approach, as the basic foundation for the design of a component-based architecture. The architecture is Web-enabled and supports the widely available COTS components. We also identify alternative Web technologies and protocols for implementation are identified as applicable in the architecture.

4.3 Develop a Prototype System

"Seeing is believing".

We build a proof-of-concept system to demonstrate the applicability of the proposed IETM Object Model and the efficacy of the proposed IETM display system architecture in supporting IETMs from various data sources. The demonstration system is file-based and covers the IETMDB procedural information. We also address issues related to database driven implementations.

CHAPTER 5: IETM OBJECT MODEL (IETMOM)

5.1 IETMDB Specification Overview

The IETMDB specification employs a two layered approach to define technical information in SGML:

1. The first layer is called the "Generic Layer". It defines general elements, which are common across all technical information applications.
2. The top layer is the "Content Specific Layer" which employs the generic layer to define elements for application specific information. The specification provides a default Content Specific Layer called Organizational Level.

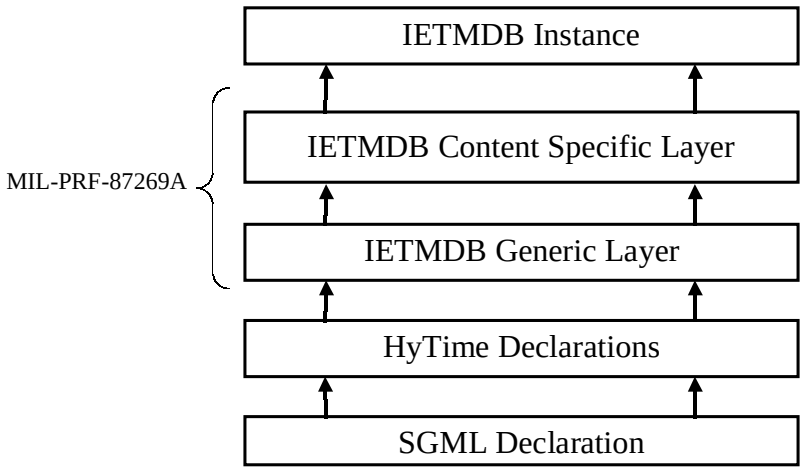

Figure 5.1 The IETMDB Specification Model

Figure 5.1 describes the layers that compose the model of the IETMDB. Each layer provides definitional requirements for the layer above it.

The SGML declaration specifies SGML features and syntax support for the MIL-PRF-87269A specification.

HyTime [8] declarations are instructions that declare which modules of the HyTime standard are to be supported by the MIL-PRF-87269A specification.

The IETMDB instance is the data file, containing information marked up in accordance with the requirements of the specified content specific layer

5.1.1 IETMDB Generic Layer

The generic layer can be best viewed as a layer of resource objects that can be used by the content specific layer and any IETM application.

The IETMDB generic layer consists of:

- Architectural Forms
- Primitive Data Elements
- User Interaction Elements called Dialog Elements
- Context Filtering Elements
- Link Elements
- SGML Parameter Entities
- SGML Notation Definitions

Architectural Forms

Generic layer architectural forms are templates, which are used to define content specific elements. They define a list of attributes and a set of semantic rules that govern the activities of the template and all conforming elements.

Architectural forms logically describe building blocks without constraining the application-specific information. They enable an IETMDB designer to create a content specific layer SGML Document Type Definition (DTD) that meets the needs of an application.

The generic layer architectural forms are five: Node, Node Alternatives, Node Sequence, If Node, and Loop Node. Each template has two components:

1. A set of semantic rules that govern the template's activities
2. A list of attributes.

The following is a description of the five templates and their semantics.

Node template

A node is an independent unit of information. All elements conforming to the Node template provide the capability for creating composite structures. Composite structures contain `primitives`, `links`, `preconditions` and `postconditions`. When a composite structure contains other composite structures within its content model, this implies hierarchy. Elements employing the node template must have a set of required attributes. A cross-reference attribute can be used to reduce the number of redundant elements by referencing common elements.

The following semantic rules apply to any element employing the Node template:

1. The element may contain a list of `preconditions` that identify the element's applicability. The list of `preconditions` will be evaluated at presentation time, and if all `preconditions` evaluate to true, that node will be presented.
2. The element may contain relational links to other data items.
3. The element may contain sub-components that employ one of the following templates: Node, Node Alternatives, or Node Sequence templates.
4. The element may contain a list of `postconditions`, which record presentation events. The `postconditions` will be evaluated after the node and all its sub-components have

been presented. The `postcondition` values will then be assigned to their specified `properties`.

Node Alternatives template

All elements conforming to the Node Alternatives template contain a list of mutually exclusive nodes, only one of which will be used at the time of presentation.

The following semantic rules apply to any content specific element employing the Node Alternatives template:

1. The element must contain components that employ the Node template.
2. The components must be of the same element type and at the same level in the hierarchy.
3. At presentation time, the `precondition` for each Node alternative will be evaluated. The Node whose `precondition` evaluates to "true" will be selected for presentation.
4. These components must contain mutually exclusive `preconditions`. In any specific situation, at most one node would have a `precondition`, which evaluates to true.
5. There need not be an applicable component for every possible situation.

Node Sequence template

All elements conforming to the Node Sequence template group elements together as well as providing an order or presentation sequence to the elements. The elements conforming to the Node Sequence also allow an author to define branching logic within the technical information.

The following semantic rules apply to any content specific element employing the Node Sequence template:

1. Any content specific element employing Node Sequence must contain components that employ the `Node`, `Node Alternatives`, `If Node`, or `Loop Node` templates.

2. The components of a Node Sequence are always traversed in the order they appear. This traversal includes the branching and iteration implicit in any `If Nodes or Loop Nodes` in the sequence logic.

If Node template

Elements conforming to the `If Node` template provide a method for conditional branching. These elements use the same logic as the “if-then-else” statement in a programming language. The "if" part is the `expression` in the content model. The "then" part is the first Node Sequence, and is selected when the `expression` evaluates to true. The "else" part is the second Node Sequence, which is optional, and is selected when the `expression` evaluates to not true.

The following semantic rules apply to any content specific element employing the If Node template:

1. The `expression` will be evaluated at presentation time
2. If the `expression` evaluates to "true" the first Node Sequence will be traversed
3. If the `expression` evaluates to anything but "true", and the second Node Sequence is present, the second Node Sequence is traversed.
4. If the `expression` evaluates to anything but "true", and the second Node Sequence is not present, the next element in the sequence will be presented.

Loop Node template

The Loop Node template provides the equivalent of a loop in a programming language. This element provides the capability to create either a "for" loop or a "while" loop within the data. The `expressions` and `assertions` contained in a Loop Node template provide the testing criteria for the loop. The template also contains a Node Sequence that holds the elements to be repeated within the loop.

The following semantic rules apply to any content specific element employing the Loop Node template:

1. If the first `assertion` exists, at the beginning of the loop, the `assertion` is evaluated and the value is assigned to the specified property.
2. The `expression` is evaluated and if the `expression` evaluates to anything but "true", the loop is terminated.
3. If the `expression` evaluates to true, the Node Sequence is traversed.
4. If the second `assertion` exists, at the end of each iteration, the second `assertion` is evaluated and the value is assigned to the specified property.
5. Steps 2-4 are continued until the loop terminates.

Primitive Data Elements

The generic layer defines a set of `primitive` elements: `text`, `table`, `graphic`, `audio`, `video`, `process`, `dialog`, `expression` and `assertion`. Those elements are available to any content specific layer DTD. The content specific elements using the primitive elements are restricted to the structure of the primitive elements as defined within the generic layer.

User Interaction Elements (Dialogs)

Dialogs are the basic elements, which provide the capability for user interaction with the technical manual information. During a presentation these elements are used to prompt the user to input a response (`fillin`), select a `choice` from a set of alternatives (`menu`), or to select items from within a text, table or graphic (`selection`).

Context Filtering Elements

Context dependent filtering is accomplished through author-defined `preconditions` and `postconditions`. A `precondition` contains an `expression`, which contains all the

information necessary to identify what conditions must be present to display the associated technical information. `Postconditions` assert the value of an `expression` to a `property`. Once these `properties` values are asserted, they become accessible to the presentation software for later processing to determine the user's situation.

Link Elements

Elements can have relationships to other elements in the technical manual, when applicable. These relationships are represented through two or more link ends. The `link` element provides the capability to create a relationship between several elements. Links can reference elements within the IETMDB as well as information sources outside the IETMDB.

SGML Parameter Entities

Entity declarations provide a mechanism for substituting elements in the Generic Layer, and within any Content Specific DTD. It defines a named set of element types, attribute specifications or even other entity references. This name is then used wherever the set can be substituted as content.

SGML Notation Definitions

SGML notation definitions enable an application to use information encoded in a notation that is not SGML. In the generic layer of the IETMDB, this feature is used to specify standard notations for graphics formats used to encode a particular graphic primitive such as Computer Graphics Metafile (`CGM`) format.

5.1.2 IETMDB Content Specific Layer (Organizational Level maintenance)

The Organizational Level layer is included in the specification as the default content specific layer. It identifies all the content specific elements required for the display of organizational level technical information to a technician. This layer breaks down organizational level data into a hierarchy based upon the system/subsystem structure of the weapon system.

System hierarchy

The vehicle, weapon system, or other equipment that is being maintained and operated is composed of several layers of subsystems, components, and parts. This hierarchical representation of the equipment being maintained and operated is described by a system element that is used recursively, and which decomposes the equipment into only those components that are being maintained or operated. Each component of this hierarchy has associated with it one or more of the following four categories of information:

- Descriptive information
- Procedural information
- Troubleshooting information
- Parts information

Descriptive information

Descriptive information provides information on system (subsystem, component, part) physical arrangement, functional behavior, theory of operation, and other aspects. Descriptive information contains a hierarchy of narrative paragraphs. Paragraphs, in turn, may refer to primitive elements defined by the generic layer.

Parts information

Two types of parts information are defined:

1. Maintainer/operator information
2. Supply information.

Elements containing either type refer explicitly to corresponding elements of the other type.

Parts information for the maintainer or operator

Parts information provided for a system maintainer and/or operator include such items as units per assembly, usable-on code, Mean Time Between Failures (MTBF), and reference designator.

Parts information provided for parts supply

Parts information provided for the parts supply process constitute unambiguous identification of a part so that it can be reordered. It consists of such items as the part number; Commercial And Government Entity (CAGE) code; Source, Maintenance, and Recoverability (SMR) code; Hardness Critical Item (HCI) identification; and National Stock Number (NSN).

Procedural information

Procedural information is composed primarily of `task` statements. Each `task` element is associated with attributes which provide related information such as: estimated completion time; maintenance level(s) where the task is to be performed; required conditions which must be met before performing the task; and the number of people required to perform the task. A procedural element may be linked to other elements, which define the support equipment and consumables that task requires, through the establishment of appropriate relationships.

Fault or Troubleshooting information

Troubleshooting information contain data necessary to isolate faults found in a system. Troubleshooting information contain `fault` elements, `fault state` elements, `test` elements, `outcome` elements, and `rectification` elements.

Fault elements

Fault elements identify potential faults, which might occur in the system.

Fault state elements

Fault state elements present a list of faults implicated as the result of a test that has been performed. Each suspected `fault` in the list is weighted, based on the probability that it is the cause of the observed malfunction. The fault state element may also present a list of possible faults that have been eliminated from consideration as the result of tests performed.

Test elements

Test elements contain a link to the procedural instructions a technician must follow to carry out a required task at a particular juncture in the troubleshooting procedure. Test elements provide all possible test outcomes.

Outcome elements

`Outcome` elements contain definitions of new fault states associated with the results of a particular test. `Outcome` elements also contain a description of the state of the item being maintained. An `outcome` is based on one or more expressions (i.e., system states that must be established for the specific outcome to apply). The final outcome element of a fault isolation procedure is associated with an identified `fault`. The identified fault is, in turn, associated with the initial element of the appropriate corrective maintenance action.

Rectification elements

Rectification (i.e., corrective maintenance actions) elements contain references to procedural rectification tasks, checkout tests used to report the success of completed rectification tasks and a list of all faults that the rectification repairs.

5.2 IETMDB Specification in UML

5.2.1 General Mapping Strategy

The following are general mapping rules used to create the IETM object-oriented model from the SGML specification:

- An element is mapped to a class
- All element attributes become class attributes in the element's class.
- The contents of an element become class attributes in the form of object references. Aggregation relationships and multiplicity are used to model this "has-a" relationship between the element's class and its contents classes.
- A group of IETM elements is modeled as a package.
- Parameter entities are modeled as abstract base classes as encountered in their respective package. The elements referenced in the entity are represented by classes derived from the entity's base class.

5.2.2 IETMDB Generic Layer in UML

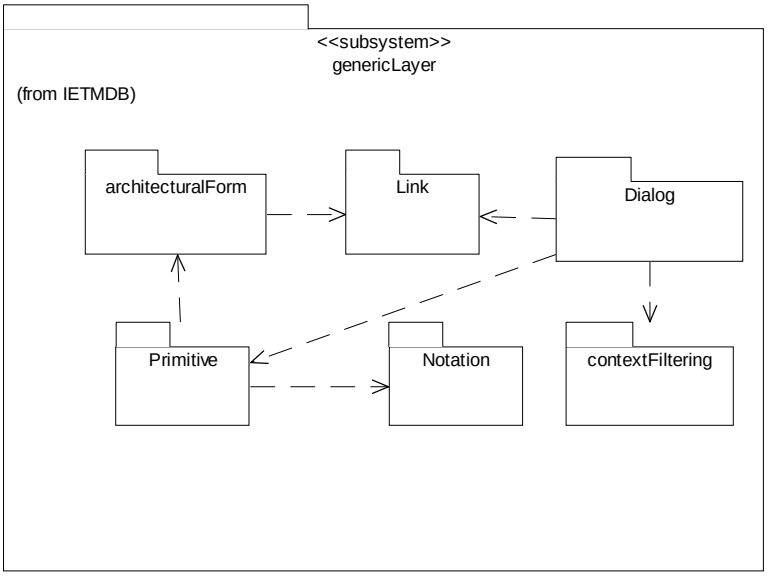

Figure 5.1 Generic Layer in UML

In UML, the IETMDB is modeled as a system package containing two subsystem packages: the generic layer and the content specific layer packages.

The generic layer is modeled as a subsystem package that contains a set of packages each representing one of the generic layer groups of elements listed above as shown in Figure 5.2. The figure also shows the dependencies that exist between those packages.

Architectural Forms Package

IETMDB architectural forms might map to classes in the logical view of the object model. Yet, simply mapping architectural forms to UML classes doesn't clearly distinguish them as building blocks and as a mechanism for producing elements. Moreover, architectural forms are abstractions that often appear in the IETM domain, and defining them as classes in the model would produce complex class diagrams that would be difficult to understand.

To capture the relevant semantics of a particular domain or architecture, UML defines a formal extension mechanism [6] to extend the modeling vocabulary. Stereotypes, tagged values, and constraints are the UML extension mechanisms that can be applied to add new building blocks, create new properties, and specify new semantics. A stereotype is a new class of modeling element that is introduced at modeling time. It represents a subclass of an existing modeling element but with a different intent and usage. Tagged values are key value pairs that can be associated with a modeling element to allow storage of a piece of information about the element. Constraints extend the semantics of a UML element, to allow addition of new rules or modification of existing ones. They can be expressed as free form text or with the more formal Object Constraint Language (OCL). We used UML's extension mechanism to define stereotypes, tagged values, constraints and icons for each architectural form. Thus, the five node templates were mapped to stereotypes in the IETM object model.

Architectural forms establish a default behavior for their conforming elements. MIL-PRF-87269A doesn't provide a clear or a full description of these behaviors or functions and of the way, they are bound to the instance data. Therefore, we based the definition of each template's set of methods on its interpretation and the expected display system behavior.

For example, method `evalPreconditions` in class `node` evaluates all preconditions and return their logical "and" result. In addition, method `XMLDisplay` enables contents of `node` conforming classes to be displayed in XML format. This method is overwritten by each node conforming class to provide the appropriate display for its contents. The model can be enhanced further to include operations for contents display in other formats e.g. Wireless Markup Language (WML) for small handheld devices in the field.

Method `selectFlow` in class `if-node` evaluates the condition associated and returns either the `then sequence` or the `else sequence` accordingly.

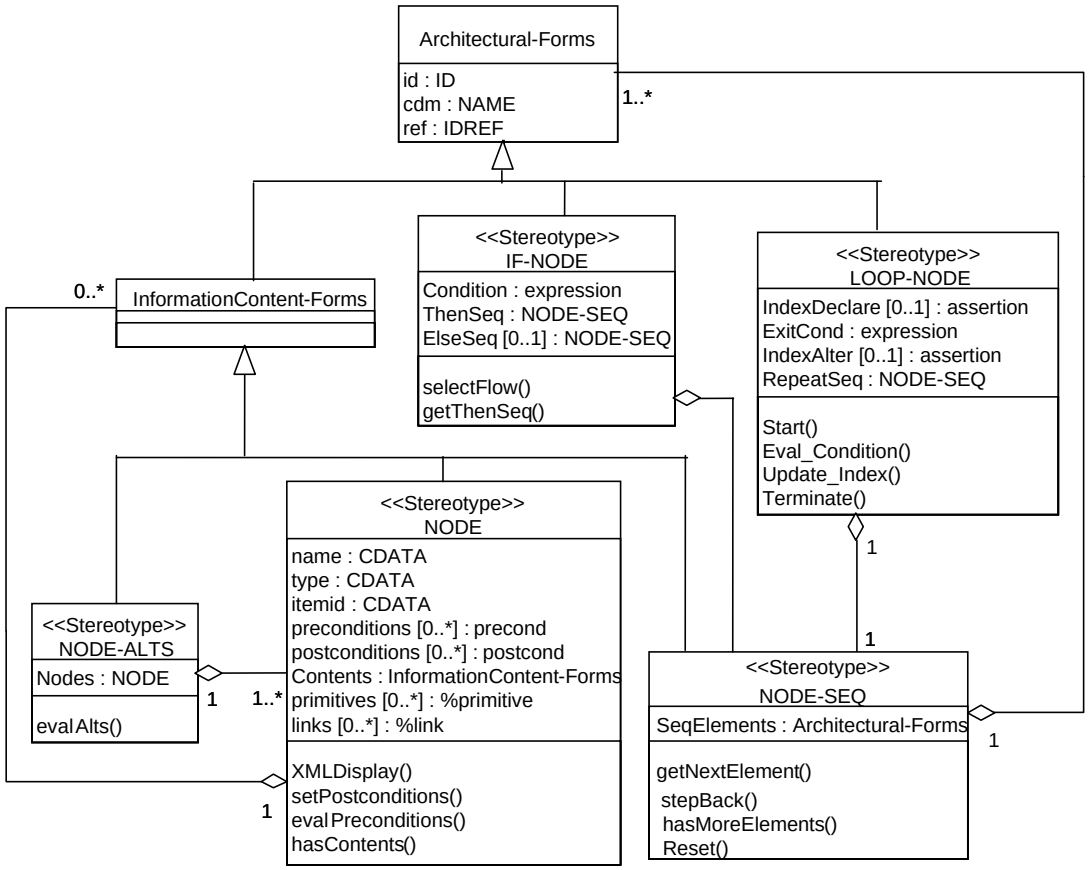

Figure 5.1 Architectural Forms in UML

Methods `getNextElement`, `hasMoreElements`, `stepBack` and `Reset` in class `node-seq` provide means to traverse a node sequence compliant element

`Loop-node` class contains the operations: `Start`, `evalCondition`, `updateIndex` and `Terminate` to allow for loop processing.

Likewise, class `node-alts` provides the method `evalAlts` which evaluates all nodes `preconditions` and returns the one whose `preconditions` evaluate to true. Architectural forms class diagram is illustrated in Figure 5.3.

Primitive Package

The primitive package shown in Figure 5.4 contains classes representing:

1. Elements conforming to the Node template: `text`, `table`, `graphic`, `grphprim` (graphic primitive), `dialog`, `audio`, `video` and `process`
2. Their counterpart: `text-alts`, `table-alts`, `graphic-alts`, `grphprim-alts`, `dialog-alts`, `audio-alts`, `video-alts` and `process-alts` conforming to the Node Alternative template.

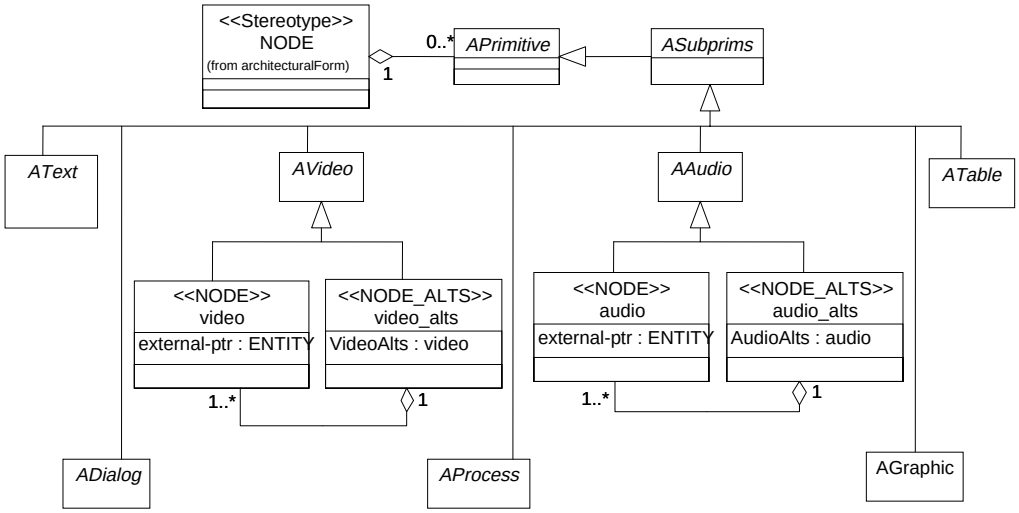

Figure 5.1 Primitive Elements in UML

The specification defines an entity declaration to reference each element and its alternative counterpart; e.g.; `AText` entity refers to both `text` and `text-alts` elements. These entities are modeled in UML as abstract classes from which these elements classes are derived. In addition, all these entities are referenced by another entity called `APrimitive`. `APrimitive` entity is modeled as an abstract base class at the top of the hierarchy from which all other entities abstract classes are derived.

The package contains other classes representing elements defined in the specification as follows:

`Rowhddef` and `colhddef` which define a table row header and a table column header respectively

- `Entry` which defines an entry for a cell in a table
- `Parameter` to be used by `process` elements.

Two more classes stereotyped `enumeration` meaning a list of possible data values were added:

- Class `graphic-code` provides a list of valid graphic formats for the attribute `code` of class `grphprim`
- Class `param-passing` lists the possible methods of parameter passing between the technical information and an external software process. This class is used by the attribute `mode` of class `parameter`.

Dialog Package

`Dialog` class from the primitive package contains a reference to any combination of one or more `menu`, `fillin` or `selection` elements. The later three elements represent the type of user interface (`UI`) supported by the IETMDB specification. In our model, we defined an abstract class `UI` from which `menu`, `fillin` and `selection` classes are derived.

Fillin Class

A `fillin` element according to the specification defines how a fill-in shall be constructed. It contains a `prompt`, a `property`, an optional default value and a `generic-range` element that may be used to provide a range for the value(s) of the fill-in.

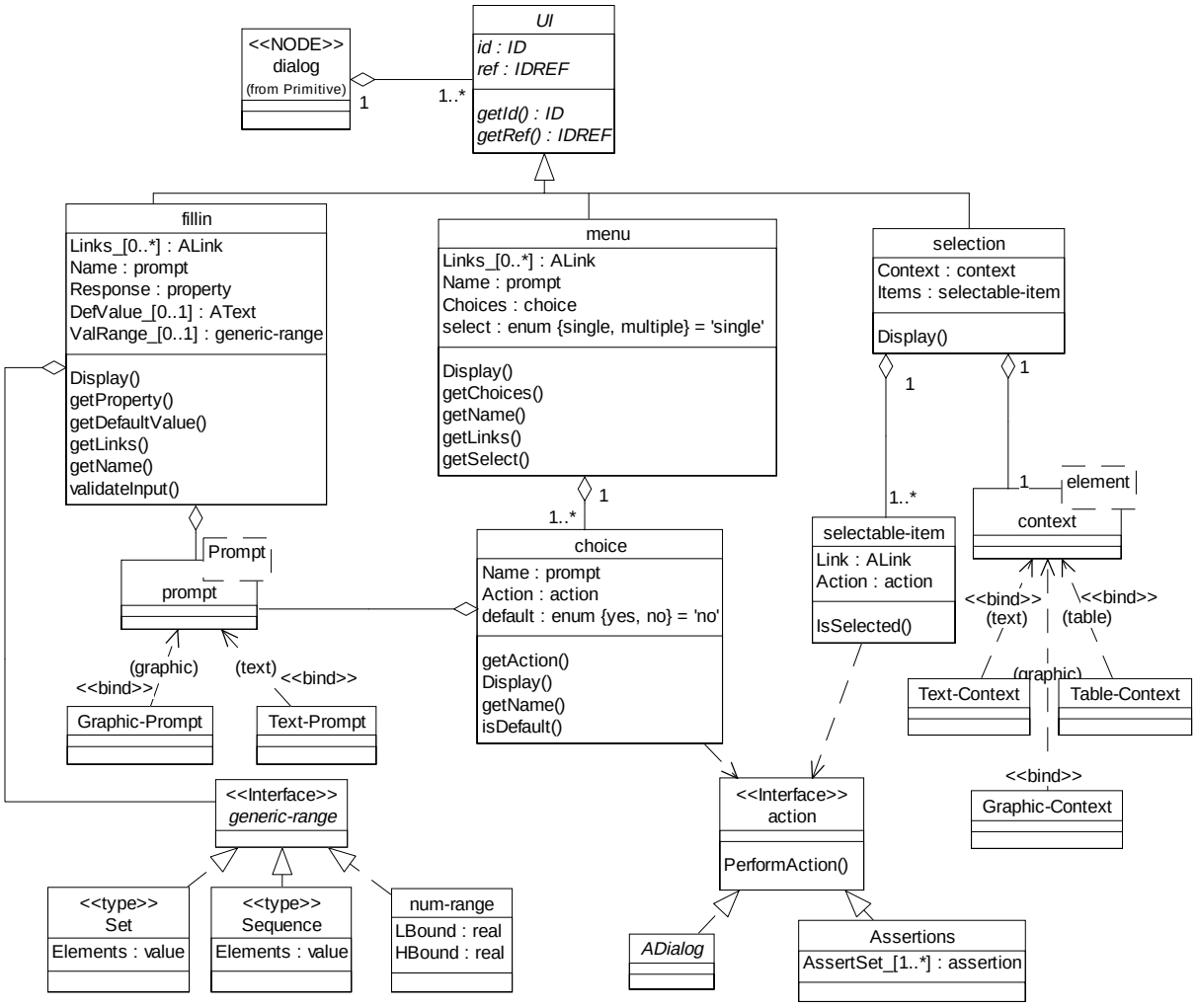

Figure 5.1 Dialog Elements in UML

The `generic-range` element is modeled as an interface. It declares the operation `isOutofRange` which returns true if the value entered by the user is out of the specified range and false otherwise. This interface is implemented by the classes `Set`, `Sequence` and `num-range` as illustrated in Figure 5.5.

Class `fillin` provides a method called `validateInput` which validates the user input by calling the method `isOutofRange` on its `generic-range` object.

Choice Class

This element defines the choices for a menu. A `choice` contains an `assertion` element or `dialog` element. The assertion or dialog identifies the action to be taken if the user selects that choice. The presentation system will either assert one or more conditions or branch to another dialog.

This action is modeled as a service `performAction` provided by the interface `action`. Both `assertions` and `dialog` implement this interface.

Context Filtering Package

Context dependent filtering provides the capability to present the user with only the information that applies to his specific situation. The `precondition` and `postcondition` elements provide the mechanism for context dependent filtering.

Precondition Class

It enables the selection of the appropriate information for presentation. Class `precondition` contains an `expression` element, which represents the required condition. The class provides the method `Evaluate` to achieve its purpose. Method `Evaluate` in turn calls method `Evaluate` on the `expression` object referenced to by the `precondition` and returns true if the `expression` evaluates to true and false otherwise.

Postcondition Class

It enables the recording of presentation events for later filtering. Class `postcondition` provides the method `Evaluate` to compute an `expression` and assert its value to a `property`. Method `Evaluate` invokes `Evaluate` again on the `assertion` object referenced by the `postcondition` and returns the computed value as an object of type `value`.

Assertion Class

An `assertion` provides a mechanism for assigning a value to a variable. It contains:

- An `expression` which evaluates to a value.
- A `property` which represents a variable

Method `Evaluate` of class assertion invokes `Evaluate` again on the expression object and returns the computed `value`. It also assigns the new `value` to the `property` object by invoking its `setValue` operation.

Expression Class

The specification defines four types of sub-expressions:

- A binary operation between two `expressions` e.g. append, plus, minus
- A unary operation upon an `expression` e.g. not, index, size
- A `property`, where the value of the `expression` is obtained by looking up the current value of the `property`.
- A `value`, where that `value` is returned as the result.

The specification defines the allowed value types, the valid operators and the semantics of each operation.

Unary operations, binary operations and value are defined as entities referencing the set of valid operations and value types respectively within the specification.

In the UML model, `expression` is an abstract class with two abstract methods:

- `getValue`: Returns the `expression` result.
- `getProperties`. Returns the list of all properties making up the `expression`, allowing the presentation system to look up their values before evaluating the expression.

The four types of sub-expressions: `binary operation`, `unary-operation`, property and `value` are derived classes from the abstract class `expression`. Each derived class provides his own implementation for each of the above operations.

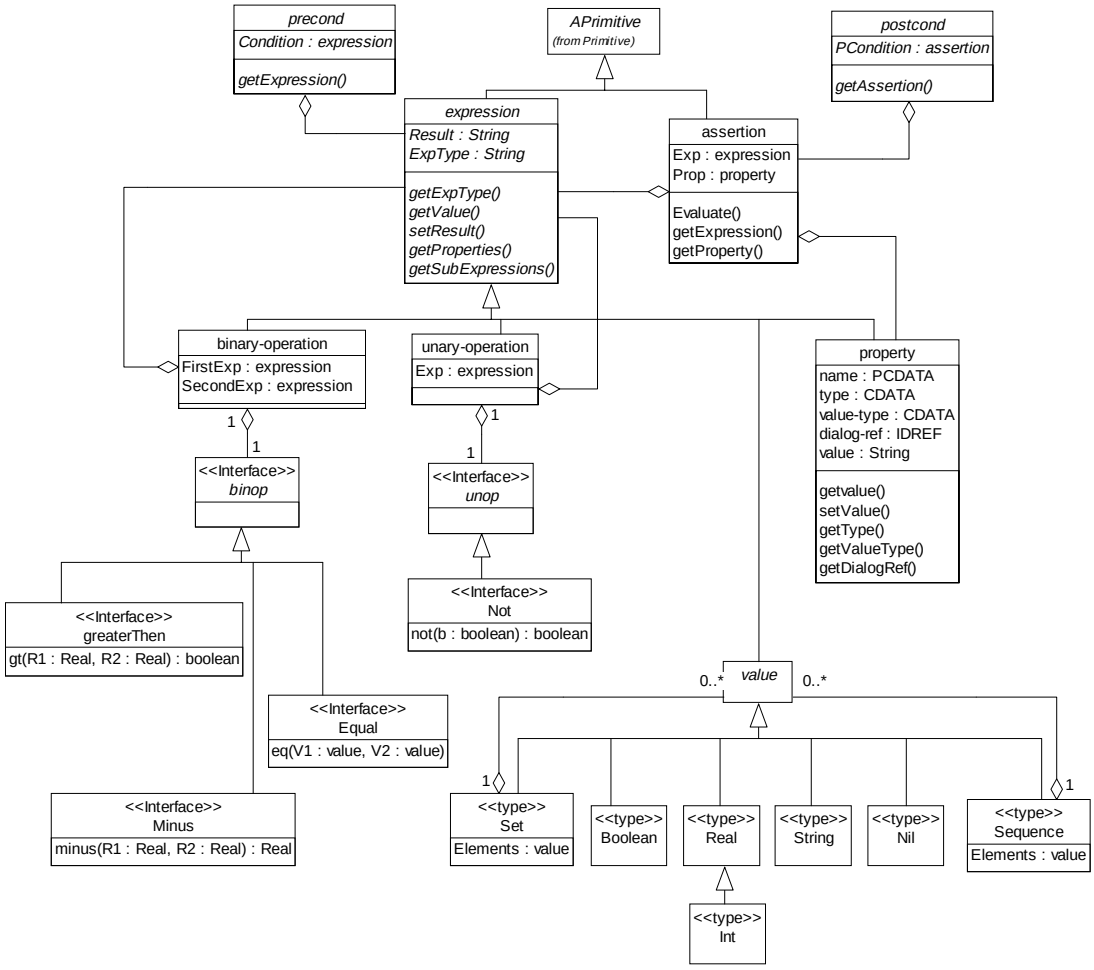

Figure 5.1 Context Filtering Package

As Figure 5.6 illustrates, binary operation (`binop`) and unary operation (`unop`) are parent interfaces. Each operation defined in the specification is modeled as an interface that provides the actual method representing this operation; these interfaces are derived from the parent interface `binop` or `unop` as appropriate.

A binary operation expression contains a reference to a `binop` object. Similarly, a unary operation expression contains a reference to a `unop` object.

Class property provides methods to get and set its value.

Class value is extended by a group of classes: Sequence, Set, Boolean, String, Real, Nil, Int. They represent the allowed data types for an expression value.

Link Package

Link is included within the content model of the Node template; therefore, any element employing the Node template may include relational links.
Link information in IETMDB is based on the HyTime standard. HyTime makes a distinct difference between links and their connected endpoints (locations or anchors). A link is a reference between two or more locations. A location is the address of a potential anchor point, which is the actual, physical point where the link ends.

The specification defines two types of links, hylink and link. Linkends attribute common to both types, lists the Ids of the named locations this link targets.

The named location address or nameloc contains one or several nmlist elements, where the nmlist element contains the Ids of the ultimate link target elements.

The link end list Linkendlist element declares the elements, which are valid targets of a link. The value of reftype attribute of the hylink element is linkendlist and the endtypes attribute of the link element is of linkendlist type.

Figure 5.7 shows the link package class diagram. Classes hylink and link are child classes derived from the abstract class alink. alink contains the common attribute linkends which references one or more named location (nameloc) class which in turn contains zero or more object references to a named element list (nmlist) class. Linkendlist is an interface class that is implemented by all classes defined in the specification as valid targets of a link.

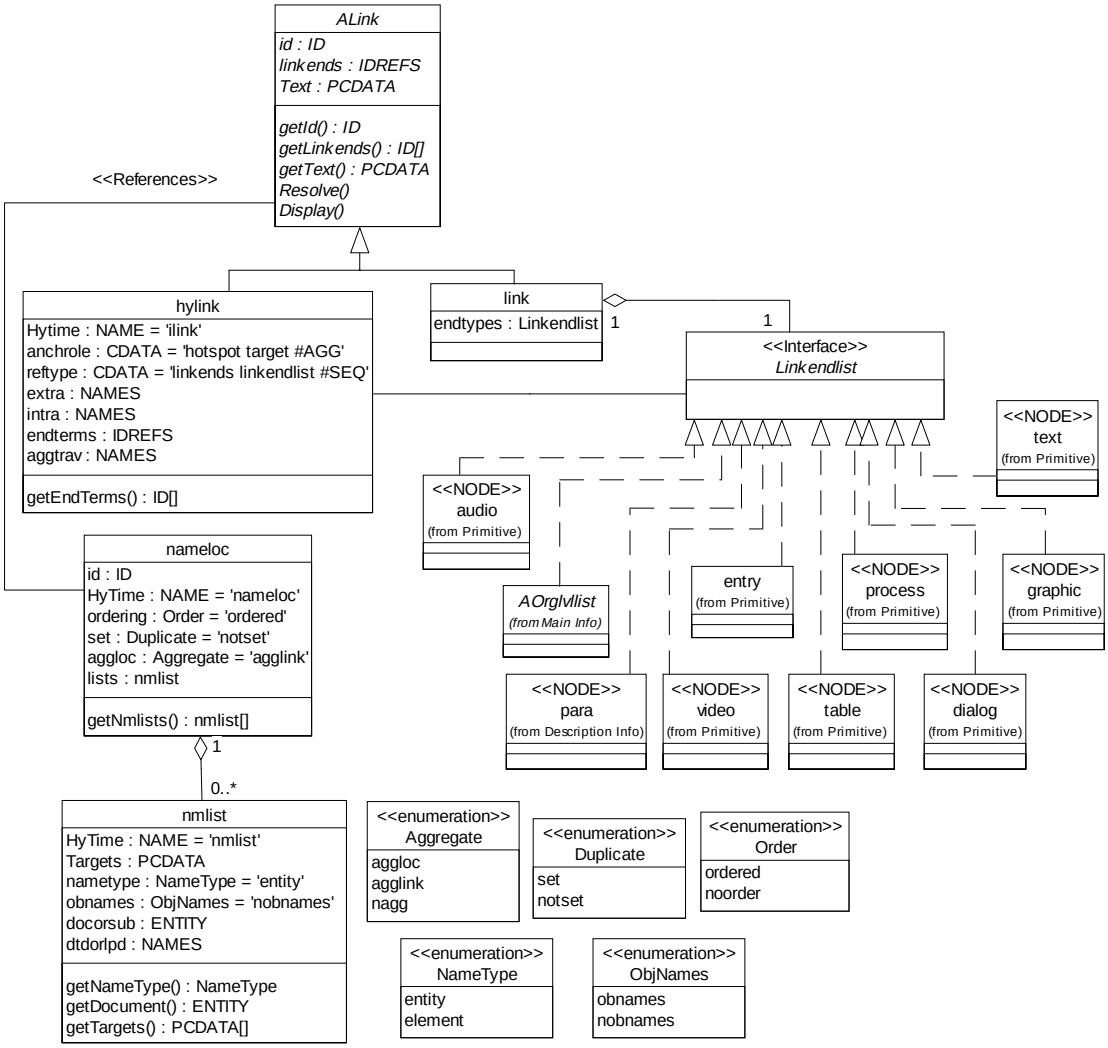

Figure 5.1 Link Package

Notation Package

Each graphic format defined in the specification is modeled as a class marked `format` via stereotypes. The public identifiers of the graphics standards are indicated by using a constraint. See Figure 5.8.

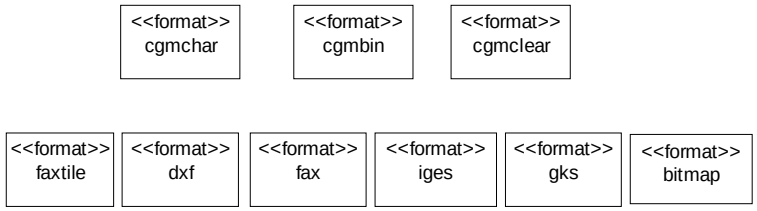

Figure 5.1 Notation Package

5.2.3 IETMDB Content Specific Layer in UML

The content specific layer is modeled as a subsystem package that contains a set of packages each representing one of the content specific groups of elements listed above.

Main Info package

Class `Techinfo` is of type `root` and it represents the top element of the information contained in this layer. It contains a reference to the top level systems objects as well as a reference to a version object. The technical information hierarchy begins at this level.

The system element defines a component (i.e., vehicle, system, subsystem, subassembly) which has associated technical information (i.e., descriptive, procedural, fault, or part information).

The system and version classes are stereotyped `NODE` because they employ the `'Node'` template from the generic layer. A `system` contains sub-system elements and descriptive, task, part, and fault information about the system.

These relationships are shown on Figure 5.9 as aggregation relationships with a multiplicity of zero or many between system class and `ASystem`, `ADescinfo`, `ATask`, `APartinfo` and AFaultinf classes.

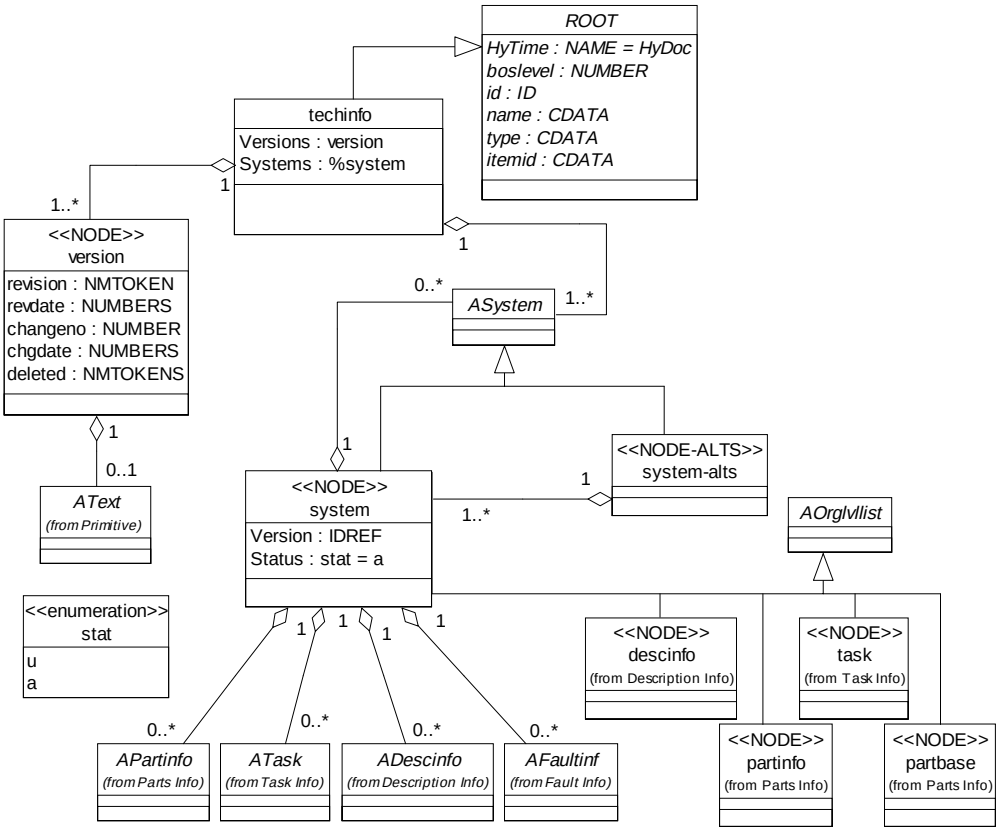

Figure 5.1 Main Info Package

Sub-primitives package

`ASubprims` entity defines a subset of the list of primitives `APrimitive` defined in the generic layer. Figure 5.10 shows `ASubprims` class as a child class of the `APrimitive` class

from the primitive package of the generic layer. The sub-primitives classes are `AText`, `ATable`, `AGraphic`, `ADialog`, `AAudio`, `AVideo` and `AProcess`.

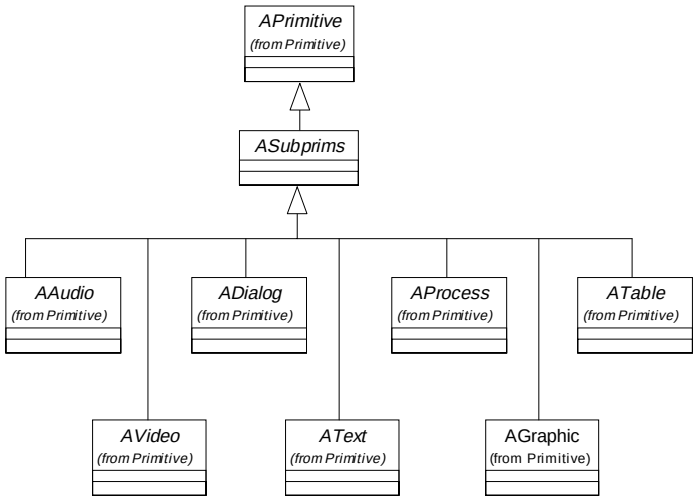

Figure 5.1 Sub-primitives Package

Description Info package

The class `descinfo` is used to define general purpose, non-procedural, narrative information. Derived from the abstract class ADescinfo, classes descinfo of type node and `descinfo-alts` of type `node-alts`, are the basic classes of the description info package. Classes `para`, `para-alts`, `loop-para`, `if-para` and `para-seq` represent five forms of a paragraph based on the generic layer templates node, node alternatives, loop node, if node and node sequence respectively. They follow their templates semantics as mentioned earlier.

Class `para` represents the simplest form of a paragraph. It describes the descriptive paragraph as having one or more sub-primitive element. It might contain a reference to a `para-seq` object.

Class `descinfo` contains a reference to one `para-seq` object. A `para-seq` class in turn contains one or more objects references of type `ADescinfo`, `para`, `para-alts`, `if-para`, or `loop-para`. These later classes form a group of child classes derived from the abstract class

`information`. Figure 5.11 shows the relationships between classes in the description info package.

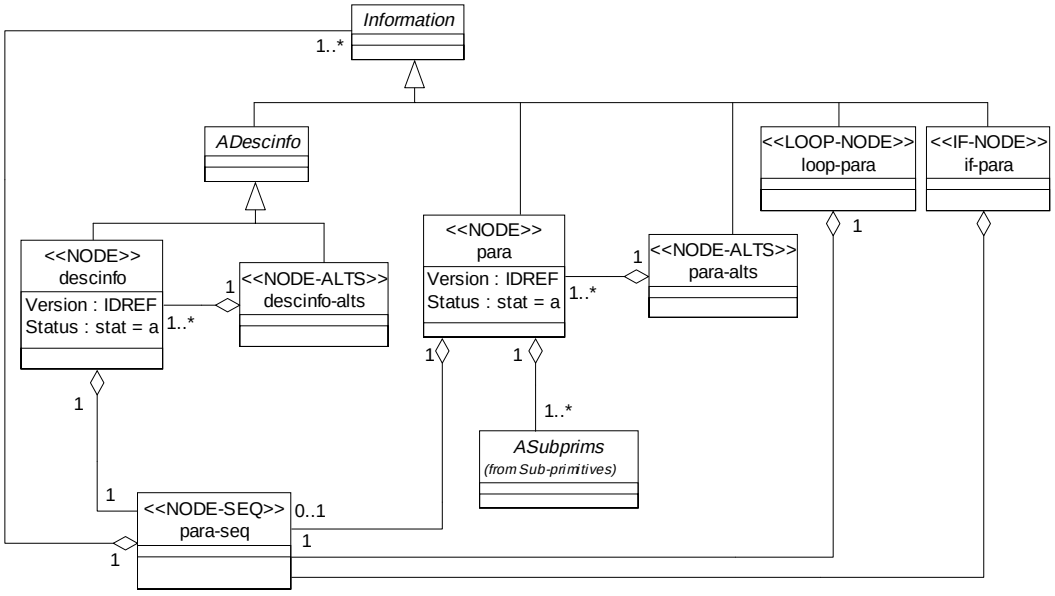

Figure 5.1 Description Info Package

Parts Info package

Parts package is broken-down to two diagrams: maintenance parts and supply parts for visual separation.

Supply system's view

`Partbase` describes the supply system's view of the part information. It describes the item in terms of its part number `partnum` attribute. The location element provides information for physical assessment. It contains x, y, z locations for a system.

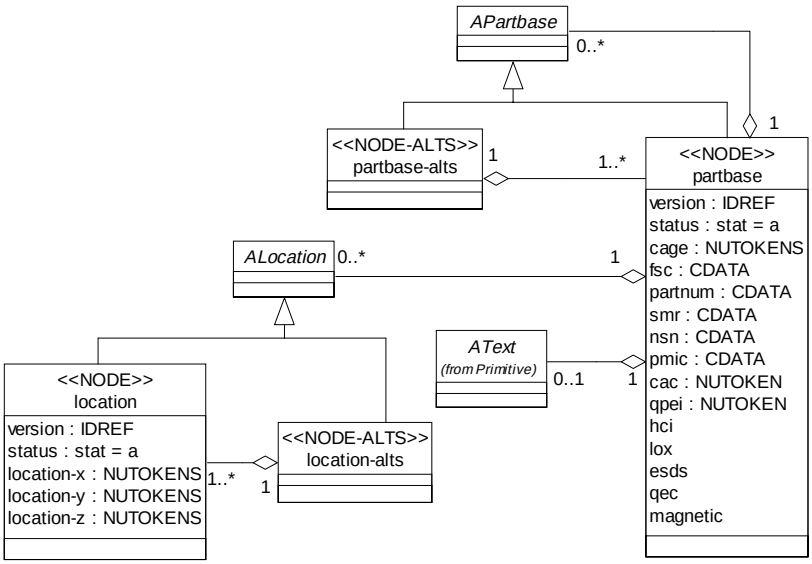

Figure 5.1 Parts Info: Supply System's View Package

In Figure 5.12, classes `partbase` and `location` employ the Node template; thus, stereotyped `NODE` and classes `partbase-alts` and `location-alts` employ the Node Alternatives template; thus, stereotyped `NODE-ALTS`. `Partbase` and `partbase-alts` are derived from the abstract class `APartbase`; similarly, `location` and `location-alts` are derived from the abstract class `ALocation`. `Partbase` class might contain a descriptive text and zero or many objects of type `ALocation`.

Maintainer's view

`Partinfo` describes the maintainer's view of the part information. It identifies parts information within its relative position in the weapon system. The `partinfo` class employs the Node template. Class `partinfo` contains zero or more alternate parts information (`APartinfo`). `Partinfo` also contains a reference to the components of the part (`APartbase`), any connecting parts (`AConnection`), attaching parts (`AAttach`-part), a

formal name for the part (`AText`), a picture of the part (`AGraphic`), and the location of the part in reference to the weapon system (`ALocation`).

`AAttach-part` and `AConnection` are abstract classes. Two class are derived from each of them, one stereotyped `NODE` and the other is stereotyped `NODE-ALTS`. Classes `attach-part` and `connection` contain a reference to one or more `APartinfo` object(s) that define the end part(s) of the attachment or connection. Figure 5.13 shows these classes and the relationships between them.

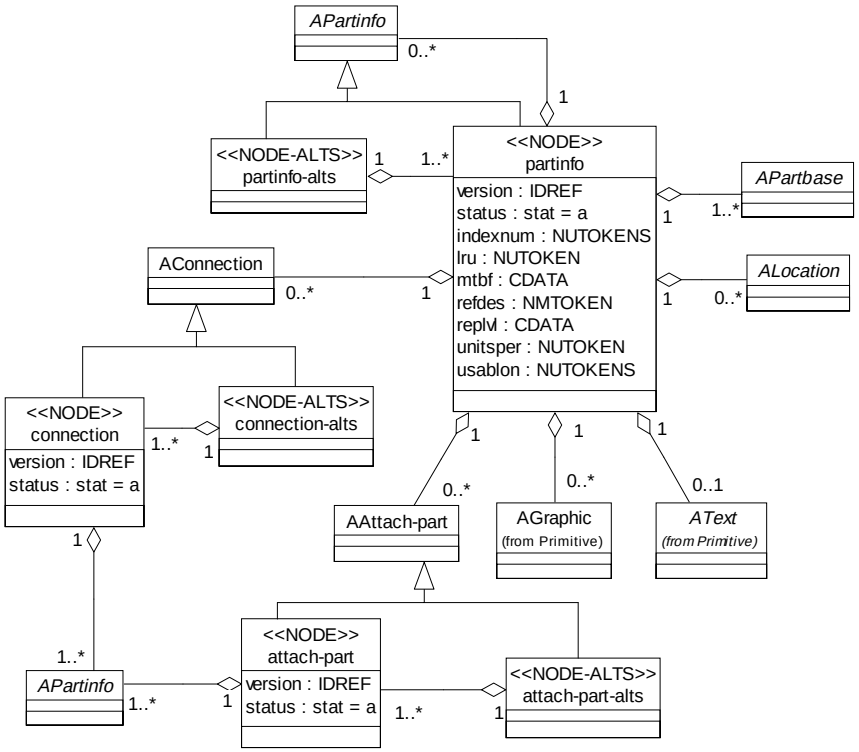

Figure 5.1 Parts Info: Maintainer's View Package

Procedural Info package

The procedural information is further divided into three packages: task information, input information and step information

Input Information

Input information is illustrated in three diagrams. The `AInput` class shown in Figure 5.14, contains references to objects of type `ARefmat` (reference material), `AExpend` (expendable), `AConsum` (consumable), `AEquip` (equipment), `APerson`, `AAlert` and `AReqcond` (required conditions) which represent the input for accomplishing a given task.

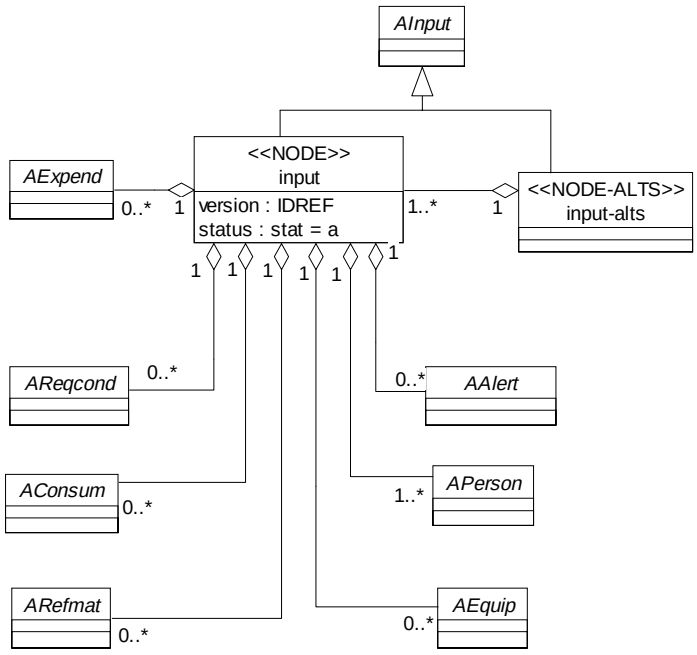

Figure 5.1 Procedural Info: Input Information Package

Figure 5.15 depicts the class representation of the expendable, consumable, equipment, personnel, reference material elements and the relationship between them.

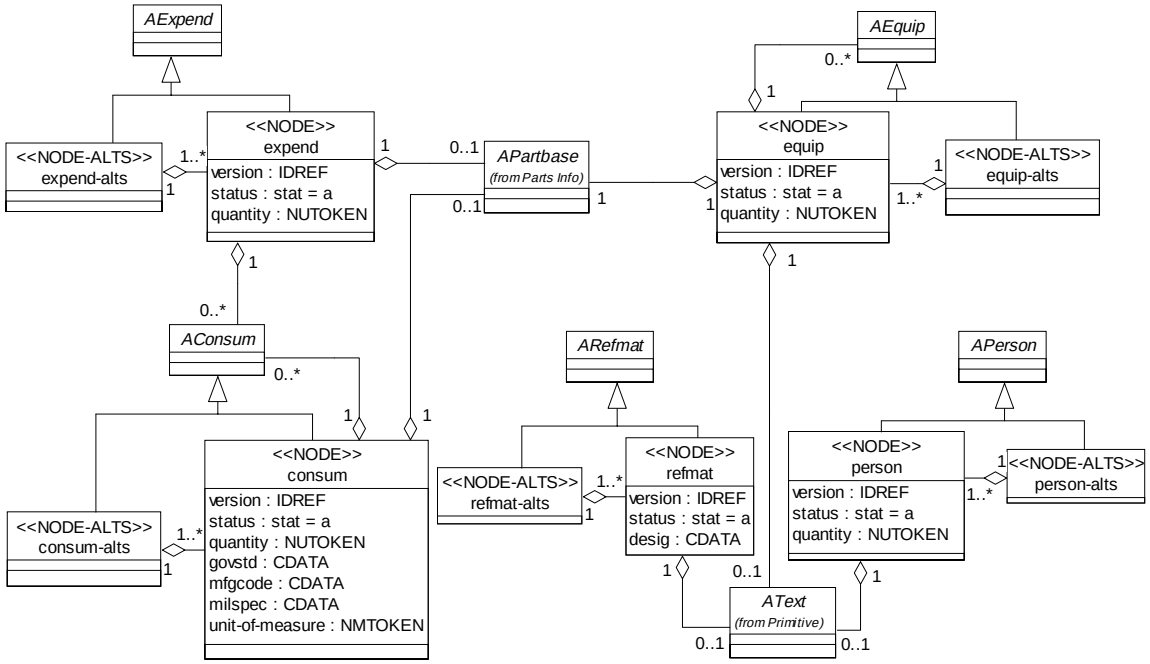

Figure 5.2 Procedural Info: Input Resources Information

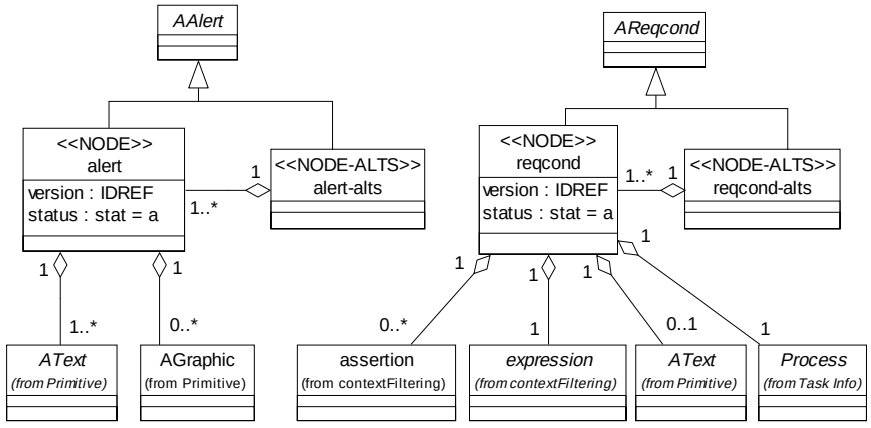

Figure 5.3 Procedural Info: Input Conditions Information

Figure 5.16 shows the alerts classes, an `alert` can be in the form of a text or graphic. It also shows the `reqconditions` classes, a `reqcond` contains a list of preliminary conditions, which must be met prior to beginning a task. If any condition is not met, it references an object of type `Process` (a task or step) which will satisfy the condition.

Step Information

A step class contains zero or more `AAlert` objects and zero or more `ASubprims` objects. It might also refer to an object of type `step-seq` (a step class conformant to the `node-seq` template). The abstract class `Procedure` is a base class for the `Process` class (parent of `AStep` and `ATask`), the `loop-step` (a step conformant to the `loop-node` template) and the `if-step` (a step conformant to the `if-node` template) classes. A `step-seq` object references one or more `Procedure` objects. Figure 5.17 illustrates the classes making up the step information and the relationship between them.

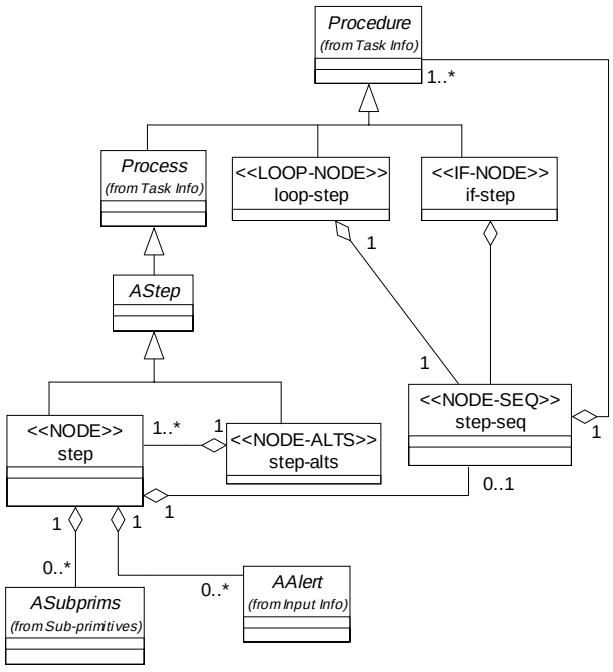

Figure 5.1 Procedural Info: Step Information Package

Task Information

The main class in this package is the `task` class. A `task` contains reference to an `input` object, a `step-seq` object and zero or more `follow-on` objects. A `follow-on` is a maintenance condition which must be accomplished sometime following the completion of a task to clean-up or undo actions performed during the task. The `follow-on` class references an object of type `Process` meaning any of its derived classes: `ATask` or `Astep`. Figure 5.18 shows the class diagram for the task info package

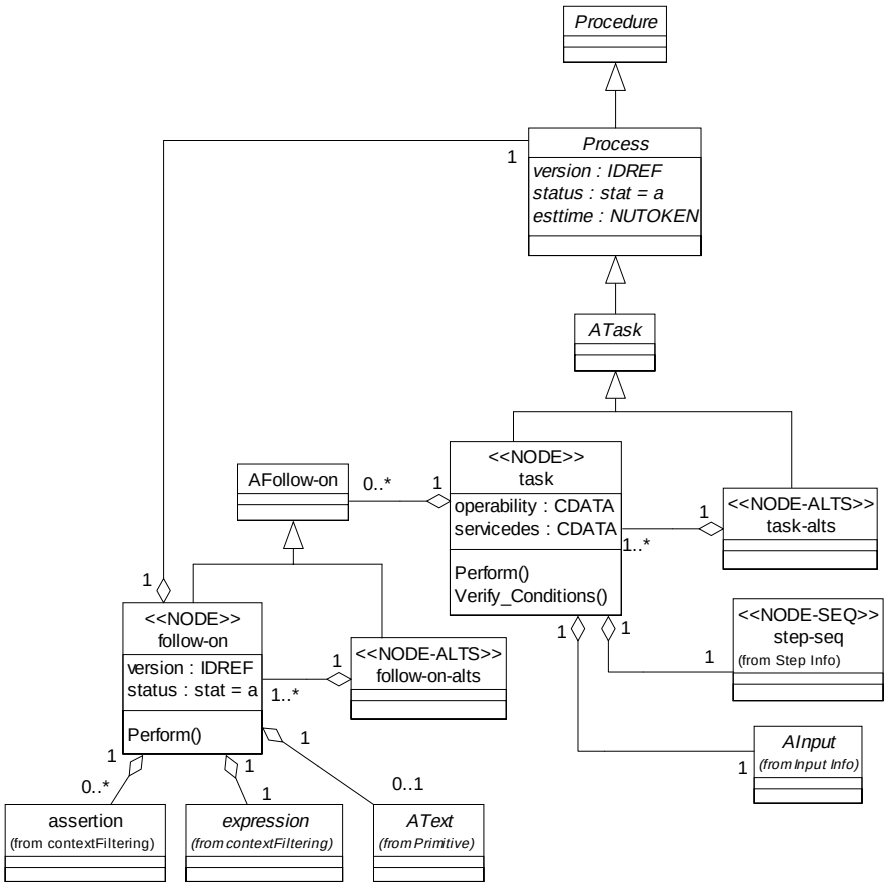

Figure 5.1 Procedural Info: Task Information Package

Fault Info package

The `faultinf` class contains all the tests and faults associated with the system that refers to it. A `fault` class contains one or more `ASystem` objects, which identify the potential failure, which may occur on a system. The `fltstate` (fault state) class contains one or more fault implicated or eliminated from consideration as the result of a test that has been performed. A fault contains the appropriate `rect` (rectification) objects to correct the discrepancy. Figure 5.19 shows the fault class diagram.

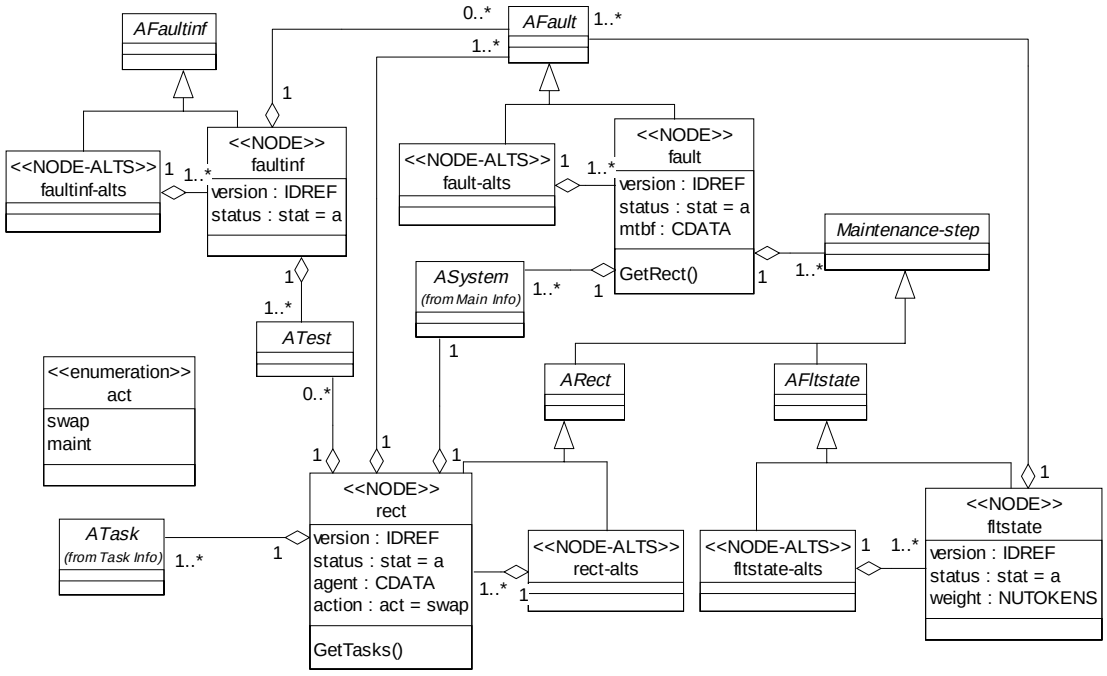

Figure 5.1 Fault Info: Fault Information Package

A `test` class contains a `task` to perform and one or more `outcome` objects. The `outcome` class identifies a result of a `test`. The `outcome` precondition list is evaluated against the result of the test, and the appropriate outcome is selected. In a dynamic troubleshooting

model, the outcome will contain a `fltstate` (fault state) object that identifies an implicated or exculpated set of faults. In a static troubleshooting model, the `outcome` will contain another `test` or a `fault` object (both are derived from the abstract class `Fault-Tree-Step`). Figure 5.20 shows the class diagram for the test classes.

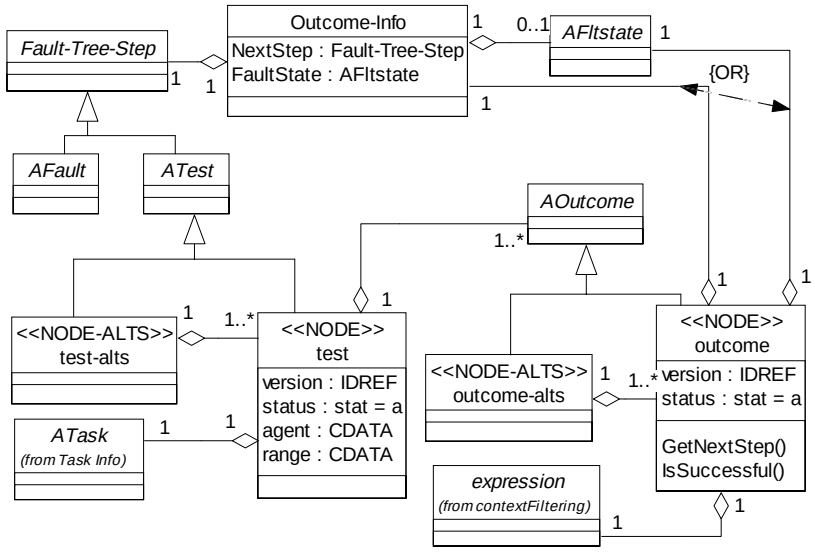

Figure 5.2 Fault Info: Test Information Package

5.2.4 Modeling IETMDB Dynamic Behavior

UML behavior diagrams are used to illustrate the behavior of the IETM display system.

Figure 5.21 is a UML use case diagram that shows the top-level functions, provided by an IETM display system for the user and their relationships. The display system processes information by first collecting information from the user, then it retrieves the next applicable (based on preconditions evaluation) piece of information and displays it to the user.

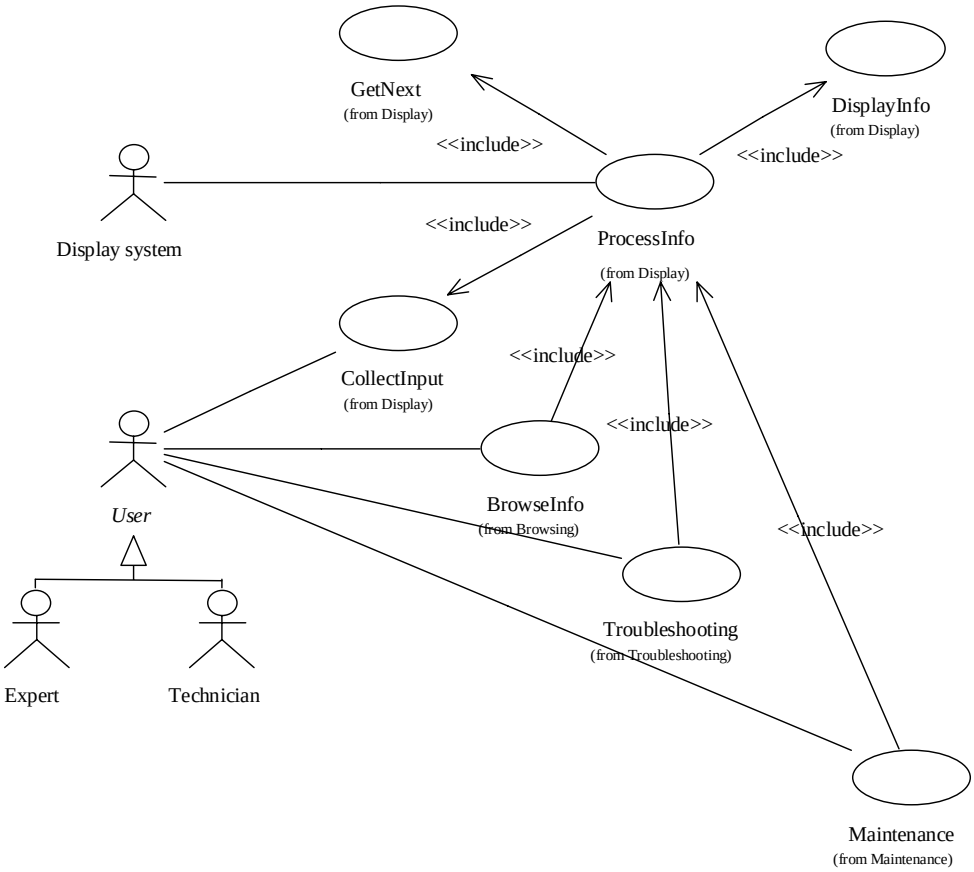

Figure 5.1 Top Level Fnctions

Figure 5.22 illustrates the main activities the user can perform on the IETM and their breakdown.

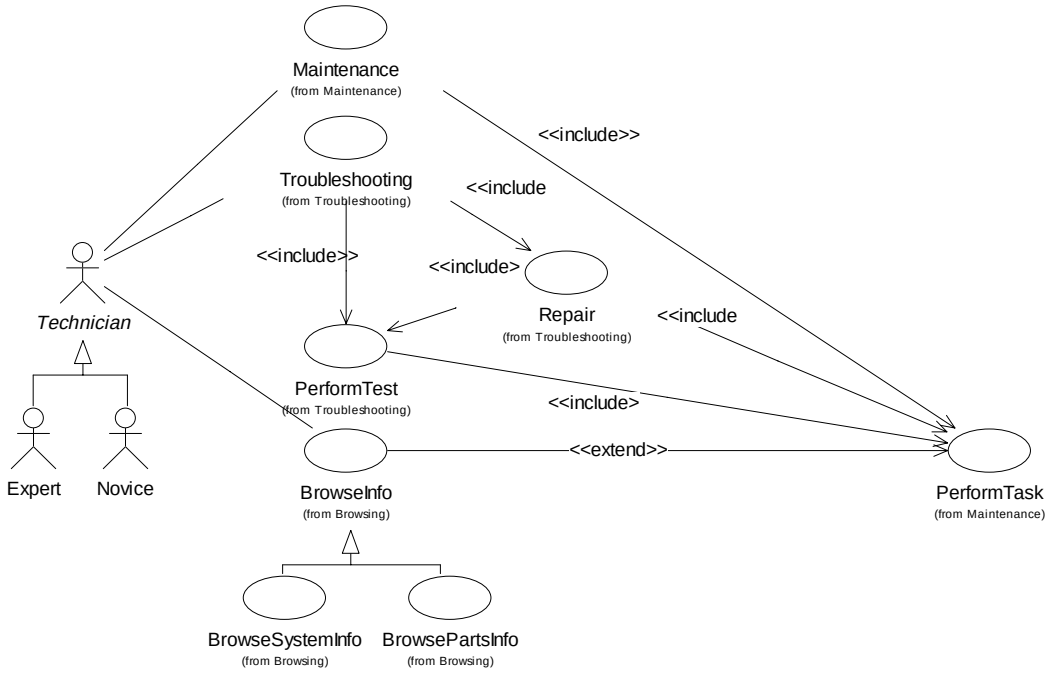

Figure 5.2 User Activities

The following sequence diagram 5.23 shows the sequence of interactions required to retrieve an element based on its template and Figure 5.24 shows the sequence of events required when displaying an object of type Node.

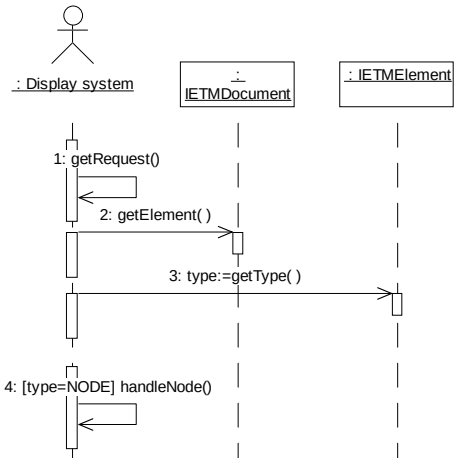

Figure 5.3 Get Element Type Scenario

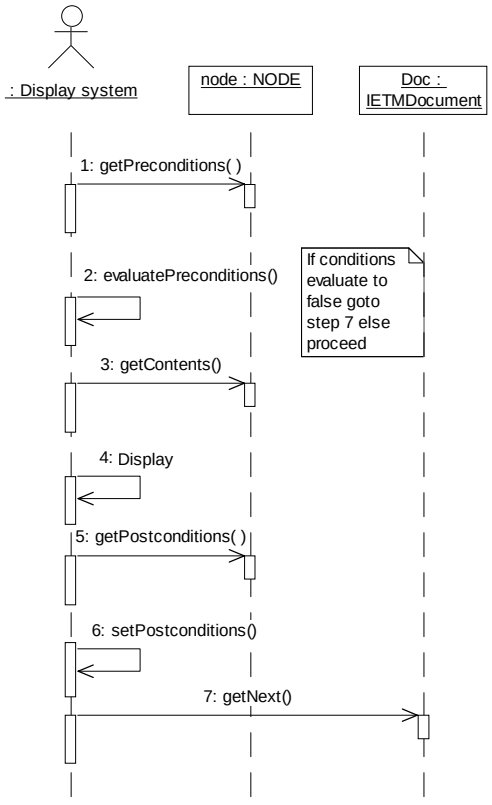

Figure 5.4 Handling a Node Element Scenario

The sequence diagram in Figure 5.25 depicts the flow of messages between IETM objects when information being presented depends on the user skill level.

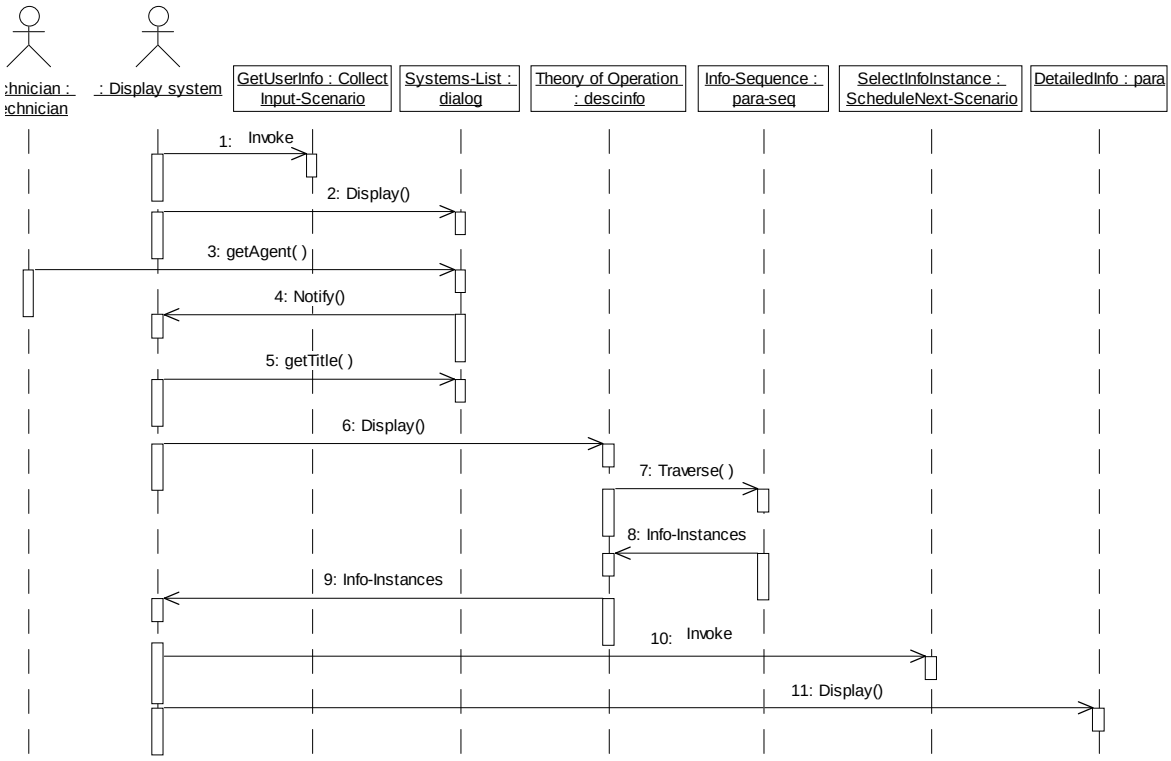

Figure 5.5 Information Based on Skill Level Scenario

The following Figure 5.26 shows interactions between the user and the IETM display system in a basic task performance scenario.

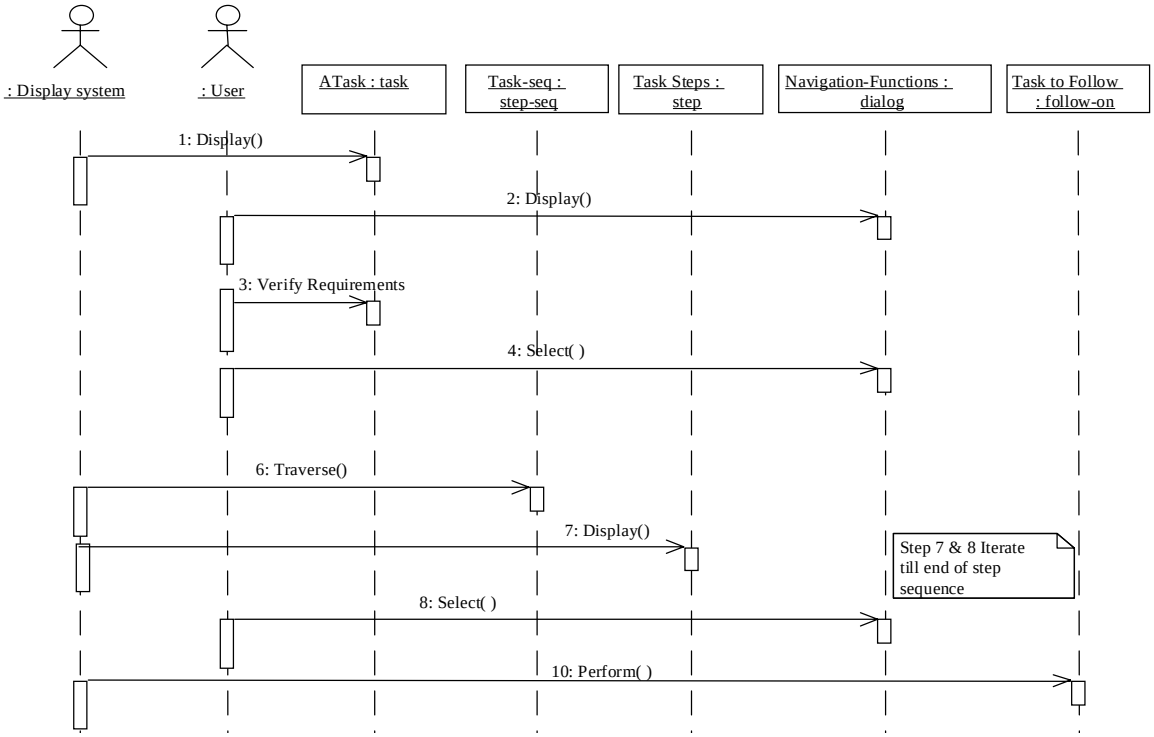

Figure 5.6 Basic Task Performance Scenario

Figure 5.27 illustrates the overall view of the IETM Object Model interfaces.

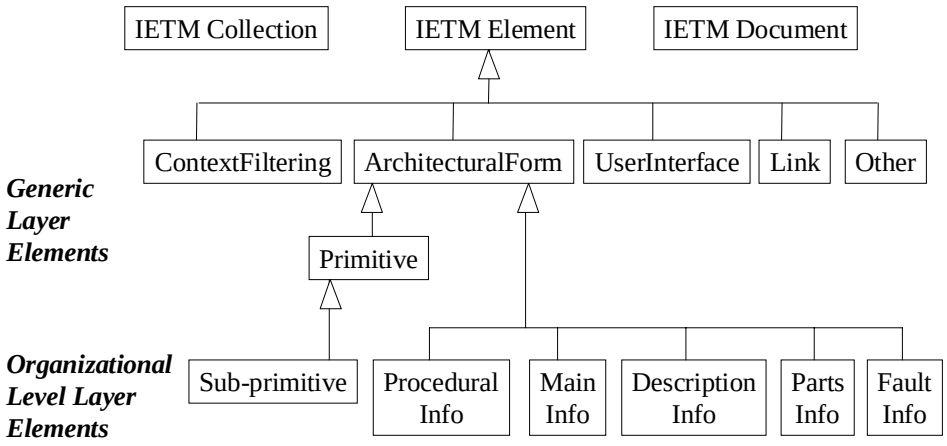

Figure 5.7 Overall IETM Object Model Interfaces

CHAPTER 6: PROPOSED IETM DISPLAY SYSTEM ARCHITECTURE

6.1 Component-based Development

Recently there has been renewed interest in the notion of software development through the planned integration of pre-existing software components. This is often called component-based development (CBD), component-based software engineering (CBSE), or simply componentware. The interest in CBD is based on a long history of work in modular systems, structured design, and most recently in object-oriented systems. CBD extends these ideas, emphasizing the outsourcing of pieces of the application system and focusing on controlled assembly of those pieces through well-defined interfaces.

The use of object technology in our work was driven by a number of factors including the desire to build software from reusable components. Object technology provides mechanisms, such as encapsulation and inheritance, that have the potential to support more rapid and flexible software development, higher levels of reuse, and the definition of software artifacts that more directly model our system concepts.

It is important to mention that an object-oriented approach does not automatically ensure the reusability of the produced software. Reuse-oriented architecture and component-based development provide a good framework structure for successful reuse.

System architecture is the blueprint (skeleton) of the application. Architectures are described as components and connectors and hence they give a component-based view of the structure of the application under development. Components are plugged in and out of an architecture.

The next step in our approach aims to link the developed object model to the IETM display system requirements through a COTS-based system architecture. UML notation and component diagrams are used to show the various system components and the way they interact.

6.2 JIA Architecture Types

The JIA is designed to be flexible and to accommodate various technologies for all aspects of the Internet and the Web. The JIA technical guidance supports four architecture types that represent the variants of IETM Intranet implementation:

- Architecture types C1 and C2: They are client-centric and require only a browser and a generic Web server.
- Architecture types S1 and S2: They are server-centric architectures. They require a Web server with database server and/or application server for server-side processing.

The proposed architecture falls into the architecture type S category. It involves server-side computations and might include a database server.

6.3 The Proposed Component-based Architecture

We advocate a three-tier architecture approach for structuring our application. The three-tier architectural approach supports the creation of large, complex client/server applications. The intent of three-tiered modeling is to support a more powerful form of object-oriented modeling, design, and programming. The cornerstone of our approach is a three-tier architectural, which means the creation of a layered software architecture in which there is complete separation of the Data Services and the Business Services (domain model), from the User Services (user interface). In such an architecture, business rules map into classes and/or operations within business objects.

This architectural approach promotes the distribution and reuse of business objects and libraries (logical packages) across applications. It also supports teams in managing an iterative, development process.

According to GartnerGroup, one should use 3-tier architecture if the application in hand has any of the following characteristics [23]:

- Large and complex, many application classes

- Applications programmed in different languages or written by different organizations
- Has two or more heterogeneous data sources
- An application life that is longer than three years.
- Significant inter-application communication

The IETM display system satisfies all the previous conditions. It is large, the IETMOM itself is over 150 classes, and it should be able to access different data sources and to incorporate components from various providers. The IETM display system should also provide support for IETMs of class 5 that allow for integration with other applications.

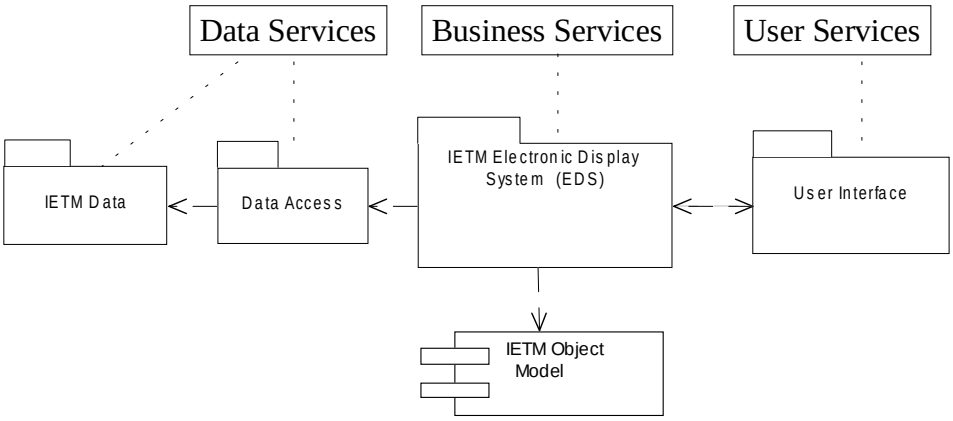

Figure 6.1 Top-level of Proposed Architecture

The proposed architecture for the IETM display system is a three-tier component-based architecture. The top-level system packages and components as shown in Figure 6.1 are:

- *User interface.* The user interface component is responsible for rendering information on the user side and for handling all user interactions.
- *IETM Electronic Display System (EDS).* The EDS system receives and handles user's requests for IETM data access and navigation. It uses context dependent filtering provided within

the IETM data to select the appropriate information for presentation. It also creates IETM document objects out of the IETM data based on the IETM object model.

- *Data access.* The data access component reads the IETM content information and process it to produce a document representation according to the interface specification with the EDS component.
- *IETM object model.* The IETM object model component is an implementation of the IETM object model specification. The later identifies the objects and interfaces, used to represent and manipulate an IETM document and the relationships between these objects and interfaces. This model is based on the IETM database specification MIL-PRF-87269A.
- *IETM data.* The IETM data represents the IETM content information. It can be stored as files on a file system or in a database in the form of files or data.

In the following subsections, we further break down the top-level architecture components to the level of well-defined components desired to assess the capabilities of Web COTS components to support the IETM system requirements.

6.3.1 User Interface Package

The UML component diagram representing the components of the user interface package is shown in Figure 6.2.

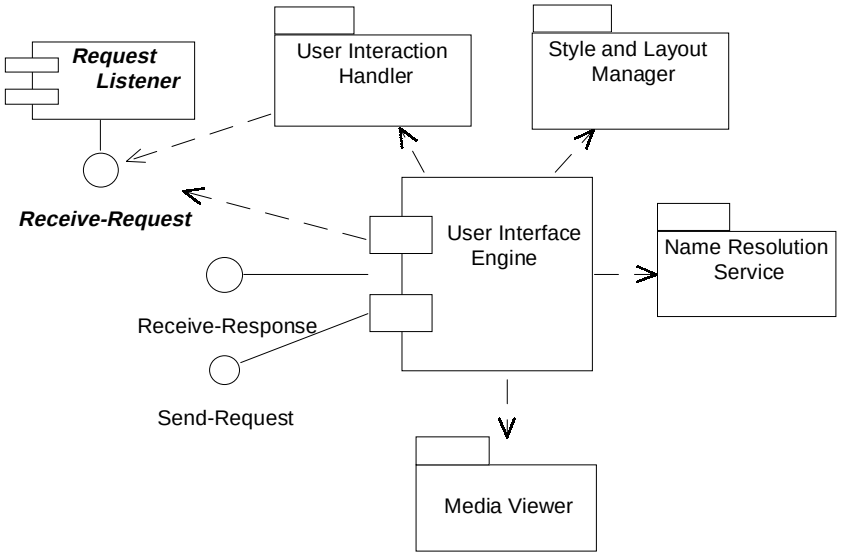

Figure 6.1 User Interface Package

At the heart of the package is the user interface engine. The user interface engine component implements two interfaces, send-request and receive-response, which allow for user interaction with the IETM display engine. To achieve its required functionality, the user interface engine depends on several packages, media viewer, user interaction handler, style and layout manager and name resolution service. A description of these packages follows.

Media Viewer

The IETM specification provides support for the inclusion of primitive elements such as: audio, video, graphics and process in an IETM document. Media viewer package contains the following components to handle the display of these elements, audio and video player, graphics viewer and process manager. They implement a display interface used by the user interface manger to invoke them. These components might require a media server component on the server side as illustrated in Figure 6.3.

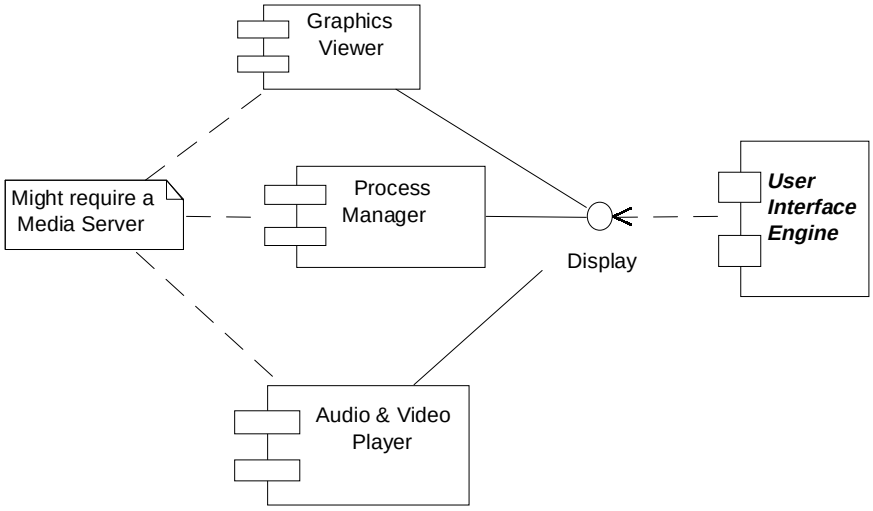

Figure 6.1 User Interface: Media Viewer

User Interaction Handler

The user interaction handler component reacts to user events. User interactions are organized into two main functions: screen display functions and IETM navigation functions. The IETM user interface specification (MIL-PRF-87268A) identifies a minimum set of navigation and control functions which should be available to the user and common to all IETMs. Figure 6.4 shows the various components of this package, screen display controls, IETM navigation controls and their event handlers screen display interaction handler and IETM navigation handler respectively.

The diagram also shows the dependency between them as well as the dependency of the user interface engine component on the interface perform action implemented by both handlers.

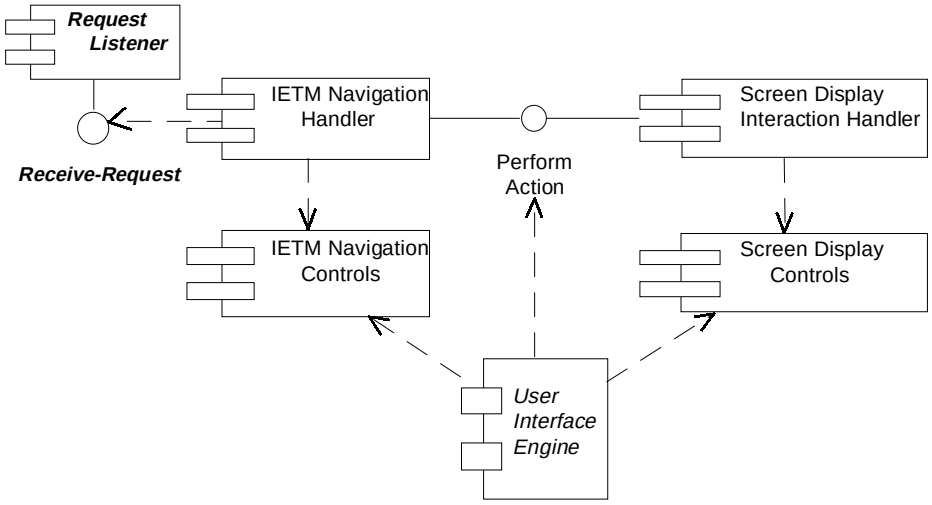

Figure 6.1 User Interface: User Interaction Handler

Style and Layout Manager

A GUI manager component is responsible for managing the graphical user interface. It implements the interface `render info` and uses display rules, layout templates, style library and user interface components. The user interface engine component depends on the `render info` interface the GUI manager provides. See Figure 6.5.

Name Resolution Service

A name resolution service is required to resolve an IETM document name and locate it on a server.

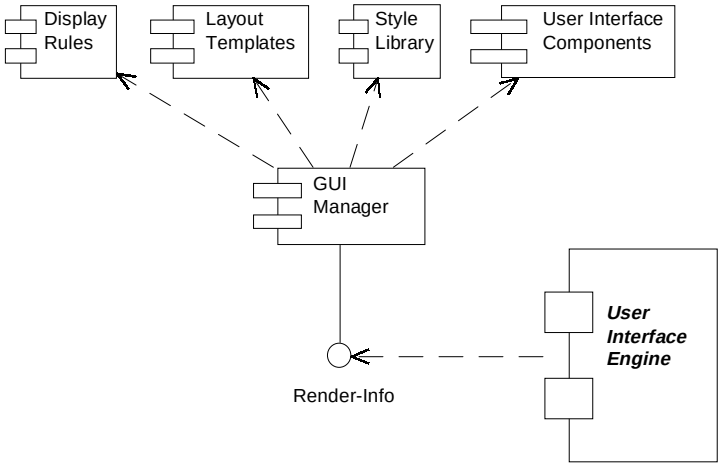

Figure 6.1 User Interface: Style and Layout Manager

6.3.2 IETM Electronic Display System (EDS) Package

The components in the IETM Electronic Display System (EDS) package perform the information processing required for presenting an IETM document to the user and responding to user requests. The UML component diagram for this package is shown in Figure 6.6.

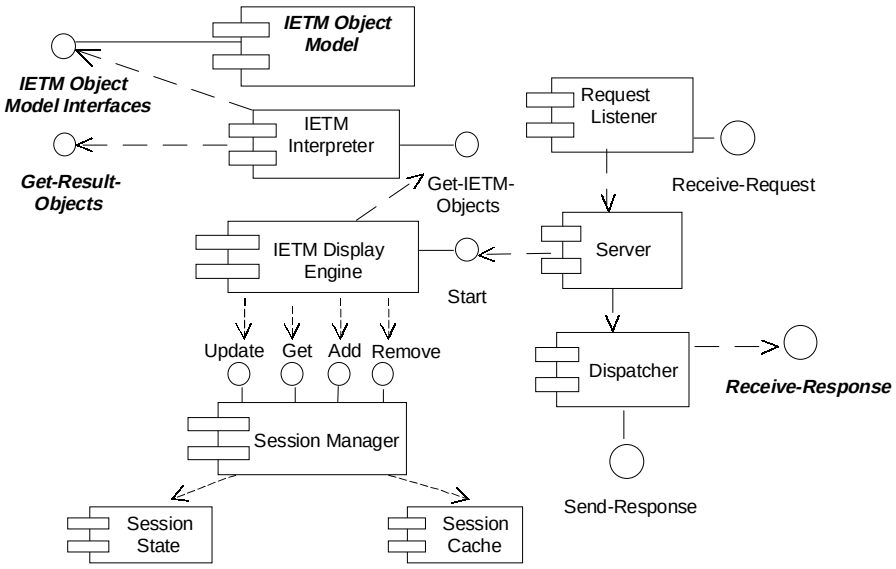

Figure 6.1 IETM Electronic Display System (EDS)

The Server

The server is the component that interfaces with the user interface package. It receives requests from the user interface engine, forwards them to the IETM display engine and sends back replies generated by the IETM display engine.

IETM Object Model (IETMOM) Component

The IETM object model defines APIs for manipulating an IETM document. These APIs are interfaces, meaning an implementation is required to provide the body of the methods and to expose them with the defined names and specified operations. The IETM Object Model component is an implementation of the IETMOM interfaces.

IETM Interpreter

The EDS package interfaces to the data access package through the IETM interpreter component. The IETM interpreter converts an object representation of the IETM document required for display into a representation that is based on the IETM object model. Its main functions are:

- Traverse the input tree of objects, which are the object representation of the IETM document instance (the actual technical manual).
- Create an IETM document object, which is the root object of the IETMOM tree.
- Generate the IETM objects tree by instantiating IETM objects based on the IETMOM and appending them to the IETM document object.

IETM Display Engine

The IETM display engine receives user requests and acts upon the objects representing the IETM document instance in memory to retrieve the required information and format it in a reply to the user. Its main functions are:

- Get user requests & generate replies
- Handle each object according to its node type (`node`, `node-alts`, `node-seq`, `if-node` or `loop-node`)
- When available, verify the preconditions of an object before displaying it
- Record the events associated with an object after it is displayed.
- Retrieve a particular object by ID or name
- Keep record of the current object being displayed

Session Management

Session State

The IETM specification defines an element called property that represents a variable. The IETM document contains references to these variables whenever their values are used to control the subsequent flow of information display. The specification anticipated that a global state table would be maintained by the display system to represent the current situation through property value pairs. Thus, the IETM display system should hold state information about each user of the system. Moreover, in a display system where a user can establish more than one session with the server, these state information become session specific information. Session state component maintains state data as name-value pair, example; a property can be named `Track` which is a string that holds the user skill level, valid values are the strings `Novice` and `Expert`.

Session Cache

One of the required IETM navigation functions is the back function, which allows the user to move to the previously displayed piece of information. In addition, the variables whose values have changed in the last display should be reset to their previous values. To achieve this, the session cache component acting like a history keeps track of the recently displayed objects.

Session Manager

The session manager component stores and retrieves state data and cached objects. The session manager uses the methods `get`, `set`, `update` and `remove` provided by the session state and session cache components.

6.3.3 Data Access Package

The data access package shown in Figure 6.7, reads the IETM data and represents it in form of objects. These objects are further transformed by the IETM Interpreter into IETMOM objects.

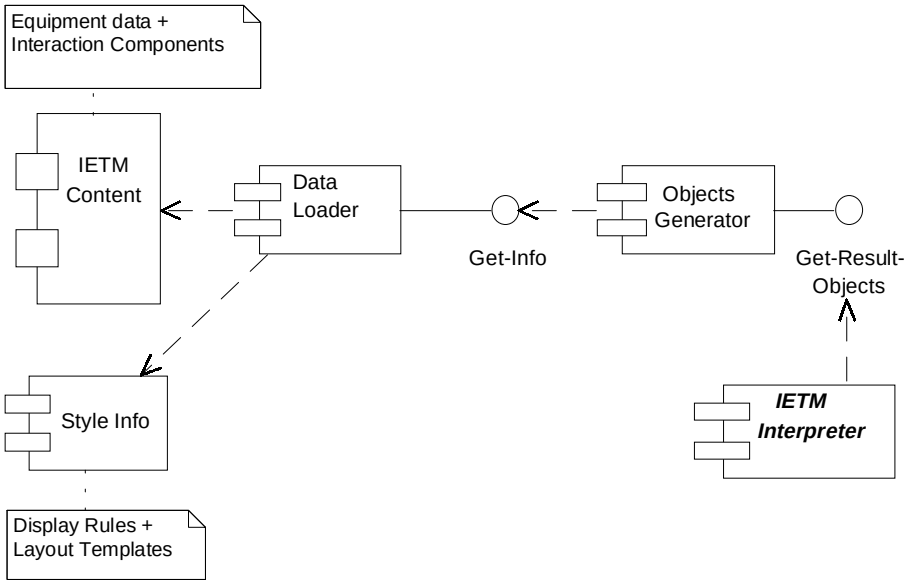

Figure 6.1 Data Access

Data Loader

The data loader component accesses the requested IETM data and loads it into memory.

Objects Generator

The document objects generator provides an object representation of the IETM data retrieved by the data loader.

6.3.4 IETM Data Package

The IETM data is mainly composed of two components:

- The IETM technical manual data conforming to the IETMDB specification.
- The style information including display rules and layout templates needed to render the IETM data on the user screen according to the IETM user interface specification.

6.4 The Web-enabled Component-based IETM Display System Architecture

The proposed architecture described in the previous section is developed based on the knowledge of existence of reusable components in the Web environment.

Alternative Web components are investigated to determine their fitness for use in the system. Qualification of a component is ascertained by assessing its functionality, interfaces and limitations to support the system requirements.

In developing the Web-enabled architecture, we consider the following:

1. COTS components within the proposed architecture, these components include Web Browser, Web Server, Media Viewer, XSL Processor, etc...
2. Components developed for reuse, these components include components specific to the IETM display application such as the IETM display engine. They also include reusable components in applications within the IETM domain such as the IETM Object Model.
3. Alternative technologies and transfer protocols. We identify alternative technologies and/or standards where they apply in the architecture, such as the use of HyperText Transfer Protocol (HTTP) between the browser and the server.

We elaborate on these three categories in the following subsections. The following diagrams illustrate the Web-enabled architecture.

Figure 6.8 shows the user interface components where the COTS components and alternative technologies are identified.

COTS components are the GUI manager, User Interface Engine which are represented by a Web browser and the Media Viewer. The User Interface engine communicates with the Web server using one of the following standards: HTTP, IIOP and DCOM. The IETM Navigation Handler can be implemented using various technologies such as: Java applets, ActiveX and forms.

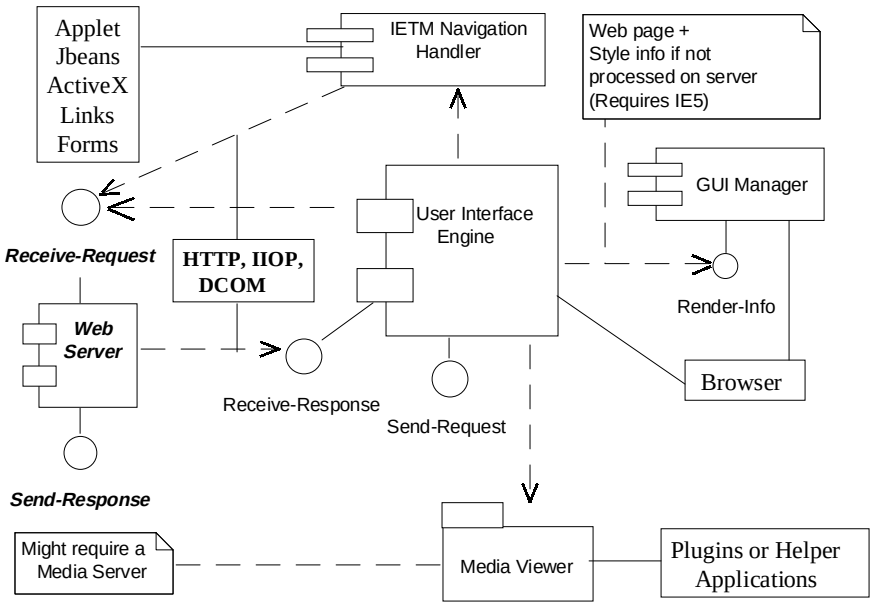

Figure 6.1 User Interface Components

Figure 6.9 shows the components of the IETM Electronic Display System. COTS components are the web server and XSL processor. The IETM display engine can be implemented using one of alternative Web technologies such as CGI, servlet, etc...

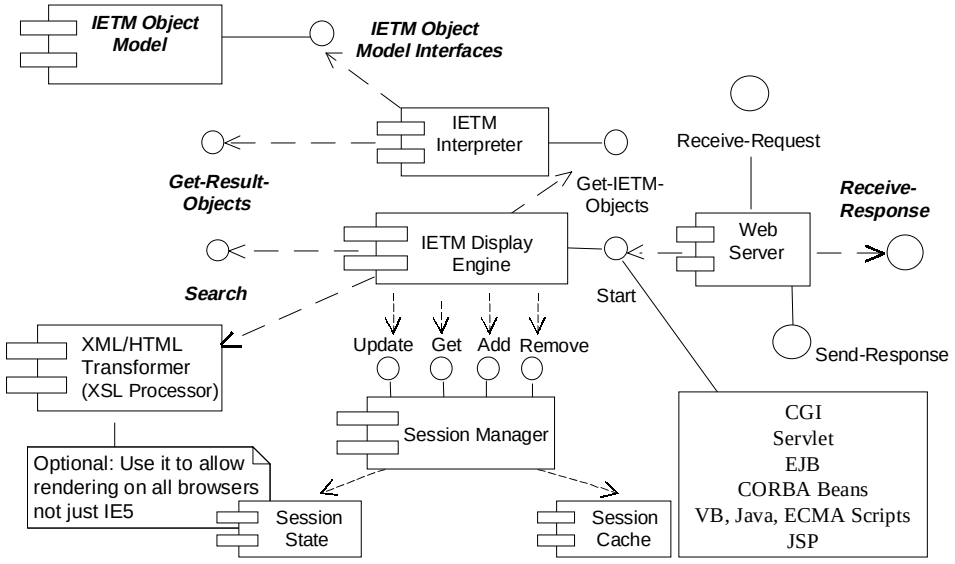

Figure 6.2 IETM EDS Components

Figure 6.10 illustrates the components dealing with the data access in a file-based system. The data loader components, the parsing engine and the XML/HTML Transformer or XSL Processor are all widely available COTS components. To allow Web access to IETM data, we propose to use XML as the IETM document format. XSL contains the styling information needed to render the XML IETM data.

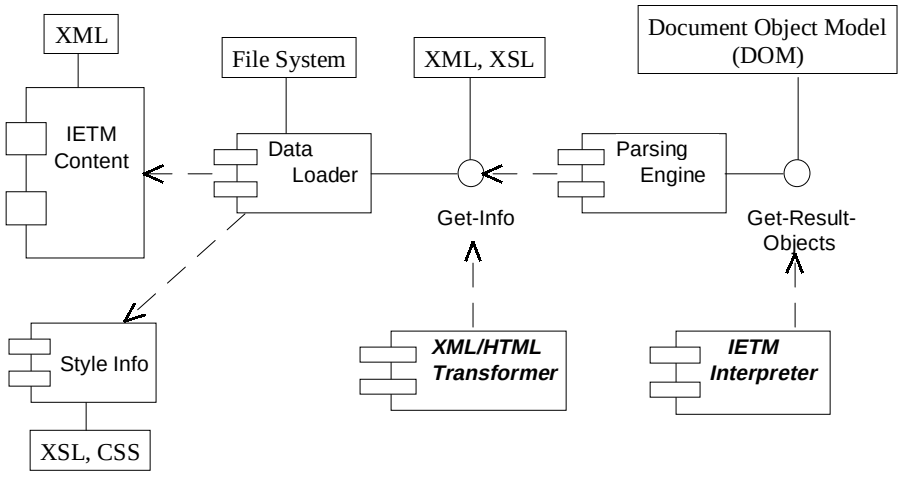

Figure 6.3 Data Access Components in File-based System

Figure 6.11 illustrates the components dealing with data access in a database-driven system. COTS components exist that represent the database information in the form of objects conformant to DOM or to other models. The data loader is also a COTS component.

CORBA technology can be used to allow the access of various remote IETM databases using distributed objects technology and standards.

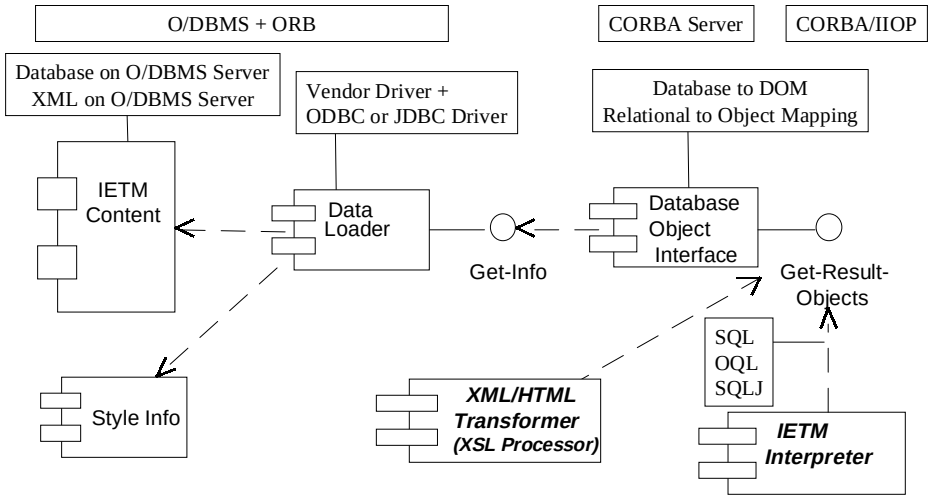

Figure 6.4 Data Access in a Database-driven System

6.4.1 COTS Components

Web Browser

The commercial Web browsers available provide the user interface functionality required by the IETM system and more. A Web browser uses HTML format for data representation (at the time this thesis is written only Internet Explorer 5 supports XML document format) and communicates with the server using the HTTP protocol

Media Viewer

A media viewer component allows viewing of audio and video clips over the Internet without having to completely download the clip. Most of them are either Plugins or helper applications that work in any Web browser. In cases where the streaming media technology is used, a media server might be required. Plugins can be downloaded when needed.

Web Server

The Web server represented in the proposed architecture is a generic COTS Web server. It requires no proprietary extensions and its main function is to forward requests and replies between the user and the IETM display engine. It interfaces with the Web browser using HTTP. Most commercial Web servers support several standard APIs to interface with server-side components e.g. Common Gateway Interface (CGI) APIs and Servlet APIs.

XSL Processor

In an XSL transformation, an XSL processor reads both an XML document and an XSL style sheet. Based on the instructions the processor finds in the XSL style sheet, it outputs an HTML document. Though it is designed primarily for transforming an XML representation to another, many implementations support outputting HTML.

An XSL processor might be needed at the server end to transform XML data to HTML so that it can be viewed on all browsers. This component is optional; the other alternative is to use Internet Explorer, again this is based on the current status of XML support in the browsers available.

Data Loader

The data loader as the name indicates retrieves the requested data from its storage media. It can be a simple file system or a database driver.

Objects Generator

The objects generator interfaces with the data loader component and transforms the retrieved data into objects. The resultant objects are either conforming to the Document Object Model (DOM) or proprietary depending on the data loader component.

The objects generator component can be an XML parser; DOM-compliant parsers are widely available. They return a DOM tree structure that represents the entire document.

In the case where IETM data is stored in a database, the object generator component can be a middleware that transfers data from the database to a DOM tree of objects. Most middleware are database vendors independent and work well with any relational database type through Open Database Connectivity (ODBC) or Java Database Connectivity (JDBC) or both.

Session Manager

The Hypertext Transfer Protocol (HTTP) is by design a stateless protocol. The need to build effective Web applications gave rise to a number of approaches to associate a series of different requests from a particular client with each other. Many strategies for session tracking have evolved over time. It is possible to adopt any of these strategies or to reuse an existing component that implements the session management functionality.

Session State

A session state component is a small component that behaves like a dynamic data storage bound into a session and exposing specified interfaces for getting and setting session variables. The Java language provides components such as `Vector` and `Hashtable` that can be used to satisfy these requirements.

Session Cache

The session cache component is very much similar to a session state component except that the objects stored in this case represent previously displayed IETM information instead of variables. Thus, the same Java components can be used in this case.

6.4.2 Components developed for reuse

Components specific to the IETM display application

The IETM Display Engine, IETM Navigation Handler and Style Information components are particular to the IETM display application. They can be mixed and matched to suit different IETM delivery formats and different procuring environments and configurations. The same IETM display engine can be used to display any IETM data source. Style information are dependent on the document structure specified in the IETMDB only, therefore, any IETM data source compliant with the specification and represented in XML can be styled using the same XSL stylesheets.

These components are flexible and self-describing, hence, they can be executed at the client or at the server e.g. they can be downloaded and executed on the client side in the case where a standalone mode is required. In addition, they can be used in a distributed environment. IETM vendors can therefore expend their efforts in providing IETM source data rather than having to constantly "re-invent the wheel" of an entire display system for each implementation. The IETM display engine component interfaces with the data source using the IETMOM, allowing several interoperability alternatives to emerge:

- Data source can be arranged in XML only with no added software components. In this case, a previously existing or developed DOM-compliant parser and DOM to IETM interpreter can be reused to retrieve the required IETM objects.
- Data source can provide the information in the form of DOM-compliant objects. A generic DOM to IETM interpreter can thus be brought into play.

- Data source in a proprietary format can be extended with a driver that converts this format to IETMOM directly, thus combining both objects generator and IETM interpreter into one component.

Within the proposed architecture, the IETM source data should comply with the IETMDB specifications and should provide a mean to represent it in the form of XML, DOM or IETMOM.

This architecture will give rise to a market of reusable objects generators that convert from various data formats to a standard object representation such as DOM and reusable IETM interpreters that transform a certain object model to IETMOM.

Components suitable in applications within the IETM domain

The IETM Object Model implementation and the IETM Interpreter components are means to represent an IETMDB compliant data. Hence, they are independent of the application that uses this data.

As an example, these components can be used to provide a generic IETM authoring and update system. The authoring and/or update application can interface to the IETM data source using the IETMOM.

Moreover, they can provide a base for a standard interface to other applications like expert systems and parts supply systems as supported by IETMs of class 5.

CHAPTER 7: CASE STUDY AND IMPLEMENTATION

To determine the applicability of the proposed solution and to demonstrate the ability of the IETM Object Model to achieve interoperability in handling source data, we implement a demonstration system as a proof-of-concept.

The demonstration system is strictly file based and is supported with a widely available COTS Web server. We ensure that the system exercises the dynamic behavior encapsulated in the IETMOM.

7.1 File Based IETM Display System

7.1.1 Case Study

The case study is an SGML file taken from the F16 aircraft technical manual. The manual provides procedural information for three systems within the aircraft: Air data system, air conditioning and horizontal situation indicating. Each system contains a set of maintenance tasks relevant to its function.

The manual is carefully chosen to represent different data structures specified by the IETMDB. It encompasses elements conforming to various node templates such as: `node, if-node` and `node-seq`. Dynamic behavior is also represented in the `preconditions`, `postconditions` and `dialog` (user interaction) elements present in the manual.

The implementation is not dependent on the case study. Therefore, this demonstration system can be used to display any IETM manual that covers the same information as the case study and a full implementation can display any IETMDB conformant document.

7.1.2 Functional Overview

The procedural information also called task information within the interactive electronic technical manual aims to assist operators performing maintenance tasks on the equipment.

Information about maintenance tasks, include task requirements like tools, materials and persons, steps describing the procedure and follow-on tasks.

Most IETMs provide two separate and complete tracks, each representing a differing level of detail: one level for a novice skill level and another for an expert skill level. The novice level contains all information necessary for an inexperienced user to perform the task involved. The expert level functions as a checklist, presenting only the steps required to complete a task, assuming a higher level of theoretical knowledge. The expert user is given the ability to access information at the novice level but the novice user can not access information at the expert level.

The way this is implemented is usually through a state variable called `track` that is set based on the user login information. Once available, this variable is used throughout the whole document to define `preconditions` for information display based on the user skill level.

The user is first presented with the login screen where he identifies his skill level. The IETM display engine stores this information and responds with a list of systems and tasks. The user then selects the task he wants to perform. The display engine then displays the task information in order, starting by the task requirements, then the step sequence one step at a time and then the follow-on tasks. While performing a task, interactions between the user and the engine are driven by the IETM content itself.

At all times during IETM information display, the user is presented with a set of navigation buttons. The navigation button `next` allows the user to proceed to the next piece of information. A `back` button is also available to go to the previously displayed piece of information. In `perform` mode, clicking the button `back` will reset all state variables to their previous values before the current information was displayed. However, in a `browse` mode, state variables are not reset.

The button `switch mode` is used to switch between `browse` mode and `perform` mode. The button `switch track` is used by the expert user to switch to the novice level seeking more detailed information.

7.1.3 Implementation Decisions

The demonstration system implementation is based on the following design decisions:

1. For demonstration purposes and time limitations, not all parts of the IETM Object Model are implemented. The objects selected for implementation are based on the case study manual for example the `table` object in the IETMOM is not implemented since there is no `table` elements in the manual we use. The IETMOM parts covered are objects from the procedural information, context filtering, dialog and architectural forms packages and `text` from the primitive package.
2. The source data and the IETM display engine output to the user are in XML format compliant with the IETMDB specification.
3. The system uses an XML-enabled browser to avoid dealing with an extra component (the XSL processor component).
4. The IETM display engine is implemented as a servlet due to the following advantages servlets have over other server extension mechanisms:
    - They use a standard API that is supported by many web servers.
    - They provide support for session tracking.
    - They have all the advantages of the Java programming language, including ease of development and platform independence.
    - They can access the large set of APIs available for the Java platform.
    - They are much faster than CGI scripts because a different process model is used.

The following sections discuss the tools and the components used in the implementation of the demonstration system.

Platform: Windows 98

For ease of development, we choose to run both client and server on the same machine running the operating system Windows 98.

Development Language: Java (JDK 1.1.8)

Java enforces modular development thereby making future additions and modifications to the system simpler. Any application built in Java is portable across multiple platforms.

Web Server: Apache Web Server

We used the Apache Web server because it is robust, extensible and in accord with the current HTTP standards.

Servlet Engine: Apache JServ

Servlet APIs are either supported by a Web server or available as add-on through server vendors and industry partners. The servlet engine of choice has to work with the Apache Web Server. Apache JServ is a pure Java servlet engine fully compliant with the JavaSoft Java Servlet APIs 2.0 specification. It works on any "version 1.1 compliant" Java Virtual Machine and may execute any Java servlet compliant with version 2.0.

Web Browser: Internet Explorer 5

For this prototype, we used Internet Explorer 5 for its support to XML rendering using XSL. However, the solution described earlier (using an XSL processor at the server side) can be used to overcome the dependency on a single Web browser in the system.

XML Parser: XML Parser for Java from IBM alphaWorks

XML Parser for Java is a validating XML parser written in 100% pure Java. It supports the DOM specification and contains classes and methods for parsing, generating, manipulating, and

validating XML documents. This parser is believed to be the most robust XML processor currently available and conforms most closely to the XML 1.0 Recommendation. For comprehensive testing of XML Parsers, please refer to this URL [16].

7.1.4 Top Level Design

The top-level design of the demonstration system is illustrated in the following Figures 7.1, 7.2 and 7.3.

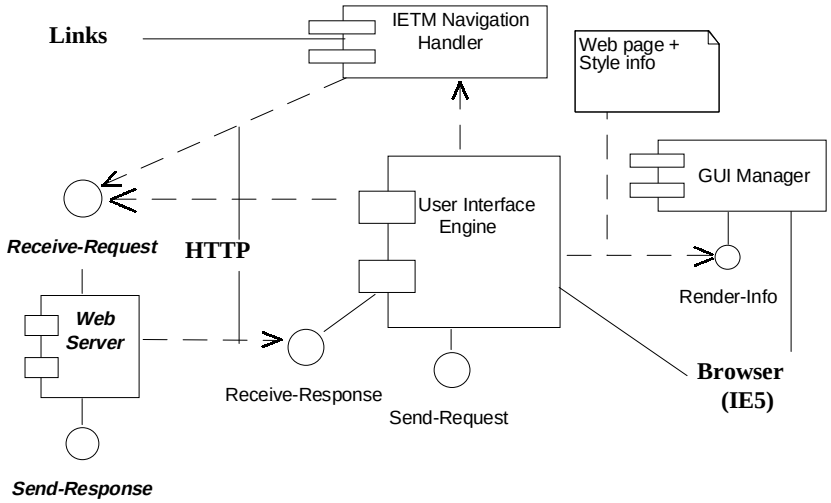

Figure 7.1 Demo System: User Interface Components

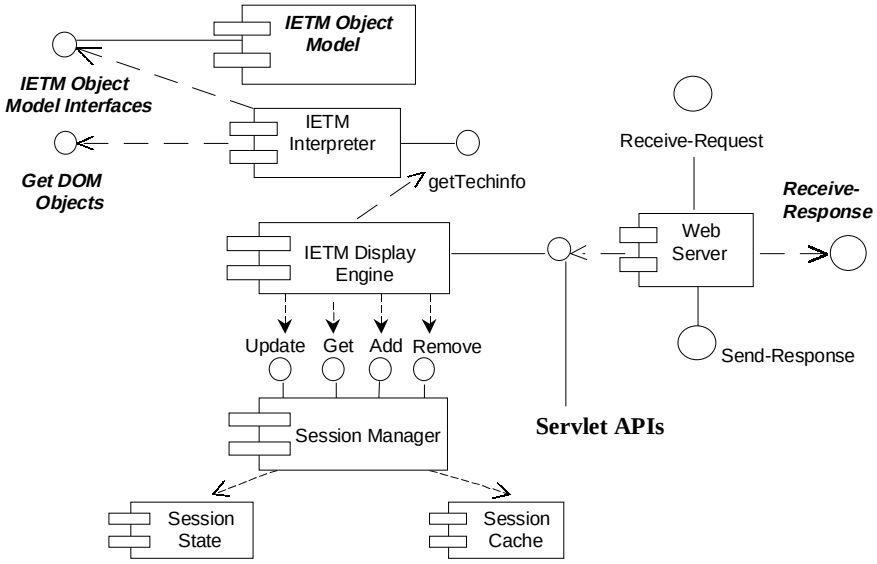

Figure 7.2 Demo System: IETM Display System Components

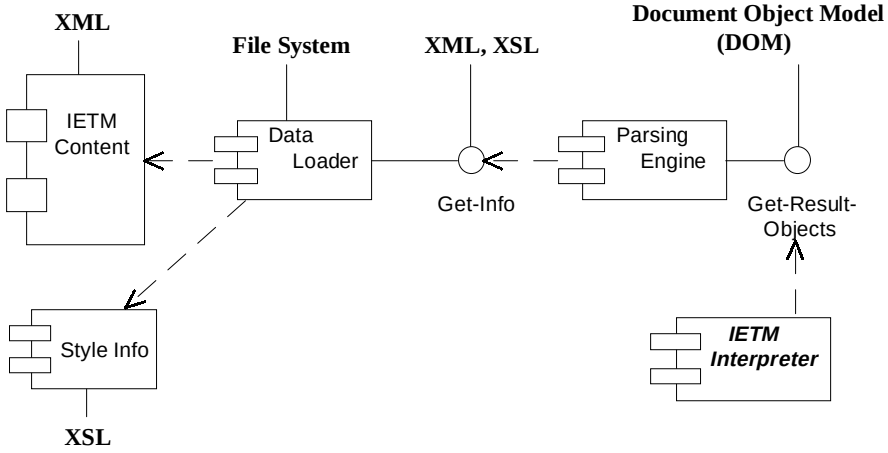

Figure 7.3 Demo System: Data Access Components

IETM Object Model Interfaces

Rational Rose, the UML modeling tool allows for code generation from the UML model in different languages. This feature is used to create the IETMOM interfaces in OMG Interface Definition Language (IDL) [24]. IDL is a precise, language-independent and implementation-neutral way to specify interfaces.

IETM Object Model Implementation

The IETM Object Model implementation is a set of Java packages directly mapping to the IETMOM specification. It provides the methods implementation for the interfaces defined by the model. It is worth mentioning that this implementation is by no means the only concrete implementation. Other implementations can be developed as long as they maintain support to the IETMOM interfaces.

Rational Rose is used again to generate the Java files representing the IETMOM. The Java language doesn't support all object-oriented concepts e.g., multiple inheritance and templates. Rational Rose doesn't provide a workaround; it simply disregards the class that uses any of these features. These classes are mapped manually to satisfy the Java language constraints while still conforming to the IETMOM specification.

Source File

The source IETM data is required to be in a well formed XML format. To turn the original manual written in SGML into a well formed XML document, we just include a closing tag for each opened one and change its extension to XML. This modification doesn't affect the document structure nor contradicts the IETMDB specification; those are basic requirements for the system.

XSL Style Sheet

Based on the IETM user interface specifications, we develop a stylesheet to render the static information in the XML source document. The XSL file resides on the server and is sent together with the XML output from the servlet to the user's browser.

XML Parser

A DOM-compliant XML parser creates a set of DOM objects that represents the information stored in the XML document. The DOM to IETM interpreter interfaces to the parser to retrieve the `Document` object, which represents the root of the document tree of objects. The following is the method provided by the IBM's parser to instantiate a DOM object.

```
Document readStream(InputStream istream)
```

DOM to IETM Interpreter

The DOM to IETM interpreter provides the following method to return the IETM `techinfo` object, which is the root of the IETM tree of objects.

```
techinfo getTechinfo(String fname)
```

The following methods are used internally by the interpreter; `traverseDOMBranch` is used to traverse the DOM tree and `createIETM` is called repeatedly during the DOM traversal to create the appropriate IETM object. The interpreter contains a set of functions `createXXX` that the method `createIETM` uses to create a particular IETM object type e.g. `task`, `step`.

```
void traverseDOMBranch(Node node)
```

```
void createIETM(Node node)
```

IETM Display Engine

The IETM display engine provides methods to handle each IETM object based on its generic layer template (node type) e.g. `if-node`.

```
architecturalForm handleIfNode(IF_NODE ifnode)

architecturalForm handleNodeSeq(NODE_SEQ nodeSeq)
```

The `doGet` method is part of the servlet API and it is called by the Web server when a user sends a request to the servlet. This method handles the user request and calls the appropriate methods to retrieve the required IETM information and to send the reply.

```
void doGet (HttpServletRequest request, HttpServletResponse response)
```

The methods `shouldPresent` and `recordEvents` are responsible respectively for evaluating the preconditions of an object before displaying it and for setting an object's postconditions and assertions after being displayed.

```
boolean shouldPresent(NODE node)

void recordEvents(NODE node, HttpServletRequest request)
```

`RollBack` method is called when the user hits the `back` button. It is used to reset the variables affected by displaying the last piece of information to their original values.

```
void rollBack(NODE node)
```

The information presented to the user at a time is made up from a set of objects. An example is the task's input information that includes reference materials objects and parts objects. The method `displayXXX` goes over the object's contents, evaluates their preconditions and creates a list of the objects valid for display.

```
void displayXXX()
```

The method `formatOutput` creates the reply in XML format compliant with the IETMDB specification from the list of objects selected for presentation and sends it to the user.

```
void formatOutput(HttpServletResponse response, String data, short output)
```

Session Management

Servlet engines provide a session tracking mechanism. A session Id is sent to the servlet with every user's request. The session component provides interfaces to get, put, update and remove object-value pairs. These interfaces are used to store and retrieve both state variables and caching information.

IETM Navigation Handler

For the sake of demonstration, navigation functions and their handler are not implemented as software components rather as buttons with URL links to the servlet IETM engine.

7.1.5 System Scenario

The UML scenario diagram in Figure 7.1, illustrates interactions among the various system components in a basic task performance operation.

The user starts by sending his information to log in to the server site and access a particular IETM. The IETM display engine receives the user request and calls the interpreter to convert the required document into IETM objects (`techinfo`). The engine then presents the user with a list of the IETM document contents in the form of links. In this case study for example, the user is presented with the list of systems and related tasks covered by the manual.

The user then selects the task he needs to perform. The IETM display engine fetches the inputs of the selected task, select the objects to present by evaluating their preconditions and sends the reply back to the user. To evaluate preconditions the engine needs to retrieve relevant state values from the session object.

When the user hits next, the display engine records the presentation events by doing two main things: retrieve input from the user's request and set postconditions and assertions that are identified in the objects presented. For example, the engine sets a particular variable managed by the session object to a certain value based on the user's selection for the presented choice.

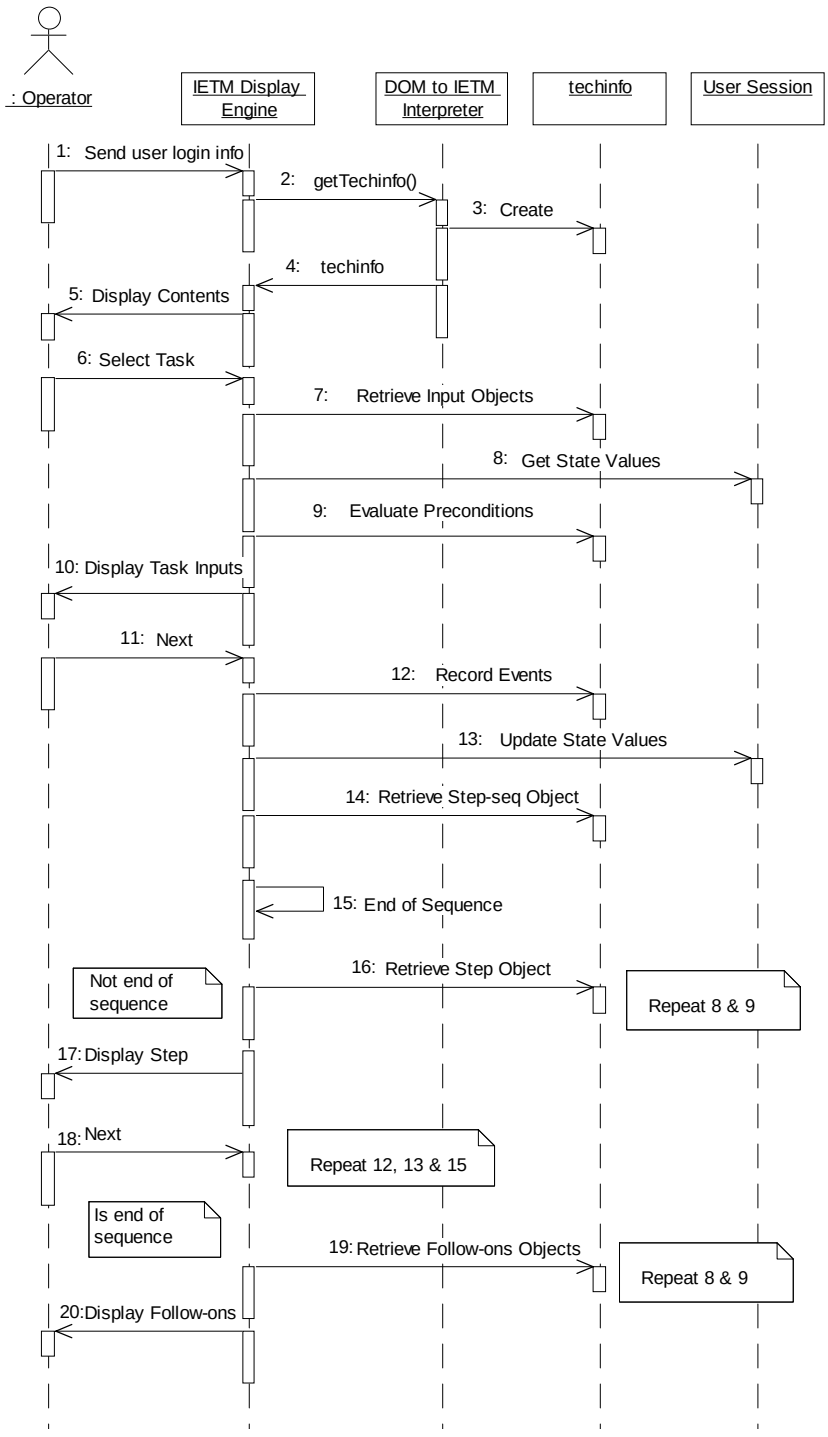

Figure 7.1 Main Flow System Scenario

The next thing to present to the user is one step at a time from the task's sequence of steps. The engine retrieves the step sequence and checks if all steps have been presented, if no the engine evaluates the preconditions of the next steps and replies with the first one valid for presentation. When the user hits next again, the engine records the presentation events, then checks the step sequence end and loops through the previous steps until it reaches the end of the step sequence. After all steps are presented, the engine fetches the task's follow-ons objects, evaluates their preconditions and sends the valid ones to the user.

7.1.6 Distributed Objects using CORBA

Distributed objects are packaged as independent pieces of code that can be accessed by remote clients via method invocations. Distributed objects technology allows components to interoperate across networks, run on different platforms, roam on networks and coexist with old applications through object wrappers.

To harvest information from a variety of IETM sources residing on a different hardware and software system, there must be some common ground between all of these sources. Common Object Request Broker Architecture (CORBA) ties disparate systems together based on the interfaces that certain remote objects implement.

CORBA is used to develop a small proof-of-concept system. In this system, the IETM display engine runs on the Web server and interacts with users while the IETM data as well as the interpreter are on another remote server. The interface between the IETM display engine and the interpreter is based on the IETMOM however in this case the method invocations and responses run on top of the CORBA Object Request Broker (ORB) middleware.

7.2 Database driven IETM Display Systems

Though we didn't develop a database driven demonstration system, we summarize here issues related to the development of such a system.

Many of the existing IETM implementations use databases to store, manage and allow efficient retrieval of IETM documents. We envision that almost all new IETM systems will also be database-driven due to the robustness, efficiency and ease of management of databases.

In the proposed architecture, the eXtensible Markup Language (XML) is the standard language for data transport. The following characteristics make XML documents suitable for data interchange and storage as well as document presentation:

- XML markup is more complete and disciplined than HTML markup.
- XML supports the definition of application-oriented markup tags.
- XML separates formatting and presentation specifications from document content.

This leads to a requirement to store XML documents, or the data that they contain, in databases, and to reference and update elements of such documents. Databases and XML offer complementary functionality for storing data. Databases store data for efficient retrieval, whereas XML offers an easy information exchange that enables interoperability between applications.

In the IETM domain, IETM data can be stored in a database (relational, object-oriented, or hierarchical) that supports XML (most of the recent ones) or in a database that doesn't support XML (mostly old systems). The proposed architecture supports both types, however non-XML databases will require additional development effort.

7.2.1 XML support in databases

XML databases support XML documents storage in one of the following three basic ways:

- *XML data storage:* Data elements are extracted from the XML document and stored as data rows and columns in the database. It allows for fast retrieval yet has the overhead of assembling and disassembling the XML documents for interchange.

- *XML document storage:* The XML document is stored entirely in a single column in the database. It is slower than data storage. However, it eliminates the need for assembling and disassembling the data for interchange.
- *Hybrid storage:* XML document is stored in a single column in the database and some of its data elements are extracted into separate columns for faster and more convenient access. This approach balances the advantages of data storage and document storage, but suffers from the cost and complexity of redundant storage of the extracted data.

The number of products for using XML with databases is growing with amazing speed. These products are divided into the following categories:

- *Database and XML Middleware*: They are used to transfer data between XML documents and databases. It is written in a variety of languages, but almost all of it uses ODBC, JDBC, or OLE DB.
- *XML-enabled database*: They are databases with extensions for transferring data between XML documents and themselves.
- *Content management systems*: Content management systems are systems for storing, retrieving, and assembling XML documents from document fragments (content). They generally include such features as editors, version control, and multi-user access. Although they use a database, this is generally hidden from the user.
- *Persistent DOM implementations*: Persistent DOM implementations are DOM implementations that use a database for speed and to avoid the memory problems that occur when using the DOM with large documents. Like content management systems, persistent DOM implementations can be used to store and retrieve XML documents, as well as create new documents from fragments of existing documents. Unlike content management systems, they do not include such capabilities as version control and editing. However, they can be used programmatically to build DOM applications.

7.2.2 Use of Databases in The Proposed Architecture

Several techniques can be used to fit any database in the proposed architecture.

IETMOM Database Driver

Provide a database dependent implementation of the IETM Object Model (which we can call IETMOM database driver), which is used like an ODBC driver that encapsulates the particular data structure of the database. The implementation of the IETMOM APIs will use query statements to handle the data. These queries can be in the Structured Query Language (SQL) or in a proprietary format as long as the driver exposes the IETMOM interfaces to the display engine.

Any Database as a Virtual XML Document

In JavaWorld magazine, an article titled XML APIs for databases [15] describes a way to implement the Simple API for XML (SAX) and DOM APIs directly over a database, enabling the DOM to IETM interpreter to treat databases as if they were XML documents. That way, we can obviate the need of converting an existing database to an XML document. The implementation uses Java and works with any database with a JDBC driver. For databases that don't support JDBC, a similar implementation can be developed using ODBC.

Database and XML Middleware

A commercial middleware can be used to sit on top of the existing ODBC, JDBC or OLE DB compliant database. The middleware transfers data from the database to an XML document or its equivalent DOM objects.

XML-enabled Databases

XML-enabled databases and content management systems support the retrieval of the stored XML document whether they use data storage or document storage approach. The resulting XML document can then be fed to the DOM-compliant parser.

Moreover, most content management systems and XML-enabled databases support the retrieval of an XML document in the form of DOM objects. From this point, the database or content management can be integrated with the rest of the architecture through the DOM to IETM interpreter component.

CHAPTER 8: CONCLUSIONS AND FUTURE WORK

8.1 Lessons Learned

Based on the experience gained developing a prototype, we learned the following lessons, which should be useful as an input in the design process of a Web-enabled IETM display system.

8.1.1 The Choice of The Implementation Language

The use of the Java language is a good choice, It provides a rich source of reusable components e.g. the Java session component is used as the session manager in our developed system. Java allows easy and fast integration between components since most Web COTS components are either written in Java or support Java. In addition, Java is a robust object-oriented language that supports error handling, threads and automatic garbage collection. Moreover, in a distributed system environment, Java and CORBA complement each other, Java deals with implementation transparency while CORBA deals with network transparency.

8.1.2 Debugging in a COTS-based Architecture

While testing your code outside of the IDE, logs from different components become important. The Web server as well as the servlet engine used provides logs that help us identify the source of the problem.

8.1.3 XSL Processors

Microsoft Internet Explorer 5 supports a subset of the transformation part of the Extensible Stylesheet Language (XSL) based on the W3C December 18th 1998 Working Draft. Microsoft Web site [9] provides information on unimplemented features, workarounds, extensions and known conformance bugs. A better approach for rendering XML documents is to use a generic XSL processor to transform XML documents to HTML.

No software yet implements all of the latest XSL recommendation. All products available now implement different subsets of the different draft versions of the specification. Furthermore, many products, including Internet Explorer 5 add elements not actually present in the current specification for XSL. Consequently, very few examples will work exactly the same way in different XSL processors. Eventually, of course, this should be straightened out as the standard evolves toward its final incarnation, as vendors fix the bugs in their products and implement the unimplemented parts, and as more Web browsers that support XSL become commercially available.

8.1.4 Code Generation

From our experience with the code generation in the case study, we find that code generation is not a straightforward process. Some modeling tools like Rational Rose lets you set up a number of options for the generated code so you don't have to go through all the generated files to fix them the way you want. However, Rational Rose doesn't provide a solution when there is no direct mapping between the UML model and the implementation language e.g. Java doesn't support multiple inheritance and templates. As a result, we had to do this manually.

8.2 Conclusions

Current IETM applications suffer from the tight coupling between the client's viewer and the data source. User interface controls, business logic and database statements are all interwoven throughout the application source code. Consequently, IETM applications are very much dependent on the source data and hence they are difficult to maintain. The reason behind this interoperability problem is the complexity and dynamic behavior of the IETMDB specification.

This thesis addresses interoperability issues in handling source data for IETM applications. Addressing the issues associated with interoperability will improve the reusability and universal accessibility of information sources.

We developed an IETM Object Model, which represents the IETMDB specification and fully encapsulates the behavior that is expected from the IETM documents. Encapsulation is what allows a client application to interact with an object without knowledge of its implementation details, a primary premise for loose coupling.

The IETM Object Model described in chapter 5 solves the IETMDB specification problems. The IETM Object Model (IETMOM):

- Encapsulates the logic and behavior of IETMs and defines interfaces for IETM data manipulation.
- Can be implemented in various programming languages and in a variety of configurations/architectures.
- Can be implemented by existing IETM programs to Web-enable them with minimum effort.

The IETMOM addresses the interoperability issues in handling the IETM source data. The IETMOM:

- Isolates the implementation of the IETM content representation and manipulation from the IETM display system, thus achieving interoperability among IETM content providers.
- Improves compatibility and reuse of IETM presentation software
- Can be reused to access and manipulate IETM documents not only for display but also for applications throughout an IETM life cycle such as, authoring, testing and configuration management.

In this thesis, we also propose a component-based IETM system architecture that is built from components with well-defined interfaces. The component-based architecture establishes the conceptual foundation for assembling our system out of Web COTS components and allows us to explore multiple solutions effectively. We also illustrate the use of UML to model complex IETM systems suitable for complex military and commercial implementations.

Using UML to model the IETM Object Model and the display system architecture and behavior, we are able to address the IETMDB specification complexity. UML enables us to:

- Build expressive models for visualizing, specifying, and documenting the proposed IETM system.
- Model the static structure as well as the dynamic behavior of the system.
- Manage system complexity.
- Create an IETMDB model independent of the implementation language.
- Forward engineer the system components.
- Free IETM system developers from having an SGML knowledge.

In this thesis, we also describe a proof-of-concept implementation that uses data from an F16 aircraft technical manual. The demonstration system proves:

- The success of the IETM Object Model to provide an interoperable alternative to the IETMDB specification and to encapsulate its behavior.
- That differing IETM data could be efficiently delivered across the Web using the IETMOM APIs.
- The ability to build and integrate an efficient Web-based IETM display system from reusable components.
- The competence of the proposed architecture to support distributed objects technology.

Due to time limitations, we didn't not implement a database version of the prototype. However, we discuss design approaches and issues related to such an implementation.

8.3 Future Work

8.3.1 Analyze The Proposed Architecture

We need to evaluate the quality of the proposed web-enabled architecture. Quality can be measured in terms of performance, scalability, change propagation and management, complexity, etc.

8.3.2 Explore Alternative Technologies

Based on the required quality attributes, we can assess the capabilities of the various system's components and explore and use different technology alternatives such as Enterprise Java Beans (EJB).

8.3.3 Integrated Authoring Environment for IETMs

IETM authoring and maintenance is a long and costly process that involves a lot of SGML and military standards expertise. IETM information update depends highly on how modular the design of the IETM authoring systems is. The IETM Object Model can be extended into a more generic and integrated one that accounts more for reuse of product data. Using the current IETM UML model, we can investigate the design of IETM Authoring systems to minimize the effort and cost of integrating IETM parts and updating IETM data.

8.3.4 Detecting Inconsistencies in IETMs

The IETM Object Model can be used as a standardized interface in applications capable of detecting inconsistency and incompleteness of preconditions and postconditions in the delivered IETMs.

8.3.5 Client to IETM Display Engine Interface

Query strings are the technique HTML uses to send information from the browser to the server. HTML documents can declare a form that contains one or more items that a user must

supply. For each item, the form specifies an item name. When the user presses the `submit` button in a form, the browser appends a "?" to the form action (a URL defined in the form), followed by the names and contents of the fields in the form, respectively, and then sends it to the server.

The developed demonstration system uses forms and query strings to support user interactions between the Web browser and the IETM display engine. Although the variable-value pairs used are intuitive, they still create a dependency between the way we render XML information (represented in the XSL stylesheet) and the IETM display engine servlet. To solve this problem, standard interfaces for parameter passing need to be defined between the user and the display engine. In case of query strings usage, form items names, values and order need to be specified. When the IETM navigation handler component is implemented as an applet, a set of APIs should be defined for the interactions between the applet and the display engine. CORBA is a suitable technology to use in this case.

8.3.6 Use Abstract Factory Pattern

The IETMOM API does not define a standard way to create IETMOM Implementation or objects; actual implementations must provide some proprietary way for creating these IETMOM objects. IETM objects can not be created directly using constructors without being dependent on the IETM Object Model implementation classes. The IETM interpreter and the IETM display engine developed depend on the implementation we provide for the IETMOM. Object-oriented design provides a solution to this problem through the abstract factory pattern.

BIBLIOGRAPHY

1. Naval Surface Warfare Center, Carderock Division. *Data Base, Revisable - Interactive Electronic Technical Manuals, For the Support Of. MIL-PRF-87269A*. <http://navysgml.dt.navy.mil/ietm/ietm>.
2. Naval Surface Warfare Center, Carderock Division. *Manuals, Interactive Electronic Technical – General Content, Style, Format, And User-Interaction Requirements. MIL-PRF-87268A*. <http://navysgml.dt.navy.mil/ietm/ietm>.
3. International Organization for Standardization. *Information Processing - Text and Office Systems - Standard Generalized Markup Language (SGML). ISO 8879:1986*. <http://www.oasis-open.org/cover/general.html>.
4. Eric L. Jorgensen. *Proposed Web-Based Joint IETM Architecture (JIA) for the Interoperability of DoD IETMs*. August 1998 <http://navysgml.dt.navy.mil/ietm/ietm>.
5. ManTech Advanced Systems International Inc. *Internet MID Demonstration*. <http://www2.dcnicn.com/DefaultDocs/cals_ide/task05/default1>.
6. Booch, G., Rumbaugh, J., Jacobson, I.: The Unified Modeling Language User Guide. Addison-Wesley, 1999.
7. Ann M Wrightson: Safety in IETP. IEE Aerospace group symposium on Certification of Ground-Air Systems, London, February 1998.
8. International Organization for Standardization. *Information Technology - Hypermedia/Time-based Structuring Language (HyTime) ISO/IEC 10744:1992(E)*. <http://www.oasis-open.org/cover/hytime.html>.
9. Microsoft Corporation. *XSL Working Draft Conformance Notes*. <http://msdn.microsoft.com/xml/xslguide/conformance.asp>.
10. W3C (World Wide Web Consortium). *Extensible Markup Language (XML)*. <http://www.w3.org/XML/>.
11. W3C (World Wide Web Consortium). *HTML 4.0 Specification*. <http://www.w3.org/TR/REC-html40>.
12. OMG (Object Management Group). *The Common Object Request Broker: Architecture and Specification*. <http://www.omg.org/corba/corbiiop.htm>.
13. Sun Microsystems Inc. *The Java Language Specification*. <http://java.sun.com/docs/books/jls/>.
14. ECMA (European Computer Manufacturers Association). *ECMAScript Language Specification*. <http://www.ecma.ch/stand/ECMA-262.htm>.

15. Ramnivas Laddad. *XML APIs for databases*. January 2000. <http://www.javaworld.com/javaworld/jw-01-2000/jw-01-dbxml.html>.

16. Nazmul Idris: *Developer Life, Parser Testing*. <http://www.developerlife.com/parsertesting.html>.

17. Shlaer, S., and S. Mellor: Object-Oriented Systems Analysis: Modeling the World in Data, Englewood Cliffs, NJ, Yourdon Press, 1988

18. Coad, P. and E. Yourdon: Object Oriented Analysis, Englewood Cliffs, NJ Prentice Hall 1990

19. Wirfs-Brock, R., B. Wilkerson, and L. Wiener: Designing Object Oriented Software, Englewood Cliffs, NJ Prentice Hall 1990

20. Rubin, K., and A. Goldberg: Object Behavior Analysis, *Communications of the ACM* 35(9):48-62, September 1992

21. Jacobson, I., M. Christerson, P. Jonsson, and G. Overgaard, Object Oriented Software Engineering, Workingham, England: Addison-Wesley 1992

22. Booch, G.: Object Oriented Analysis and Design with Applications, The Benjamin/Cummings Publishing Company, Inc, 1994

23. Robert Orfali, Dan Harkey and Jeri Edwards: Client/Server Survival Guide, third edition. John Wiley & Sons, Inc. 1999.

24. Object Management Group. *Interface Definition Language (IDL)*. <http://www.omg.org/>.

25. Naval Air Warfare Center. *Using MID for Interactive Electronic Technical Manuals*. <http://www.nawcsti.navy.mil/mid/ietms.html>.

26. Eric L. Jorgensen. *DoD Classes of Electronic Technical Manuals*. April 1994. <http://navysgml.dt.navy.mil/ietm/ietm.html>.

APPENDIX A GLOSSARY

Child. A child is an immediate descendant node of a node

Content model. The content model is a simple grammar governing the allowed types of the child elements and the order in which they appear

CORBA. The Common Object Request Broker Architecture from the OMG . This architecture is a collection of objects and libraries that allow the creation of applications containing objects that make and receive requests and responses in a distributed environment.

Data model. A data model is a collection of descriptions of data structures and their contained fields, together with the operations or functions that manipulate them.

DTD. Document Type Definition

DOM. Document Object Model

ECMAScript. The programming language defined by the ECMA-262 standard. As stated in the standard, the originating technology for ECMAScript was JavaScript.

Element. Each document contains one or more elements, the boundaries of which are delimited either by start-tags and end-tags or, for empty elements by an empty-element tag. Each element has a type, identified by name, and may have a set of attributes. Each attribute has a name and a value.

HTML. The HyperText Markup Language (HTML) is a simple markup language used to create hypertext documents that are portable from one platform to another. HTML documents are SGML documents with generic semantics that are appropriate for representing information from a wide range of applications.

HyTime. Hypermedia/Time-based Structuring Language

IDL. An Interface Definition Language (IDL) is used to define the interfaces for accessing and operating upon objects. Examples of IDLs are the Object Management Group's IDL, Microsoft's IDL, and Sun's Java IDL.

Interface. An interface is a declaration of a set of methods with no information given about their implementation. In object systems that support interfaces and inheritance, interfaces can usually inherit from one another.

JDBC. JDBC (Java Database Connectivity) is an application program interface (API) specification for connecting programs written in Java to the data in popular databases.

Language binding. A programming language binding for an IDL specification is an implementation of the interfaces in the specification for the given language. For example, a Java language binding for the Document Object Model IDL specification would implement the concrete Java interfaces that provide the functionality exposed by the interfaces.

Method. A method is an operation or function that is associated with an object and is allowed to manipulate the object's data.

Model. A model is the actual data representation for the information at hand. Examples are the structural model and the style model representing the parse structure and the style information associated with a document. The model might be a tree, or a directed graph, or something else.

Object model. An object model is a collection of descriptions of classes or interfaces, together with their member data, member functions, and class-static operations.

ODBC. Open Database Connectivity (ODBC) is a standard or open application programming interface (API) for accessing a database.

Plugins Plug-ins are software programs that extend the capabilities of Web browsers in a specific way giving the user, for example, the ability to play audio samples or view video movies from within a Web browser

SGML. Standard Generalized Markup Language

UML. Unified Modeling Language

XML. Extensible Markup Language (XML) is an extremely simple dialect of SGML which is completely described in this document. The goal is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML.

XSL eXtensible Stylesheet Language

APPENDIX B CASE STUDY

The following is a part of the F16 IETM manual use as a case study. This part represents the document root element as well as the first task in the first system defined in the manual.

```
<techinfo name='Aircraft Technical Manual' id='november23'>
<system id='i110102' name='Air Data System' itemid='3411'>
<task id='i147144' name='CENTRAL AIR DATA COMPUTER, 3411A1, REMOVAL (34-11-01).' type='3' itemid='34-11-01A' esttime='3'>
<input id = 'i53146'>
<reqcond id = 'i34148'>
<text id = 'i7953110'>Aircraft safe for maintenance</text>
<expression id = 'i1070177'>
<expression id = 'i1071177'>
<property id = 'i590176' type = 'dynamic' value-type = 'boolean'>AIRCRAFT_SAFE_FOR_MAINTENANCE_103001</property>
</expression>
<eq></eq>
<expression id = 'i1072177'>
<boolean>FALSE</boolean>
</expression>
</expression>
<task id='i148144' name='MAKE AIRCRAFT SAFE FOR MAINTENANCE MENU (10-30-01)' type='0' itemid='10-30-01M' esttime='1' ref='i83144'>
</task>
</reqcond>
<reqcond id = 'i35148' type='do'>
<text id = 'i7954110'>Access door 1202 open</text>
<expression id = 'i1074177'>
<expression id = 'i1075177'>
<property id = 'i592176' type = 'dynamic' value-type = 'string'>DOOR_1202</property>
```

```
</expression>
<eq></eq>
<expression id = 'i1076177'>
<string>CLOSED</string>
</expression>
</expression>
<task id='i149144'  name='OPEN DOOR 1202' type='2' itemid='1202A' esttime='7' ref='i5957144'>
</task>
<assertion id = 'i351182'>
<property id = 'i593176' type = 'dynamic' value-type = 'string'>DOOR_1202</property>
<expression id = 'i1077177'>
<string>OPEN</string>
</expression>
</assertion>
</reqcond>
<person id = 'i64150' quantity = '1'></person>
<equip id = 'i20152' quantity='1'>
<text id = 'i7955110'>Maintenance Platform, Type C-1 or equivalent</text>
</equip>
</input>
<step-seq id = 'i85160'>
<step id = 'i865158'>
<precond id = 'i227178' type='context'>
<expression id = 'i1078177'>
<expression id = 'i1079177'>
<property id = 'i594176' type = 'dynamic' value-type = 'string'>TRACK</property>
</expression>
<eq></eq>
```

```
<expression id = 'i1080177'>
<string>NOVICE</string>
</expression>
</expression>
</precond>
<alert id = 'i148156' type='n'>
<text id = 'i7956110'>Retain all serviceable parts for reinstallation.</text>
</alert>
<alert id = 'i149156' type='n'>
<text id = 'i7957110'>Install protective devices on all open hoses, ports or unions, and electrical disconnects.</text>
</alert>
<text id = 'i7958110'>Disconnect three electrical connectors.</text>
<postcond id = 'i60179'>
<assertion id = 'i352182'>
<property id = 'i595176' type = 'dynamic' value-type = 'string'>CONNECTOR_3411P101</property>
<expression id = 'i1081177'>
<string>DISCONNECTED</string>
</expression>
</assertion>
</postcond>
<postcond id = 'i61179'>
<assertion id = 'i353182'>
<property id = 'i596176' type = 'dynamic' value-type = 'string'>CONNECTOR_3411P102</property>
<expression id = 'i1082177'>
<string>DISCONNECTED</string>
</expression>
</assertion>
</postcond>
```

```
<postcond id = 'i62179'>
<assertion id = 'i354182'>
<property id = 'i597176' type = 'dynamic' value-type = 'string'>CONNECTOR_3411P105</property>
<expression id = 'i1083177'>
<string>DISCONNECTED</string>
</expression>
</assertion>
</postcond>
</step>
<step id = 'i866158'>
<precond id = 'i228178' type='context'>
<expression id = 'i1084177'>
<expression id = 'i1085177'>
<property id = 'i598176' type = 'dynamic' value-type = 'string'>TRACK</property>
</expression>
<eq></eq>
<expression id = 'i1086177'>
<string>NOVICE</string>
</expression>
</expression>
</precond>
<text id = 'i7959110'>Disconnect two hoses.</text>
<postcond id = 'i63179'>
<assertion id = 'i355182'>
<property id = 'i599176' type = 'dynamic' value-type = 'string'>PS1_HOSE</property>
<expression id = 'i1087177'>
<string>DISCONNECTED</string>
</expression>
</assertion>
```

```
</postcond>
<postcond id = 'i64179'>
<assertion id = 'i356182'>
<property id = 'i600176' type = 'dynamic' value-type = 'string'>PT1_HOSE</property>
<expression id = 'i1088177'>
<string>DISCONNECTED</string>
</expression>
</assertion>
</postcond>
</step>
<step id = 'i867158'>
<precond id = 'i229178' type='context'>
<expression id = 'i1089177'>
<expression id = 'i1090177'>
<property id = 'i601176' type = 'dynamic' value-type = 'string'>TRACK</property>
</expression>
<eq></eq>
<expression id = 'i1091177'>
<string>NOVICE</string>
</expression>
</expression>
</precond>
<dialog id = 'i69123' >
<menu id = 'i69126' select = 'single'>
<prompt id = 'i69127'>
<text id = 'i7960110'>Will the same CADC be reinstalled or does the replacement CADC already have unions installed?</text>
</prompt>
<choice id = 'i158128' default='yes'>
<text id = 'i7961110'>YES</text>
```

```
<assertion id = 'i357182'>
<property id = 'i602176' type = 'dynamic' value-type = 'boolean'>UNIONS_INSTALLED</property>
<expression id = 'i1092177'>
<boolean>TRUE</boolean>
</expression>
</assertion>
</choice>
<choice id = 'i159128' default='no'>
<text id = 'i7962110'>NO</text>
<assertion id = 'i358182'>
<property id = 'i603176' type = 'dynamic' value-type = 'boolean'>UNIONS_INSTALLED</property>
<expression id = 'i1093177'>
<boolean>FALSE</boolean>
</expression>
</assertion>
</choice>
</menu>
</dialog>
</step>
<step id = 'i868158'>
<precond id = 'i230178' type='context'>
<expression id = 'i1094177'>
<expression id = 'i1095177'>
<property id = 'i604176' type = 'dynamic' value-type = 'string'>TRACK</property>
</expression>
<eq></eq>
<expression id = 'i1096177'>
<string>NOVICE</string>
</expression>
```

```
</expression>
</precond>
<text id = 'i7963110'>Remove two unions and discard two packings.</text>
</step>
<step id = 'i869158'>
<precond id = 'i232178' type='context'>
<expression id = 'i1100177'>
<expression id = 'i1101177'>
<property id = 'i606176' type = 'dynamic' value-type = 'string'>TRACK</property>
</expression>
<eq></eq>
<expression id = 'i1102177'>
<string>NOVICE</string>
</expression>
</expression>
</precond>
<text id = 'i7964110'>Disengage two latch nuts and two latches.</text>
</step>
<step id = 'i870158'>
<precond id = 'i233178' type='context'>
<expression id = 'i1103177'>
<expression id = 'i1104177'>
<property id = 'i607176' type = 'dynamic' value-type = 'string'>TRACK</property>
</expression>
<eq></eq>
<expression id = 'i1105177'>
<string>NOVICE</string>
</expression>
```

```
</expression>
</precond>
<text id = 'i7965110'>Slide CADC Novice out of mount.</text>
<postcond id = 'i65179'>
<assertion id = 'i359182'>
<property id = 'i608176' type = 'dynamic' value-type = 'string'>CADC_3411A1</property>
<expression id = 'i1106177'>
<string>REMOVED</string>
</expression>
</assertion>
</postcond>
</step>
<step id = 'i871158'>
<precond id = 'i234178' type='context'>
<expression id = 'i1107177'>
<expression id = 'i1108177'>
<property id = 'i609176' type = 'dynamic' value-type = 'string'>TRACK</property>
</expression>
<eq></eq>
<expression id = 'i1109177'>
<string>EXPERT</string>
</expression>
</expression>
</precond>
<text id = 'i7966110'>Slide CADC out of mount.</text>
<postcond id = 'i66179'>
<assertion id = 'i360182'>
<property id = 'i610176' type = 'dynamic' value-type = 'string'>PS1_HOSE</property>
<expression id = 'i1110177'>
```

<string>DISCONNECTED</string>

</expression>

</assertion>

</postcond>

<postcond id = 'i67179'>

<assertion id = 'i361182'>

<property id = 'i611176' type = 'dynamic' value-type = 'string'>PT1_HOSE</property>

<expression id = 'i1111177'>

<string>DISCONNECTED</string>

</expression>

</assertion>

</postcond>

<postcond id = 'i68179'>

<assertion id = 'i362182'>

<property id = 'i612176' type = 'dynamic' value-type = 'string'>CONNECTOR_3411P101</property>

<expression id = 'i1112177'>

<string>DISCONNECTED</string>

</expression>

</assertion>

</postcond>

<postcond id = 'i69179'>

<assertion id = 'i363182'>

<property id = 'i613176' type = 'dynamic' value-type = 'string'>CONNECTOR_3411P102</property>

<expression id = 'i1113177'>

<string>DISCONNECTED</string>

</expression>

</assertion>

</postcond>

<postcond id = 'i70179'>

<assertion id = 'i364182'>

<property id = 'i614176' type = 'dynamic' value-type = 'string'>CONNECTOR_3411P105</property>

<expression id = 'i1114177'>

<string>DISCONNECTED</string>

</expression>

</assertion>

</postcond>

<postcond id = 'i71179'>

<assertion id = 'i365182'>

<property id = 'i615176' type = 'dynamic' value-type = 'string'>CADC_3411A1</property>

<expression id = 'i1115177'>

<string>REMOVED</string>

</expression>

</assertion>

</postcond>

<postcond id = 'i4486179'>

<assertion id = 'i17622182'>

<property id = 'i39104176' type = 'dynamic' value-type = 'boolean'>PITOT_STATIC_PROBE_PNEU_LEAK_OP_CHKOUT_REQD</property>

<expression id = 'i79890177'>

<boolean>TRUE</boolean>

</expression>

</assertion>

</postcond>

<postcond id = 'i4487179'>

<assertion id = 'i17623182'>

<property id = 'i39105176' type = 'dynamic' value-type = 'boolean'>DFLCS_BIT_REQD_270010</property>

<expression id = 'i79891177'>

<boolean>TRUE</boolean>

```
</expression>
</assertion>
</postcond>
</step>
</step-seq>
<follow-on id = 'i21163' type='mi'>
<text id = 'i7968110'>Install CENTRAL AIR DATA COMPUTER, 3411A1.</text>
<expression id = 'i1120177'>
<expression id = 'i1121177'>
<property id = 'i618176' type = 'dynamic' value-type = 'string'>CADC_3411A1</property>
</expression>
<eq></eq>
<expression id = 'i1122177'>
<string>REMOVED</string>
</expression>
</expression>
<task id='i151144' name='CENTRAL AIR DATA COMPUTER, 3411A1, INSTALLATION (34-11-01).' type='5' itemid='34-11-01B' esttime='5' ref='i152144'>
</task>
<assertion id = 'i367182'>
<property id = 'i619176' type = 'dynamic' value-type = 'string'>CADC_3411A1</property>
<expression id = 'i1123177'>
<string>INSTALLED</string>
</expression>
</assertion>
</follow-on>
</task>
```

DINA GHOBASHY

OBJECTIVE

An opportunity in the computer engineering and software field where I can make effective use of my experience and studies while contributing to my work environment.

EDUCATION

Spring 1999 – Present West Virginia University WV, USA
- Master of science in Computer Engineering (MSE) student, GPA 4.0

1995 – 1996 Information Technology Institute Cairo, Egypt
- Network engineering certificate, distinction degree

1990 - 1995 Cairo University, Faculty of Engineering Cairo, Egypt
- B.Sc. Communications and Electronics Engineering, Computer department, GPA: 3.24

1990 College du Sacre-Coeur Cairo, Egypt
- Egyptian High School certificate, GPA: 3.74

WORK EXPERIENCE

1996 - 1998 Egyptian Banks Co. for Technological Advancement
System Analyst
- Egyptian Banks Co. provides technological sharing services for the banking sector. The essence of their work is setting up and maintaining a private network between the banks as well as developing and operating banking applications. Analysis, design, installations, testing, troubleshooting, training and help desk assistance were the main activities I was deeply involved in.

PROJECTS

1999 – 2000 WVU Term Projects

- E-commerce demonstration based on CORBA, Servlet and XML technologies
- Internet Banking project using Microsoft Access and Active Server Pages (ASPs)
- Discrete event simulation of a client server queuing network model in C++

1997 – 1998 Egyptian Banks Co.

- Worked on the implementation of an Electronic Check Clearing project. Accomplishments included system analysis and design; developing an interface for a document handler machine in C++ and setting up the TCP/IP dial-up private network using NT RAS and a CISCO access server.

1996 – 1997 Egyptian Banks Co.

- Worked on the installation, configuration and testing of an EFT (ATM/POS) switch using ACI Base24 package on a Tandem machine, another area of responsibility was the setup of the X.25 private network used as the infrastructure for this service.

1996 Information Technology Institute graduation project

- Development of an Internet traffic analyzer package based on the client-server model. A Unix server performs the sniffing and calculation functions and forwards the data to a GUI client for further statistics computation and analysis.

1995 Faculty of Engineering graduation project

- Development of a video-conferencing package over an IPX LAN.

SUMMER JOBS

1999 West Virginia University Morgantown, WV USA

- Graduate Teaching Assistant for "Introduction to Java concurrent programming" course.

1994 Schlumberger wireline and testing Assen, Netherlands

- Worked as a Field Engineer, Assisted on the maintenance, calibration and operation of electronic tools used for on-site testing.

COMPUTER SKILLS

Operating systems: Unix, Windows NT/95, DOS, Tandem's Guardian

Languages: UML, Java, C/C++, Visual Basic, SQL, IDL, HTML, XML, XSL, ASP, Socket and IPX programming, assembly

Packages: Inprise Visibroker, MS-Office, MS-Access, Rational Rose, Internet applications

Technology: Object Oriented Analysis & Design, CORBA

Network protocols: NetBEUI, TCP/IP, IPX/SPX, X.25

LANGUAGES

English: Fluent written, spoken and understood (TOEFL score: 637)

French: Fluent written, spoken and understood

Arabic: Mother tongue.

REFERENCES

Available upon request