2004-09

Operation and Maintenance Support Information (OMSI) creation, management, and repurposing with XML

Raymond, Scott P.

Monterey California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

OPERATION AND MAINTENANCE SUPPORT INFORMATION (OMSI) CREATION, MANAGEMENT, AND REPURPOSING WITH XML

by

Scott P. Raymond

September 2004

Co-Advisors: Daniel R. Dolk
Gordon H. Bradley

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** September 2004	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: Title (Mix case letters) Operation And Maintenance Support Information (OMSI) Creation, Management, And Repurposing With XML			**5. FUNDING NUMBERS**
6. AUTHOR(S) Raymond, Scott P.			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE**

13. ABSTRACT ***(maximum 200 words)***

New facility construction and existing facility renovation create new or modified operation and maintenance (O&M) requirements for the maintenance responsibility organization such as a Public Works Department (PWD). This O&M requirement is fully described by an Operation and Maintenance Support Information (OMSI) package. OMSI content includes facility, systems, and product information.

This thesis will address information integration, the process of allowing information systems to cross-communicate and share data. OMSI information integrated within the framework of a Computer-Aided Facility Management (CAFM) system allows for early identification of O&M requirements, an improved planning capability for new facilities, and more efficient economies of scale. In addition to PWD manpower savings, OMSI-CAFM integration will also allow a revolution in the way O&M requirements are planned and created. Preliminary OMSI information would be ideally created by the design A/E after having considered work force capability from both a workload and expertise perspective. While this may be impractical due to the changing nature of workforce capability and the lengthy planning and design cycle of military construction, OMSI-CAFM integration will certainly allow O&M planning to begin early in the OMSI development stages. OMSI submittals can be layered to provide preliminary planning information in the first submittal and add additional detailed information in later submittals. In such a manner, PWD O&M planners can begin an incremental planning effort early in the facility construction phase.

This thesis provides a non-proprietary, no-cost solution to OMSI-CAFM information integration that minimizes specialized knowledge on the part of the OMSI AE. This will allow a broad applicability of the solution to all OMSI developers, including those for smaller non-MILCON projects that aren't specifically funded for OMSI generation. An effective solution must also provide for easy and inexpensive repurposing of OMSI information for future (and as yet unknown) uses. The solution uses XML technologies (XML, XSD, XLS, XLST, XPath, XQuery, etc) and XML storage systems for the content creation, management, and repurposing of OMSI information.

14. SUBJECT TERMS Information Integration, Operation and Maintenance, O&M, Support Information, OMSI, building, facility, relational, XML, data models, transformation, repurposing, XSLT, XQuery				**15. NUMBER OF PAGES** 139
				16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL	

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

OPERATION AND MAINTENANCE SUPPORT INFORMATION (OMSI) CREATION, MANAGEMENT, AND REPURPOSING WITH XML

Scott P. Raymond
Lieutenant Commander, Unites States Navy
B.S., Rensselaer Polytechnic Institute, 1990

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN INFORMATION TECHNOLOGY MANAGEMENT

from the

NAVAL POSTGRADUATE SCHOOL
September 2004

Author: Scott P. Raymond

Approved by: Professor Daniel R. Dolk
Thesis Co-Advisor

Professor Gordon H. Bradley
Thesis Co-Advisor

Professor Dan C. Boger
Chairman, Department of Information Systems Technology

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

New facility construction and existing facility renovation create new or modified operation and maintenance (O&M) requirements for the maintenance responsibility organization such as a Public Works Department (PWD). This O&M requirement is fully described by an Operation and Maintenance Support Information (OMSI) package. OMSI content includes facility, systems, and product information.

This thesis will address information integration, the process of allowing information systems to cross-communicate and share data. OMSI information integrated within the framework of a Computer-Aided Facility Management (CAFM) system allows for early identification of O&M requirements, an improved planning capability for new facilities, and more efficient economies of scale. In addition to PWD manpower savings, OMSI-CAFM integration will also allow a revolution in the way O&M requirements are planned and created. Preliminary OMSI information would be ideally created by the design A/E after having considered work force capability from both a workload and expertise perspective. While this may be impractical due to the changing nature of workforce capability and the lengthy planning and design cycle of military construction, OMSI-CAFM integration will certainly allow O&M planning to begin early in the OMSI development stages. OMSI submittals can be layered to provide preliminary planning information in the first submittal and add additional detailed information in later submittals. In such a manner, PWD O&M planners can begin an incremental planning effort early in the facility construction phase.

This thesis provides a non-proprietary, no-cost solution to OMSI-CAFM information integration that minimizes specialized knowledge on the part of the OMSI AE. This will allow a broad applicability of the solution to all OMSI developers, including those for smaller non-MILCON projects that aren't specifically funded for OMSI generation. An effective solution must also provide for easy and inexpensive repurposing of OMSI information for future (and as yet unknown) uses. The solution uses XML technologies (XML, XSD, XLS, XLST, XPath, XQuery, etc) and XML storage systems for the content creation, management, and repurposing of OMSI information.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGEMENTS

I thank the faculty at the Naval Postgraduate School for their excellent education and outstanding resources. I owe much gratitude to my thesis advisors because they were invaluable to the completion of this thesis, but more importantly, they are great educators who have set me up for long-standing success.

I also thank George Oberlander Jr. for teaching me that software doesn't drive the process, rather the process drives the software. While this may seem a simple lesson, it is often overlooked and has a profound impact on any software-related development project.

THIS PAGE INTENTIONALLY LEFT BLANK

I: PROBLEM CHARACTERIZATION

Where is the Life we have lost in living?
Where is the wisdom we have lost in knowledge?
Where is the knowledge we have lost in information?
The cycles of Heaven in twenty centuries
Bring us farther from GOD and nearer to the Dust.

- T.S. Eliot; *Choruses from the Rock*

A. PROBLEM DESCRIPTION

While lamenting our distancing from Life, T.S. Eliot characterized the first known hierarchy of knowledge: Information → Knowledge → Wisdom. Had he written *The Rock* in the 21st century, he surely would have added the line "Where is the information we have lost in data?"

Russell Ackoff describes five levels of knowledge in the human mind: data (symbols), information (useful data), knowledge (application of data and information), understanding (why a state came to be), and wisdom (vision and design) (Ackoff 1989). Avoiding philosophical considerations of this knowledge hierarchy at the upper three levels, the lower two levels (data and information) are fundamentally important to any information system.

The general problem addressed in this thesis is information integration. Many legacy systems are stove-piped, meaning that data comes in or out at the tops and bottoms, but there is very little cross-communication between adjacent systems. Stovepipes significantly reduce the value of data used in an information context because it is very difficult, and sometimes impossible, to leverage the data beyond anything other than its originally intended purpose. Quite simply, many information systems, especially those that are stove-piped, leave unused a great deal of potential information contained in data.

Data is created or collected as a pure analytical exercise; measurements are made, tolerances are estimated, pieces are counted, and processes are described. Data can be represented as electronic bits, printed text, memorized facts, images, or any other suitable method of recording. While such data may have context or semantics, it does not by itself provide any useful information. Information systems (those systems that collect

data, apply context or semantics, and represent the information) are able to provide greater understanding and connectivity of data. Unfortunately they are often not able to share their representations with other systems.

We consider information integration to be the process of allowing information systems to cross-communicate and share data. This integration allows both structured data (e.g., data contained in databases) and semi-structured data (e.g., data contained in documents) to be collected once and purposed as new information across any number of systems. This creates the ability to repurpose data without manual intervention or re-authoring. Unfortunately, information integration is often an afterthought considered only when requirements for a new information system are identified. We will address a method of integrating information a priori as opposed to post facto, using a process that begins the moment data is collected.

B. METHODOLOGY

The process of information integration begins with an analysis of how information is stored and accessed. We examine both relational database management system (RDBMS) and XML representations of relational information. These two information "models" will be used to represent the same data. Information integration is successfully achieved when one model can be transformed to all other models. This will result in a framework that supports the entire information life-cycle: creation, management, and repurposing. *[N.B. I am not referring to "Information Lifecycle Management" which is the process of determining how data and information flow from the moment it is created or identified to the end of its use by way of retention policy.]* Using multiple models of the same information will allow for more elegant and efficient repurposing to satisfy myriad end-user needs. We develop these models and then apply them to a specific application, namely facilities construction and management, to show the feasibility of this design.

C. SCOPE OF THESIS

This thesis addresses information integration in the realm of facilities management. Computer-Aided Facility Management (CAFM) systems are complex

information management systems that support the life-cycle management process of facilities from design, through construction and commissioning, occupancy, and demolition. The United States Navy is responsible for the operation and maintenance of 31,000 buildings located on 2,000 bases worldwide. Given such a large span of control, information integration within the realm of the Navy's CAFMs is essential for effective oversight and planning. The two major CAFMs currently in use by the Navy are Archibus [ARCHIBUS][1] in the European region and Maximo [MAXIMO] in the remaining regions. We consider information integration only for the Archibus system, which includes the authoring process of the facility management information (known as Operation and Maintenance Support Information, or OMSI), the synchronization of the information within the CAFM, and repurposing the information for non-CAFM use.

D. SUMMARY OF THESIS

The remainder of this thesis is divided into eight chapters. Chapter II provides a more detailed background of the problem and includes overviews of the organizations involved in the Navy's OMSI program. Chapter III reviews the technologies involved in data modeling and data storage systems. Chapter IV documents the development of the Archibus schema for representing OMSI data and includes an entity-relationship (ER) diagram and database schema diagram. Chapter V reviews XML schema technologies, established schema design considerations and modeling rules, and documents an XML Schema that is equivalent to the ER diagram of the previous chapter. Chapter VI contains an analysis of the storage of XML-based OMSI information. Data-centric and document-centric systems are described and compared for use with OMSI. Chapter VII provides descriptions of the XML transformation technologies that are needed for OMSI deliverables. Chapter VIII details the development and implementation of XML-based deliverables to be used with OMSI. Some deliverables are proof-of-concepts while others are production-ready. Chapter IX summarizes the conclusions and implications of this thesis

[1] Software applications and standards referenced by square brackets are listed at the end of this thesis.

THIS PAGE INTENTIONALLY LEFT BLANK

II: BACKGROUND OF PROBLEM

A. ORGANIZATIONAL OVERVIEW

This thesis deals with the operation and maintenance (O&M) of Navy facilities. While there are countless organizations involved with the life-cycle creation and management of the O&M process throughout the entire Navy, I focus on the process at a single overseas base in Sicily, Italy. The organizations involved in this process are Atlantic Division, Naval Engineering Facilities Command; Naval Air Station Sigonella; Archibus Solutions Center – Research Triangle; and Syska. While other bases include parallel or even different organizations, the fundamental process is similar and can be easily extended from this thesis.

The following sections provide a basic overview of these organizations as context for the discussion of the relevant problem and issues.

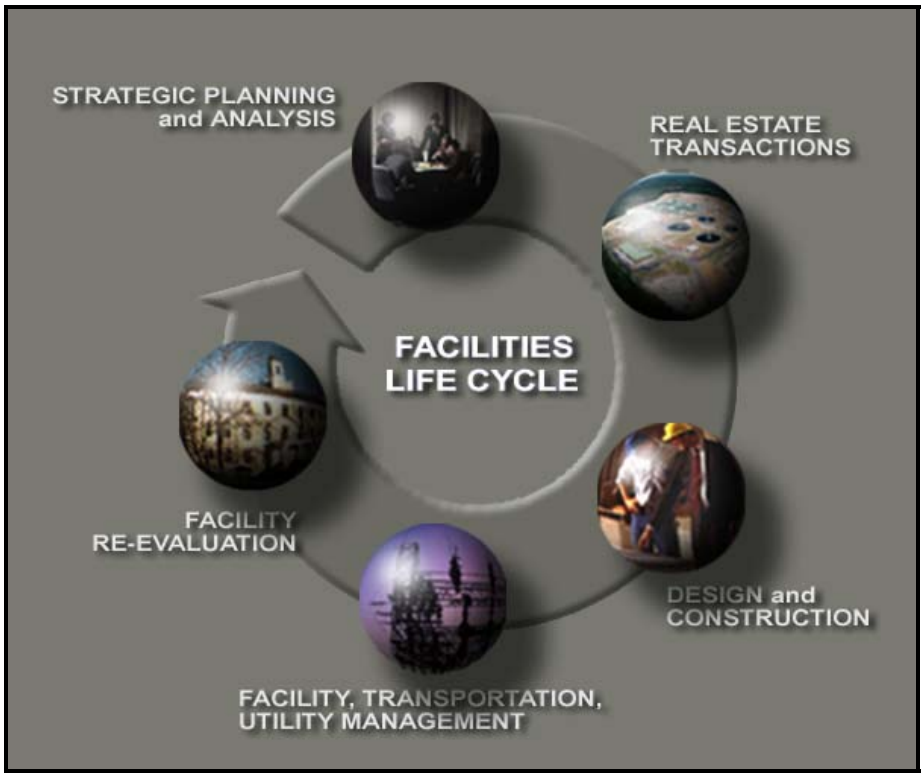
1. Atlantic Division, Naval Engineering Facilities Command (LANTDIV)

The Atlantic Division, Naval Engineering Facilities Command (LANTDIV) has as its mission to provide quality facilities, proactive operational support, and expert engineering services to military and non-military government agencies. LANTDIV is an Engineering Field Division (EFD) under Naval Facilities Engineering Command (NAVFAC) authority. All information concerning LANTDIV has been taken from their website (LANTDIV 2004).

LANTDIV was established in 1942 with the intent to decentralize and expedite NAVFAC actions in the Atlantic area, to provide liaison between NAVFAC and field organizations, and to provide competent liaisons between theater commanders and Seabee Construction Battalions. It has since evolved into the primary engineering advisor to operational commanders.

LANTDIV uses a Facilities Life Cycle model described in Figure 1 to manage all facets of the Navy's public works and facility planning requirements.

Figure 1. LANTDIV's Facilities Life Cycle



SOURCE: From the LANDTIV Homepages (LANTDIV 2004)

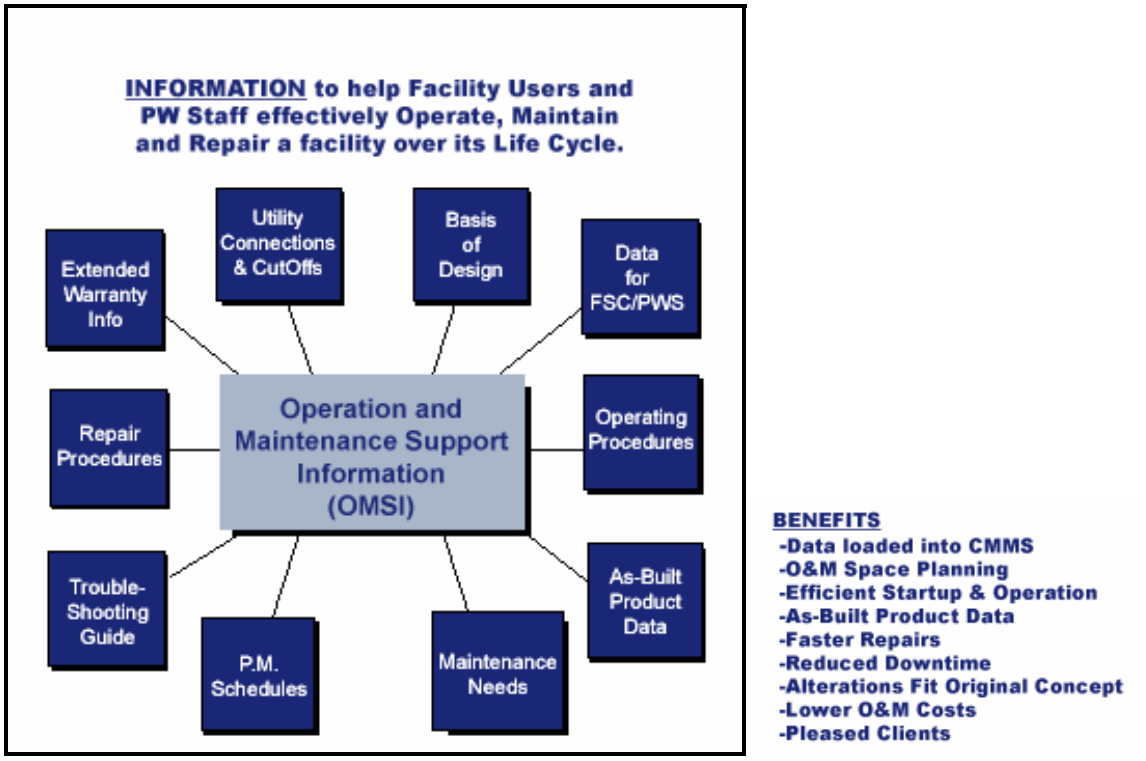
Of particular interest among LANTDIV's services provided to the Navy is the "Base Operations" Business Line, which is part of the "Facility, Transportation, Utility Management" life cycle element. Base Operations contains a branch called "Facilities Maintenance and Engineering Branch", which is summarized by the following description:

> [The] Facilities Management and Engineering Branch of the Base Operations Support Business Line of the Atlantic Division, Naval Facilities Engineering Command … provides facilities engineering and technical assistance in managing the Navy's Public Works Management Program as related to maintenance engineering applications and implementation at the Claimant, Regional and Activities levels. In support of the Navy's Base Operations requirements, the Branch provides Sustainment, Restoration and Modernization (SRM) program management, Facilities Condition Assessment program (FCAP) management, Operation Maintenance Support Information (OMSI) program management, and Cathodic Protection (CP) technical expertise. These programs have been established to focus on the resource and life-cycle management of facilities and assets belonging to the Navy. As your

partner in the Facilities Management and Engineering business, we are committed to providing you the best possible support and tools for effective and efficient management of your assets.

This thesis focuses on the OMSI product: information to help the facility user and maintenance staff effectively operate, maintain, and repair a facility. Figure 2 describes the components of LANTDIV's OMSI deliverables.

Figure 2. OMSI Components



SOURCE: From the LANTDIV's Facilities Management and Engineering Webpage (LANTDIV 2004)

2. Naval Air Station Sigonella (NASSIG)

The Naval Air Station Sigonella (NAS Sigonella) is the end-user (or consumer) of the OMSI deliverable. More specifically, their Public Works Department (PWD) manages the operation and maintenance (O&M) of all facilities and infrastructure. The PWD's Program Management Office (PMO) provides cradle-to-grave oversight for the design, construction, commissioning, and delivery of construction contracts. The Resident Officer-in-Charge of Construction (ROICC) Office, a tenant command who is headed by the dual-hatted Public Works Officer (who has concurrent reporting seniors), oversees the actual construction contracts that build, renovate, or repair facilities and

infrastructure. A portion of these contracts requires the OMSI deliverable. In 2002, the PWD developed the "Storefront" mission, vision, and focus areas identified in Figure 3.

Figure 3. PWD Storefront Mission, Vision, and Focus Areas

Public Works Storefront

The Public Works Storefront is the synthesis of the Public Works Department (NASSIG) and the ROICC Sicily (EFA-MED) offices into one entity that provides Sigonella customers a single point-of-service for their needs.

Our Mission
We proudly build, fix, maintain, and support Sigonella safely and quickly to meet our customer's needs by providing high quality and cost effective facilities, utilities, transportation, environmental, and contracting services.

Our Vision
We are an integral part of the Sigonella team. We are valued for our ability to provide responsive best-value solutions for Public Works requirements. Our innovative team executes our mission with a strong commitment to excellence and customer satisfaction.

Our Focus Areas
For our PEOPLE, we will increase personnel skills and capabilities as well as the level of satisfaction with our recognition programs. We will improve the safety and functionality of our workplace, and improve communications at all levels within our organization.

To be INNOVATIVE, we will implement best business practices that: reduce cycle times and costs; better use our facilities and utilities; and provide the means to achieve better, faster, cheaper, and safer products and services.

For our CLIENTS, we will improve: response; execution; and the sharing of information. We will increase community involvement and the use of private investment to improve mission performance.

Within our OPERATIONS, we will reduce total facility cost, increase our overall operating efficiency, and improve our return on investments.

SOURCE: From the NAS Sigonella Public Works Department, February 2002

It is significant to note that the Storefront recognizes the importance of providing integrated services to their customers. In a similar fashion, the Storefront must also provide for the integrated delivery of OMSI within its own organization (employees are, after all, internal customers).

The PWD contains approximately 150 local national employees, 170 enlisted Seabees, five chiefs, ten senior U.S. civilians (GS-12 and GS-13s), and six officers. Among its many divisions, the Operations Division (PWOps) is directly responsible for the O&M requirements. PWOps is also responsible for the management of the Archibus Computer-Aided Facility Management (CAFM) system. The PMO Division is responsible for ensuring that construction contracts contain OMSI deliverables where appropriate; the determining factor is usually budget availability within the MILCON programs.

3. Support Contractors

The creation of OMSI requires significant effort by an Architect/Engineer (A/E) contractor to collect equipment manufacturer's data from construction contractors and develop the OMSI submittal. LANTDIV manages this OMSI A/E contractor for all Military Construction (MILCON) contracts at NAS Sigonella and typically uses a single contractor for all OMSI deliverables during a contract period. The first OMSI A/E to begin implementing OMSI deliverables into the Archibus Computer-Aided Facility Management (CAFM) system at NAS Sigonella was Syska in 2003.

Delivering OMSI into the framework of a CAFM requires coordination with a database administrator. NAS Sigonella (and all of Navy Region Europe) uses Archibus Solutions Center – Research Triangle (ASC-RT) as their subject matter expert for Archibus. Indeed, any changes to the Archibus schema must be coordinated through the Archibus Program Manager, who in turn, contracts with ASC-RT to make the changes and disseminate them throughout Navy Region Europe. Syska also uses ASC-RT as their Archibus subject matter expert and has contracted with them in the past to deliver OMSI into Archibus.

B. SUMMARY OF PROBLEM

New facility construction or existing facility renovation creates new or modified operation and maintenance (O&M) requirements for the maintenance responsibility such as a Public Works Department (PWD). This O&M requirement is fully described by an Operation and Maintenance Support Information (OMSI) "package" delivered by the project's Design A/E, the construction contractor, or a specialized OMSI A/E. OMSI content includes facility information, primary systems information, and product data. Ten years ago, this content was delivered on paper (usually two to five volumes). In the last five years, this data has been provided as an electronic delivery (PDF files), although it is still document-based and maintains a printed-page style layout. The benefits of such electronic delivery included reduced expense (it was always created electronically as a first step), the searchable nature of electronic data, and the ease of distribution.

Operation and maintenance organizations often utilize a Computer-Aided Facility Management (CAFM) system to manage and oversee the scheduling of O&M

requirements. The CAFM also provides database information on assets, generates historical performance reports, and plans for current and future O&M requirements.

Historically OMSI is delivered at the time of facility acceptance (i.e. when the construction is complete and the facility is ready for occupancy) and the PWD is left to incorporate the data into the CAFM system using their own manpower, and in whatever manner is deemed appropriate at the time. Atlantic Division, Naval Facilities Engineering Command (LANTDIV) and NAS Sigonella are now in the process of integrating OMSI delivery within the framework of the CAFM.

The benefits of creating and delivering OMSI within the framework of the CAFM system include economies of scale, early identification of operation and maintenance (O&M) requirements, and planning capability for new facilities. In addition to these PWD manpower savings, OMSI-CAFM integration will also allow a revolution in the way O&M requirements are planned and generated. Ideally, preliminary OMSI information would be created by the design A/E after having considered work force capability from both a workload and expertise perspective. While this may be impractical due to the changing nature of workforce capability and the lengthy planning and design cycle of military construction, OMSI-CAFM integration will certainly allow O&M efforts to begin early in the OMSI development stages. OMSI submittals can be layered to provide preliminary planning information in the first submittal and add additional detailed information in later submittals. In such a manner, PWD O&M planners can begin an incremental planning effort early in the facility construction phase. This is critical to the successful management of facility commissioning because contracted maintenance must be budgeted years in advance and some maintenance agencies require advance notice of new requirements. Incremental planning ensures a proactive approach to facility management and avoids the necessity for reactive last-minute planning. In extreme cases, new facilities must go without O&M because the organic workforce is unable to assume the responsibility and contracts cannot be setup or modified in time to start maintenance upon facility commissioning.

NAS Sigonella is currently in the fourth year of a ten-year, $750M Recapitalization effort that will require $5M to $10M to generate OMSI deliverables. Significant strides were made in the summer of 2002 to require future OMSI submissions

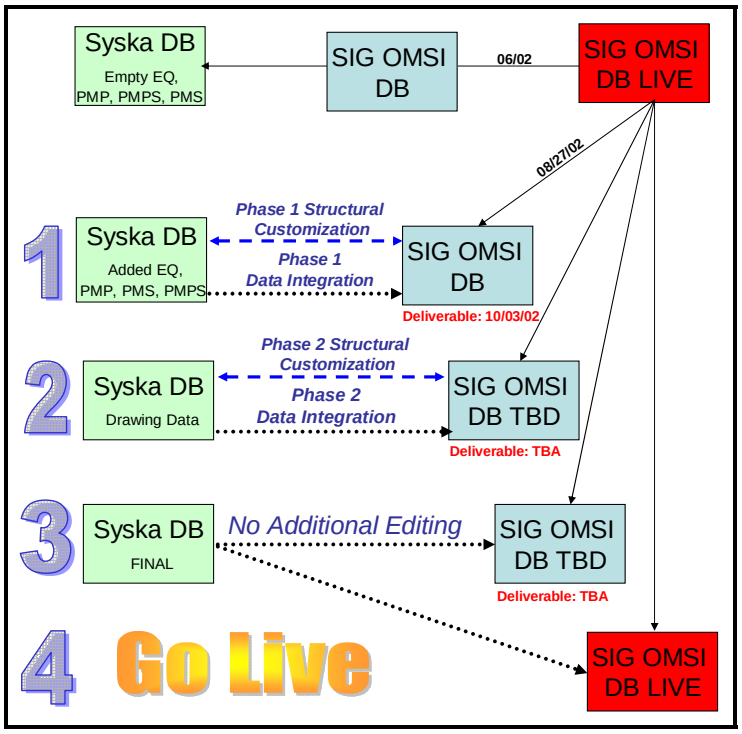
to be delivered directly into the CAFM system. However, this effort is narrowly focused to solve a specific problem for one OMSI AE and one OMSI contract. The solution involves proprietary knowledge and proprietary software – items that prohibit an immediate and general application of the solution to other OMSI deliverables. There is also the added expense of subcontracting with the Archibus contractor each time the deliverable must be incorporated into the existing database.

The objective is to seek a solution to OMSI-CAFM integration that minimizes specialized knowledge on the part of the OMSI AE. This will allow a broad applicability of the solution to all OMSI developers, including those for smaller non-MILCON projects that aren't specifically funded for OMSI generation. An effective solution must also provide for easy and inexpensive repurposing of OMSI information for future (as well as yet unknown) uses. It is expected that the cost of creating the deliverable will also decrease with Archibus contractor independence.

1. Current OMSI Delivery Process

The current OMSI process was reviewed to gain an understanding of the issues which impact the final deliverables. In general, the process of creating database content, synchronizing new content with existing content, and presenting database content for varied purposes can be difficult – sometimes overwhelmingly so. The creation of OMSI deliverables is an example that demonstrates such difficulty. Figure 4 is a diagram of the current method for synchronizing OMSI deliverables from Syska (the current OMSI A/E under contract with LANTDIV) with a larger CAFM database at NAS Sigonella.

Figure 4. Current OMSI Database Synchronization Method



SOURCE: From the ASC-RT delivery documentation flowchart by Sherri Johnston dated 03 October 2002

Each transport stage that connects the "Syska DB" (the development Archibus database) to the "SIG OMSI DB" (the production Archibus database), or vice versa, requires specialized work by a contractor familiar with Archibus in general and NAS Sigonella's implementation of Archibus in particular. Of course, this work is an effort that requires time and money. The term for software that performs these transports is generally known as "middleware". Though there are many commercial middleware applications, ASC-RT chose to develop their own "one-off" middleware solution for synchronizing the development and production Archibus databases. This has resulted in an expensive, closed-source solution.

Archibus has built-in functionality as well as an API for importing and exporting its data. Unfortunately this functionality is not currently being used to synchronize OMSI deliverables. This represents a great untapped improvement for creating a free and open methodology for avoiding a middleware solution.

2. The Future of OMSI

As the last ten years of OMSI deliverables at NASSIG have shown, it is difficult to predict future OMSI processes. Nonetheless, we can learn from our past missteps and avoid repeating the same pitfalls.

Consider the case of the oldest OMSI deliverables: printed manuals. While it is presumed these manuals met all the needs of their consumers when originally delivered, they now sit on shelves collecting dust – if they haven't been lost, thrown out, or destroyed. Updates to the manuals were not made, which calls into question their accuracy. As facility management processes have changed over the years, the printed manuals cannot be incorporated easily into new O&M systems. For example, it is an expensive and time-consuming task to convert these printed manuals into some electronic form suitable for Archibus. In short, the printed manuals are of limited value to the current OMSI program.

The decision to move to an electronic OMSI deliverable did not solve either of the two problems previously discussed: current-day accuracy and repurposing for new modes of consumption. PDF documents are as static as the printed manual and there is no easy way to convert them to use in any database, except using rudimentary techniques such as Binary Large Objects (BLOBs) to store the entire document in a database field. It would seem the only improvement gained by PDF deliverables is the reduction of production costs (printing can be expensive) and the liberation of library shelves.

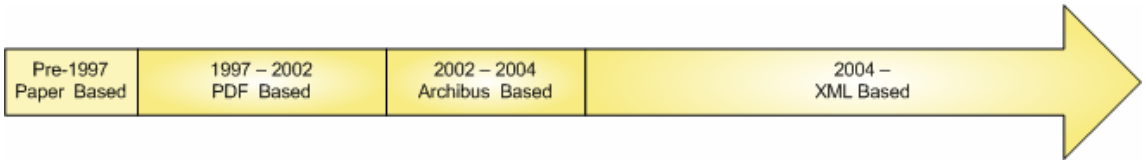
The decision two years ago to deliver OMSI directly into Archibus only solves one of the problems still associated with PDF deliverables. While NASSIG is now able to ensure current-day accuracy by maintaining the electronic database, repurposing is still a difficult prospect. This solution, while revolutionary, did not avoid the same pitfall we experienced ten years ago!

The results of this thesis describe a better process for delivering OMSI - "better" in the sense that it both avoids currently identifiable pitfalls and hedges against future, as yet unidentified, pitfalls. We make the case that a delivery framework of Extensible Markup Language (XML) is the foundation of a better process for OMSI creation, management, and repurposing. XML provides a method of OMSI creation, the means for its storage, the mode of its delivery, and the feasibility of its cross-platform and cross-

purpose utilization. XML solves both of the problems discussed: current-day accuracy is assured by incorporating the deliverables directly into Archibus (or any other CAFM for that matter) and repurposing for new modes of consumption is easily done using standard (and free) technologies and programs.

Figure 5 shows the timeline of OMSI deliverables. XML-based delivery will take OMSI far into the future, hedging against new CAFMs, modified O&M practices, different planning tools, and even new maintenance organizations (e.g. contracting out all maintenance efforts.)

Figure 5. Timeline of OMSI Deliverables



This thesis will create a delivery mechanism that can persist well into the future – certainly longer than the two years of Archibus-based and five years of PDF-based deliverables. The future of OMSI will include inexpensive modes of content creation and flexible manners of repurposing. The OMSI process will influence a larger portion of the facility management life-cycle by providing information at all stages of a facility's life.

III: REVIEW OF TECHNOLOGIES

A. BACKGROUND CONCEPTS

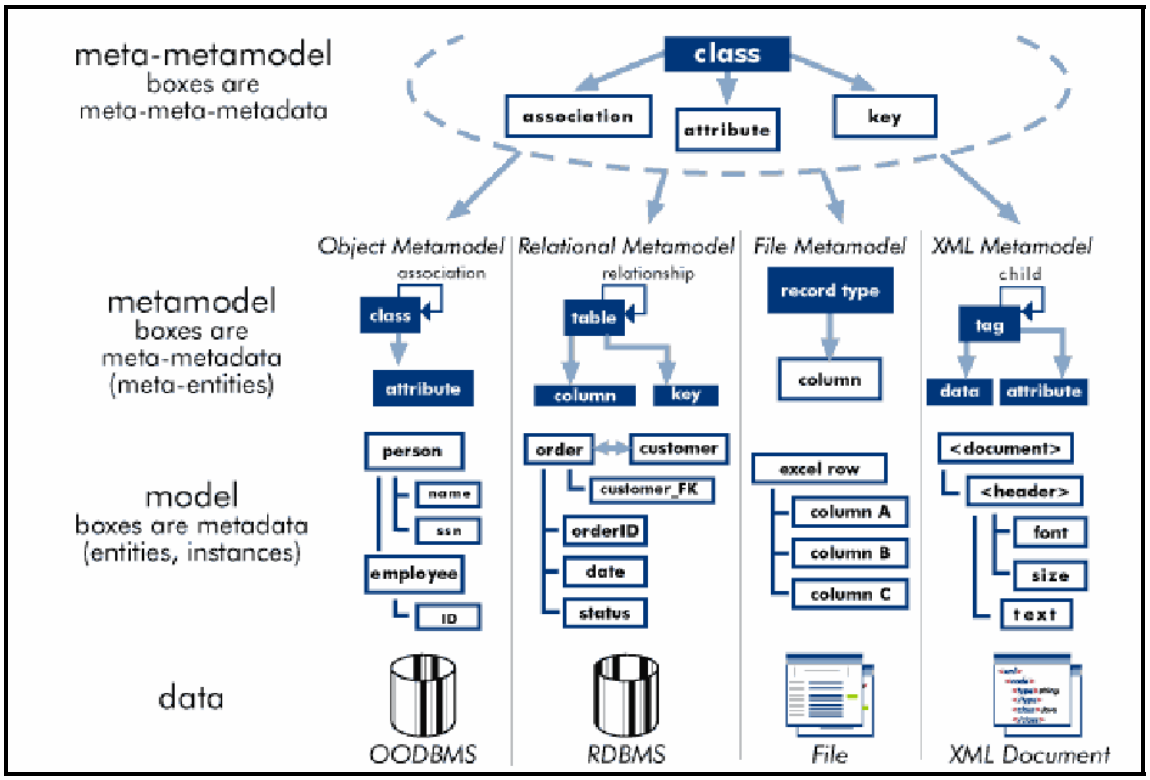
This chapter lays out the two concepts necessary for understanding the foundations of integrating OMSI with the Archibus CAFM: data modeling and data storage systems. Data modeling is the more important of the two concepts because the choice of model will influence data storage systems that must access the information captured in the model.

B. DATA, MODELS, AND METAMODELS

Taking a step back to the concept of data storage systems sets the stage for dramatic improvements to the current OMSI database synchronization process. Data storage systems rely on models of data to provide semantics for the content stored within the system. The combination of the data and its semantics gives rise to information, which is really the whole point behind a data storage system. Data models can also be modeled themselves – these "models of models" are termed metamodels to help prevent confusion. Figure 6 demonstrates a description of common metamodels that define languages for expressing models themselves.

The Relational and XML metamodels are of particular interest to this OMSI project. Both metamodels have seen prolific use in managing complex information structures and offer powerful ways of accessing and exchanging information. We will describe both of these models and compare them in the following sections.

Figure 6. Data, Models, and Metamodels



SOURCE: From "Model-Driven Information Integration" (MetaMatrix 2002)

1. The Relational Data Model

Edgar F. Codd developed the first database model in 1970 while working for IBM. He published "A Relational Model of Data for Large Shared Data Banks", using mathematical relations to represent data. Although hierarchical and network databases were in use before 1970, Codd was the first to formally describe a data model; it wasn't until later that the hierarchical and network models were retrofitted to describe pre-1970 models.

a. Characteristics of the Relational Data Model

The relational data model represents all data as mathematical relations. At the model's foundation are *domains* (or data types) which describe the allowed values of data. Unordered pairs of domain and value are called *attributes* and sets of attributes are called *tuples*. Unordered sets of tuples are called *relations*. In traditional database vernacular, relations are called tables and tuples are called rows. The properties of

relations further constrain the relational data model. Attributes must be single-valued; multiple values are represented by using more than one attribute.

One of the main tenets of the relational model is the separation of the logical and physical views of the data. Mathematical relations (e.g. subsets of the Cartesian product of n sets) are applied to the logical model and reasoning about data is accomplished by using the true/false evaluation of a given proposition. Relational calculus and algebra allow operations upon the data for retrieval and manipulation, and a relational database management system (DBMS) provides support for these operations to define a database and business rules about the data.

The concept of normalization is deeply rooted in the relational database realm; database designers recognize that while there can be many ways to model the same data, not all relations are equally attractive. Some relations can create instances where changing the data can have undesirable consequences. These consequences are called “modification anomalies” and through careful consideration they can be avoided whenever desired. The process of redefining relations to avoid these anomalies is called “normalization”. Normalization is essential to the long-term integrity of a database and the data model should be normalized prior to using the database for production purposes.

b. Benefits of the Relational Data Model

The relational model or relational databases in general, have many useful benefits. Such benefits are usually the result of a strict separation of the logical and physical views of the data which allow for great speed and capability and the sharing of data amongst multiple applications.

Physical integrity is an essential aspect of any data model. In the relational data model, such "model integrity" can be looked at on the domain, attribute, tuple, and relation level. Domain and attribute integrity are the most fundamental requirements of the relational model. Tuple integrity, also known as entity integrity, requires that the primary key can never be null and that database operations must maintain the existence and uniqueness of all the primary keys. Relation integrity requires that foreign keys must be NULL or match the values of the primary keys to which they relate.

The physical integrity of the relational data model leads to a very useful aspect of the logical view. Multiple tables can be dynamically joined to present the end-user with a single table. The two dimensional nature of "data as a table" is a more natural construct for end-users than multi-dimensional relations. Of course, operations on the virtual table must maintain the physical integrity of the underlying relations.

In addition to the physical integrity of the relational model itself, the DBMS must also ensure integrity of the data it manages. Data integrity can be maintained using the ACID model of transactions: Atomicity, Consistency, Isolation, and Durability. Atomicity requires each database transaction to be atomic – if a part of the transaction fails, the whole transaction must fail. Consistency maintains that only valid data can be written to the database. Isolation permits concurrent transactions but prevents impact between them. Durability ensures that no executed transactions are lost.

Relational constraints such as keys, dependencies, and referential integrity allow the expression of the model beyond what the data model itself requires. These relational constraints are maintained by the DBMS and allow for the creation of a database schema based upon identified business rules. Indexing is another very powerful function of the relational model. It allows for quick access, facilitated sorting, and prevention of duplicates for a given attribute.

2. The XML Data Model

XML serves as the technological foundation of this study and so it is important to understand the fundamental nature of the XML data model. The W3C essay *The XML Data Model* [XMLDATAMODEL] describes it in the following way:

> The data model for XML is very simple - or very abstract, depending on one's point of view. XML provides no more than a baseline on which more complex models can be built. All those more restricted applications will share some common invariants, however, and it is those that are given below.
>
> Think of an XML document as a linearization of a tree structure. At every node in the tree there are several character strings. The tree structure and the character strings together form the information content of an XML document. Almost everything will follow naturally from that. Some of the characters in the document are only there to support the linearization, others are part of the information content.

It is not an overstatement that the XML data model is, indeed, quite simple. Yet XML can be used to model very complex information. It is no less powerful than the relational model and provides many unique characteristics that will be useful to the OMSI process.

XML stands for "eXtensible Markup Language" – a standardized and extensible meta-language used to define other markup languages. W3C's Extensible Markup Language (XML) 1.0 (Third Edition) Recommendation [XML1.0] documents the details of the XML standard. At their foundations, XML markup languages are a hierarchical collection of "elements" [N.B. There are other "node" types in an XML document which will be discussed later.] One example of an XML markup language is XHTML – similar to HTML but conforming to all the XML rules. Other examples are CML, MathML, SVG, etc.

The XML data model describes the construction of XML documents. These documents are Unicode "text" documents (so they are processor and platform independent) that contain both data and metadata (data about data.) Well-formed (not the same as "valid", which will be discussed later) XML documents must conform to the following basic rules:

1) every start tag must have a matching end-tag, or be an "empty" tag;
2) tags must be properly nested – all children elements must be closed before the parent element can be closed;
3) an XML document can have only one root element;
4) element and attribute names must begin with a letter or a "_", contain only letters, digits, "_", "-", and ".", and contain no spaces. They can't start with "xml" and case sensitivity must be respected;
5) white space is retained within PCDATA; and
6) values of attribute key-value pairs must be enclosed in quotes or apostrophes.

Data is represented as element "content" (the stuff between the tags) and sub-elements. Metadata is contained in the element tag names and attributes which describe the elements. Data and metadata combined in the same document give rise to information – that is, data with meaning. This allows XML to solve problems of semantics, structure, and style all at once.

XML documents can be "validated" against a standardized schema by using XML schemas; validation ensures that the format of the document is in accordance with the

markup language specifications. For example, the XML schema can require that each <Book> element must have an <Isbn> child element. XML schemas can use regular expressions which allow exceptional control over element/attribute presence and text content.

An XML document is essentially just a hierarchical tree of "nodes." These nodes can be described using the Document Object Model (DOM) [DOM1.0], which provides a standard set of objects for representing both HTML and XML documents. The DOM also serves as a standard interface for accessing and manipulating HTML and XML objects.

The DOM represents XML documents as a hierarchy of Node objects, of which there are twelve types: Element, Attribute, Text, CDATA Section, Entity Reference, Entity, Processing Instruction, Comment, Document, Document Type, Document Fragment, and Notation.

An XPath tree [XPATH1.0] is an alternate model for representing an XML document. There are only seven XPath node types: root node, element node, text node, attribute node, comment node, processing instruction node, and namespace node. These node types correspond fairly directly to the DOM Node object. Each XPath node type is described in Table 1.

Table 1. XPath Node Types

Node Type	Description
Root node	There is one root node for each document. This is similar to the DOM's document node. Do not confuse the root node with the document element, which in a well-formed document is the outermost element that contains all other elements.
Element node	An element node is a part of the document bounded by start and end tags, or represented by a single empty element tag such as <Tag/>
Text node	A text node is a sequence of consecutive characters in a PCDATA part of an element. There can never be two adjacent text nodes in the tree because a text node is made as big as possible (they will be merged together.)
Attribute node	An attribute node includes the name and value of an attribute written within an element start tag (or empty element tag.)
Comment node	A comment node represents a comment written in the XML source document between the delimiters "<!--" and "-->".
Processing Instruction Node	A processing instruction node represent a processing instruction written in the XML source document between the delimiters "<?" and "?>". Note that the XML declaration ("<?xml version='1.0'?>") is not a processing instruction, even though it looks like one.
Namespace node	A namespace node represents a namespace declaration, except that it is copied to each element that it applies to. So each element node has one namespace node for every namespace declaration that is in scope for the element.

SOURCE: From "XSLT, 2nd Edition" (Kay 2003)

The concept of a node is important. XPath (as well as other XML technologies that use XPath) does not provide direct access to tags, attributes, or other markup. Instead, it provides access to the logical nodes established by the particular markup. The relationships between nodes are central to XPath; the document tree can be traversed based on how elements relate to one another or how attributes relate to elements, etc.

3. Comparison Between the Relational and XML Data Models

Table 2 summarizes the differences between the relational and XML data model.

Table 2. Comparison Between the Relational and XML Data Model

	Relational	XML
Structure:	Data is stored in a two-dimensional array of rows and columns. The structure is flat, but relations between tables provide depth.	Data is stored in a nested tree of limitless depth.
Homogeneity:	Data is regular and homogenous. Every row of a particular table has the same columns with identical names and data types.	Data is irregular and heterogonous. Each data instance can have a different structure.
Relations and Keys:	Table joins are at the heart of a database schema. Keys and indices are used to increase performance.	Hierarchical relationships are easy to navigate. Relations between sibling or cousin elements can only be done with XML schemas.
Query Results:	The result of a query is flat, regular, and homogenous.	The result of a query can be an irregular and heterogeneous tree.
Density:	Data is dense – every column of every row must have a value. NULLS are used as placeholders for non-existing data.	Data is sparse. Because no two elements (even of the same type) need have the same structure, inapplicable data may simply be removed from the structure.
Order:	Data is structurally unordered. Rows of a table have no inherent ordering, although order can sometimes be derived from data values.	Data is intrinsically ordered according to its location in the tree.

SOURCE: Summarized from "XQuery from the Experts: A Guide to the W3C XML Query Language" (Chamberlin, Draper et al. 2003)

The most noteworthy difference is the homogeneous nature of the relational model vs. the heterogeneous nature of the XML model. This difference makes the relational model well-suited to instances where the data is uniform and predictable while the XML model is more appropriate where the data structure may change between instances.

C. DATA STORAGE SYSTEMS

Data storage systems can be divided into three broad categories: data-centric, document-centric, and information-centric models. The best known data storage system

is the relational database, which conforms to the data-centric model. Examples of document-centric models include spreadsheets, XML documents, and other file-based storage systems such as Native XML databases. There are only a few examples of information-centric models, of which Neocore's XML Information Management System [NEOCOREXMS] may best fit this category. In September 2003 Xpriori™ acquired all intellectual property and products from NeoCore Inc. and made [NEOCOREXMS] its flagship product. For more information on Xpriori™ and NeoCore, see the Xpriori™ homepages (Xpriori 2004).

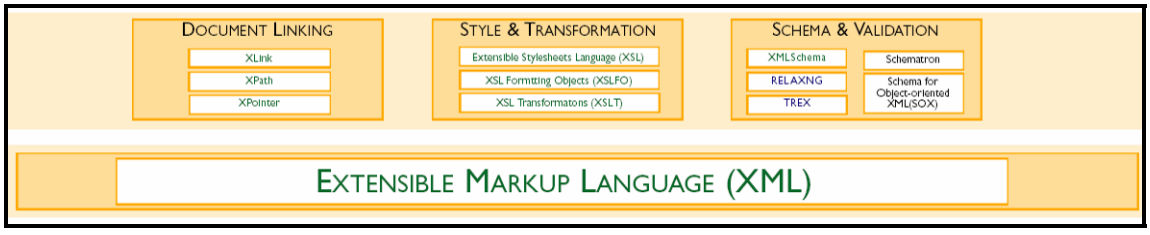
The current OMSI deliverables do not all fit entirely within a single category; rather some parts are document-centric, some parts are data-centric, and some parts are a mixture of both. For example, start-up and shut-down procedures, environmental considerations, etc. are purely document-centric. Preventive Maintenance (PM) procedures and equipment schedules are purely data-centric. PM libraries are a mixture of both.

Although relational databases are fundamentally data-centric, they can store documents as binary large objects (BLOBs) within database fields or they can store the document filename in a field for reference use through hyperlinks. These solutions are not without disadvantages: manipulating the document for presentation can be difficult and the requirement for third-party document viewers must be taken into account.

D. XML AS A DATA STORAGE SYSTEM

XML is equally appropriate for both data-centric and document-centric systems. The "XML Family" of technologies includes document linking, style and transformation, and schema and validation specifications, as shown in Figure 7.

Figure 7. XML Technologies



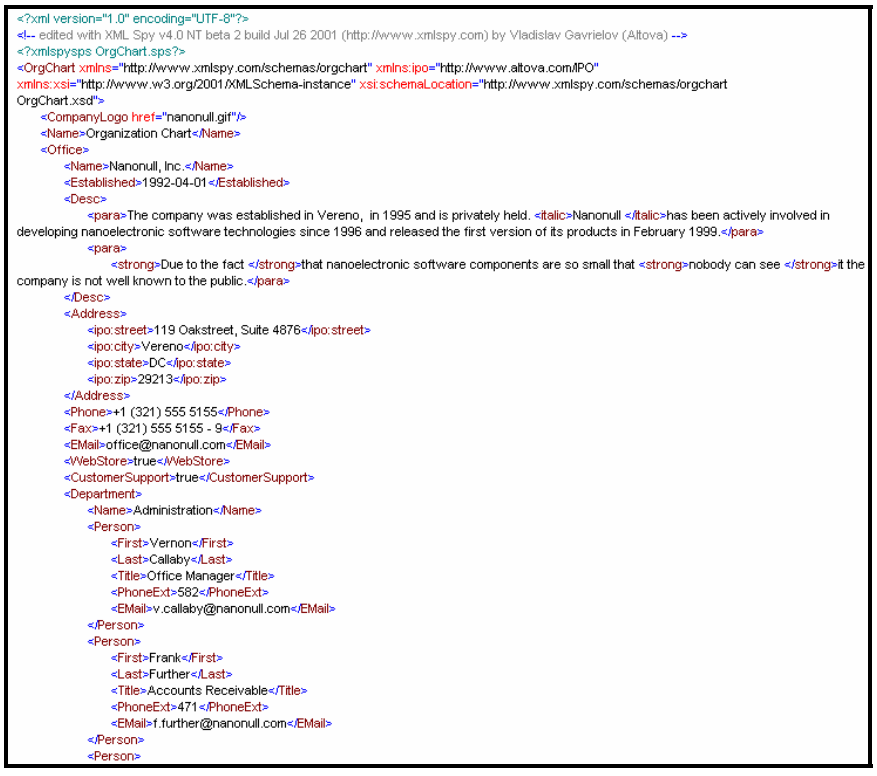
SOURCE: From "Key XML Specifications and Standards" (ZapThink 2002).

XML provides significant advantages over relational databases because of its ability to separate data from presentation and to provide a platform-independent way of transporting data.

1. Separating Data from Presentation

The presentation of data can be separated from the data itself through the use of the Extensible Stylesheets Language (XSL). XSL Transformations (XSLT) is one of the most common XSL technologies – it is even built into the latest versions of Microsoft's Internet Explorer. Figure 8 is an example XML document. It is "raw" in the sense that no transformations have yet been applied. Figure 9 shows the result of applying an XSLT – the XML document has been transformed into an XHTML document that is viewable with any Internet browser. It is significant to note that the transformation can be applied in real-time on the client side: each client can apply its own XSLT to achieve the preferred presentation.

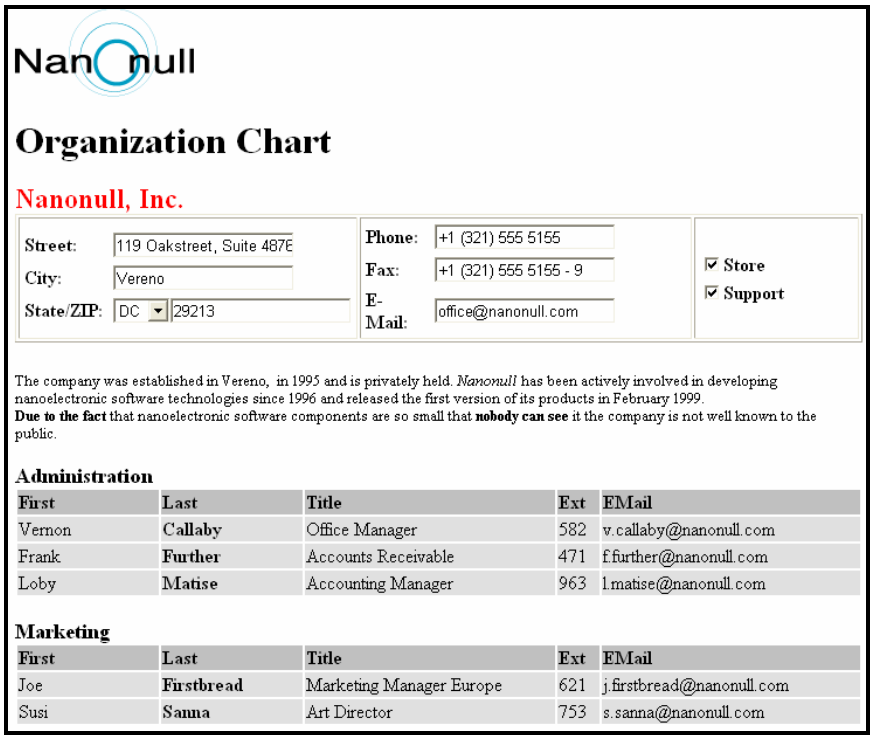
Figure 8. "Raw" XML Document



```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.0 NT beta 2 build Jul 26 2001 (http://www.xmlspy.com) by Vladislav Gavrielov (Altova) -->
<?xmlspysps OrgChart.sps?>
<OrgChart xmlns="http://www.xmlspy.com/schemas/orgchart" xmlns:ipo="http://www.altova.com/IPO"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.xmlspy.com/schemas/orgchart
OrgChart.xsd">
    <CompanyLogo href="nanonull.gif"/>
    <Name>Organization Chart</Name>
    <Office>
        <Name>Nanonull, Inc.</Name>
        <Established>1992-04-01</Established>
        <Desc>
            <para>The company was established in Vereno, in 1995 and is privately held. <italic>Nanonull </italic>has been actively involved in
developing nanoelectronic software technologies since 1996 and released the first version of its products in February 1999.</para>
            <para>
                <strong>Due to the fact </strong>that nanoelectronic software components are so small that <strong>nobody can see </strong>it the
company is not well known to the public.</para>
        </Desc>
        <Address>
            <ipo:street>119 Oakstreet, Suite 4876</ipo:street>
            <ipo:city>Vereno</ipo:city>
            <ipo:state>DC</ipo:state>
            <ipo:zip>29213</ipo:zip>
        </Address>
        <Phone>+1 (321) 555 5155</Phone>
        <Fax>+1 (321) 555 5155 - 9</Fax>
        <EMail>office@nanonull.com</EMail>
        <WebStore>true</WebStore>
        <CustomerSupport>true</CustomerSupport>
        <Department>
            <Name>Administration</Name>
            <Person>
                <First>Vernon</First>
                <Last>Callaby</Last>
                <Title>Office Manager</Title>
                <PhoneExt>582</PhoneExt>
                <EMail>v.callaby@nanonull.com</EMail>
            </Person>
            <Person>
                <First>Frank</First>
                <Last>Further</Last>
                <Title>Accounts Receivable</Title>
                <PhoneExt>471</PhoneExt>
                <EMail>f.further@nanonull.com</EMail>
            </Person>
            <Person>
```

SOURCE: From Altova's XMLSpy 2004 on-line tutorial.

Figure 9. XML "Transformed" into HTML



Nanonull

Organization Chart

Nanonull, Inc.

Street:	119 Oakstreet, Suite 487E	**Phone:**	+1 (321) 555 5155	☑ **Store**	
City:	Vereno	**Fax:**	+1 (321) 555 5155 - 9	☑ **Support**	
State/ZIP:	DC 29213	**E-Mail:**	office@nanonull.com		

The company was established in Vereno, in 1995 and is privately held. *Nanonull* has been actively involved in developing nanoelectronic software technologies since 1996 and released the first version of its products in February 1999.
Due to the fact that nanoelectronic software components are so small that **nobody can see** it the company is not well known to the public.

Administration

First	Last	Title	Ext	EMail
Vernon	**Callaby**	Office Manager	582	v.callaby@nanonull.com
Frank	**Further**	Accounts Receivable	471	f.further@nanonull.com
Loby	**Matise**	Accounting Manager	963	l.matise@nanonull.com

Marketing

First	Last	Title	Ext	EMail
Joe	**Firstbread**	Marketing Manager Europe	621	j.firstbread@nanonull.com
Susi	**Sanna**	Art Director	753	s.sanna@nanonull.com

SOURCE: From Altova's XMLSpy 2004 on-line tutorial.

2. Platform Independent Transportation of Data

A platform independent method of transporting data is important in enterprise-wide applications. Many legacy systems are stove-piped, meaning that data comes in or out at the tops and bottoms, but there is very little cross-communication between adjacent systems. Stovepipes significantly reduce the value of data because it can't be leveraged for anything other than its originally intended purpose. As work processes and planning efforts are commingled, the end-user is left to manually translate data (even if done electronically) among the varied sources. The problem is especially exacerbated by proprietary and closed database structures used on different operating systems where electronic translation is difficult.

XML offers a solution to this problem of cross-communication. It is a completely open standard controlled by the World Wide Web Consortium (W3C) and has achieved wide support among software developers. There are literally hundreds of "markup languages" (think of them as dialects or vocabularies which conform to the XML standard) currently in use: MathML, ChemML, VoiceML, etc. Adopting an XML

vocabulary ensures that applications and processes can interchange and exchange information in a common format.

XML Transformations allow markup languages to be "translated" between one another. For example, the XYZ chemical manufacturer company could transform a supplier's data conforming to ChemML into their own proprietary "ChemXYZML". In this case, the ChemML serves as a platform independent method of transporting the data. The supplier's data might reside in a Unix relational database and the XYZ company data might reside on a WindowsNT hierarchical database… XML lets them exchange information without concern for the lower level physical system details.

E. MIDDLEWARE

Middleware is a general class of software used by applications to transfer data between databases or data storage systems. Most middleware is aimed at accessing data in relational databases using ODBC, JDBC, or OLE DB drivers, although some products exist for other types of structures such as multi-valued databases. Middleware can also transfer data between hierarchical databases, including XML. Middleware products range from home-grown projects to data conversion engines that cost tens of thousands of dollars.

There are many middleware applications on the commercial and open-source market. The "Big Four" database vendors (Oracle, IBM, Microsoft, and Sybase) all have integrated middleware functionality. Stand-alone middleware applications fill particular niches by focusing on graphical user interfaces (GUIs), application program interfaces (APIs), etc.

A form of middleware is being used by ASC-RT to synchronize the development and production Archibus databases. While they chose to develop a "one-off" home-grown middleware application, they could have chosen to implement an off-the-shelf product to accomplish the same goal.

I have discarded middleware as an acceptable technology for OMSI deliverables because it requires specialized knowledge to maintain changing data structures and new information. OMSI can be used by many different processes and it is too great a burden to develop subsequent middleware applications for each new use. The challenge is to

utilize a better technology that meets both the localized need of NAS Sigonella while better accommodating potential future needs.

THIS PAGE INTENTIONALLY LEFT BLANK

IV: Archibus Schema for Representing OMSI Data

A. Development History

Database-enabled OMSI data efforts began in earnest in the Spring of 2002. Representatives from LANTDIV, NAS Sigonella, ASC-RT, and the Syska met to discuss the nature of changes to the Archibus database schema needed to support OMSI integration. The addition of building systems was the first challenge. Archibus did not contain any building system information, although this would be essential for managing the OMSI information. After setting a direction for the inclusion of building systems, we proceeded to identify other shortcomings of Archibus for effective management of OMSI information.

1. Building Systems

The biggest change to the Archibus database schema was the addition of building systems. LANTDIV divides equipment into 12 building systems: conveying, electrical, exterior circulation, exterior closure, fire suppression, HVAC, interior construction, plumbing, roofing, site, specialties, and structural. A standardized system classification methodology was desirable in order to facilitate the decomposition of buildings into building systems and building systems into subcomponents.

It was agreed that the LANTDIV systems would fit well within either the Construction Specifications Institute (CSI) MasterFormat or the ASTM E1557-97 "Standard Classification of Building Elements and Related Sitework - UNIFORMAT II" [UNIFORMATII].

UNIFORMAT II is a National Institute of Standards and Technology (NIST) supported format for classifying building elements and related sitework. Elements are defined as major components common to most buildings and usually perform a given function, regardless of the design specification, construction method, or materials used. UNIFORMAT II ensures consistency at all stages of a building life cycle – planning, programming, design, construction, operations, and disposal.

UNIFORMAT II standardizes four levels of classification, as shown in the following table:

Table 3. UNIFORMAT Classification Levels

	Description
Level 1	This level, the largest element grouping, identifies Major Group Elements, such as the Substructure, Shell, and Interiors, etc.
Level 2	This level subdivides Level 1 Major Group Elements into Group Elements. The Shell Major Group, for example, includes the Superstructure, Exterior Closure, and Roofing Groups Elements.
Level 3	This level breaks the Level 2 Group Elements further into Individual Elements. The Exterior Closure Group Element, for example, includes Exterior Walls, Exterior Windows, and Exterior Doors Individual Elements.
Level 4	This proposed level breaks the Level 3 individual Elements into yet smaller sub-elements. Standard Foundation sub-elements, for example, include wall foundations, column foundations, perimeter drainage, and insulation.

UNIFORMAT II's most significant benefit is its applicability to a robust group of users. Owners, developers, programmers, cost planners, project managers, schedulers, architects and engineers, operating and maintenance staff, manufacturers, specification writers, and educators should all find the classification useful. For these reasons, it was decided that Archibus would categorize building systems by implementing UNIFORMAT II.

The OMSI specification was a "Scope of Work" text that described the information required in an OMSI delivery. As a starting point, each paragraph requirement was mapped to an Archibus table if it existed. If no Archibus table existed to hold the data requirement, it was flagged for later consideration to create new Archibus tables.

2. Document Management

The ability to manage documents did not exist within the framework of Archibus. However, it was clear that documents were essential to the creation of OMSI deliverables. As an intermediate solution to a full-fledged document management system (DMS), we agreed to create a database reference to documents stored on the network file system as a URL. This would allow CAFM users to query through building system data to find the appropriate URL, which could then be opened by its native application (i.e. Microsoft Word, Adobe Reader, etc.) through the use of a stored application (e.g. script)

in Archibus. A better solution for incorporating DMS functionality within the framework of OMSI deliverables is discussed below.

B. OMSI Entity Relationship Diagram

It is beneficial to diagram the entity relationships between the major OMSI data items. The diagram is useful for creating the database and performing validation such as normalization and relational integrity constraints.

Figure 10 uses IDEF1X [IDEF1X] notation to describe the OMSI data model as entities that have attributes and participate in relationships. IDEF1X is typically used to create a graphical information model which represents the structure and semantics of information within an environment or system.

Each relation (i.e. table) is shown as a rectangle with its name directly above and its attributes inside. Identifier dependency, shown by rounded corners, indicates a constraint between two related entities that requires the primary key in one (child entity) to contain the entire primary key of the other (parent entity). Identifying relationships (those in which every attribute in the primary key of the parent entity is contained in the primary key of the child entity) are shown as solid lines and non-identifying relationships are shown as dashed lines. The use of this notation will be useful in developing equivalent XML schemas later in the thesis. Relationships are show as lines with the following cardinality notations:

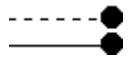
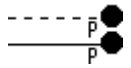
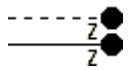
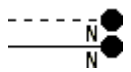
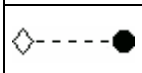
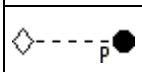
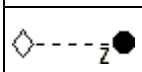
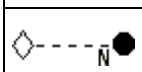
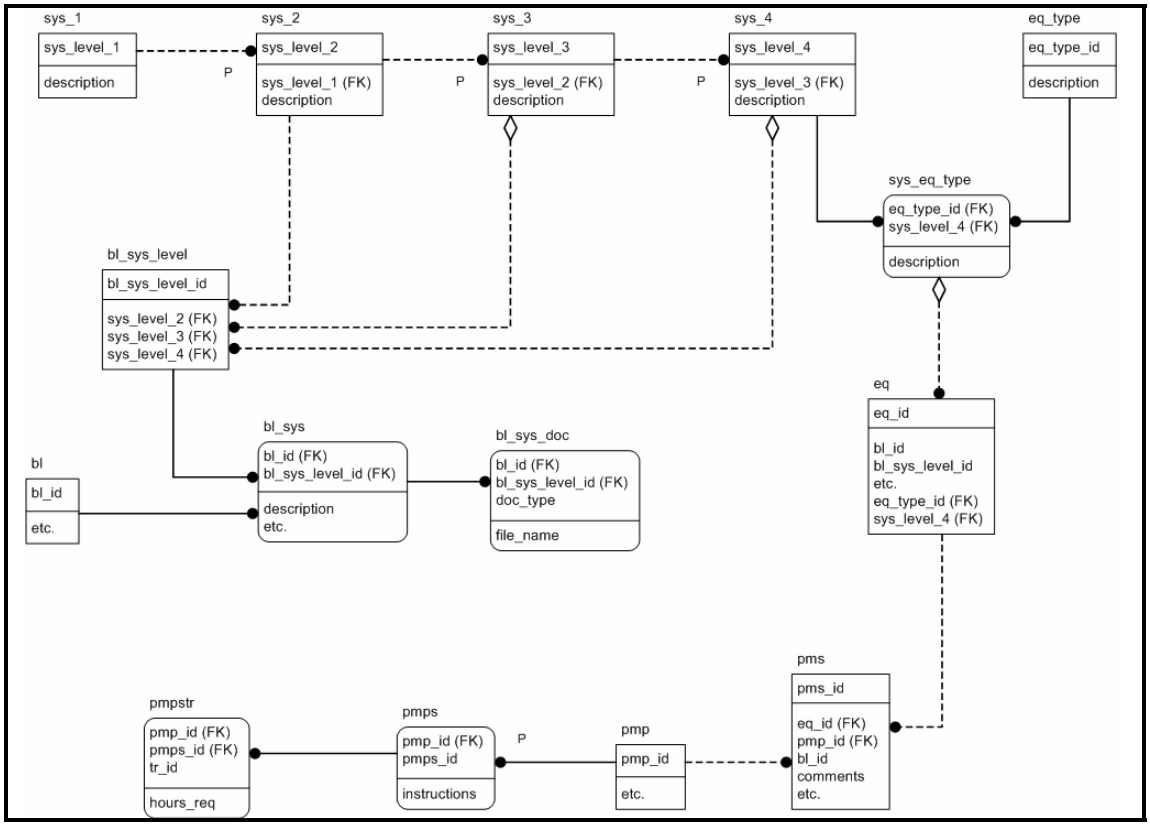
	One to zero or more
	One to one or more
	One to zero or one
	One to exactly N
	Zero or one to zero or more
	Zero or one to one or more
	Zero or one to zero or one
	Zero or one to exactly N

Figure 10. Entity Relationship Diagram for OMSI

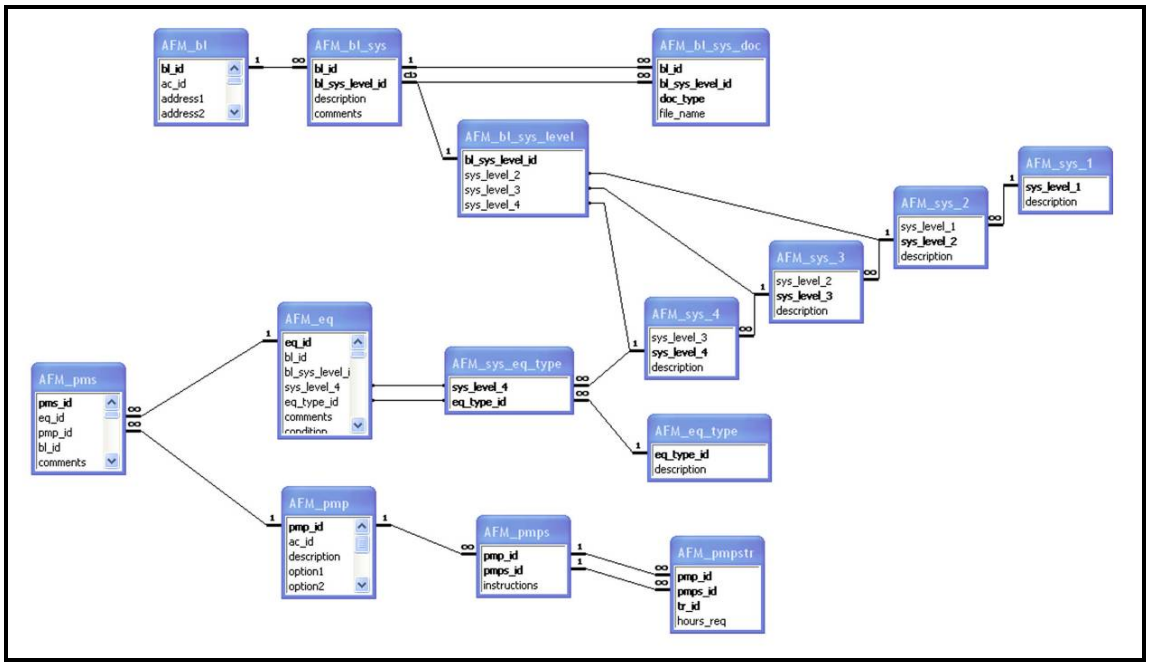


Not all the entities required to make complete OMSI deliverables have been included in Figure 11. In this sense, the results of my thesis are not production-ready (i.e. immediately ready for use), even if the additions are straightforward and uncomplicated. One reasoning for this incompleteness is the changing data requirements of Archibus. I did not model subcomponents of building areas (such as floors, rooms, and areas), equipment models and manufactures, equipment parts and warranties, or employee skill levels. Nonetheless, these entities are straightforward and the model can be easily extended include them. In addition, Navy Region Europe's Archibus Program Manager must decide on the scope of the Archibus schema changes that remain economical, considering the full life-cycle costs of propagating the changes to all European installations.

C. Database Relationships Diagram

In order for Archibus to receive OMSI deliverables, a database schema was developed using the Entity Relationship diagram in the previous section. This schema was created with the cooperation and assistance of ASC-RT, although the current production Archibus schema may differ. The Archibus schema relationships diagram is shown in Figure 10. Again, there are additional tables that must be populated by a complete OMSI deliverable, but they are uninteresting and not complicated.

Figure 11. Archibus Database Schema for OMSI



A list of the tables in the database schema, along with their brief descriptions, is shown in Table 4.

Table 4. Database Table Descriptions

Table	Description
AFM_sys_1	SYSTEM LEVEL *X*. The UNIFORMAT II system levels are represented as a set of related tables that are part of a strict hierarchy. See Table 3 for a more detailed description.
AFM_sys_2	
AFM_sys_3	
AFM_sys_4	
AFM_bl	BUILDING. This represents a physical building on an installation. It also represents abstract items such as roads, antennas, etc.
AFM_bl_sys	BUILDING SYSTEM. Each building can contain building systems such as HVAC, Fire Suppression, Plumbing, etc. A building system is a collection of equipment items that work in an interconnected fashion.
AFM_bl_sys_level	BUILDING SYSTEM LEVEL. Building systems are not all at the same UNIFORMAT II level. Most systems are at Level 3 but some systems are at Level 2 and Level 4.
AFM_bl_sys_doc	BUILDING SYSTEM DOCUMENT. Each building system has supporting documentation such as Safety Instructions, Normal Operating Procedures, etc.
AFM_eq_type	EQUIPMENT TYPE. Similar equipment items are grouped into an equipment type such as MOT (motor), PUM (pump), etc.
AFM_sys_eq_type	SYSTEM EQUIPMENT TYPE. Each equipment type is associated with one or more System Level 4s.
AFM_eq	EQUIPMENT. This represents any item that requires preventive maintenance. It is often the smallest piece of equipment that can receive "stand-alone" maintenance.
AFM_pms	PM SCHEDULE. This represents a one-to-one association between a piece of equipment and one of its required PM Procedures. Equipment can have many required PM procedures and the same PM procedure can be assigned to many pieces of equipment.
AFM_pmp	PM PROCEDURE. This represents a maintenance action that can be performed.
AFM_pmps	PM PROCEDURE STEP. This represents a discrete step of a PM procedure. In almost all cases, there is only one step per PM Procedure – this simplifies many aspects of PM management.
AFM_pmpstr	PM PROCEDURE TRADE. This represents a trade resource requirement needed to complete a PM Procedure Step.

The schema is straightforward, with the exception of the AFM_bl_sys_level and AFM_sys_eq_type tables. The AFM_bl_sys_level is necessary because a "Building System" (a record in the AFM_bl_sys table) may reside at the UNIFORMAT Level 2, Level 3, or Level 4. Given the nature of UNIFORMAT's strict hierarchy, it was preferred to model each level as its own table in the database. This would allow the systems to be easily extended in the future with a Level 5. However this created a problem with an AFM_bl_sys's primary keys. Specifically, in order to uniquely identify an AFM_bl_sys record, you need a primary key field for bl_id and either sys_level_2, sys_level_3, or sys_level_4. A primary key can never have a NULL value, and so the lack of a Level 3 or Level 4 causes a problem. To solve this issue of NULL primary keys, we could have put "pseudo-null" values in the AFM_sys_*x* tables, however this was discounted as an inelegant solution and the decision, and the AFM_bl_sys_level table was added instead.

The AFM_sys_eq_type table is used to ensure that the same equipment type is not inadvertently assigned to two different Level 4s albeit with a different name (perhaps due to a typographical error.)

D. DOCUMENT-CENTRIC INFORMATION

For the purposes of schema modeling, it is useful to distinguish between data- and document-centric information. However, this distinction is not always well established. Data-centric information is sometimes known as "structured" data, while document-centric information is sometimes known as semi-structured data. Given this, the structure of information can often lend clues to the nature of the information itself. When structures are rigid or applications that use the information demand a rigid structure (i.e. homogeneous), we can label the information data-centric. When the structures are free-form (though nonetheless following constraints) and information can be created within heterogeneous structures, we might label the information document-centric. There is undoubtedly a large domain of mixed information where data- and document-centric entities are intermixed.

Another difference between data- and document-centric information is the nature of how the information itself is used. Data-centric information tends to revolve around

data transport, that is, information must be transported between applications for aggregation, reporting, and analysis. Document-centric information, on the other hand, primarily tends to get transformed rather than transported, that is, information is repurposed for differing end-user consumption.

The information described in the database relationship diagram (Figure 11) is clearly all data-centric. A database is the prototypical data-centric storage system. However, the bl_sys_doc entity, which contains a significant portion of OMSI information, obviously contains reference to document-centric information by way of the document filename URL. These documents include:

- Training Recommendations
- System Description
- Start-Up and Shutdown Procedures
- Normal Operating Procedures
- Alternate Operating Procedures
- Emergency Operating Procedures
- System Flow Diagrams
- Environmental Considerations
- Operator Servicing Requirements
- Safety Instructions
- Troubleshooting Guides and Diagnostic Techniques

If these documents were to be represented as an XML document (perhaps as XHTML, DocBook, or a new non-standard markup language) they could be stored directly in a database. Each of the Big Four database vendors have created solutions for converting XML documents into their relational tables. Oracle's "XML SQL Utility", IBM's "XML Extender", Microsoft's "OPENXML row set", and Sybase's "ResultSetXML Java class" allow for the automatic conversion of XML data into and out of relational databases (Dayen 2001).

Nonetheless, it would require considerable work to place the document entities directly into Archibus and perform the necessary programming. Such a solution would also require rework if Archibus were converted to a different database engine. To avoid these prospects, I have proposed to create a field in Archibus to hold a URI that refers to an XML fragment or document that can be rendered in a consumer-specific presentation. The structure of this XML fragment or document will be discussed in Chapter VI.

Having developed and examined a relational data model for representing OMSI information in Archibus, we can now use its Entity Relationship diagram and general model rules to develop an equivalent XML data model. In the following chapter we will discuss the schema technologies for describing XML data models, schema structure considerations, and modeling rules. With these in mind, we will create an XML schema that represents the same information that Archibus holds.

THIS PAGE INTENTIONALLY LEFT BLANK

V: XML SCHEMA FOR REPRESENTING OMSI DATA

A. BACKGROUND

Chapter III discussed the requirements for well-formed XML. Yet well-formed XML places no restrictions on the structure of the XML document – and it is structure that provides semantic context to the information. While database schemas are an intrinsic feature of relational databases, XML uses extrinsic technologies called XML schemas to define the structure of XML documents. XML schemas, by defining structure, give rise to XML vocabularies or XML markup languages.

While there are numerous XML schema technologies, we only address the more popular ones: DTD, XML Schema, and RELAX NG. Some schema technologies have been standardized by the W3C (such as DTD and XML Schema) while others (e.g. RELAX NG) are de facto standards that are in the process of formal standardization.

The XML Schema presented here is able to represent all the information described by the Archibus database schema in the previous chapter. This includes a majority of the OMSI data required to support present-day deliverables needed by NASSIG. In this vein, the schema is appropriate for use in an XML environment while at the same time inserting structure suitable for conversion to a database. No attempt was made to represent each and every database schema object because, as discussed below, it is not necessary to explicitly represent all relational data entities and attributes in an XML schema. We call the vocabulary specified by this XML Schema OMSIML.

B. SCHEMA TECHNOLOGIES

XML schemas allow for additional control of XML documents beyond the XML syntax of being well-formed. Schemas can be used to validate an XML document against markup structure, identity integrity (e.g. key relations), data type constraints, and business rules. While there are many schema technologies no single one can be considered the "overall best"; each has some strength that gives rise to it being the best for a specific application.

1. DTDs

Document type declarations are defined by [XML1.0] to contain or point to markup declarations that provide a grammar for a class of documents. This grammar is known as document type definitions, or DTDs, and is the only schema embedded in [XML1.0] itself. XML Processors (a software module used to read XML documents and provide access to their content and structure) are classified as "validating" and "non-validating". [XML1.0] requires that "validating processors must, at user option, report violations of the constraints expressed by the declarations in the DTD, and failures to fulfill the validity constraints given in this specification." This will allow applications to off-load the validating function to a standard XML Processor.

While DTDs are an attractive technology for validating OMSIML documents, they have some weaknesses that make them generally unsuitable for use in conjunction with databases. The most glaring absence is data typing. DTD-specified elements have only four types of content models: Empty, Element, Mixed, and Any. These types specify whether an element may be empty, have text, have children elements, or have a combination of text and children. While elements that contain children can be specified as sequences or choices, there is no way to validate data types. This becomes important when using the XML to populate databases as will be done with OMSIML. DTD attributes have better typing, but they are still limited to just nine attribute types (CDATA, ID, IDREF, IDREFS, ENTITY, ENTITIES, NMTOKEN, NMTOKENS, and Enumerated List.)

DTDs also have a weakness in their method of element referencing. DTD's ID and IDREFs can be used for modeling relationships but unfortunately IDs must be unique within the XML document. This makes it difficult to model such things as database auto-numbered primary keys in two or more tables where there would surely be duplicate "1", "2", "3", etc. values.

Another weakness of DTDs is their syntax: they are not well-formed XML. While this doesn't currently create a problem for OMSIML, it has the unfortunate side-effect that DTDs cannot be accessed by XML Processors or modified by XML transformations such as XSLT.

2. XML Schema

The May 2001 W3C Recommendation XML Schema (consisting of Part 0: Primer, Part 1: Structures, and Part 2: Datatypes) [XMLSCHEMA] defines another schema vocabulary. "XML Schema" should not be mistaken for "XML schema" (note the capital "S") although the choice of name is certainly confusing. [XMLSCHEMA] has gained widespread use in the XML industry and is also a basis for the type system of other XML technologies such as XQuery, XPath 2.0 and XSLT 2.0 drafts.

[XMLSCHEMA], unlike DTDs, are represented using well-formed XML syntax (i.e. they are XML documents). This allows them to be processed using any XML Processor or transformed using XSLT. This isn't currently a necessity for use with OMSIML, but it does provide some measure of protection against future changes.

It is beyond this thesis to provide a complete description and explanation of [XMLSCHEMA]. It is sufficient to recognize that [XMLSCHEMA] has strong data typing as well as cardinality control, and it is this, combined with a method of intra-document referencing, that is essential to validating XML documents for use with relational databases.

3. RELAX NG

The Organization for the Advancement of Structured Information Standards (OASIS) is a not-for-profit, international consortium that drives the development, convergence, and adoption of e-business standards. In 2001, OASIS established a "Call for Participation" to establish a Technical Committee (TC) for creating a specification of a schema language for XML based on the TREX proposal. The TC eventually created RELAX NG [RELAXNG], a "specification for a language that validates XML documents, otherwise characterized as a simple schema language for XML which focuses upon description and validation of the structure and content of an XML document without attempting to specify application processing semantics" (OASIS 2003). It is billed as a simple, easy-to-learn schema language that includes both an XML and compact non-XML syntax. It supports XML namespaces, treats attributes uniformly within elements (so far as possible), has unrestricted support for unordered and mixed content, and can partner with a separate data typing language such as [XMLSCHEMA].

[RELAXNG] is also included as Part 2 of the ISO 19575 Document Schema Definition Language (DSDL) standard.

[RELAXNG] has some advantages over [XMLSCHEMA] such as being able to describe content dependencies (i.e. where elements or attributes are valid based on the value/presence of other elements or attributes). However, [RELAXNG] is missing one important capability necessary when validating XML for use with relational data, namely imposing identity constraints. As will be discussed later, identity constraint is an important method for intra-document referencing.

C. SCHEMA STRUCTURE CONSIDERATIONS

There are a few general considerations that should be given to any schema structure. While it is difficult to generate an exhaustive cookbook of guidelines, we discuss three reflections that provided significant improvement to the schema structure: the use of elements vs. attributes, the extent of hierarchical decomposition, and the concept of normalization of relations within the document. A brief examination of hierarchical composition led to an assessment of relation representation in XML which will be discussed in the section on modeling rules.

1. Elements vs. Attributes

On the surface it might seem an inconsequential act of schema construction to decide on when to use elements or attributes. After all, any attribute can be modeled as an element without changing the semantics of the schema. Nonetheless it is necessary to give considerable thought to determining a guideline for choosing element vs. attribute. The Department of the Navy first promulgated an XML Developer's Guide in October 2001. In April 2002 the XML Working Group of the U.S. Federal CIO Council's Architecture and Infrastructure Committee promulgated a draft Federal XML Developer's Guide (U.S. Federal CIO Council) [XMLGUIDE] which is a DON-approved adaptation of the consensus draft of the DON XML Developer's Guide v1.1.

[XMLGUIDE] provides direction on the choice of element vs. attribute as well as other useful rules for component naming and case convention, schema design, and document versioning. [XMLGUIDE] uses the terms MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and

OPTIONAL to denote the nature of conformance. Given the presence of "must" conformance, the guide should be reviewed in its entirety prior to implementing OMSIML in a wide-scale production environment. [XMLGUIDE] provides the following direction on element vs. attribute selection:

> The use of attributes SHOULD be carefully considered. Attributes SHOULD only be used to convey metadata that will not be parsed. Attributes, if used, SHOULD provide extra metadata required to better understand the business value of an element.
>
> Some additional guidelines are:
>
> - Attribute values SHOULD be short, preferably numbers or conforming to the XML Name Token convention. Attributes with long string values SHOULD NOT be created.
> - Attributes SHOULD only be used to describe information units that cannot or will not be further extended or subdivided.
> - Information specific to an application or database MUST NOT be expressed as values of attributes (see Section 4.3.1).
> - Use attributes to provide metadata that describes the entire contents of an element. If the element has children, any attributes should be generally applicable to all the children.

In consideration of [XMLGUIDE]'s admonitions against attribute use, we decided to use attributes only in the instance of adding auto-numbered identifiers to be used as primary keys in the relational database. However, attributes were not used when referencing these auto-numbered identifiers in other elements for the purpose of maintaining referential integrity. This not only complies with [XMLGUIDE], but more importantly, ensures that the schema can be easily extended in the future.

There are also other considerations for deciding on elements vs. attributes. If elements are considered the containers for data, attributes can provide additional information on the content of the element. Chris Brandin discusses six pitfalls that often occur when modeling information with XML (Brandin 2003):

- Inadequate context describing what a data element is (incomplete use of tags)
- Inadequate instructions on how to interpret data elements (incomplete use of attributes)
- Use of attributes as data elements (improper use of attributes)

- Use of data elements as metadata instead of using tags (indirection through use of name/value pairings)
- Unnecessary, unrelated, or redundant tags (poor hierarchy construction)
- Attributes that have nothing to do with data element interpretation (poor hierarchy construction or misuse of attributes)

The difficulty with XML modeling lies with the fact that two semantically equal models can require very different levels of effort to maintain. The pitfalls discussed by Brandin can certainly cause significant maintenance efforts as the model is progressively refined. The schema we present here have minimized the use of attributes and, accordingly, avoid Brandin's pitfalls.

2. Hierarchical Composition

Hierarchical composition is a fundamental construct of the XML data model. It creates a parent/child/sibling context that defines inherent relations in the data. Using hierarchical composition we simplify the OMSIML schema design by avoiding the complication of representing relations between non-hierarchical entities. This makes for an elegant schema that provides a straightforward method of ensuring the consistency and integrity of data.

Hierarchical composition can be taken too far when it becomes awkward to traverse the tree from the root to a leaf many levels away. This is especially true when creating XML transformations that require XPath statements. The guideline adopted is to limit the schema to five levels of hierarchy but, in a few instances, it was determined that the elegance of using more levels outweighed the burden of traversing the nodes in XPath.

3. Normalization

The concept of normalization is deeply rooted in the relational database realm. Database designers recognize that while there can be many ways to model the same data, not all relations are equally attractive. Some relations can create instances where changing the data can have undesirable consequences. These consequences are called "modification anomalies" and through careful consideration they can be avoided whenever desired. The process of redefining relations to avoid these anomalies is called "normalization".

Normalization of XML schemas is also a desirable process. *[N.B. Normalization of XML schemas is not to be confused with normalization of XML documents, which entails the removal of whitespace to facilitate document comparison.]* Normalized schemas can eliminate ambiguity of data expression, minimize redundancy, and help maintain data consistency. Will Provost developed guidelines for designing XML Schemas that achieve these goals in "Normalizing XML" (Provost 2002). The following paragraphs highlight the aspects of Provost's guidelines relevant to OMSIML.

First normal form is inherent for all valid relations (tables) in a relational database – it requires that attributes be single valued and each tuple (record) must have the same attributes in the same order. These requirements are clearly not necessary for XML given its heterogeneous nature which allows great flexibility in modeling data.

Second normal form requires that all of a relation's nonkey attributes be dependent on all of its keys. Third normal form requires there be no transitive dependencies among a relation's keys. Together, these two normal forms translate to the common vernacular of "nonkey attributes must depend on the keys, the whole keys, and nothing but the keys." The importance of second and third normal forms is their avoidance of insertion and deletion anomalies. They ensure that tuples can be added independently of one another and that as tuples are deleted there is no loss of data dependency information. Second and third normal form can be accomplished in XML Schemas through the use of the key() and keyref() definitions. Unfortunately, XML is not as straightforward as relational data in the application of second and third normal form. It is not always desirable to perform this type of normalization. While relational data keys must be unique within the scope of the relation, XML Schema keys are unique within the scope of an element and XML Schema keyrefs cannot be defined to traverse multiple scopes. XML Schema designers must take care to ensure that all desired associations are asserted even when complicated by differing scopes. Provost offers some design standards for avoiding these issues.

Fourth normal form requires that a relation have no multi-valued dependencies. Multi-value dependencies are instances where a relation has two or more nonkey attributes that are determined by the same key but the nonkey attributes are not dependent on each other. Failure to maintain fourth normal form requires additional tuples to avoid

misleading assumptions about nonkey dependencies. This is patently inefficient and can create update anomalies. Fourth normal form is easily achieved by placing the non-dependent attributes into separate tables. XML's heterogeneous nature obviates the need to maintain fourth normal form. Without the need to preserve rectangular structure, XML documents can easily remain consistent in the presence of multi-valued dependencies.

D. MODELING RULES

In addition to the schema structure considerations of the previous section, there are also modeling rules that can help generate an elegant XML schema based on a known relational data model. Prior to beginning work on an XML schema for OMSI, we created a relational model for holding the OMSI information. This model is desirable as a guideline for developing the XML schema for two reasons: leveraging an existing model is less effort than reinventing the wheel, and similar schemas enable the transformation from XML to Archibus in an easier fashion. This class of schema translation is generally known as "schema conversion". It is different from "schema matching" (which transforms data between two known schemas) although a converted schema would be an easy candidate for schema matching as well.

The NIKE (Nittany Information, Knowledge and wEb) Research Group published "Schema Conversion Methods between XML and Relational Models" (Lee, Mani et al. 2002) which detailed methods of schema conversion. Their challenge was to develop methods which capture both the structure of the schema as well as the semantic constraints. One of these methods, the Constraints-based Translation Algorithm (CoT) "capture[s] the overall picture of relational schema where multiple tables are interconnected … [by] consider[ing] inclusion dependencies during the translation, and merg[ing] multiple inter-connected tables into a coherent and hierarchical parent-child structure in the final XML schema." While such a rigorous algorithm was unjustified given the scope of this thesis, it nonetheless represents an interesting approach to automating the work required to generate XML schemas.

In this section we discuss a modeling "recipe" for developing an XML schema from a relational schema and approaches for representing relationships in XML. These two considerations were essential to the development of OMSIML: whatever XML

content is created by OMSI deliverables, some of its information has ultimately to be stored in Archibus. Given the existence of a relational schema for this information, it is necessary to create an equivalent schema in XML and create transformations between the two.

1. Modeling Recipes

"Professional XML Databases" (Williams, Brundage et al. 2000) offers the following 11 rules for developing an XML structure from a relational database:

Rule 1: Choose the Data to Include.

Based on the business requirement the XML document will be fulfilling, we decide which tables and columns from your relational database will need to be included in our documents.

Rule 2: Create a Root Element.

Create a root element for the document. We add the root element to our DTD, and declare any attributes of that element that are required to hold additional semantic information (such as routing information). Root element's names should describe their content.

Rule 3: Model the Content Tables.

Create an element in the DTD for each content table we have chosen to model. Declare these elements as EMPTY for now.

Rule 4: Modeling Non-Foreign Key Columns.

Create an attribute for each column we have chosen to include in our XML document (except foreign key columns). These attributes should appear in the !ATTLIST declaration of the element corresponding to the table in which they appear. Declare each of these attributes as CDATA, and declare it as #IMPLIED or #REQUIRED depending on whether the original column allows NULLS or not.

Rule 5: Add ID Attributes to the Elements.

Add an ID attribute to each of the elements you have created in our XML structure (with the exception of the root element). Use the element name followed by ID for the name of the new attribute, watching as always for name collisions. Declare the attribute as type ID, and #REQUIRED.

Rule 6: Representing Lookup Tables.

For each foreign key that we have chosen to include in our XML structures that references a lookup table:

1. Create an attribute on the element representing the table in which the foreign key is found.
2. Give the attribute the same name as the table referenced by the foreign key, and make it #REQUIRED if the foreign key does not allow NULLS or #IMPLIED otherwise.
3. Make the attribute of the enumerated list type. The allowable values should be some human-readable form of the description column for all rows in the lookup table.

Rule 7: Adding Element Content to Root elements.

Add a child element or elements to the allowable content of the root element for each table that models the type of information we want to represent in our document.

Rule 8: Adding Relationships through Containment.

For each relationship we have defined, if the relationship is one-to-one or one -to-many in the direction it is being navigated, and no other relationship leads to the child within the selected subset, then add the child element as element content of the parent element with the appropriate cardinality.

Rule 9: Adding Relationships using IDREF/IDREFS.

Identify each relationship that is many-to-one in the direction we have defined it, or whose child is the child in more than one relationship we have defined. For each of these relationships, add an IDREF or IDREFS attribute to the element on the parent side of the relationship, which points to the ID of the element on the child side of the relationship.

Rule 10: Add Missing Elements.

For any element that is only pointed to in the structure created so far, add that element as allowable element content of the root element. Set the cardinality suffix of the element being added to *.

Rule 11: Remove Unwanted ID Attributes.

Remove ID attributes that are not referenced by IDREF or IDREFS attributes elsewhere in the XML structures.

Although these rules are for generating DTDs as opposed to XML Schemas, the logic is certainly valid, and relevant to developing OMSIML. Many of the rules are obvious, but Rules 8 and 9 (adding relationships) will merit further discussion.

Ron Bourret has also written extensively on the topic of mapping DTDs and relational database schemas (Bourret 2002). He classified two mapping strategies (table-based mapping and object-relation mapping) that are good candidates for mapping data-centric XML documents to relational databases. Bourret's approach is much more methodical than the recipe of Williams, Brundage et al, though it is more detailed than OMSIML warrants.

2. Representing Relationships in XML

Jeff Ryan describes three methods for representing relationships in XML in his article "Modeling One-To-Many Relationships With XML" (Ryan 2003). The first two methods are containment and intra-document references. Containment is the case of one element being contained within another element. Intra-document references are key/keyrefs (or ID/IDREFs with DTDs) that allow pointing within the XML document to be validated by using a schema. The third method, inter-document relations, can also be used to maintain relationships between XML documents. Similar to intra-document relations, it allows a pointer to an entity, but in a separate XML document. Its only notable advantage over intra-document relations is flexibility, which is not specifically needed within the scope of OMSIML.

Containment is the "stronger" and more elegant method of representing relationships if the semantic structure of the model permits. However, it can also be slower in some applications (Ruyak, Mathwani et al. 2003) and containment and normality can be mutually exclusive, since keys are the only way to enforce normality of many-to-many relationships. Intra-document referencing requires more thought to develop the schema but it can be extremely flexible. Table 5 rates the key decision factors for choosing containment vs. intra-document referencing.

Table 5. Comparison of Containment and Intra-document Referencing

	Containment	Intra-document Reference
Processing Speed[†]	Good	Excellent
Data Passing	Excellent	Good
Flexibility	Fair	Good
Ease of Use	Excellent	Good

[†] Processing speed comparison is based on results presented in "Optimizing XML Processing for Performance" (Ruyak, Mathwani et al. 2003)

SOURCE: Adapted from "Modeling One-to-Many Relationships with XML" (Ryan 2003)

The rules presented in the modeling recipe of the previous section provide some useful guidance about when to choose containment vs. intra-document reference in the cases where the model demands a specific choice. But there are also many instances when either decision is semantically correct. In such cases, containment may be preferred for data passing or ease of use while intra-document reference may be preferred for processing speed or flexibility. Containment may also be avoided in cases where the hierarchical composition creates a burden for traversing the nodes or accessing the leaves with long XPath statements.

E. OMSI XML SCHEMA

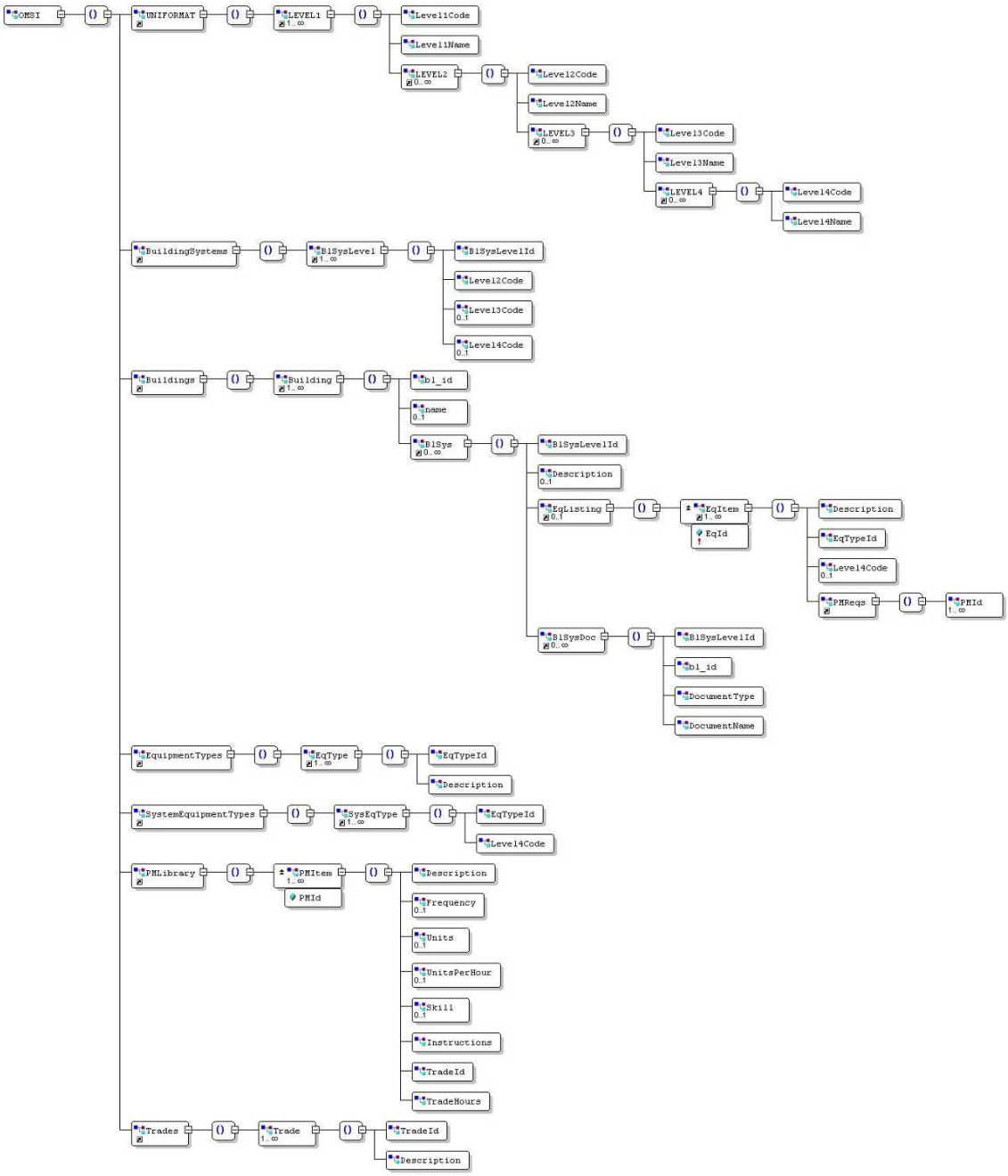
The OMSI XML Schema diagram shown in Figure 12 describes the OMSIML model. The complete documentation of the XML Schema can be found in Appendix B.

This schema was developed using the IDEF1x diagram of the OMSI entity relationships. This ensures that the XML schema is modeled in a consistent and efficient manner by considering the presence of identifier dependency entities and identifying relations. As discussed in Chapter IV, the identifier dependency parent entities have all their primary keys contained in the child entity. This creates an obvious choice for representing the relationship through containment because the primary keys can be left out of the children entities and regenerated (if need be) from the parent entity. Non-identifier dependency entities merit special attention because containment must be augmented with keys to ensure the non-identifying relations are modeled properly.

Starting with the model of the UNIFORMAT system levels, these levels are a clear hierarchical dependency and can be appropriately modeled with containment. We chose to limit the containment to just the sys_levels in order to ensure a sustainable model that can easily be adapted with additional levels in the future. The eq_type and sys_eq_type were modeled as their own elements (EquipmentTypes and SystemEquipmentTypes) and keys (XML Schema keys) were created on their primary keys for use with keyrefs in the appropriate entities. bl_sys_level was also represented as its own entity (BuildingSystems) because these are seen as a master list developed by the maintenance organizations and OMSI developers are not be free to create their own. Again, a key was created for use with keyrefs in related entities.

The Building model was generated using containment for the identifier dependencies (bl_sys and bl_sys_doc). The eq entity was added because in the real-world model, equipment is clearly contained in buildings. Non-identifying relations were modeled to to eq using keyrefs to the appropriate keys. The pms entity models a many-to-many intersection between the pmp and eq entities to capture the concept of PM requirements for equipment. This required embedding the appropriate PMId keyref in EqItem to represent the relationship.

Figure 12. OMSI XML Schema Diagram



In this chapter we have created an XML data model for representing OMSI information. The corresponding schema will be used to create instance documents that hold OMSI deliverables. In the next chapter we will address the storage of collections of

these instance documents by using the file system, a relational database, or a native-XML database. We will provide a cursory review of XML database products and suggest a storage mechanism for both data- and document-centric information.

THIS PAGE INTENTIONALLY LEFT BLANK

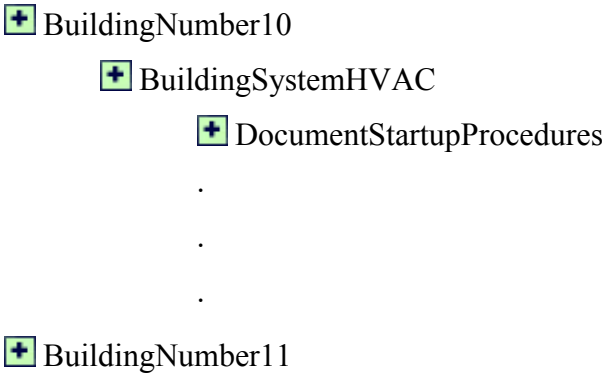
VI: STORAGE OF XML-BASED OMSI INFORMATION

A. STORAGE METHODS

There are three basic methods for storing XML documents: file systems, relational databases, and native XML databases. Ronald Bourret discusses these mechanisms in-depth in his seminal "XML and Databases" (Bourret 2003). In a strict sense, the use of a file system for storing XML documents isn't a storage method by itself. The "system" must include some means of adding, modifying, deleting, and querying the documents and it is only through other applications (even if they're built into the operating system) that these actions can be done with a file system. Relational and native XML databases, however, include such add, modify, delete, and query functions internally in their application.

1. File Systems

File systems can easily store and access XML documents. Metadata can be attached to the document itself or a directory structure can be used to imply semantics. For example, a directory structure of the form



makes it clear that Building System documents are grouped on Building Systems, which are in turn grouped on Buildings. With this knowledge, we can mentally query the metadata implied by the directory structure.

While it is easy to add or delete documents, file systems do not include a way of querying XML documents, with the exception of executing simple text searches. It is possible to build an application that could access these documents and apply XML technologies such as XPath to better query them, but it is clear that such a solution would be extremely cumbersome as well as quite difficult to maintain and extend.

2. Relational Databases

Relational databases can be used to store XML documents using either of two methods. The first method is to simply store the document in a table using a CLOB or BLOB and creating an API that can store, retrieve, and delete the entities. For example, XML DB, a feature of the Oracle Database, provides a native method of XML query, update, and transform. XML documents can be loaded into XMLType tables or XMLType columns in a database using Oracle's Procedural Language/SQL (PL/SQL) or JDBC. The other major database vendors (e.g. Microsoft, Sybase, etc.) include a similar manner of handling natively XML documents stored in a database table or column.

The second method of using a relational database to store XML documents is to create an equivalent relational schema to model the XML document. Chapter IV discussed the schema conversion methods of Lee, Mani, et al. These rules can be applied to convert the XML document to a relational schema and store the content in any relational database. While such a process is straightforward and there exist algorithms for performing the conversion, this method is not without a downside. Once the XML document is represented as relational data there is no possibility of applying other XML technologies such as XSLT, XQuery, and XPath. Perhaps more importantly, however, is the problem of round-tripping. Round-tripping is a situation in which an original XML document is put somewhere (in this case, a relational database) and then retrieved. One would expect the retrieved document to be identical to the original after taking the "round trip", but unfortunately, this is often not the case. Schema conversion is not a lossless process and semantics can be lost along the way. Although this problem is not particularly troublesome for data-centric XML, it can be disastrous for document-centric XML where the user expects the presentation of the document to be the same after round-tripping.

3. Native XML Databases

Native XML databases are the utopia for storing XML documents. First defined by the XML:DB Initiative, Kimbro Staken offers the following description of a native XML database in "Introduction to Native XML Databases" (Staken 2001):

> - Defines a (logical) model for an XML document -- as opposed to the data in that document -- and stores and retrieves documents according

to that model. At a minimum, the model must include elements, attributes, PCDATA, and document order. Examples of such models are the XPath data model, the XML Infoset, and the models implied by the DOM and the events in SAX 1.0.

- Has an XML document as its fundamental unit of (logical) storage, just as a relational database has a row in a table as its fundamental unit of (logical) storage.

- Is not required to have any particular underlying physical storage model. For example, it can be built on a relational, hierarchical, or object-oriented database, or use a proprietary storage format such as indexed, compressed files.

This definition is quite useful because it requires that the "database" act on whole XML documents using a document-like model such as XPath, but it doesn't require a specific database technology. Clearly, native XML databases go beyond the data store. Jim Tivy suggests that one can recognize a native XML database by examining how the data is modeled to the programmer (Chamberlin, Draper et al. 2003). Native XML databases use a model that respects the structure of XML in addition to working with other XML technologies such as XML Schema, XPath, XQuery, etc. This fits quite well with the XML:DB Initiative's definition.

In addition to having XML documents (the fundamental unit in the data store), native XML databases also have "collections". These collections, analogous to relational tables, are also a defining feature. Relational tables are constrained so that each record in the table must conform to the same schema. There is no such explicit requirement in native XML databases. While some products can require that all documents in a collection conform to the same schema (by performing validation against a DTD, XML Schema, etc.), many products will allow the store of any well-formed XML document in a collection. This schema-independent nature of a collection can be quite powerful by facilitating queries across a collection of diverse documents.

Native XML databases were first queried using XPath 1.0 [XPATH1.0], a sublanguage of XSLT 1.0 [XSLT1.0]. However, these technologies were not designed with database querying in mind; rather they were intended as a language for transforming an XML document. Users of the native XML databases were addressing a very different

scenario of wanting to extract information from the large collections of documents, and this, in part, led to the development of a new XML query language called XQuery 1.0 [XQUERY]. We will discuss these two "competing" technologies, especially as they relate to transforming, in Chapter VII.

Native XML databases are especially fitting for the OMSI content because they support both data- and document-centric objects equally well. Bourret lists over 35 native XML database products, including dbXML (dbXML Group), eXist (Wolfgang Meier), eXtc (M/Gateway Developments Ltd.), eXtensible Information Server (XIS) (eXcelon Corp.), GoXML DB (XML Global), Ipedo (Ipedo), Neocore XML Management System (NeoCore), ozone (ozone-db.org), SQL/XML-IMDB (QuiLogic), Tamino (Software AG), TOTAL XML (Cincom), X-Hive/DB (X-Hive Corporation), and Xindice (Apache Software Foundation).

B. XML DATABASE PRODUCTS

Ron Bourret maintains an extensive listing of XML database products, which he classifies into eight categories (Table 6).

Table 6. Classifications of XML Database Products

	Description	Application
Middleware	Software you call from your application to transfer data between XML documents and databases.	data-centric
XML-Enabled Databases	Databases with extensions for transferring data between XML documents and themselves.	data-centric
Native XML Databases	Databases that store XML in "native" form, generally as some variant of the DOM mapped to an underlying data store. This includes the category formerly known as persistent DOM (PDOM) implementations.	data- and document-centric
XML Servers	XML-aware J2EE servers, Web application servers, integration engines, and custom servers. Some of these are used to build distributed applications while others are used simply to publish XML documents to the Web. Includes the category formerly known as XML application servers.	data- and document-centric
Wrappers	Software that treats XML documents as a source of relational data. These products typically query XML documents using SQL.	data-centric
Content Management Systems	Applications built on top of native XML databases and/or the file system for content/document management. Include features such as check-in/check-out, versioning, and editors.	document-centric
XML Query Engines	Standalone engines that can query XML documents.	data- and document-centric
XML Data Binding	Products that can bind XML documents to objects. Some of these can also store/retrieve objects from the database.	data-centric

SOURCE: From "XML Database Products" (Bourret 2004)

These classifications cover an overwhelming number of individual products; Bourret lists almost 200 of them. It is patently clear that XML-based OMSI information can be stored

and accessed in many different manners. For the purposes of this thesis we have chosen to sidestep this important aspect of developing an XML framework for OMSI for two principle reasons:

(1) The manner in which OMSI information might be consumed (apart from within the CAFM) is uncertain. Undoubtedly this is because there are no current users communicating their needs to consume OMSI in non-traditional ways. In this sense, it is our belief that this thesis will be quite useful in demonstrating the value of OMSI deliverables in the context of new modes of consumption.

(2) The evaluation of such a large number of products is beyond the scope of the thesis.

C. DATA-CENTRIC STORAGE

The bulk of the data generated for OMSI is inherently data-centric. Indeed, it was the data centricity and a desire to incorporate it into a CAFM that provided the impetus for this thesis. Given the XML Schema used to represent this data, it is feasible to create a single XML document for each OMSI deliverable (typically a building or small group of buildings). As the final deliverable is imported into the CAFM, there is no long-term concern of creating an unmanageable list of documents because a single XML document could be generated from Archibus. In the event that the OMSI deliverable contains information not represented in Archibus, it is still possible to parse all the XML documents and combine them into a single XML document.

D. DOCUMENT-CENTRIC STORAGE

Although only a small part of the total OMSI data is document-centric, there are nonetheless noteworthy requirements for generating documents. The twelve document types listed in Chapter IV have historically been created in Microsoft Word, although very little presentation markup has been used. This makes for a very static mode of consumption.

These documents could easily be stored in a Document Management System (DMS). In the preliminary discussions between LANTDIV and NAS Sigonella on how OMSI information would best be delivered electronically, it was requested that the OMSI

A/E integrate the Word documents into the Microsoft Sharepoint Portal Server (SPS) maintained by the Public Works Department. Of course, hyperlinks to the documents stored in SPS would need to be inserted into Archibus for the CAFM users, but it was envisioned other end-users appreciating the ability to query the documents in a more robust environment.

Preliminary investigations into the XML database products soon revealed the benefits of storing the 12 OMSI document types using an XML model. Apart from being able to query the document content in the XML framework used for the rest of OMSI, it would be quite useful to be able to dynamically repurpose the content for other consumers. Another important improvement of the XML model is the addition of semantics to the information content. As an example, consider the previous deliveries of PDF documents that might have included safety admonitions. While it is possible to index a PDF document on SPS, it is quite difficult to modify the document for different uses. It is also challenging to add semantic markup to the PDF text to indicate the existence of safety admonitions. Imagine an HVAC technician wanting to take building air distribution and electrical schematics into the field. He'd have to print the schematics and the safety admonitions and hope he brought the ones he'd need. He could even overlook a critical safety admonition. If the schematics and safety admonitions were stored in an XML framework, the content could be easily repurposed for a mobile device (e.g. PocketPC) on the server-side and the technician could consume just the ones he wanted while in the field.

The requirement to store the documents in an XML framework led to an important realization: there already existed an XML schema for describing the content of technical documents. This discovery of DocBook would be extremely useful for OMSI.

1. DocBook Background

DocBook is a standard set of XML markup tags that are used to describe the content of books, articles, and other technical documents. The DocBook schema was first published in 1991 as a joint project between HaL Computer Systems and O'Reilly to facilitate the exchange of UNIX documentation originally marked up in troff. In 1994 DocBook maintenance was taken over by the Davenport Group. A Technical Committee (TC) of the Organization for the Advancement of Structured Information Standards

(OASIS) was formed in 1998 to further develop and maintain the DocBook DTD. In its present form, DocBook v4.3, there is not an official XML Schema, although the DTD has been used to derive Schemas for general use. The TC intends to publish an official XML Schema with V5.0.

DocBook can be used as a foundation for a publishing system. It is especially well-suited in cases where there are large quantities of content which is highly structured, and which is to be interchanged between incompatible systems, or rendered in multiple output forms and versions (Strayton 2003). A DocBook is an article, a book, or a set (of books). Books may contain BookInfo (title, author, copyright, etc.), prefaces, chapters, and appendixes, bibliographies, glossaries, indices, and a colophon. An article contains just a body (similar to a book chapter), appendices, bibliographies, indices, and glossaries.

DocBook has been used in a wide range of applications, including electronic books and articles, books for print (especially by O'Reilly), website maintenance, computer documentation, training material, Questions and Answer FAQs, etc. These broad applications have generated a need for a "simple" DocBook with a reduced number of elements to keep new users from being overwhelmed; Simplified DocBook is currently a Working Draft 1.1b3 that contains just 106 elements, 525 entities, and 26 notations. DocBook v4.3 contains over 400 elements alone! A Simplified DocBook must be a subset of DocBook, is limited to articles only (single documents such as white papers, etc.), and must support online browser transformations (meaning that is must be small enough to download transparently to the user). Unfortunately, Simplified DocBook may not contain all the elements needed for OMSI documentation.

The DocBook schema is well-suited for OMSI documents; the elements shown in Table 7 are of particular use.

Table 7. DocBook Elements Appropriate for OMSI

	Element	Description
Lists	CalloutList	A list of annotations or descriptions
	GlossList	A list of glossary terms and their definitions
	ItemizedList	An unordered (bulleted) list
	OrderedList	A numbered list
	SimpleList	An unadorned list of items
Admonitions	Caution	*Meanings are not specified by DocBook. OMSI can apply the standard definitions (WARNING: Misuse or failure to follow instructions properly may result in personal injury or death!; CAUTION: No risk of personal injury; however, misuse or failure to follow instructions may result in damage of equipment; and NOTE: No risk of personal injury or equipment damage; however, misuse or failure to follow instructions may prevent proper performance of the equipment)*
	Important	
	Note	
	Tip	
	Warning	
Miscellaneous	Procedure	A list of steps to be performed in a well-defined sequence
	ULink	A link that addresses its target by means of a URL

SOURCE: Descriptions from "DocBook: The Definitive Guide" (Walsh and Muellner 2003)

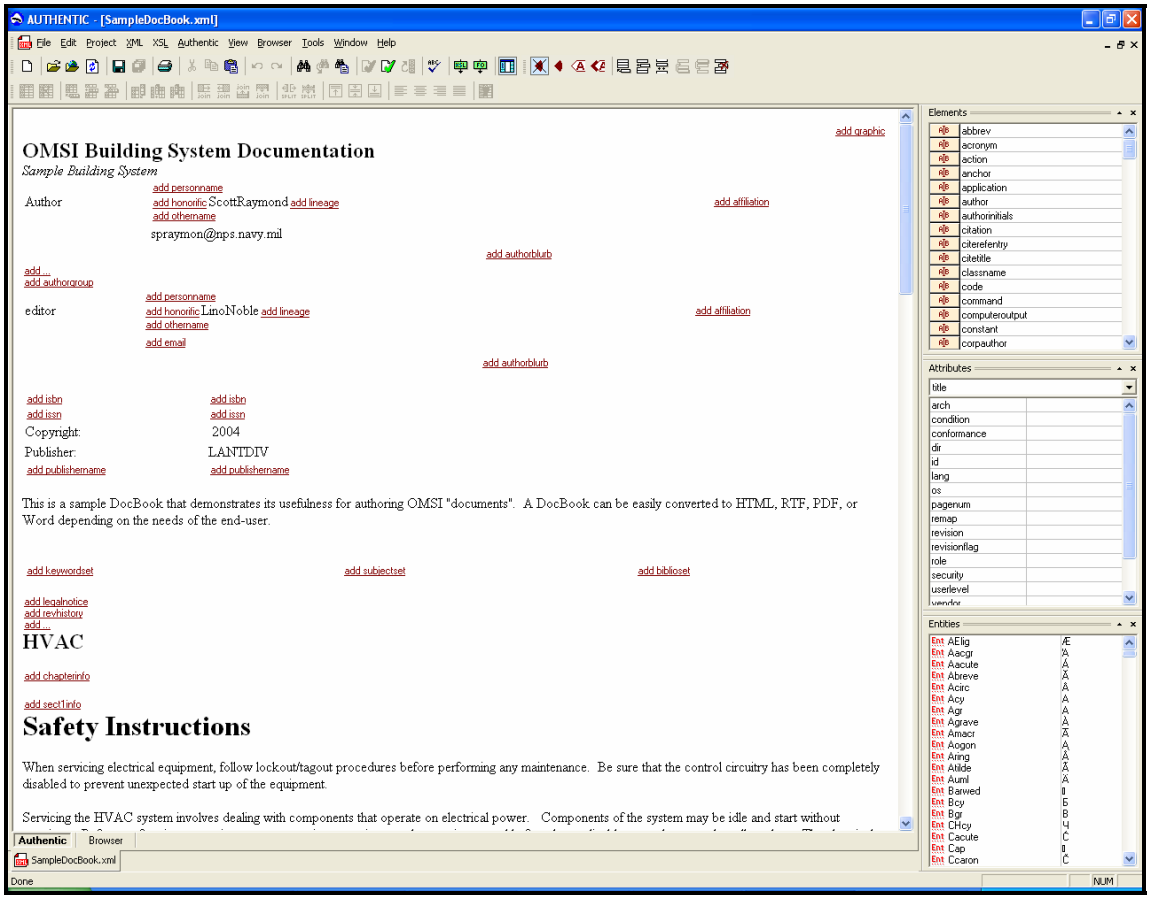
The admonitions are an excellent illustration of how semantics can be embedded into the OMSI documents. For example, consider the case of a supervisor wanting to ensure technicians consider safety requirements prior to beginning work in the field. The DocBook can be queried for admonitions and a safety checklist could be automatically generated in HTML.

2. Creating DocBooks

Although DocBooks are text documents that can be created or edited using any text editor, an integrated development environment (IDE) is essential for authoring DocBooks efficiently. I have examined two IDEs for creating DocBooks: Altova's Authentic™ [AUTHENTIC] and Pixware's XMLmind XML Editor [XXE]. Both applications provide WYSIWYG renditions of the DocBook within the editing process and include integrated spell-checkers.

Authentic comes with a stylesheet for creating DocBooks v4.2. Figure 13 shows a screenshot of Authentic being used to author a DocBook. Elements can be added using the in-document "add…" hyperlinks or by dragging an element from the list of valid elements depending on the insertion context. These methods save the author significant time and frustration by avoiding the need to manage start/end tags and ensure the XML is a valid DocBook. Authentic is free as a desktop application or a web browser plug in for Internet Explorer.

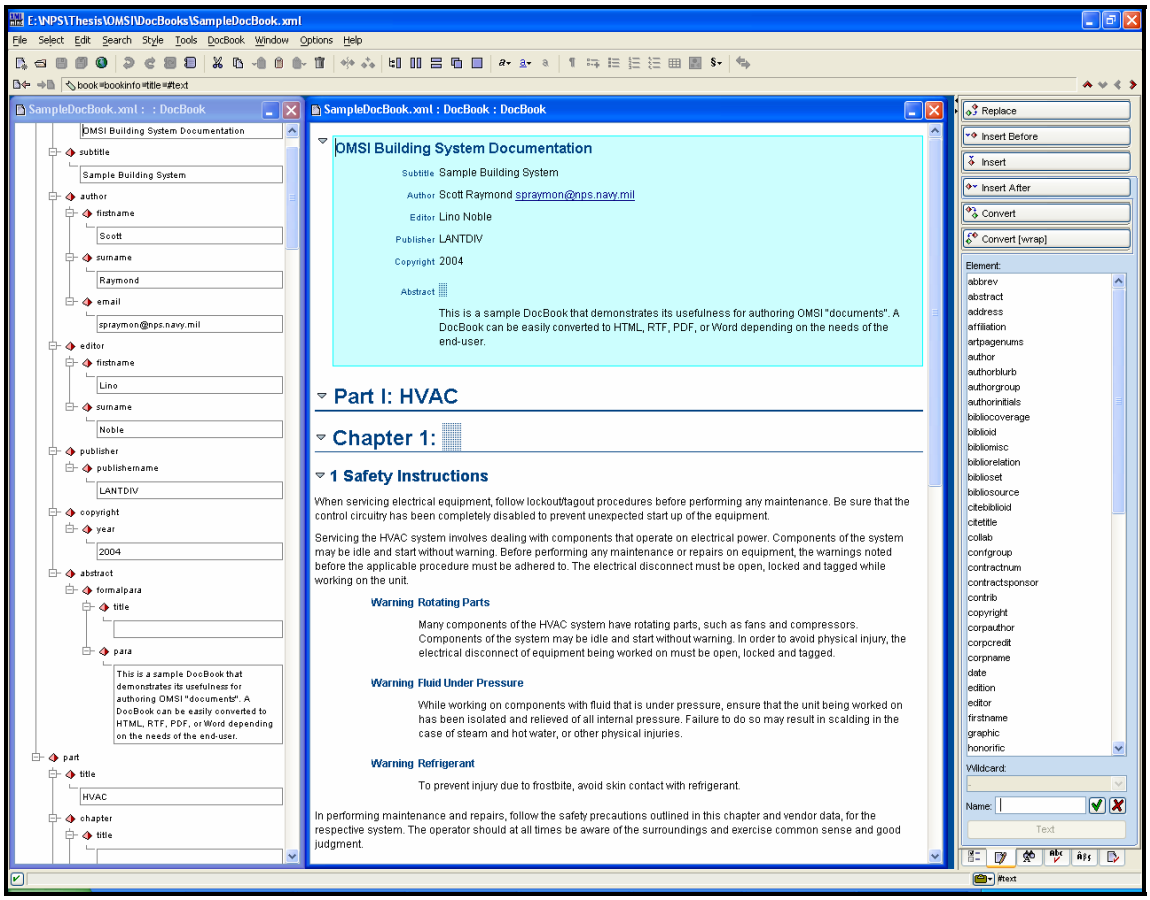
Figure 13. Using Authentic for Authoring a DocBook



XXE is available in Standard and Professional versions. While the standard version is free, it does not have the complete feature set found in the professional version. Most notably missing from the standard version is the ability to use FO processor plug-ins (which would allow rendering of HTML, PDF, or Word output from within XXE) and to modify or upload files stored on a FTP or WebDAV server. Figure 14 shows a screenshot of XXE being used to author a DocBook. Notice that there are simultaneous

tree and style views of the DocBook. Similar to Authentic, XXE presents a list of valid elements depending on context to ensure the validity of the DocBook.

Figure 14. Using XXE for Authoring a DocBook



There are many other IDEs that can be used to create DocBooks. Most are incorporated with an XML editor (in the same fashion as XXE) although some are dedicated DocBook applications. One that might warrant further investigation is DMSi's SyntoniX™ [SYNTONIX], which capitalizes on Microsoft Office's "Smart Documents" feature that guides the user through the process of creating a document. SyntoniX has the advantage of letting the author use Microsoft Word to create a Simplified DocBook[1]. This significantly decreases the learning time necessary for creating a DocBook and allows the use of a familiar environment for authoring.

[1] SYNTONIX does not have a DocBook template, perhaps because of the complexity required to support all DocBook elements. However, DMSi does create custom solutions for non-industry standards, which might present an opportunity for creating an OMSI-specific subset of DocBook elements.

3. Rendering DocBooks with Styling

DocBooks are not meant for direct consumption by the end-user (i.e. reader). They must be styled as part of the publishing system – styled in the sense that transforming and formatting must be done in a manner that creates an object viewable or printable by the reader. The W3C has authored the Extensible Stylesheet Language family (XSL) of recommendations, which consists of three parts:

- XSL Transformations (XSLT) - a language for transforming XML;
- the XML Path Language (XPath) - an expression language used by XSLT to access or refer to parts of an XML document. (XPath is also used by the XML Linking specification); and
- XSL Formatting Objects (XSL-FO) - an XML vocabulary for specifying formatting semantics

XSL-FO are XML documents that includes output information. They serve as the pillars of a DocBook publishing system by incorporating styling information that is needed for paginated rendering. The XSL-FO drive a Formatting Objects processor which is responsible for creating the final (consumable) object, which can be a PDF, PCL, PS, SVG, XML, Print, AWT, MIF or TXT document. PDFs are often the primary output because PDF readers are ubiquitous and cross-platform. (Holman 2003)

Two well-known Formatting Objects processors are Apache's FOP (Formatting Objects Processor) [FOP] and the XEP Rendering Engine [XEP]. FOP, licensed under the Apache Software License, is freely available whereas XEP must be purchased.

We have examined two applications for transforming DocBooks into PDF documents: XXE (as discussed above) and the DocBook Toolchain Manager (DocMan) [DOCMAN]. DocMan is a free Java program that transforms DocBooks into HTML, XHTML, PDF and CHM documents. It has a simple interface that also allows batch processing of files. Both applications represent an effective and efficient method of producing PDFs for OMSI deliverables.

Figure 15 shows a sample DocBook fragment and its equivalent automatically rendered presentation. The Table of Contents is automatically generated by parsing the entire DocBook for section tags. Notice that the sections are not explicitly numbered; the rendering process can assign numbers (using whatever convention desired) as the DocBook is parsed. Another useful feature of rendering DocBook presentations is the

insertion of standard admonition (e.g. Warning) graphics. Other features, such as automatic numbering of procedure elements, are also useful.

Figure 15. DocBook Fragment and Automatically Rendered Presentation

```
<part>
   <title>HVAC</title>
   <chapter>
      <title></title>
      <sect1>
         <title>Safety Instructions</title>
         <para>When servicing electrical equipment, follow lockout/tagout procedures
before performing any maintenance.  Be sure that the control circuitry has been
completely disabled to prevent unexpected start up of the equipment.</para>
         <para>Servicing the HVAC system involves dealing with components that operate
on electrical power.  Components of the system may be idle and start without warning.
Before performing any maintenance or repairs on equipment, the warnings noted before the
applicable procedure must be adhered to.  The electrical disconnect must be open, locked
and tagged while working on the unit.</para>
         <warning>
            <title>Rotating Parts</title>
            <para>Many components of the HVAC system have rotating parts, such as fans
and compressors.  Components of the system may be idle and start without warning.  In
order to avoid physical injury, the electrical disconnect of equipment being worked on
must be open, locked and tagged.</para>
         </warning>
```

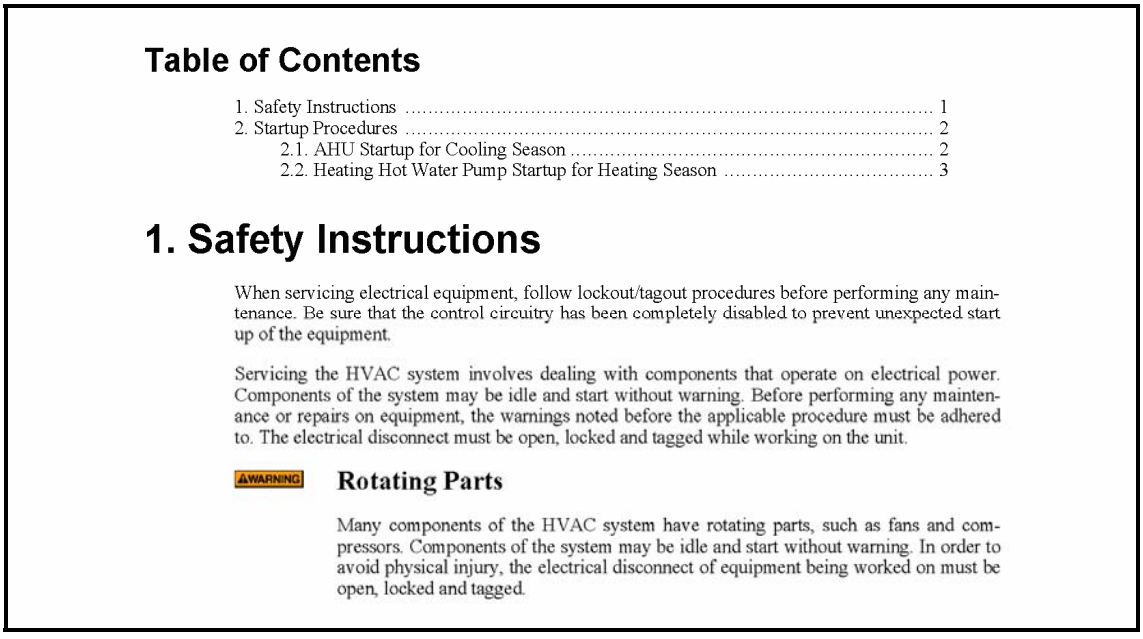


Table of Contents

1. Safety Instructions

When servicing electrical equipment, follow lockout/tagout procedures before performing any maintenance. Be sure that the control circuitry has been completely disabled to prevent unexpected start up of the equipment.

Servicing the HVAC system involves dealing with components that operate on electrical power. Components of the system may be idle and start without warning. Before performing any maintenance or repairs on equipment, the warnings noted before the applicable procedure must be adhered to. The electrical disconnect must be open, locked and tagged while working on the unit.

WARNING

Rotating Parts

Many components of the HVAC system have rotating parts, such as fans and compressors. Components of the system may be idle and start without warning. In order to avoid physical injury, the electrical disconnect of equipment being worked on must be open, locked and tagged.

Note: The Table of Contents is automatically generated based on the <sect> tags. Admonition (e.g. Warning) graphics are also automatically inserted.

4. Sample OMSI DocBook

We have developed a sample DocBook for OMSI. Appendix C contains both the sample DocBook and its transformation into a PDF file. The sample and transformation have resolved any doubt about the benefit of creating OMSI DocBooks; one can easily create the original information in DocBook, deliver the desired transformation (e.g. a

PDF document), and deliver the DocBook for future use. Even if the modes of consumption never change (an unlikely prospect), there is very little extra effort involved. Perhaps the only downside is the requirement to learn a new authoring environment (everyone is familiar with Microsoft Word, but few have seen XXE), but we believe this to be a worthy investment for protection against future changes.

This chapter has examined methods of storing data- and document-centric XML-based OMSI information. While addressing document-centric storage, we discussed the rendering of DocBooks for end-user consumption as PDF or HTML documents. In the next chapter, we address the more general topic of XML transformations which are essential to the development and implementation of integrated OMSIML deliverables.

VII: XML-BASED OMSI INFORMATION TRANSFORMATIONS

A. TRANSFORMATION TECHNOLOGIES

It is beyond the scope of this thesis to provide an in-depth analysis of XML transformation technologies. Nonetheless it was important to choose a technology that meets the needs of the OMSI process and is sustainable well into the future. This chapter briefly describes the two leading technologies, XSL Transformations and XQuery, and discusses which is better suited for use with OMSI.

1. XSL Transformations (XSLT)

The W3C's latest XSL Transformations recommendation [XSLT1.0] was designed for use as part of XSL. The authors explicitly note:

> XSLT is also designed to be used independently of XSL. However, XSLT is not intended as a completely general-purpose XML transformation language. Rather it is designed primarily for the kinds of transformations that are needed when XSLT is used as part of XSL.

Given such a strongly worded design intention, it would seem strange that over 80% of actual XSLT usage is for transforming XML to HTML and only 20% is used for rendering XML into other display formats which include XSL (Chamberlin, Draper et al. 2003). This indicates that XSLT has indeed become a general-purpose XML transformation language.

Many of the characteristics of XSLT as a transformation language have led to its widespread use. An XSLT stylesheet is itself an XML document, which gives it the ability to modify other XSLT stylesheet. This can be quite useful in large applications where general stylesheets are modified for specific use. XSLT is a functional programming language, which is quite different from the more common procedural languages such as Java, C, and VisualBasic. Functional languages emphasize rules and pattern-matching instead of the procedural construct of specifying a sequence of steps (albeit with condition branching) in order to achieve the desired result. XSLT cannot specify an explicit order of execution nor can it use updateable variables. These characteristics can make it daunting for traditional programmers to master the language,

but once the functional paradigm is embraced, XSLT programmers can become quite efficient.

XSLT includes the use of a sub-language for selecting nodes from the source tree. This sub-language, XML Path Language (XPath) [XPATH1.0] was published by W3C as a separate recommendation because its use clearly extends beyond XSLT. It was intended to be a single language for addressing XML documents and was specifically designed for use with XSLT and XPointer.

XSLT 2.0 [XSLT2.0] and XPath 2.0 [XPATH2.0] are currently in working draft status and continue to be complementary products. Some long-awaited functionality is included in the new versions, most notably conversion of result tree fragments to node-sets, multiple output documents, and built-in support for distinct-value grouping. While the functions have always been available through extensions (e.g. EXSLT), it will soon be possible to ensure all XSLT compliant processors support these functions. [XPATH2.0] also includes support for data types beyond [XPATH1.0]'s string, Boolean, node-set, and number.

We use only [XSLT1.0] for XSLT transformations since there is no missing functionality and the transforms should work long into the future given the excellent backward compatibility of [XSLT2.0].

2. XQuery

"XQuery 1.0: An XML Query Language" [XQUERY] is a W3C Working Draft for a specification designed to be broadly applicable across many types of XML data sources. It is obviously intended as a different language from [XSLT2.0] and differs in many significant ways. These differences can be explained by two principle reasons: the different design requirements led to different design decisions and the authors of each specification came from very different communities. It is important to recognize these differences and understand the context in which XQuery was developed.

The heart of XQuery is the FLWOR expression. FLWOR is an acronym comprised of the first letter of each clause that may occur together: For, Let, Where, Order, and Return. FLWOR expressions are almost self-explanatory; the following example certainly needs no explanation:

```
for $b in doc("books.xml")//book
let $a := $b//author
where count($a) > 1
order by $b/title
return $b/title
```

Notice that the FLWOR statement is quite similar to SQL's "SELECT columnlist FROM tablename WHERE criteria ORDER BY columnlist" statement. Such similarity makes XQuery compact and especially appropriate for use with data-centric information.

3. Comparison of XSLT and XQuery

When comparing XSLT and XQuery, the answer to the question "which is better?" is not straightforward. As much as the debate between XML and RDBs is similar to a diametric discussion of religion, so too does the debate between XSLT and XQuery tend to polarize the two sides. Here are two differing opinions on the answer to the question:

> "The strength of XQuery is that it is a simpler language than XSLT, which makes it much more feasible to implement efficient searching of very large XML databases. Its other strength is that for simple problems, the XQuery code is much shorter than the XSLT code." (Kay 2004a)

> "My take on it is that in order to do anything of interest, you need to know XPath to a fairly solid degree. By the time you get there, XSLT is more expressive and capable than XQuery." (Kurt 2004)

Neither author is likely to convince the other that he is mistaken. Nonetheless, there are strengths and weaknesses of the two technologies that are indisputable. In an interview with Ivan Pedruzzi (developer of Sonic's Stylus Studio [STYLUSSTUDIO]), Michael Kay, the developer of Saxon [SAXON] lays out a clean distinction between the two technologies (Kay 2004b). He makes the observation that most of the XQuery proponents come from an RDB environment, whereas for RDB developers XSLT isn't a language they can easily relate to. Instead, XQuery offers them SQL-like semantics that can be easily understood and visualized. On the other hand, XSLT developers are very familiar with their language – to the point they've learned to appreciate its strengths and ignore its weaknesses. It's also true that almost anything done with XQuery can still be done with XSLT, with the exception of querying relational or XML databases.

There is certainly a middle ground where either XSLT or XQuery is appropriate, even if one is the more efficient technology. The case of managing and repurposing OMSI falls into this category. I have developed both XSLT and XQuery transformations to generate the same HTML output. As will be discussed later, there are some notable performance differences, although we did not investigate the source of the bottlenecks to determine if optimizations could be made to improve performance.

Both technologies can be used in a wide array of command-line or IDE processors. Even though XQuery has not reached recommendation status it has been implemented in commercial products such as Stylus Studio and the open-source Saxon.

Given the similar functionality of these two transformation technologies, I found them both to be suitable for use in developing OMSIML. However, there is currently one difference between them that makes a very strong case for using XSLT in any delivered OMSI content: integrated web browser support.

B. INTEGRATED WEB BROWSER SUPPORT

Web browsers offer varying levels of integrated XSL functionality. Internet Explorer 6, Netscape 6, Mozilla and Firebird all provide built-in [XSLT1.0] processing. Internet Explorer Version 5 doesn't actually support XSLT, but rather a precursor to XSLT known as XSL-WD. Table 8 lists XSLT support by the major web browsers. Over 99% of the global web browsers support [XSLT1.0].

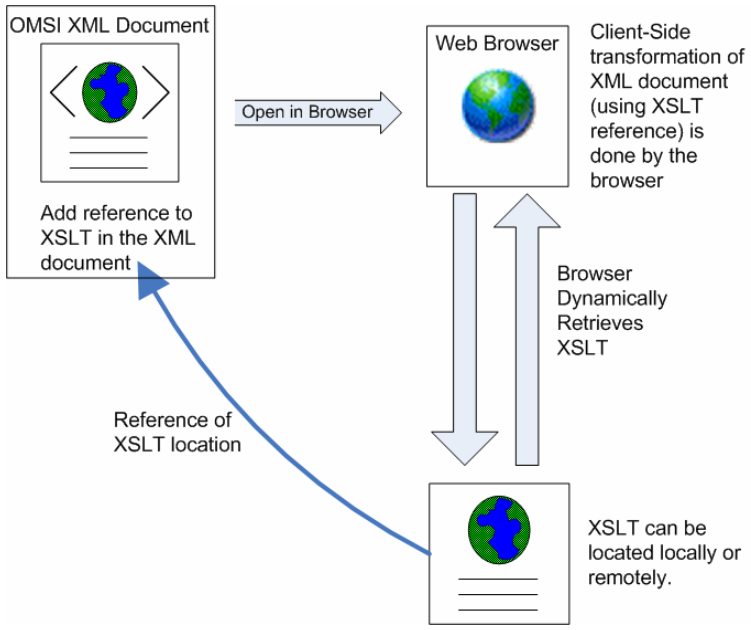
Table 8. XSLT Support in Web Browsers

Browser	XSLT version	XSLT processor
Internet Explorer 6.0	XSLT 1.0 or XSL-WD	MSXML 3.0
Internet Explorer 5.5	XSL-WD	MSXML 2.0
Internet Explorer 5	XSL-WD	MSXML 2.0
Internet Explorer 5 for Mac	XSL-WD	MSXML 2.0
Netscape 6+	XSLT 1.0	TransforMiiX
Mozilla/Firebird	XSLT 1.0	TransforMiiX
Opera	No support	-

SOURCE: From "Practical XML for the Web" (Shiell, James et al. 2002)

Integrated browser support is important because it allows for client-side XSLT transformations of XML documents. A diagram of client-side transformation process is shown in Figure 16. A processing-instruction is added to the XML document that instructs the browser to perform a transformation by giving reference to the XSLT document. When opening the XML document, the browser performs the transformation and renders the result.

Figure 16. Client-side XML Transformations



Client-side transformations offer the benefit over server-side transformations of offloading the processing from the server to the client. This saves bandwidth and reduces server processing requirements. This is especially appealing when stylesheets are cached locally and only the XML data needs to be transmitted. As the number of concurrent clients increases, this methodology scales better than server-side transformations.

More importantly, client-side transformations allow the server to send semantically rich data which is then transformed as late as possible. The final transformation is used only to perform the rendering of the presentation. This preserves the original data for other potential uses: follow-on queries, later presentations, etc. If transformations occur too soon, at best the consumer can be forced to perform screen-scraping to capture the original data. Performing the transformation client-side also allows the user to pass parameters to the XSLT just prior to executing the transformation. While this is not a design issue for the OMSI framework, it leads to another very important benefit of client-side transformations: user customizations.

Using client-side transformations, users are free to choose their own aesthetic customizations (i.e. styles or skins) or even render the data in completely new ways such as charts, summaries, or even speech. This has very important implications when considering the varied uses of OMSI deliverables. For example, LANTDIV reviews the

deliverables for content (i.e. is all the information present?), NAS Sigonella technicians review the deliverables for accuracy (i.e. is the information present accurate?), and the NAS Sigonella planners review the deliverables for summary purpose (i.e. what does the information imply vis-à-vis workload?). Without client-side transformations, a complex server-side framework must be developed (which is beyond the scope of this thesis) or the OMSI A/E must create separate XML documents for each of the OMSI users. The prospect of maintaining separate documents implies an exceptional burden. With client-side transformations it is a simple matter to develop different XSLTs for different uses.

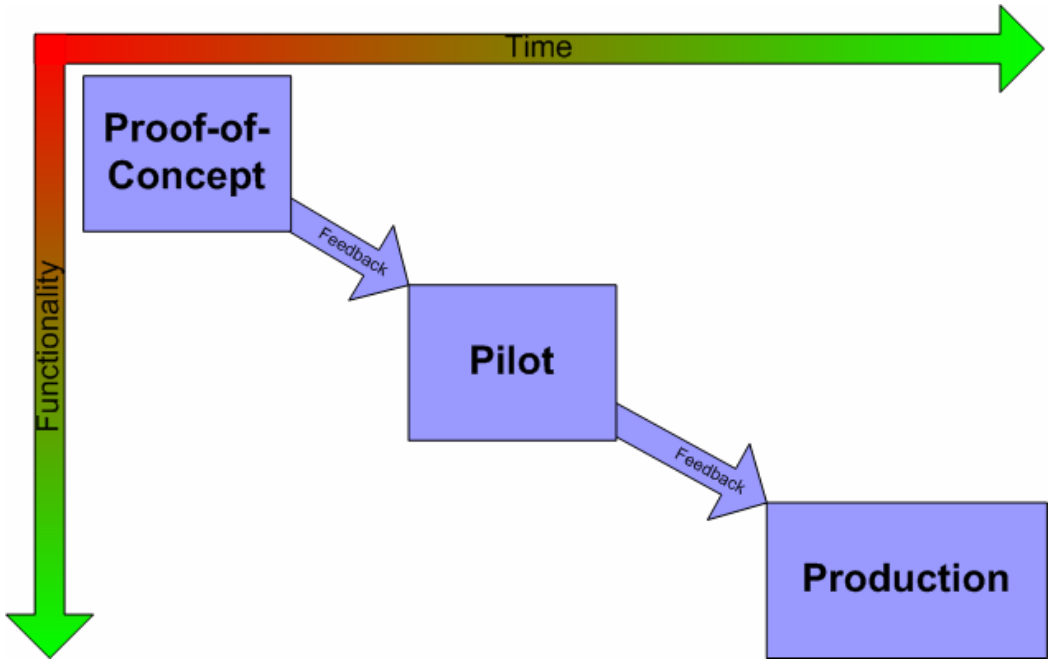
Given these benefits of client-side XSLT transformations, we have developed XSLT transformations specific to the consumer of the information. The Design-Based Planning Submittal described in the next chapter offers an excellent example of this methodology.

THIS PAGE INTENTIONALLY LEFT BLANK

VIII: DEVELOPMENT AND IMPLEMENTATION

We chose to use an iterative "Proof-of-Concept → Pilot → Production" (P3) methodology for developing the OMSI/ML product line as described in Figure 17. Rather than assume a traditional methodology such as the Waterfall, Spiral, Evolutionary, or Extreme models, we focused on what the delivery could do for the end-user. The P3 approach is most similar to the Prototyping model (Lantz 1985), albeit on a much smaller scale, and shares some of its same concerns. The biggest criticism is the tendency to over-promise and under-deliver. This happens because the end-user can get the sense that the product is finished, when in fact it is has only established the framework of the system. It should be clearly stated here that the results of this thesis are not 100% of the solution. Rather, they demonstrate the existence of a superior framework for managing the OMSI information lifecycle.

Figure 17. Development Methodology



The P3 methodology helps ensure client and stakeholder buy-in by using manageable steps. The Proof-of-Concept stage gives users an opportunity to visualize what the product will do for them. The Pilot stage is where much of the functionality gets added. It is especially useful for receiving lots of feedback to help refine the

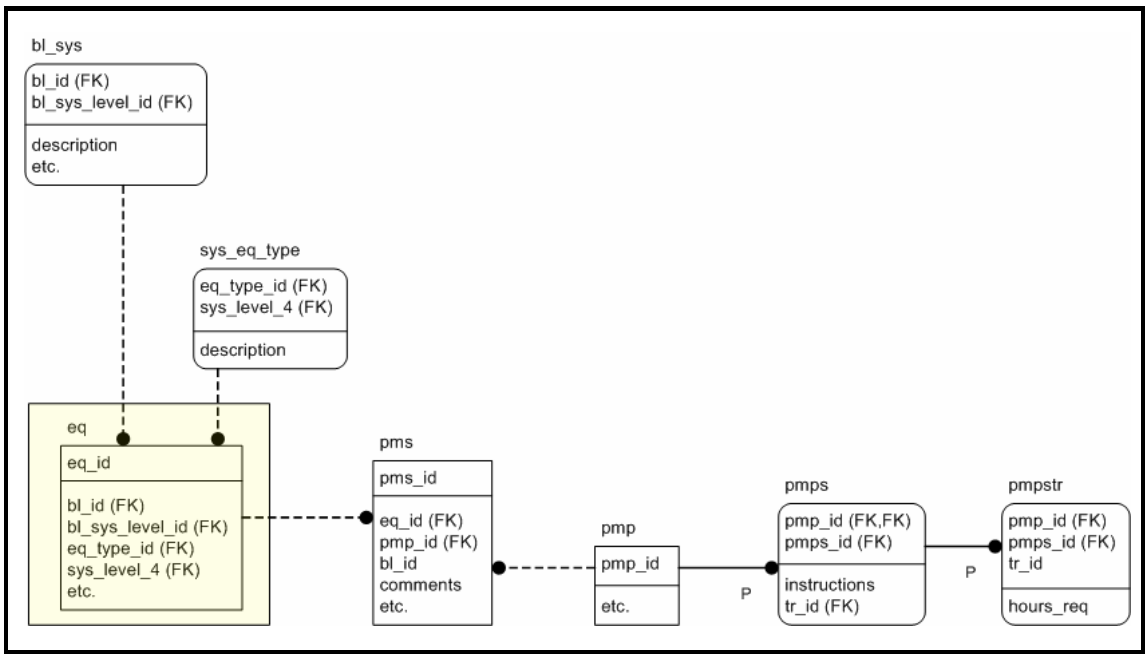
product. Most of the buy-in occurs in this stage. In the Production stage additional feedback is received and final buy-in is obtained.

As part of the Proof-of-Concept, methods were first developed to transform ArchibusML to OMSIML. Then a Proof-of-Concept and Pilot HTML output of OMSI information were developed that represented a "Design-Based Planning Submittal". Finally, a Proof-of-Concept, Pilot, and Production PM Library were developed that can be used for managing a library of PM procedures.

A. ARCHIBUS TO OMSIML TRANSFORMATION

In order to develop a working example of an OMSIML document, we must first be able to transform freely between Archibus and OMSIML. Such a capability is necessary before considering other opportunities where baseline development data is required from which to work. We first accessed the Archibus database using Microsoft Access and ODBC. This had the significant advantage of easily creating SQL queries that, once exported as XML (using Access's built-in Export as XML functionality), would be much easier to transform with XSLT. It was also an easy endeavor to link Archibus with the developmental database schema (as shown in Figure 11 on page 33) which had already been mostly implemented in the production schema of Archibus. Of course, maintaining a "shadow" Access database is not feasible in the long-term and the Archibus data must eventually be accessed through its ArchibusML export API to ensure compatibility with future Archibus changes..

After establish database connectivity, we created an Access query containing all the keys in the AFM_eq table as well as all the keys of any table containing the AFM_eq table's primary key (eq_id). This to ensure that any additional data exports could tie into the EquipmentListing without needing to make any changes to the EquipmentListing itself. Using the IDEF1x model, partially shown below, it was an easy matter to accomplish this.

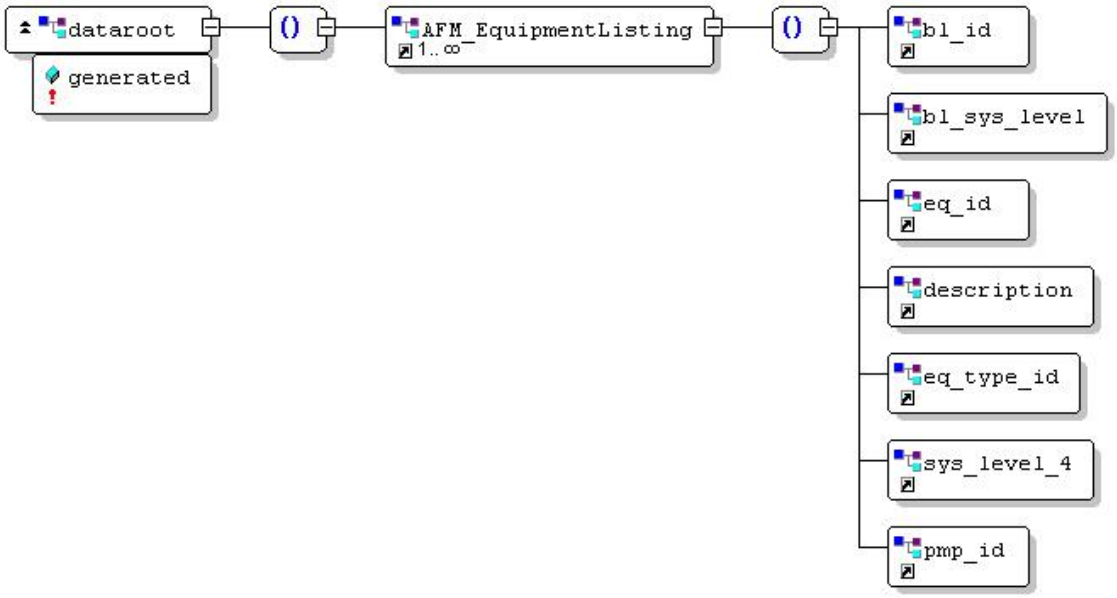


All identifier dependency parent entities (shown by rounded corners) connected to the eq entity already had their primary keys contained in the eq relation. It was then only needed to add the primary keys of those entities connected to the eq relation through non-identifying relations (shown as dashed lines) because in these cases, the primary keys of the parent entity weren't contained in the child entity.

The importance of ensuring the presence of all these keys in the EquipmentListing is best shown by example. In the event that pmp_id was not included in EquipmentListing, it would not be possible to later export AFM_pmpstr and relate it to the EquipmentListing because the pms relation could not be navigated. This would require creation of an export including the pms relation which would then have to ensure that its eq_id attributes were synchronized with those in EquipmentListing. While such a process is certainly feasible, it creates an opportunity for data mismatch and needlessly complicates the synchronization process.

The schema of Access' Equipment Listing query exported as XML (EquipmentListing.xml) is shown in Figure 18.

Figure 18. XML Schema of Access' Equipment Listing Exported as XML



Having generated a schema instance document EquipmentListing.xml, the following straightforward XQuery transforms it to an XML document conforming to the OMSIML schema:

```
<OMSI>
 <Buildings>
  {
   for $bl in distinct-values(/dataroot/AFM_EquipmentListing/bl_id)
   order by $bl
   return
   <Building>
    <bl_id>{$bl/text()}</bl_id>
     { distinct-values (
     for $blsys in /dataroot/AFM_EquipmentListing/bl_sys_level_id
     where $blsys/../bl_id = $bl
     order by $blsys
     return
     <BlSys>
      <BlSysLevelId>{$blsys/text()}</BlSysLevelId>
      <EqListing>
       { distinct-values (
       for $EqItem in /dataroot/AFM_EquipmentListing
       where $EqItem/bl_sys_level_id = $blsys and $EqItem/bl_id =
$bl
       return
       <EqItem>
        <EqId>{$EqItem/eq_id/text()}</EqId>
        <Description> </Description>
        <EqTypeId>{$EqItem/eq_type_id/text()}</EqTypeId>
         <PMReqs>
         { distinct-values(
         for $PMItem in /dataroot/AFM_EquipmentListing
         where $PMItem/eq_id = $EqItem/eq_id
         return
         <PMId>{$PMItem/pmp_id/text()}</PMId>
         )
         }
         </PMReqs>
       </EqItem>
       )
       }
      </EqListing>
     </BlSys>
     )
     }
   </Building>
  }
 </Buildings>
</OMSI>
```

Using a similar process enables the transform of any Archibus data into an OMSIML instance. Our next development was to create a pilot application that built upon OMSIML.

B. THE DESIGN-BASED PLANNING SUBMITTAL

The first development taken to the pilot stage was a design-based planning submittal (DBPS). The NAS Sigonella PWD requested this submittal for use in planning the maintenance requirements for new facilities not yet delivered. They needed a way to estimate the man-hours required by Trade to maintain a new Building System. This type of submittal was an excellent showcase for the merits of OMSIML. It would allow for a delivery that was previously unfeasible because the OMSI A/E would not generate anything but the pre-established deliveries – any additional deliverables would require a modification to the contract.

NAS Sigonella requested an HTML delivery shown by their prototype in Figure 19.

Figure 19. Design-Based Planning Submittal Request by NAS Sigonella

Naval Air Station Sigonella

Operations, Maintenance, and Support Information

Design-Based Planning Submittal

Construction Project: P-620

Submittal Delivery Date: 1 June 2003

Prepared by: Syska

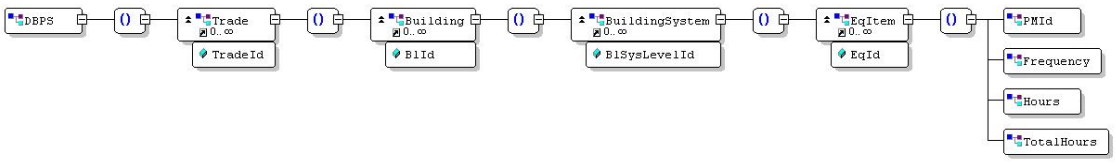
Maintenance Requirement by Trade

Trade					Trade Hours
	Building			Building Hours	
		Building System	Building System Hours		
22					254
	1			150	
		BISys 1	100		
		BISys 2	50		
	2			104	
		BISys 1	104		
57					115
	1			40	
		BISys 1	40		
	2			75	
		BISys 1	25		
		BISys 2	30		
		BISys 3	20		

In developing this delivery, it was decided to transform OMSIML into an intermediate markup language that could then be transformed into a final consumable format. The benefit of creating this intermediary is the ability to separate its presentation

from its content; the intermediate XML is not intended for end-user consumption. A final transformation will be necessary to generate the HTML requested by NAS Sigonella. We could have easily applied a single transformation that created the final HTML from OMSIML, but it would have made it more cumbersome to adjust the deliverable if new modes of consumption were requested. As an example, suppose NAS Sigonella determined they needed the DBPS information in an Excel spreadsheet. With the intermediate XML, it is a simple matter to repurpose it as comma-separated-values (CSV) that could be directly imported into Excel. Given the simple structure of the intermediate XML, anyone with even a rudimentary understanding of XSLT could create this new transformation. The XML Schema diagram of the intermediate XML, is shown in Figure 20.

Figure 20. DBPS XML Schema Diagram



To generate the intermediate markup from OMSIML the performance of both XSLT and XQuery were compared. XQuery was the first choice because it seemed a much simpler endeavor given the straightforward nature of the FLWOR construct. Unfortunately, its performance was unacceptable; it took over 300 seconds to run against the prototype OMSIML. The final production OMSIML, likely ten to twenty times the size of the prototype, would push the run times to an unacceptable level. The equivalent XSLT performance was found to be quite acceptable. The comparison of the two transformations are summarized in Table 9. The complete listings of the two transformations are contained in Appendix D.

Table 9. DBPS XSLT and XQuery Performance

Profile Type:	XSLT
Transform Engine:	Stylus
Transformer Script:	OMSI2DBPS.xsl
Initial Document:	OMSI23.xml
Total Elapsed Time:	22268259µs (µs=microseconds, or $^{1}/_{1000000}$ second)

Note: 99.9% of the elapsed time occurred processing the innermost FLWR statement

Profile Type:	XQuery Transform
Transform Engine:	Stylus
Transformer Script:	DBPS23.xquery
Initial Document:	OMSI23.xml
Total Elapsed Time:	360387709µs (µs=microseconds, or $^{1}/_{1000000}$ second)

Note: 92.4% of the elapsed time occurred processing the <xsl:if test="/OMSI/PMLibrary/PMItem[@PMID=$PMID]/TradeID = $TradeID">

Once the intermediate markup was generated, creating the final presentation XSLT was a straightforward endeavor. The XSLT listed in Appendix D was applied to the intermediate markup and resulted in the HTML shown in Figure 21.

Figure 21. Design-Based Planning Submittal Delivery

Naval Air Station Sigonella

Operations, Maintenance, and Support Information

Design-Based Planning Submittal

Construction Project: P-620

Submittal Delivery Date: 1 June 2003

Prepared by: Syska

Trade	Buildings Systems by Building				Total Trade Hours
	Building	Building System	Total BISys Hours	Total Building Hours	
11	722	D30	10	51	105
		UNK	38		
		D40	3		
	724	UNK	2	5	
		D40	3		
	725	D30	13	49	
		UNK	18		
		D40	18		
21	724	UNK	1	1	1
22	722	D30	16	16	88
	724	D30	13	13	
	725	D30	59	59	
51	722	D30	9	9	17
	725	D30	9	9	

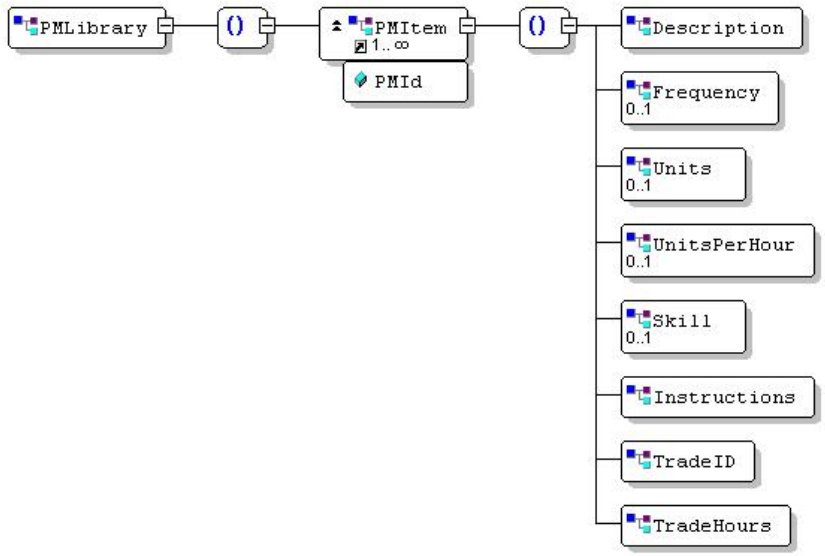
The intermediate XML (DBPS.xml) and presentation XSLT (DBPS.xsl) are delivered to the end-user. When opening DBPS.xml in Microsoft Internet Explorer 6, the DBPS.xsl is automatically applied and the end-user is presented just the output of the XSL.

C. THE PM LIBRARY

As a final development we intended to demonstrate the process all the way through the P3 methodology by developing an XML Schema to represent information required for a PM Library. We will refer to the schema and instance documents as PMLibrary.xsd and PMLibrary.xml, respectively. Figure 22 shows a diagram of the

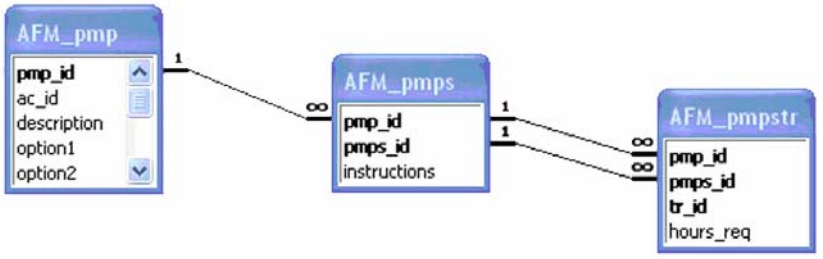
PMLibrary.xsd which is a part of the larger OMSIML Schema documented in Appendix B.

Figure 22. PMLibrary XML Schema Diagram



Comparing the schema diagram to the relational schema of the equivalent Archibus data shown in Figure 23 reveals the XML model to be much simpler.

Figure 23. Archibus PM Procedure Schema



This is possible because the business rules used by NAS Sigonella limit the PM Procedure to just one Step. In the event this business rule were to change, it would be a trivial matter to adjust the PMLibrary to account for more than one Step per Procedure and more than one Trade per Step. *[N.B. We did not implement such a model because it is unlikely that more than one step would ever be added. Indeed, this is the attraction of XML modeling – PMLibrary could easily be extended if the future warrants, which leaves the schema designer to concentrate on current processes.]*

Each <PMItem> element represents a PM procedure. It is uniquely identified by its attribute "PMId" which must be unique among all PMItems. The child elements further describes the details of the PM procedure.

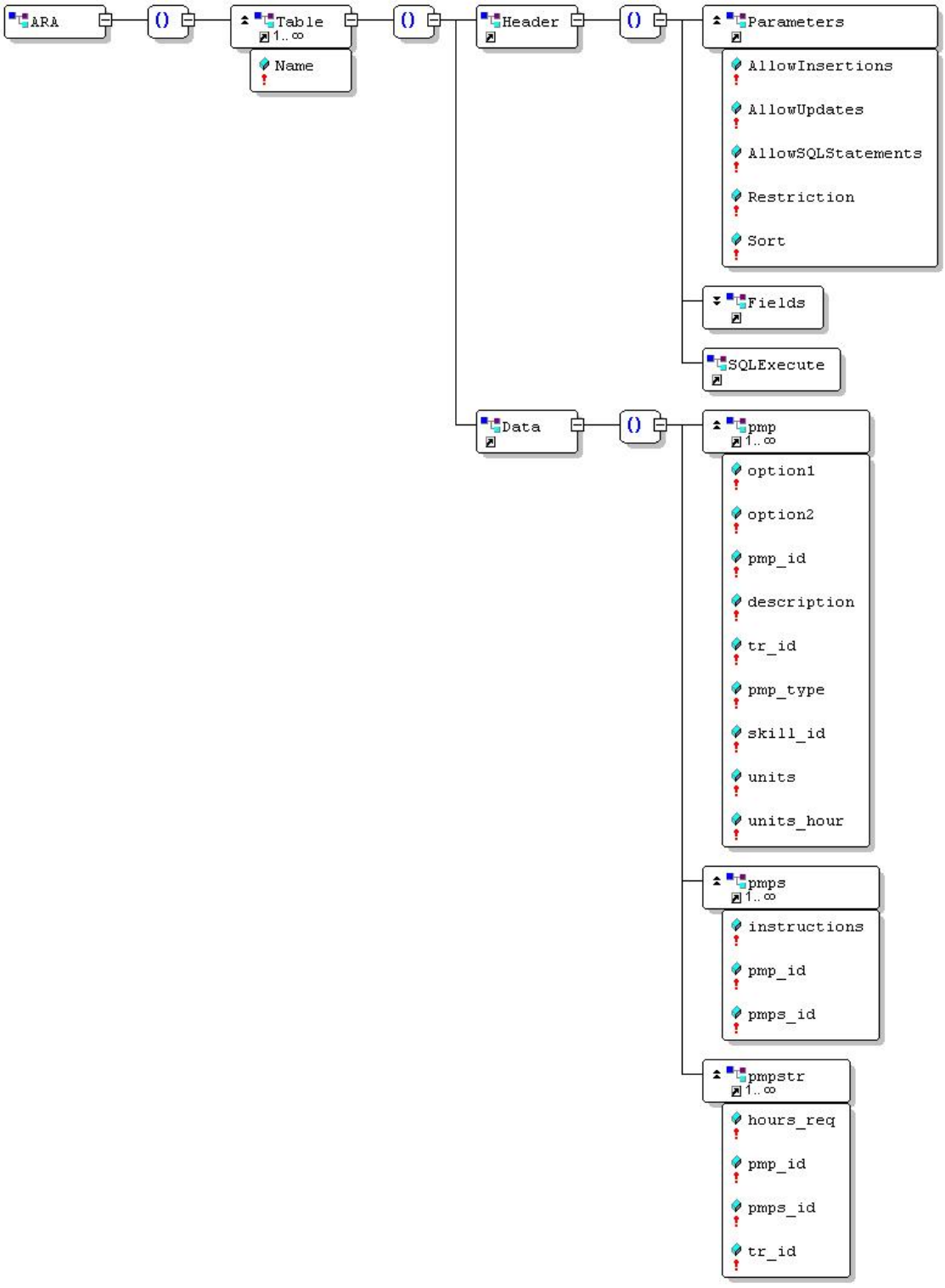
The <Frequency> element is enumerated and contains the number of maintenance actions required in one year. The following is a list of acceptable enumeration values and their verbal equivalents:

```
<selectoption description="Daily" value="365"/>
<selectoption description="Weekly" value="52"/>
<selectoption description="Monthly" value="12"/>
<selectoption description="Quarterly" value="4"/>
<selectoption description="Semi-Annual" value="2"/>
<selectoption description="Annual" value="1"/>
<selectoption description="Bi-Annual" value="0.5"/>
<selectoption description="Every Five Years" value="0.2"/>
<selectoption description="UNK" value="0"/>
```

If an otherwise similar PM procedure has two possible frequencies, it will be necessary to create separate PMItems. While this is not an elegant model for such a scenario, the nature of its rare occurrence fails to justify the added complication of modeling the possibility of more than one frequency per PMItem.

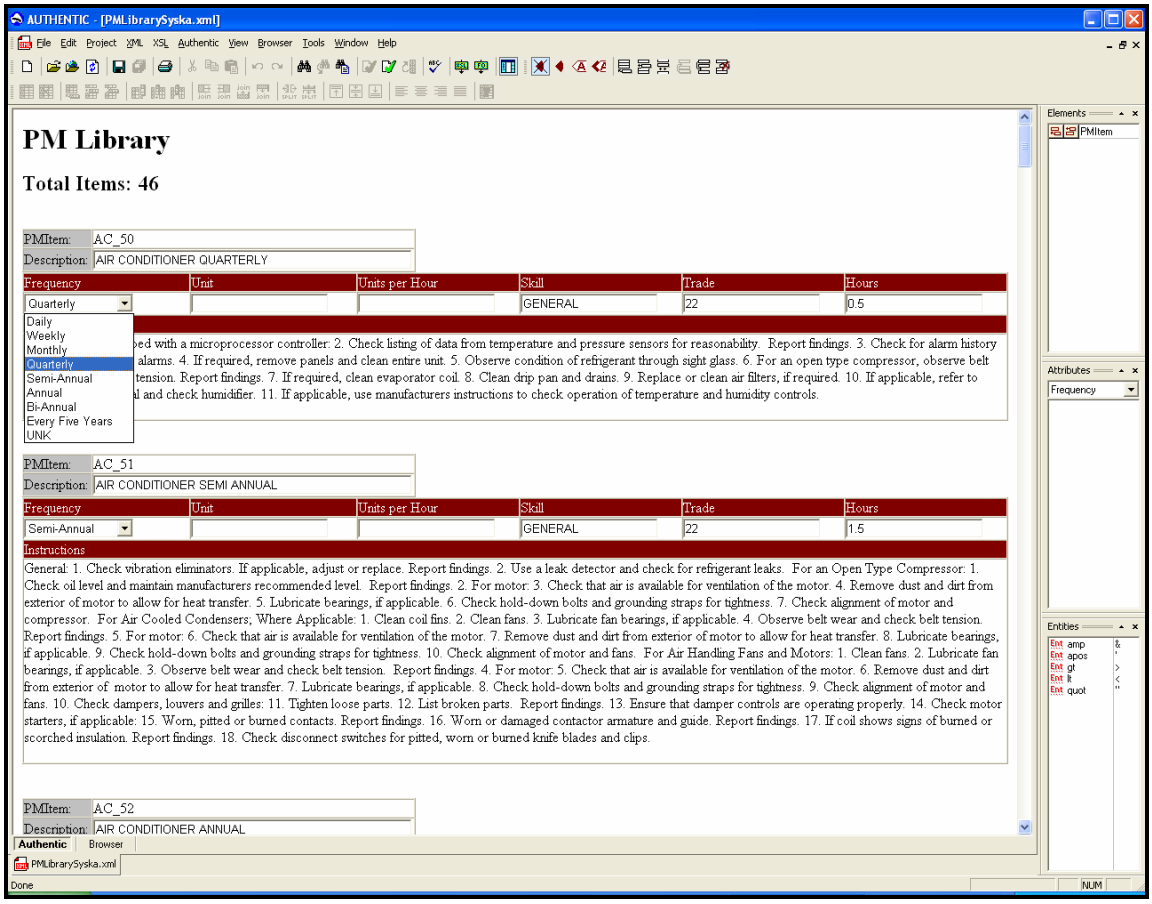
Figure 24 shows a diagram of the XML Schema corresponding to an XML document generated by using the Archibus Data Transfer Command to export the AFM_pmp, AFM_pmps, and AFM_pmpstr tables.

Figure 24. Archibus/FM XML Export Schema for PM Library



In order to conduct a proof-of-concept, we executed an Archibus Data Transfer export to generate information to feed into the PMLibrary. The PM procedures were taken from a Syska OMSI deliverable made in November 2003 consisting of approximately 50 items. These items, while not a complete list of all PM procedures, served as a sufficiently representative sample of the procedures used at NAS Sigonella. Finishing the proof-of-concept, we developed an Authentic™ view for managing the PM Library as shown in Figure 25.

Figure 25. Authentic View of PM Library



Note that the user is presented with a pull-down list of the allowable enumerated values for Frequency. This is useful in hiding the abstraction of Frequency as a value for the number of times the procedure must be performed each year; the user need only select a verbal description of frequency and Authentic inserts the value of the description.

Having proven that it was feasible from a usability perspective to represent the PM Library in an XML document, we proceeded to conduct a pilot implementation that

would take the information in PMLibrary.xml and transform it into an XML document that conformed to ArchibusML. This would allow changes, deletions, or insertions to the development PMLibrary which would then be reflected automatically in the production Archibus database. The XSLT that accomplishes this transformation is listed in Appendix D.

We also extended the OMSIML to ArchibusML transformations to include all the information in a completely populated OMSIML instance document which includes UNIFORMAT, Buildings, Building Systems, Equipment Types, Equipment Listings, and relevant look-up lists. This is perhaps the most useful production-ready result of this thesis as it will allow OMSI authors to create information in XML as a first step, rather than having to use Archibus to author content. While OMSIML may not include all the information needed for an OMSI deliverable, it is easily extendable.

It is our expectation that the next OMSI delivery order for NAS Sigonella will create information using XML documents. This also includes the use of DocBook for creating all document-centric OMSI information. Indeed, DocBooks should be used immediately. They can be created using free editors that ensure conformance to the DocBook schema or they can be authored with any text or XML editor. Some of the free editors use a GUI that is almost as easy to use as Microsoft Word. The benefit of freely capturing semantics of the text could prove invaluable to future applications.

IX: CONCLUSIONS AND IMPLICATIONS

A. CONCLUSIONS

This thesis began with consideration of an information integration problem. After data has been created or collected it must be managed by an information system, many systems are often stove-piped and unable to share their information representations with other systems. Information is satisfactorily integrated when information systems can cross-communicate and share their data that has been given semantics or context. Such integration allows both structured and semi-structured data to be collected once and repurposed as new information across any number of systems.

We approached the research of improving the OMSI creation, management, and repurposing processes by first performing a ground-up review of modeling OMSI content. Information integration considered a priori (as opposed to post facto integration) allowed us to redesign OMSI models in a manner that maximizes our ability to repurpose data without manual intervention or re-authoring. Rather than accept the traditional model of OMSI content as strictly document-based, we chose to examine relational and XML models of the information. A requirement of any useful OMSI model is the ability to integrate with a CAFM. This thesis addressed a specific CAFM (Archibus) and began the modeling of OMSI as a relational model. We used the IDEF1X information modeling technique to diagram the relational model and then used the diagram and some general modeling recipes to create an XML schema for OMSI which. We called this schema OMSIML and used it to develop proof-of-concepts, pilots, and a production-ready implementation of a PM Library. We also described the DocBook schema and justified its use for creating the purely document-centric OMSI information.

Methods of storing and transforming XML-based OMSI information were also examined. The benefit of OMSIML is in its ability to store and transform XML instances to support both current deliverables and future, yet to be identified, requirements. We have not identified a system for storing and querying the OMSI XML documents; the current processes are quite manageable using the XML documents accessed directly by an Operating System's file management services. We compared the suitability of XSLT

and XQuery as XML transformation technologies and developed transformations for the proof-of-concept, pilot, and production implementations.

The production-ready implementation of the PM Library includes the use of Altova's Authentic to manage the information in an intuitive and efficient graphical environment. This free application allows the management of XML documents in a manner that is comparable to traditional database interfaces using queries and forms.

The most useful result of this thesis is the OMSIML to ArchibusML transformations and the use of DocBook for creating document-centric OMSI information. These two implementations will allow OMSI authors to create information in XML documents as a first step, thereby capturing semantics of the data for use in current and future applications. DocBook is extremely attractive because it is a standardized schema that makes use of free transformations to create deliverables in PDF, Word, or HTML.

B. AREAS FOR FURTHER RESEARCH AND DEVELOPMENT

While this thesis has produced useful deliverables for authoring OMSI information in XML documents, there are many areas that warrant further research and development. The four most significant areas are schema conversions, transformation efficiency, XQuery implementations, and native XML databases.

We have described general recipes for modeling XML based on relational schemas. We also briefly addressed the topic of schema conversion and a "Constraints-based Translation" algorithm for automatically converting a relational schema to an XML schema (Lee, Mani et al. 2002). The prospect of automating the creation of an XML schema from a relational schema would greatly facilitate future implementations of OMSI without needing to give considerable thought to optimum schema design.

While XSLT and XQuery can both perform a given XML transformation, we found significant differences in efficiency to XQuery's detriment. As more XQuery processors become available it would be useful to identify bottlenecks in either XQuery construction or processor implementations. Although the SQL-like nature of XQuery FLWOR expressions greatly simplifies the development of transformations by non-XSLT experts, unacceptable efficiencies must be eliminated. Nonetheless, XSLT functionality

is included in 99% of all web browsers and will likely continue to remain the best technology for creating content presentation-specific transformations.

We have not addressed implementations of native XML databases. However, these products represent an excellent method for managing OMSIML and DocBook instances. The ability to query collections of OMSI XML documents makes way for many repurposing opportunities for consumers of OMSI information. This may also include a publishing framework for OMSI information external to a CAFM system.

C. IMPLICATIONS

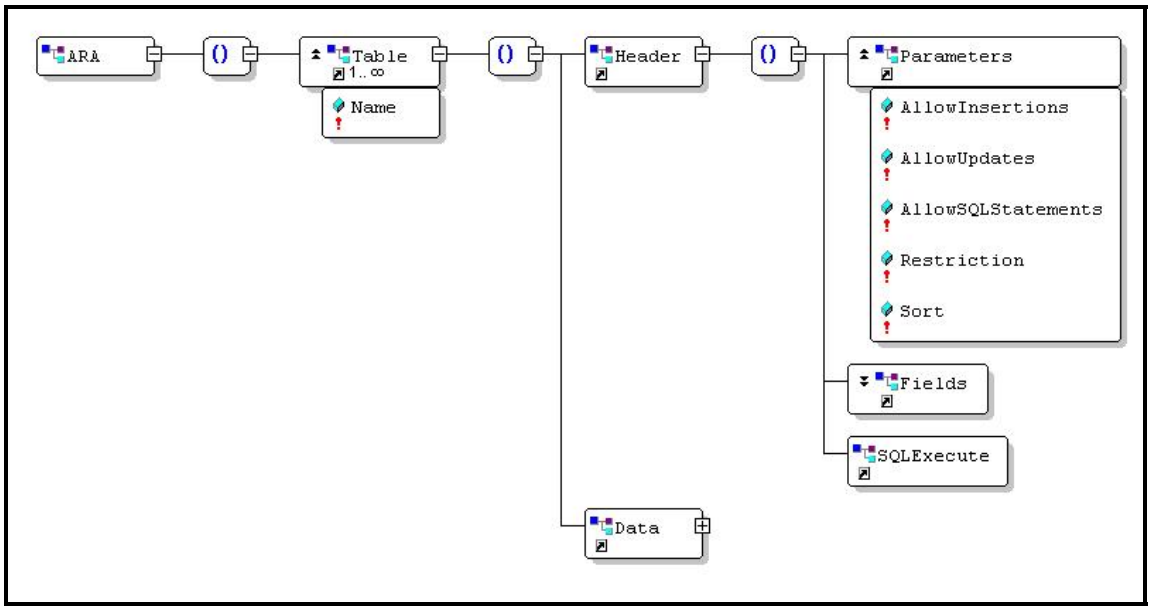
OMSIML represents a revolutionary step in the progress of OMSI deliverables. It greatly improves the flexibility of LANTDIV's OMSI program by resulting in a single statement of work for OMSI authoring that can support almost any end-user information storage system. It meets the current needs of OMSI customers and allows for easy extensibility for addressing future (and as yet unknown) needs. OMSIML also obviates the requirement for OMSI A/Es to learn the intricacies of CAFMs in order to integrate their deliverables with the customer's CAFM. This will reduce the expense of OMSI deliverables and allow for greater competition among OMSI A/Es. It also makes it more feasible to generate OMSI deliverables for non-MILCON construction and repair when fiscal resources are scarce or it is not reasonable to expect small construction contractors to understand the complete OMSI process. Any construction contractor can easily create an OMSIML document conforming to the XML Schema that can then be delivered using the automated tools of LANTDIV's OMSI process.

Most importantly, the XML framework of OMSI ensures its suitability far into the future by hedging against new CAFMs, modified O&M practices, different planning tools, and even new maintenance organizations (e.g. contracting out all maintenance efforts.). OMSIML provides deliverables that will remain relevant, accurate, integratable, and customizable long into the future by being able to evolve with changing processes. This ability is essential to the long-term success of any O&M program and adds significant value to LANDIV's OMSI program. We expect the LANTDIV customer base to grow as a result of the more robust deliverable framework and as more applications for OMSI are developed the OMSIML schema will continue to be refined.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A – ARCHIBUS DATA TRANSFER FUNCTIONALITY

Archibus has a data transfer function that allows users to manually import or export selected data as an XML document. There are also APIs provided that let external applications or scripts perform this data transfer automatically. The XML conforms to the following schema:



Each table exported is represented by a <Table> element which is identified by the Archibus table name stored in its Name attribute; there can be more than one table imported/exported. The <Header> element sets parameters, specifies which fields to be imported/exported, and holds an SQL statement to be executed by Archibus prior to the transfer. The <Data> element holds a child element for each of the records to be imported/exported. The child element is named the same as \ARA\Table\@Name and presents each field as an attribute named after that field.

APPENDIX B – OMSI XML SCHEMA

OMSIML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v5 rel. 4 U (http://www.xmlspy.com) by Scott Raymond (private)
-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
 <xs:simpleType name="Level1Type">
  <xs:restriction base="xs:string">
   <xs:minLength value="1"/>
   <xs:maxLength value="1"/>
   <xs:pattern value="[A-Z]"/>
  </xs:restriction>
 </xs:simpleType>
 <xs:simpleType name="Level2Type">
  <xs:restriction base="xs:string">
   <xs:minLength value="3"/>
   <xs:maxLength value="3"/>
   <xs:pattern value="[A-Z][1-9]0"/>
  </xs:restriction>
 </xs:simpleType>
 <xs:simpleType name="Level3Type">
  <xs:restriction base="xs:string">
   <xs:minLength value="5"/>
   <xs:maxLength value="5"/>
   <xs:pattern value="[A-Z][1-9]0[1-9][0]"/>
  </xs:restriction>
 </xs:simpleType>
 <xs:simpleType name="Level4Type">
  <xs:restriction base="xs:string">
   <xs:minLength value="5"/>
   <xs:maxLength value="5"/>
   <xs:pattern value="[A-Z][1-9]0[1-9][1-9]"/>
  </xs:restriction>
 </xs:simpleType>
 <xs:element name="OMSI">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="UNIFORMAT"/>
    <xs:element ref="BuildingSystems"/>
    <xs:element ref="Buildings"/>
    <xs:element ref="EquipmentTypes"/>
    <xs:element ref="SystemEquipmentTypes"/>
    <xs:element ref="PMLibrary"/>
    <xs:element ref="Trades"/>
   </xs:sequence>
  </xs:complexType>
  <xs:key name="Level1Code">
   <xs:selector xpath="UNIFORMAT/LEVEL1/Level1Code"/>
   <xs:field xpath="."/>
  </xs:key>
  <xs:key name="Level2Code">
   <xs:selector xpath="UNIFORMAT/LEVEL1/LEVEL2/Level2Code"/>
   <xs:field xpath="."/>
  </xs:key>
  <xs:key name="Level3Code">
   <xs:selector xpath="UNIFORMAT/LEVEL1/LEVEL2/LEVEL3/Level3Code"/>
   <xs:field xpath="."/>
  </xs:key>
  <xs:key name="Level4Code">
   <xs:selector xpath="UNIFORMAT/LEVEL1/LEVEL2/LEVEL3/LEVEL4/Level4Code"/>
   <xs:field xpath="."/>
  </xs:key>
```

OMSIML Schema (cont.)

```
    <xs:key name="BUILDING_bl_id">
      <xs:selector xpath="Buildings/AFM_bl/bl_id"/>
      <xs:field xpath="."/>
    </xs:key>
    <xs:key name="EQTYPE_EqTypeId">
      <xs:selector xpath="EquipmentTypes/EqType/EqTypeId"/>
      <xs:field xpath="."/>
    </xs:key>
    <xs:key name="SYSEQTYPE_PrimaryKeys">
      <xs:selector xpath="SystemEquipmentTypes/SysEqType"/>
      <xs:field xpath="EqTypeId"/>
      <xs:field xpath="Level4Code"/>
    </xs:key>
    <xs:keyref name="SYSEQTYPE_EqTypeId" refer="EQTYPE_EqTypeId">
      <xs:selector xpath="SystemEquipmentTypes/SysEqType/EqTypeId"/>
      <xs:field xpath="."/>
    </xs:keyref>
    <xs:keyref name="SYSEQTYPE_Level4Code" refer="Level4Code">
      <xs:selector xpath="SystemEquipmentTypes/SysEqTypeId/Level4Code"/>
      <xs:field xpath="."/>
    </xs:keyref>
    <xs:key name="BLSYSLEVEL_BlSysLevelId">
      <xs:selector xpath="BuildingSystems/BlSysLevel/BlSysLevelId"/>
      <xs:field xpath="."/>
    </xs:key>
    <xs:keyref name="BLSYSLEVEL_Level2Code" refer="Level2Code">
      <xs:selector xpath="BuildingSystems/BlSysLevel/Level2Code"/>
      <xs:field xpath="."/>
    </xs:keyref>
    <xs:keyref name="BLSYSLEVEL_Level3Code" refer="Level3Code">
      <xs:selector xpath="BuildingSystems/BlSysLevel/Level3Code"/>
      <xs:field xpath="."/>
    </xs:keyref>
    <xs:keyref name="BLSYSLEVEL_Level4Code" refer="Level4Code">
      <xs:selector xpath="BuildingSystems/BlSysLevel/Level4Code"/>
      <xs:field xpath="."/>
    </xs:keyref>
    <xs:key name="BLSYS_PrimaryKeys">
      <xs:selector xpath="Buildings/Building/BlSysl"/>
      <xs:field xpath="BlSysLevelId"/>
      <xs:field xpath="bl_id"/>
    </xs:key>
    <xs:keyref name="BLSYS_BlSysLevelId" refer="BLSYSLEVEL_BlSysLevelId">
      <xs:selector xpath="BuildingSystems/BlSysLevel/BlSysLevelId"/>
      <xs:field xpath="."/>
    </xs:keyref>
    <xs:keyref name="BLSYS_bl_id" refer="BUILDING_bl_id">
      <xs:selector xpath="BuildingSystems/BlSysLevel/bl_id"/>
      <xs:field xpath="."/>
    </xs:keyref>
    <xs:key name="BLSYSDOC_PrimaryKeys">
      <xs:selector xpath="BuildingSystems/BLSYSLEVEL/BLSYS/BLSYSDOC"/>
      <xs:field xpath="BlSysDocId_BLSYS_BlSysId_BLSYSLEVEL_BlSysLevelId_FK_FK"/>
      <xs:field xpath="BlSysDocId_BLSYS_BlSysId_AFM_bl_bl_id_FK_FK"/>
      <xs:field xpath="BlSysDocId_DocumentType"/>
    </xs:key>
    <xs:keyref name="BLSYSDOC_BLSYS_ForeignKeys" refer="BLSYS_PrimaryKeys">
      <xs:selector xpath="BuildingSystems/BLSYSLEVEL/BLSYS/BLSYSDOC"/>
      <xs:field xpath="BlSysDocId_BLSYS_BlSysId_BLSYSLEVEL_BlSysLevelId_FK_FK"/>
      <xs:field xpath="BlSysDocId_BLSYS_BlSysId_AFM_bl_bl_id_FK_FK"/>
    </xs:keyref>
  </xs:element>
  <xs:element name="UNIFORMAT">
    <xs:complexType>
      <xs:sequence>
```

OMSIML Schema (cont.)

```
      <xs:element ref="LEVEL1" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="LEVEL1">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Level1Code"/>
      <xs:element name="Level1Name">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="60"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element ref="LEVEL2" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="LEVEL2">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Level2Code"/>
      <xs:element name="Level2Name">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="60"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element ref="LEVEL3" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="LEVEL3">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Level3Code"/>
      <xs:element name="Level3Name">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="60"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element ref="LEVEL4" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="LEVEL4">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Level4Code"/>
      <xs:element name="Level4Name">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="60"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="Buildings">
  <xs:complexType>
```

OMSIML Schema (cont.)

```
    <xs:sequence>
      <xs:element ref="Building" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="Building">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="bl_id">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="10"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="name" minOccurs="0">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="25"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element ref="BlSys" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="EquipmentTypes">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="EqType" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="EqType">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="EqTypeId"/>
      <xs:element name="Description">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="50"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="SystemEquipmentTypes">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="SysEqType" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="SysEqType">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="EqTypeId">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="5"/>
            <xs:pattern value="[A-Z0-9]{1,5}"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
```

OMSIML Schema (cont.)

```
      <xs:element name="Level4Code" type="Level4Type"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="BuildingSystems">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="BlSysLevel" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="BlSysLevel">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="BlSysLevelId">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="50"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="Level2Code" type="Level2Type"/>
      <xs:element name="Level3Code" type="Level3Type" minOccurs="0"/>
      <xs:element name="Level4Code" type="Level4Type" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="BlSys">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="BlSysLevelId">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="5"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="Description" minOccurs="0">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="50"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element ref="EqListing" minOccurs="0"/>
      <xs:element ref="BlSysDoc" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="BlSysDoc">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="BlSysLevelId">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="50"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="bl_id">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="50"/>
          </xs:restriction>
        </xs:simpleType>
```

OMSIML Schema (cont.)

```
      </xs:element>
      <xs:element name="DocumentType">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="50"/>
            <xs:enumeration value="Type1"/>
            <xs:enumeration value="Type2"/>
            <xs:enumeration value="Type3"/>
            <xs:enumeration value="Type4"/>
            <xs:enumeration value="Type5"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="DocumentName">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="50"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="PMLibrary">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="PMItem" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Description" type="xs:string"/>
            <xs:element name="Frequency" minOccurs="0"/>
            <xs:element name="Units" type="xs:string" minOccurs="0"/>
            <xs:element name="UnitsPerHour" type="xs:string" minOccurs="0"/>
            <xs:element name="Skill" type="xs:string" minOccurs="0"/>
            <xs:element name="Instructions" type="xs:string"/>
            <xs:element name="TradeId" type="xs:string"/>
            <xs:element name="TradeHours" type="xs:string"/>
          </xs:sequence>
          <xs:attribute name="PMId"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="EqListing">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="EqItem" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="EqItem">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Description" type="xs:string"/>
      <xs:element name="EqTypeId">
        <xs:simpleType>
          <xs:restriction base="xs:string"/>
        </xs:simpleType>
      </xs:element>
      <xs:element name="Level4Code" minOccurs="0"/>
      <xs:element ref="PMReqs"/>
    </xs:sequence>
    <xs:attribute name="EqId" use="required">
      <xs:simpleType>
```

OMSIML Schema (cont.)

```
        <xs:restriction base="xs:NMTOKEN"/>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
 </xs:element>
 <xs:element name="PMReqs">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="PMId" maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="Trades">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="Trade" maxOccurs="unbounded">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="TradeId" type="xs:string"/>
       <xs:element name="Description" type="xs:string"/>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

APPENDIX C– SAMPLE OMSI DOCBOOK

This appendix demonstrates the use of a DocBook to author and publish document-centric OMSI data such as a building system's safety instructions. The DocBook was authored using XMLSpy and published to PDF using DocBook Toolchain Manager.

Sample DocBook XML for OMSI HVAC System Documentation

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.3CR2//EN"
"http://www.docbook.org/xml/4.3CR2/docbookx.dtd">
<book>
   <bookinfo>
      <title>OMSI Building System Documentation</title>
      <subtitle>Sample Building System</subtitle>
      <author>
         <firstname>Scott</firstname>
         <surname>Raymond</surname>
         <email>spraymon@nps.navy.mil</email>
      </author>
      <editor>
         <firstname>Lino</firstname>
         <surname>Noble</surname>
      </editor>
      <publisher>
         <publishername>LANTDIV</publishername>
      </publisher>
      <copyright>
         <year>2004</year>
      </copyright>
      <abstract>
         <formalpara>
            <title></title>
            <para>This is a sample DocBook that demonstrates its usefulness for
authoring OMSI "documents".  A DocBook can be easily converted to HTML, RTF, PDF, or
Word depending on the needs of the end-user.</para>
         </formalpara>
      </abstract>
   </bookinfo>
   <part>
      <title>HVAC</title>
      <chapter>
         <title></title>
         <sect1>
            <title>Safety Instructions</title>
            <para>When servicing electrical equipment, follow lockout/tagout procedures
before performing any maintenance.  Be sure that the control circuitry has been
completely disabled to prevent unexpected start up of the equipment.</para>
            <para>Servicing the HVAC system involves dealing with components that
operate on electrical power.   Components of the system may be idle and start without
warning.    Before performing any maintenance or repairs on equipment, the warnings
noted before the applicable procedure must be adhered to.   The electrical disconnect
must be open, locked and tagged while working on the unit.</para>
            <warning>
               <title>Rotating Parts</title>
               <para>Many components of the HVAC system have rotating parts, such as
fans and compressors.   Components of the system may be idle and start without warning.
In order to avoid physical injury, the electrical disconnect of equipment being worked
on must be open, locked and tagged.</para>
            </warning>
            <warning>
               <title>Fluid Under Pressure</title>
               <para>While working on components with fluid that is under pressure,
ensure that the unit being worked on has been isolated and relieved of all internal
pressure.   Failure to do so may result in scalding in the case of steam and hot water,
or other physical injuries.</para>
            </warning>
            <warning>
               <title>Refrigerant</title>
               <para>To prevent injury due to frostbite, avoid skin contact with
refrigerant.</para>
            </warning>
```

Sample DocBook XML for OMSI HVAC System Documentation (cont.)

```
            <para>In performing maintenance and repairs, follow the safety precautions
outlined in this chapter and vendor data, for the respective system.   The operator
should at all times be aware of the surroundings and exercise common sense and good
judgment.</para>
         </sect1>
         <sect1>
            <title>General Safety Precautions</title>
            <itemizedlist>
               <listitem>
                  <para>Some equipment may start automatically.   Follow established
lockout/tagout procedures.</para>
               </listitem>
               <listitem>
                  <para>Exercise caution in opening steam lines.  Allow ample time for
line heat-up.</para>
               </listitem>
               <listitem>
                  <para>Observe confined space procedures when entering designated
areas.</para>
               </listitem>
               <listitem>
                  <para>When performing maintenance or repairs on electrical
equipment, follow established lockout/tagout procedures.</para>
               </listitem>
               <listitem>
                  <para>When performing PM on fluid systems, isolate and safely
relieve internal pressure.   Follow established lockout/tagout procedures.</para>
               </listitem>
               <listitem>
                  <para>Wear proper protective gear as applicable, such as:</para>
                  <itemizedlist>
                     <listitem>
                        <para>Eye protection or face shield</para>
                     </listitem>
                     <listitem>
                        <para>Hearing Protection</para>
                     </listitem>
                     <listitem>
                        <para>Chemical resistant apron and/or gloves</para>
                     </listitem>
                     <listitem>
                        <para>Electrician’s insulated gloves.</para>
                     </listitem>
                     <listitem>
                        <para>Protective footwear</para>
                     </listitem>
                     <listitem>
                        <para>Respirator</para>
                     </listitem>
                  </itemizedlist>
               </listitem>
            </itemizedlist>
            <para>While servicing the HVAC system wear safety glasses or goggles.
Failure to do so may cause physical injury.</para>
            <para>In addition to these warnings, observe all other warnings noted
within the vendor data.</para>
         </sect1>
      </chapter>
      <chapter>
         <title>System Start-up</title>
         <sect1>
            <title>TestSection1</title>
            <para>This chapter details the startup procedures for various units of the
HVAC system.</para>
            <sect2 id="test2">
```

Sample DocBook XML for OMSI HVAC System Documentation (cont.)

```
            <title>AHU Startup for Cooling Season</title>
            <procedure>
               <step>
                  <para>Visually inspect unit for debris and blockage.</para>
               </step>
               <step>
                  <para>Check condensate drains for blockage and proper
prime.</para>
               </step>
               <step>
                  <para>Ensure that the chiller has been started as per the
chillier start up procedure in this section.</para>
               </step>
               <step>
                  <para>With the AHU and AHU local fans, disconnect switches in the
“OFF” position, check that the AHU and AHU fan circuit breakers are closed.</para>
               </step>
               <step>
                  <para>Switch the AHU and AHU fans to the “ON” position at the
disconnect switch.</para>
               </step>
               <step>
                  <para>Open the chilled water isolation valves.</para>
               </step>
               <step>
                  <para>Through the DDC ensure that the AHU is enabled.</para>
               </step>
               <step>
                  <para>Verify that the AHU fans are operating properly and the
dampers are properly positioned.  Verify correct fan rotation.</para>
               </step>
               <step>
                  <para>Ensure that the respective exhaust fans for the AHU are
operating.</para>
               </step>
               <step>
                  <para>Allow system to run for 30 minutes to reach steady
state.</para>
               </step>
               <step>
                  <para>Check space temperature and verify that it is within + 2°F
of the temperature set point.</para>
               </step>
               <step>
                  <para>Check drain for proper operation. Check for leaks and
blockage.</para>
               </step>
            </procedure>
         </sect2>
         <sect2>
            <title>Heating Hot Water Pump Startup for Heating Season</title>
            <procedure>
               <step>
                  <para>Verify that prestart checks have been performed on the HHW
pumps.</para>
               </step>
               <step>
                  <para>Inspect pump seal for leakage.</para>
               </step>
               <step>
                  <para>Ensure that the suction and discharge valves are
open.</para>
               </step>
               <step>
                  <para>Ensure that the HHW pump disconnect switch is in the closed
```

Sample DocBook XML for OMSI HVAC System Documentation (cont.)

```
position.</para>
                    </step>
                    <step>
                        <para>Switch  H-O-A to “hand position.</para>
                    </step>
                    <step>
                        <para>Verify flow through the system.</para>
                    </step>
                    <step>
                        <para>Verify that there are no leaks around the pump seal.</para>
                    </step>
                    <step>
                        <para>Switch H-O-A to automatic.</para>
                    </step>
                </procedure>
            </sect2>
        </sect1>
    </chapter>
  </part>
</book>
```

OMSI Building System: HVAC

Lt. Scott Raymond

This is a sample DocBook that demonstrates its usefulness for authoring OMSI "documents". A DocBook can be easily converted to HTML, RTF, PDF, or Word depending on the needs of the end-user.

Table of Contents

1. Safety Instructions

When servicing electrical equipment, follow lockout/tagout procedures before performing any maintenance. Be sure that the control circuitry has been completely disabled to prevent unexpected start up of the equipment.

Servicing the HVAC system involves dealing with components that operate on electrical power. Components of the system may be idle and start without warning. Before performing any maintenance or repairs on equipment, the warnings noted before the applicable procedure must be adhered to. The electrical disconnect must be open, locked and tagged while working on the unit.

Rotating Parts

Many components of the HVAC system have rotating parts, such as fans and compressors. Components of the system may be idle and start without warning. In order to avoid physical injury, the electrical disconnect of equipment being worked on must be open, locked and tagged.

Fluid Under Pressure

While working on components with fluid that is under pressure, ensure that the unit being worked on has been isolated and relieved of all internal pressure. Failure to do so may result in scalding in the case of steam and hot water, or other physical injuries.

Extremely Cold Refrigerant

To prevent injury due to frostbite, avoid skin contact with refrigerant.

Sample Caution

This is a sample caution. Note that a caution implies harm to equipment vice a warning's implied harm to a person.

In performing maintenance and repairs, follow the safety precautions outlined in this chapter and vendor data, for the respective system. The operator should at all times be aware of the surroundings and exercise common sense and good judgment.

- Some equipment may start automatically. Follow established lockout/tagout procedures.
- Exercise caution in opening steam lines. Allow ample time for line heat-up.
- Observe confined space procedures when entering designated areas.
- When performing maintenance or repairs on electrical equipment, follow established lockout/tagout procedures.
- When performing PM on fluid systems, isolate and safely relieve internal pressure. Follow established lockout/tagout procedures.
- Wear proper protective gear as applicable, such as:
    - Eye protection or face shield
    - Hearing Protection
    - Chemical resistant apron and/or gloves
    - Electrician's insulated gloves.
    - Protective footwear
    - Respirator

While servicing the HVAC system wear safety glasses or goggles. Failure to do so may cause physical injury.

In addition to these warnings, observe all other warnings noted within the vendor data.

2. Startup Procedures

This chapter details the startup procedures for various units of the HVAC system.

2.1. AHU Startup for Cooling Season

1. Visually inspect unit for debris and blockage.
2. Check condensate drains for blockage and proper prime.
3. Ensure that the chiller has been started as per the chillier start up procedure in this section.
4. With the AHU and AHU local fans, disconnect switches in the "OFF" position, check that the AHU and AHU fan circuit breakers are closed.
5. Switch the AHU and AHU fans to the "ON" position at the disconnect switch.
6. Open the chilled water isolation valves.
7. Through the DDC ensure that the AHU is enabled.
8. Verify that the AHU fans are operating properly and the dampers are properly positioned. Veri-

fy correct fan rotation.

9. Ensure that the respective exhaust fans for the AHU are operating.
10. Allow system to run for 30 minutes to reach steady state.
11. Check space temperature and verify that it is within + 2°F of the temperature set point.
12. Check drain for proper operation. Check for leaks and blockage.

2.2. Heating Hot Water Pump Startup for Heating Season

1. Verify that prestart checks have been performed on the HHW pumps.
2. Inspect pump seal for leakage.
3. Ensure that the suction and discharge valves are open.
4. Ensure that the HHW pump disconnect switch is in the closed position.
5. Switch H-O-A to “hand" position.
6. Verify flow through the system.
7. Verify that there are no leaks around the pump seal.
8. Switch H-O-A to automatic.

APPENDIX D – XSLT AND XQUERY TRANSFORMATIONS LISTING

This appendix lists the XSLT and XQuery Transformations used throughout the development of this thesis.

XSLT to Transform PMLibrary.xml to ArchibusML.xml

```
<?xml version="1.0"?>
<xsl:stylesheet exclude-result-prefixes="xsl" version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <ARA>
      <xsl:element name="Table">
        <xsl:attribute name="Name">pmp</xsl:attribute>
        <Header>
          <Parameters AllowUpdates="0" AllowSQLStatements="1" Sort="pmp.pmp_id"
AllowInsertions="1"/>
          <Fields pmp_id="PM Procedure" description="PM Procedure Description"
pmp_type="Procedure Type" tr_id="Primary Trade" skill_id="Skill Required" units="Std.
Units (sq. ft., etc.)" units_hour="Std. Units per Hour"/>
          <SQLExecute/>
        </Header>
        <Data>
          <xsl:for-each select="PMLibrary/PMItem">
            <pmp>
              <xsl:attribute name="pmp_id"><xsl:value-of select="@PMID"/></xsl:attribute>
              <xsl:attribute name="description"><xsl:value-of
select="Description"/></xsl:attribute>
              <xsl:attribute name="pmp_type"><xsl:text>EQ</xsl:text></xsl:attribute>
              <xsl:attribute name="tr_id"><xsl:value-of select="TradeID"/></xsl:attribute>
              <xsl:attribute name="skill_id"><xsl:value-of select="Skill"/></xsl:attribute>
              <xsl:attribute name="units"><xsl:value-of select="Units"/></xsl:attribute>
              <xsl:attribute name="units_hour"><xsl:value-of
select="UnitsPerHour"/></xsl:attribute>
            </pmp>
          </xsl:for-each>
        </Data>
      </xsl:element>
      <xsl:element name="Table">
        <xsl:attribute name="Name">pmps</xsl:attribute>
        <Header>
          <Parameters AllowUpdates="1" AllowSQLStatements="1" Restriction=""
Sort="pmps.pmp_id, pmps.pmps_id" AllowInsertions="1"/>
          <Fields instructions="Instructions" pmps_id="PM Step Code" pmp_id="PM Procedure
Code"/>
          <SQLExecute/>
        </Header>
        <Data>
          <xsl:for-each select="PMLibrary/PMItem">
            <pmps>
              <xsl:attribute name="pmp_id"><xsl:value-of select="@PMID"/></xsl:attribute>
              <xsl:attribute name="pmps_id"><xsl:text>1</xsl:text></xsl:attribute>
              <xsl:attribute name="instructions"><xsl:value-of
select="Instructions"/></xsl:attribute>
            </pmps>
          </xsl:for-each>
        </Data>
      </xsl:element>
      <xsl:element name="Table">
        <xsl:attribute name="Name">pmpstr</xsl:attribute>
        <Header>
```

XSLT to Transform PMLibrary.xml to ArchibusML.xml (cont.)

```
        <Parameters AllowUpdates="1" AllowSQLStatements="1" Restriction=""
Sort="pmpstr.pmp_id, pmpstr.pmps_id, pmpstr.tr_id" AllowInsertions="1"/>
        <Fields tr_id="Trade Code" pmps_id="PM Step Code" hours_req="Hours Required"
pmp_id="PM Procedure Code"/>
        <SQLExecute/>
      </Header>
      <Data>
        <xsl:for-each select="PMLibrary/PMItem">
          <pmpstr>
            <xsl:attribute name="pmp_id"><xsl:value-of select="@PMID"/></xsl:attribute>
            <xsl:attribute name="pmps_id"><xsl:text>1</xsl:text></xsl:attribute>
            <xsl:attribute name="hours_req"><xsl:value-of
select="TradeHours"/></xsl:attribute>
            <xsl:attribute name="tr_id"><xsl:value-of select="TradeID"/></xsl:attribute>
          </pmpstr>
        </xsl:for-each>
      </Data>
    </xsl:element>
  </ARA>
 </xsl:template>
</xsl:stylesheet>
```

XQuery to Transform OMSIML into an Intermediate Markup Language for the DBPS

```
<DBPS>
 {
 for $Trade in /OMSI/Trades/Trade
 return
 <Trade TradeID="{$Trade/TradeID/text()}">
  {
  for $Bl in /OMSI/Buildings/Building
  return
  <Building BlID="{$Bl/bl_id/text()}">
   {
   for $BlSys in $Bl/BlSys
   return
   <BuildingSystem BlSysLevelID="{$BlSys/BlSysLevelID/text()}">
    {
    for $EqItem in $BlSys/EqListing/EqItem, $PMItem in /OMSI/PMLibrary/PMItem
    where ($EqItem/PMReqs/PMID = $PMItem/@PMID) and ($PMItem/TradeID = $Trade/TradeID)
    return
    <EqItem EqID="{$EqItem/@EqID}">
     <PMID>

     </PMID>
     <Frequency>
      {$PMItem/Frequency/text()}
     </Frequency>
     <Hours>
      {$PMItem/TradeHours/text()}
     </Hours>
    </EqItem>
    }
   </BuildingSystem>
   }
  </Building>
  }
 </Trade>
 }
</DBPS>
```

XSLT to Transform OMSIML into an Intermediate Markup Language for the DBPS

```
<?xml version="1.0" ?>
<xsl:stylesheet exclude-result-prefixes="xsl" version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <DBPS xsi:noNamespaceSchemaLocation="file://k:\NPS\Thesis\OMSI\DBPS\DBPS.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
     <xsl:for-each select="OMSI/Trades/Trade">
      <xsl:variable name="TradeID" select="TradeID"/>
      <Trade>
       <xsl:attribute name="TradeID">
        <xsl:value-of select="TradeID"/>
       </xsl:attribute>
       <xsl:for-each select="../../Buildings/Building">
        <Building>
         <xsl:attribute name="BlID">
          <xsl:value-of select="bl_id"/>
         </xsl:attribute>
         <xsl:for-each select="BlSys">
          <BuildingSystem>
           <xsl:attribute name="BlSysLevelID">
            <xsl:value-of select="BlSysLevelID"/>
           </xsl:attribute>
           <xsl:for-each select="EqListing/EqItem/PMReqs/PMID">
            <xsl:variable name="PMID" select="."/>
            <xsl:if test="/OMSI/PMLibrary/PMItem[@PMID=$PMID]/TradeID = $TradeID">
             <EqItem>
              <xsl:attribute name="EqID">
               <xsl:value-of select="../../@EqID"/>
              </xsl:attribute>
              <PMID>
               <xsl:value-of select="."/>
              </PMID>
              <Frequency>
               <xsl:value-of
select="/OMSI/PMLibrary/PMItem[@PMID=$PMID]/Frequency"/>
              </Frequency>
              <Hours>
               <xsl:value-of
select="/OMSI/PMLibrary/PMItem[@PMID=$PMID]/TradeHours"/>
              </Hours>
             </EqItem>
            </xsl:if>
           </xsl:for-each>
          </BuildingSystem>
         </xsl:for-each>
        </Building>
       </xsl:for-each>
      </Trade>
     </xsl:for-each>
    </DBPS>
  </xsl:template>
</xsl:stylesheet>
```

XSLT to Render the DBPS Intermediate XML to HTML

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- Filename:  DBSPRecursion.xsl -->
<!-- Render DBPS Intermediate XML to HTML -->
<!-- This uses recursion to calculate the total hours -->

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <html>
      <head/>
      <body>
        <p align="center">
          <font face="Arial Black" size="4">Naval Air Station Sigonella</font>
        </p>
        <p align="center">
          <font face="Arial Black" size="4">Operations, Maintenance, and
Support Information</font>
        </p>
        <table border="0" cellpadding="0" cellspacing="0" style="border-collapse: collapse"
width="100%" id="AutoNumber1">
          <tr>
            <td width="101">
              <p align="center">
                <img border="0" src="graphics/NASSIGLogo.jpg" width="93" height="93"/>
              </p>
            </td>
            <td>
              <p align="center">
                <font face="Arial Black" size="6">Design-Based Planning
          Submittal</font>
              </p>
            </td>
            <td>
              <p align="center">
                <img border="0" width="93" height="93" src="graphics/LANTDIVLogo.jpg"/>
              </p>
            </td>
          </tr>
        </table>
        <p align="center">
          <font face="Arial" size="4">Construction Project:  P-620</font>
        </p>
        <p align="center">
          <font face="Arial" size="4">Submittal Delivery Date:  1
June 2003</font>
        </p>
        <p align="center">
          <font face="Arial" size="4">Prepared by:  Syska</font>
        </p>
        <p> </p>

        <xsl:for-each select="DBPS">
          <table width="100%" border="1" cellpadding="0" cellspacing="0">
            <thead>
              <tr bgcolor="#800000">
                <td width="15%" rowspan="2">
                  <div align="center">
                    <font face="Arial" size="3" color="#FFFFFF">Trade</font>
                  </div>
                </td>
```

XSLT to Render the DBPS Intermediate XML to HTML (cont.)

```
            <td width="70%">
              <div align="center">
                <font face="Arial" size="3" color="#FFFFFF">Buildings Systems by
Building</font>
              </div>
            </td>
            <td width="15%" rowspan="2">
              <div align="center">
                <font face="Arial" size="3" color="#FFFFFF">Total Trade Hours</font>
              </div>
            </td>
          </tr>
          <tr bgcolor="#800000">
            <td>
              <table width="100%" border="1" cellpadding="0" cellspacing="0">
                <thead>
                  <tr>
                    <td width="25%">
                      <div align="center">
                        <font face="Arial" size="3" color="#FFFFFF">Building</font>
                      </div>
                    </td>
                    <td>
                      <table width="100%" border="1" cellpadding="0" cellspacing="0">
                        <thead>
                          <tr>
                            <td width="50%">
                              <div align="center">
                                <font face="Arial" size="3" color="#FFFFFF">Building
System</font>
                              </div>
                            </td>
                            <td width="50%">
                              <div align="center">
                                <font face="Arial" size="3" color="#FFFFFF">Total BlSys
Hours</font>
                              </div>
                            </td>
                          </tr>
                        </thead>
                        <tbody/>
                      </table>
                    </td>
                    <td width="25%">
                      <div align="center">
                        <font face="Arial" size="3" color="#FFFFFF">Total Building
Hours</font>
                      </div>
                    </td>
                  </tr>
                </thead>
                <tbody/>
              </table>
            </td>
          </tr>
        </thead>

        <tbody>
          <xsl:for-each select="Trade">
            <xsl:if test="count(descendant::EqItem) &gt; 0">
              <xsl:variable name="trhours">
                <xsl:call-template name="sumhours">
                  <xsl:with-param name="list" select="Building/BuildingSystem/EqItem"/>
                </xsl:call-template>
              </xsl:variable>
```

XSLT to Render the DBPS Intermediate XML to HTML (cont.)

```
              <tr>
                <td width="16% ">
                  <div align="center">
                    <xsl:for-each select="@TradeId">
                      <xsl:value-of select="."/>
                    </xsl:for-each>
                  </div>
                </td>
                <td>
                  <xsl:for-each select="Building">
                    <xsl:if test="count(descendant::EqItem) &gt; 0">
                      <xsl:variable name="blhours">
                        <xsl:call-template name="sumhours">
                          <xsl:with-param name="list" select="BuildingSystem/EqItem"/>
                        </xsl:call-template>
                      </xsl:variable>
                      <table width="100%" border="1" cellpadding="0" cellspacing="0">
                        <thead/>
                        <tbody>
                          <tr>
                            <td width="25%">
                              <div align="center">
                                <xsl:value-of select="@BlId"/>
                              </div>
                            </td>
                            <td width="50%">
                              <xsl:for-each select="BuildingSystem">
                                <xsl:if test="count(descendant::EqItem) &gt; 0">
                                  <xsl:variable name="blsyshours">
                                    <xsl:call-template name="sumhours">
                                      <xsl:with-param name="list" select="EqItem"/>
                                    </xsl:call-template>
                                  </xsl:variable>
                                  <table width="100%" border="1" cellpadding="0"
cellspacing="0">
                                    <thead/>
                                    <tbody>
                                      <tr>
                                        <td width="50%">
                                          <div align="center">
                                            <xsl:value-of select="@BlSysLevelId"/>
                                          </div>
                                        </td>
                                        <td width="50%">
                                          <div align="center">
                                            <xsl:value-of select="format-number($blsyshours,
'#')"/>
                                          </div>
                                        </td>
                                      </tr>
                                    </tbody>
                                  </table>
                                </xsl:if>
                              </xsl:for-each>
                            </td>
                            <td width="25%">
                              <div align="center">
                                <xsl:value-of select="format-number($blhours, '#')"/>
                              </div>
                            </td>
                          </tr>
                        </tbody>
                      </table>
                    </xsl:if>
                  </xsl:for-each>
```

XSLT to Render the DBPS Intermediate XML to HTML (cont.)

```
              </td>
              <td>
                <div align="center">
                  <xsl:value-of select="format-number($trhours, '#')"/>
                </div>
              </td>
            </tr>
          </xsl:if>
        </xsl:for-each>
      </tbody>
    </table>
  </xsl:for-each>
 </body>
</html>
</xsl:template>

<xsl:template name="sumhours">
 <xsl:param name="list"/>
 <xsl:choose>
  <xsl:when test="$list">
   <xsl:variable name="first" select="$list[1]"/>
   <xsl:variable name="totalrest">
    <xsl:call-template name="sumhours">
     <xsl:with-param name="list" select="$list[position()!=1]"/>
    </xsl:call-template>
   </xsl:variable>
   <xsl:value-of select="$first/Frequency * $first/Hours + $totalrest"/>
  </xsl:when>
  <xsl:otherwise>0</xsl:otherwise>
 </xsl:choose>
</xsl:template>
</xsl:stylesheet>
```

LIST OF REFERENCES

Ackoff, R. L. (1989). "From Data to Wisdom." Journal of Applied Systems Analysis **16**: 3-9.

Bourret, R. Mapping DTDs to Databases. 2002. http://www.rpbourret.com/xml/DTDToDatabase.htm. Accessed 01 August 2004.

Bourret, R. XML and Databases. November 2003. http://www.rpbourret.com/xml/XMLAndDatabases.htm. Accessed 01 April 2004.

Bourret, R. XML Database Products. 26 March 2004. http://www.rpbourret.com/xml/XMLDatabaseProds.htm. Accessed 01 April 2004.

Brandin, C. (2003). Information Modeling with XML. XML Data Management. A. Chaudhri, A. Rashid and R. Zicari., Addison Wesley Professional.

Chamberlin, D., D. Draper, et al. (2003). XQuery from the Experts : A Guide to the W3C XML Query Language. Boston, Addison-Wesley.

Dayen, I. Storing XML in Relational Databases. http://www.xml.com/lpt/a/2001/06/20/databases.html. Accessed 1 April 2004.

Holman, G. K. (2003). Definitive XSL-FO. Prentice-Hall.

Kay, M. (2003). XSLT Programmer's Reference, 2nd. Indianapolis, IN, Wrox.

Kay, M. (2004a). Xquery for 'hard data probs' was RE: [xsl] XSLT vs Perl. XSL-List, Mullberry Technologies, Inc.

Kay, M. (2004b). A Conversation with Michael Kay on XML Technologies. I. Pedruzzi, The Stylus Scoop.

Kurt, K. (2004). Xquery for 'hard data probs' was RE: [xsl] XSLT vs Perl. XSL-List, Mullberry Technologies, Inc.

LANTDIV. LANTDIV Homepage. http://www.lantdiv.navfac.navy.mil. Accessed 01 August 2003.

Lantz, K. E. (1985). The Prototyping Methodology. Englewood Cliffs, N.J., Prentice-Hall.

Lee, D., M. Mani, et al. (2002). Schema Conversion Methods Between XML and Relational Models. European Conference on Artificial Intelligence (ECAI), Knowledge Transformation Workshop (ECAI-OT), Lyon, France, The NIKE (Nittany Information, Knowledge and wEb) Research Group.

MetaMatrix. Model-Driven Information Integration. http://www.metamatrix.com/whitepapers/Model-DrivenIntegration.pdf. Accessed 01 April 2004.

OASIS. Technology Reports: RELAX NG. 15 December 2003. http://xml.coverpages.org/relax-ng.html. Accessed 01 August 2004.

Provost, W. Normalizing XML. 13 November 2002. http://www.xml.com/pub/a/2002/11/13/normalizing.html. Accessed 13 June 2004.

Ruyak, B., L. Mathwani, et al. Optimizing XML Processing for Performance. http://servlet.java.sun.com/javaone/resources/content/sf2003/conf/sessions/pdfs/3420.pdf. Accessed 1 August 2004.

Ryan, J. Modeling One-to-Many Relationships with XML. 28 January 2003. http://www.developer.com/xml/article.php/1575731. Accessed 01 April 2004.

Shiell, A., J. James, et al. (2002). Practical XML for the Web. glasshaus.

Staken, K. (2001). "Introduction to Native XML Databases." O'Reilly xml.com. http://www.xml.com/pub/a/2001/10/31/nativexmldb.html

Strayton, B. (2003). DocBook XSL: The Complete Guide. Sagehill Enterprises.

U.S. Federal CIO Council (2002). Draft Federal XML Developer's Guide.

Walsh, N. and L. Muellner. DocBook: The Definitive Guide. 31 December 2003. http://www.docbook.org/tdg/en/html/docbook.html. Accessed 24 May 2004.

Williams, K., M. Brundage, et al. (2000). Professional XML Databases. Birmingham, UK; Chicago, Wrox Press.

Xpriori. Xpriori Homepages. http://www.xpriori.com. Accessed 01 April 2004.

ZapThink. Key XML Specifications and Standards. May 2002. http://www.zapthink.com/report.html?id=ZTS-GI101. Accessed 01 June 2003.

LIST OF SOFTWARE APPLICATIONS AND STANDARDS

[ARCHIBUS]. ARCHIBUS Inc., ARCHIBUS/FM v14, http://www.archibus.com/

[AUTHENTIC]. Altova, authentic v2004, http://www.altova.com/products_doc.html

[DOCMAN]. Trieloff, L., DocBook Toolchain Manager, http://trieloff.net/docbook/archive/cat_docman2.html

[DOM1.0]. W3C Recommendation, Document Object Model (DOM) Level 3 Core v1.0, 07 April 2004, http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/

[FOP]. Apache's XML Project, FOP (Formatting Ojects Processor), http://xml.apache.org/fop/

[IDEF1X]. Federal Information Processing Standards Publication, Integration Definition or Information Modeling (IDEF1X), 21 December 1993, http://www.itl.nist.gov/fipspubs/idef1x.doc

[MAXIMO]. MRO Software, MAXIMO® v5, http://www.mro.com/corporate/assetmanagement/maximo-5/index.php

[NEOCORE XMS]. Xpriori, Neocore XML Information Management System, 2004, http://www.xpriori.com/html/products.html

[RELAXNG]. OASIS Committee Specification, RELAX NG, 03 December 2001, http://www.relaxng.org/

[SAXON]. Kay, M., Saxon v8.0, http://saxon.sourceforge.net/

[STYLUSSTUDIO]. Sonic, Stylus Studio v5, http://www.stylusstudio.com/

[SYNTONIX]. DMSi, SyntoniX, http://www.dmsi-world.com/syntonix.htm

[UNIFORMATII]. ASTM, E1557-97 "Standard Classification of Building Elements and Related Sitework - UNIFORMAT II", http://www.uniformat.org/

[XEP]. xAttic, XEP XSL Rendering Engine, http://xep.xattic.com/xep/

[XML1.0]. W3C Recommendation, Extensible Markup Language (XML) 1.0 (Third Edition), 04 February 2004, http://www.w3.org/TR/REC-xml/

[XMLDATAMODEL]. W3C Essay, The XML Data Model, 1997, http://www.w3.org/XML/Datamodel.html

[XMLGUIDE]. U.S. Federal CIO Council, Draft Federal XML Developer's Guide, http://xml.gov/documents/in_progress/developersguide.pdf

[XMLSCHEMA]. W3C Recommendation, XML Schema Part 0: Primer, Part 1: Structures, and Part 2: Datatypes, 2001, http://www.w3.org/TR/xmlschema-0/

[XMLSPY]. Altova, xmlspy v2004, http://www.altova.com/products_ide.html

[XPATH1.0]. W3C Recommendation, XML Path Language (XPath) Version 1.0, 16 November 1999, http://www.w3.org/TR/xpath/

[XPATH2.0]. W3C Working Draft, XML Path Language (XPath) Version 2.0, 23 July 2004, http://www.w3.org/TR/xpath20/

[XQUERY]. W3C Working Draft, XQuery 1.0: An XML Query Languague, 23 July 2004, http://www.w3.org/TR/xpath20/

[XSLT1.0]. W3C Recommendation, XSL Transformations Version 1.0, 16 November 1999, http://www.w3.org/TR/xslt/

[XSLT2.0]. W3C Working Draft, XSL Transformations Version 2.0, 12 November 2003, http://www.w3.org/TR/xslt20/

[XXE]. Pixware, XMLmind XML Editor v2.6, http://www.xmlmind.com/xmleditor/

INITIAL DISTRIBUTION LIST

1. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

2. Mr. Roy Morris
   Facilities Management and Engineering Branch
   Atlantic Division, Naval Engineering Facilities Command
   Norfolk, Virginia

3. Mr. Lino Noble
   OMSI Team
   Atlantic Division, Naval Engineering Facilities Command
   Norfolk, Virginia

4. LT Charles Kubic
   Public Works Operations Officer
   NAS Sigonella
   Sicily, Italy

5. Professor D. C. Boger
   Naval Postgraduate School
   Monterey, California