

**Analysis and Design
For a
Next Generation
Software Release Management System**

by
ROBERT ARTHUR SMITH
B.A., Montclair State University, 1974
B.S., University of Colorado, 1993

A thesis submitted to the
Faculty of the Graduate School of the
University of Colorado in partial fulfillment
of the requirement for the degree of
Master of Science
Department of Computer Science
1999

This thesis entitled:
Analysis and Design for a Next Generation Software Release Management System
written by Robert Arthur Smith
has been approved for the Department of Computer Science

Alexander Wolf

Kenneth Anderson

Dennis Heimbigner

Date

The final copy of this thesis has been examined by the signatories, and we find that both the content and the form meet acceptable presentation standards of scholarly work in the above mentioned discipline.

Smith, Robert A. (M.S., Computer Science)

Analysis and Design of a Next Generation Software Release Manager

Thesis directed by Professor Alexander L. Wolf

SRM is a software release management system that helps developers in releasing software and users in obtaining software. Two key characteristics of SRM are its inherent support for dependencies and its support for distributed operation. However, despite its advantages, two important disadvantages exist in the current version of SRM. First, it is inflexible in terms of its distribution, which needs to be specified a priori. Second, it is inflexible in terms of its appearance, which cannot be configured and is based on a standard set of descriptive metadata.

This thesis contributes a solution to these two problems. Based on a comprehensive survey of existing release management systems, a core set of capabilities is defined. These capabilities are further refined into a detailed design of both a novel repository structure and a new XML-based method of defining the descriptive metadata of a repository.

The advantages of the design are demonstrated through the use of a prototype, which is used to demonstrate how the distribution topology can change over time and how the same repository can be configured to be a software or document release management system.

Acknowledgments

First, I would like to thank Professor Alexander L. Wolf, who was brave enough to give a person who is a fulltime employee an opportunity to do some research. I've found this past year or so of work both interesting and challenging, and for that, well, thanks.

Next, André van der Hoek, teacher, editor, and friend. SRM and NUCM are his brainchildren, and through his patience (sorely tested at times) and teaching ability, I was able to learn some of this field. Moreover, the past two weeks he has been *the* editor extraordinaire. I never realized how much could be done by email in two weeks by a person who is just flat out good at this. What can I say, except that this work would not be done without him. I wish you the best of luck as you move on, and I will miss the weekly meetings. As the immortal songsmith wrote, "what a long, strange trip it's been."

I would also like to thank Bill Hoferer of Sun Microsystems, Inc. He's my department manager, and an ex-BUFF driver, and he gave me the time to finish this work up, even when I could more profitably be used doing what I should be doing. Thanks.

And finally, I want to thank my family, Maryann, Melissa and Andrew, and my father. How can you thank people who support you, even when you're cranky from the long hours? Or who can help you regain your balance with a joke or just having fun? You can't, but this comes close. Thanks, from the bottom of my heart.

Thank you all.

CONTENTS**CHAPTER**

1	Introduction	1
2	Survey	5
2.1	Survey Dimensions	5
2.2	Survey Results	12
3	Capabilities	17
3.1	Metadata	18
3.2	Access Control	21
3.3	Consumer Operations	23
3.4	Producer Operations	24
3.5	Server Operations	25
3.6	Prioritization	27
4	Design	31
4.1	Key Entities and Relationships	31
5	Mapping the Design to a Repository	39
5.1	Defining the Physical Repository	39
5.2	Defining the Logical Repository	42
5.3	Schema, Attributes and Logs	45
5.4	Groups	46
6	Schema, Metadata and Management	50
6.1	Metadata Management	50

6.1.1	Base Data Types	51
6.1.2	Schema Definition	53
6.1.3	Release Item Definition	56
6.1.4	Remaining Metadata	60
6.1.4.1	Access Control Lists	61
6.1.4.2	Licenses	63
6.1.4.3	Key Sets	64
6.1.4.4	Groups	65
7	System Prototype	67
7.1	Creating a Repository.	67
7.2	Adding a Child Repository	70
7.3	Adding Groups	73
7.3	Managing Access Control List, Key Sets and Licenses.	77
7.4	Adding Release Items to the Repository	78
7.5	Retrieving Software from the Repository	83
7.5.1	Software Retrieval Process	84
7.5.2	Web-based Software Retrieval Process	87
7.6	Document Repository	91
8	Conclusion	96
	BIBLIOGRAPHY	97

APPENDIX

A. Document Type Definitions	98
B. Document Repository Upload Screen XSL Style Sheet	100
C. Document Repository Availability Screen XSL Style Sheet	102

FIGURES**Figure**

4.1 Entity Relationship Top Level Diagram.	33
4.2 Entity Relationship Diagram – Attributes Centered on Repository Level Facilities.	34
4.3 Entity Relationship Diagram – Attribution Centered on Release Item Facilities.	35
5.1 Physical Repository Directory Structure.	40
5.2 Logical Repository Directory Structure.	42
5.3 Logical Repository Link to a Release Item.	43
5.4 Logical Repository Links between Parent and Child Repositories.	45
5.5 Logical Repository Links Showing a Group.	47
5.6 Logical Repository Links Showing Groups and SubGroups.	48
6.1 Repository Schema Definition.	54
6.2 Sample Schema Definition.	55
6.3 Release Item Definitions.	56
6.4 Example Release List.	59
6.5 Access Control List DTD.	61
6.6 Example Access Control List.	62
6.7 License DTD.	63
6.8 An Example License XML File.	63
6.9 KeySet DTD.	64
6.10 KeySet DTD.	64

6.11 Group DTD.	65
6.12 Sample Group XML File.	66
7.1 CreateParent Application Interface.	68
7.2 Parent Repository Directory Structure.	69
7.3 CreateChild Application Interface.	71
7.4 Child Repository Directory Structure.	72
7.5 AddGroup Application Interface.	74
7.6 Creating Groups.	75
7.7 Parent Repository AllGroups Directory Contents.	76
7.8 MyGroups Directory Contents.	77
7.9 ManageItem Application Interface.	77
7.10 Sample Upload Graphical User Interface.	79
7.11 Group Selection Dialog.	80
7.12 Strong Dependency Selection Dialog.	81
7.13 Off-site Dependency Selection Dialog.	82
7.14 Directory Structure after DVS Check In.	83
7.15 Software Retrieval Dialog GUI Prototype.	84
7.16 Release Item Information Dialog GUI Prototype.	86
7.17 Sample XSL Style Sheet for License DTD.	89
7.18 Sample License Page Generated by XSL.	90
7.19 Sample Schema Definition for a Document Repository.	91
7.20 Sample Document Release List.	92
7.21 Document Repository Upload Screen.	93

7.22 Document Repository Availability Screen.

TABLES**Table**

2.1 Survey Results of System Capabilities.	14
2.2 Survey Results of System Capabilities.	15
3.1 Metadata Related Capability Definitions.	29
3.2 Access Control Capability Definitions.	29
3.3 Consumer Operations Capability Definitions.	29
3.4 Producer Operations Capability Definitions.	30
3.5 Server Operations Capability Definitions.	30

Chapter 1

Introduction

Configuration management, as a process, has traditionally been applied during the development cycle, and normally has not been extended to the point of releasing software. The objective of a software release management process is to provide, for a group or organization, a means to organize, manage, and control the release of software systems. Software systems can range from simple script files to massive systems containing numerous executable and library files. Understanding the key relationships between system components becomes more difficult as the number of components rises, and ensuring that the proper components are available for any given release of a system becomes a difficult task [12]. With the advent and development of the World Wide Web and the increased use of networks within organizations, different approaches have been developed to address the issue of managing releases of software artifacts. The Software Release Manager (SRM) [21] is one of the systems developed to address the software release management process.

SRM is a distributed application that defines a Configuration Management (CM) policy for managing the release of software artifacts. SRM has two objectives. First, it supports producers in releasing their software and specifying dependencies. Second, it supports the consumer in obtaining released software, in particular by providing the ability to retrieve all related software as one package. SRM implements its policy through the use of the Network Unified Configuration Manager (NUCM)

[22], making use of its distribution functionality to provide release management in a distributed, heterogeneous, and possibly decentralized setting.

Despite the capabilities provided by SRM, there are two major drawbacks with the current version of SRM. First, SRM currently provides a fixed set of attributes that are used to describe and manage artifacts within the repository. This inflexibility makes it difficult for repository owners to customize SRM to their release management and presentation needs. Second, server distribution is fixed, a priori, and cannot be changed. Therefore, all member servers must be configured at the time of repository creation. This limitation prevents servers from joining or leaving a federation of servers, as organizations evolve or change their center of interest.

To address some of the shortcomings of SRM, the work described in this thesis was undertaken. This thesis addresses both of the problems with the existing version of SRM. Based on a comprehensive survey of existing release management systems, a core set of release management capabilities is identified. Moreover, a comprehensive design is contributed that defines how the next generation SRM should be implemented. Finally, a prototype is constructed that demonstrates how the design addresses and solves the problem of inflexibility in the attributes that define a repository and the problem of inflexibility in the distribution process.

Chapter 2 presents a survey of existing release management systems. In particular, a large number of websites are inspected for methodology, content, structure, and relationships as a preliminary step towards providing a set of comparative results used to identify a set of commonly provided capabilities in release management systems. Since software release management is not a

widespread methodology, and evidence of release management practices is not always visible except by inference, the approach to the survey was evolutionary. A certain set of characteristics was identified as being the primary search criteria, and then a large number of websites were investigated from the business and academic areas for presence of the search criteria. As the search evolved, additional criteria were discovered.

As would be expected from such a survey, site presentation varies widely, as did the amount and type of data presented about each individual item. Artifacts are organized in different ways, and the processes for obtaining an artifact varied with almost every site. Chapter 3 organizes the survey capabilities by grouping operations or characteristics based on either similar capabilities or by related information. The capabilities are also prioritized for inclusion within the requirements for the new version of SRM.

Based on this categorization, the design of the next generation SRM is presented in Chapter 4. The design is presented as an Entity Relationship Diagram (ERD), which defines the management of all metadata and key metadata relationships to be stored in SRM. The ERD provides a precise definition of the key concepts and relationships derived from the capabilities defined in Chapter 3, specifically: a logical repository, a physical repository, groups, release items, the schema, data types, metadata, access control lists, licenses, license key sets, registration, strong dependencies, and off-site dependencies.

Information from the logical design, as defined in the ERD, is mapped onto the physical design of the repository. Chapter 5 describes how the physical repository

is the mechanism by which the release manager stores and manages artifacts, and how the distributed aspects of SRM are accomplished. Since SRM represents a Configuration Management policy, the design makes use of NUCM to implement the underlying repository structure and to support the distributed aspects of the release manager.

Accompanying the physical repository structure, as described in Chapter 5, is a flexible attribution mechanism, which allows a repository owner to define the look and attribute sets at the time when a repository is created. Chapter 6 details the attribution mechanism, which maps some of the types defined in the ERD, such as keysets and licenses, to the physical repository structure, and some of the types to XML document type definitions (DTDs) that are to be interpreted by the release management system.

Chapter 7 discusses the prototype of the next generation SRM. The objective of the prototype is to validate the design presented in the previous three chapters. The prototype does not implement the entire design. Instead, it focuses on implementing the solution to the two problems with the current version of SRM, inflexible metadata and distribution. The discussion centers around the process of creating repositories, adding specialized or shared content to the repository, adding release items to the repository, and managing the information in a repository that supports flexible attribute sets.

Chapter 8 summarizes the materials discussed, from the discovery of functionality through to the final prototype, and then finishes with a brief discussion of future directions that can be taken from this work.

Chapter 2

Survey

With the advent of the World Wide Web, many different approaches have been taken to solve the problem of software release management. The functionality provided by each approach varies widely, as does the amount and kind of metadata that is presented to the user. Each approach is generally delimited by the needs of the organization releasing their software, although many websites show a great deal of similarity with their solutions. Further inspection of release management processes at websites shows a very similar approach taken between managing the release of documents and software, and those approaches are being extended now to other items, as well, such as books at Amazon.com or music which can be downloaded over the web.

To gain an understanding of the different approaches to software release management, this chapter presents a survey of the World Wide Web for evidence of software release management. Presented first is the list of capabilities by which the websites are surveyed, followed by the results of the survey. The chapter concludes with the introduction of some observations that can be made from the results of the survey.

2.1 Survey Dimensions

Sites are evaluated for methodology, content, structure and key relationships. During the course of the survey, many capabilities were considered, and applied as

search criteria as the survey progressed. Initial factors were established as characteristics to look for, such as a licensing mechanism, access control and any indication of dependent relationships [22]. Through an iterative process of inspection of websites and evaluation of results, the following list of capabilities was determined to be most relevant to software release management.

- **Weak relationships.** Weak relationships indicate some form of relationship between artifacts where there is no dependency, but there exists some form of relationship between them that a producer may wish to capture. For example, TUCOWS [19] and freshmeat.net [7] provide numerous examples of grouping relationships. An example of a weak relationship could be listing all available software by operating system, or listing an application with all its optional components.
- **Strong relationships.** Strong Relationships are characterized as a dependency relationship between artifacts, in which one artifact requires another artifact for correctness. An example of a Strong Dependency would be the Distributed Versioning System (DVS) [5] that uses the Network Unified Configuration Manager (NUCM) [22] to manage its repositories. The current version of SRM [17] tracks strong dependencies, presents that information to the user, and gives the user the opportunity to download the dependency when the requested artifact is downloaded. A further example of managing a Strong Dependency comes from the Red Hat Packaging Manager (RPM) [15], which allows a software producer to accurately specify dependencies for any item to be released.

- **Passive licensing.** A passive licensing process occurs when the repository policy does not require the user to read or agree to license terms prior to downloading the artifact. No acceptance of licensing terms can be inferred from a passive policy. Examples of a passive licensing policy are the GNU [9] sites. GNU provides a link to the GNU Public License and requests users read the terms and conditions, but there is no requirement to read the license text prior to downloading any item
- **Intrusive licensing.** An intrusive licensing process is one where the licensing terms and conditions are displayed prior to download, and the user signifies acceptance by continuing with the download. For example, SRM [17] can present licensing terms to the user before the user can download, and the user must click a button to signify acceptance.
- **Passive registration.** Passive registration happens when the user is requested to register, but is not required to do so. For example, after a download is completed, SRM [17] presents the user with a registration page, which can be closed, without registering the user.
- **Intrusive registration.** Intrusive registration occurs when the user must register prior to receiving a download. AT&T Research [3] actually presents the license terms on the same page that the user registration information must be entered, and the artifact cannot be downloaded without the user first providing all required information. In one process, AT&T retrieves both the user information and an implicit acceptance of the licensing terms.

- **Historical versions.** Historical versions means making prior versions of an artifact available. This situation occurs when a company or an organization makes older versions of a software package available to users who require backward compatibility for their software systems. IntraNet Solutions [11], which provides document management solutions, provides the user with current and prior versions of a document. SRM [17] is the only release management system that could be determined to provide the same function for software.
- **Metadata presence.** Metadata is descriptive data about any given artifact, such as the release name or version. Typically, such metadata can be key metadata, similar in nature to primary keys in a database. The two most consistent sites for evidence of metadata presence are SRM [17] and TUCOWS [19]. SRM has a required set of metadata that the producing user must complete when uploading a release. TUCOWS exhibits a consistent approach to presenting metadata about an artifact, and for organizing the site.
- **Info beyond the bare essentials.** Information (Info) beyond the bare essentials, or optional metadata, is an indicator that not only is key metadata present, but there is a consistency of related metadata that indicates an organized approach to providing that metadata. It is important to note that there is not a consistent approach among the websites viewed as to what is key metadata and what is extended metadata. For example, GNU [9] has information or description pages for its software in which the file information and revision is usually available. However, in many instances, the software information pages contain information related to the project, project team, or other related projects or software. While

not uniform or consistent in nature, the pages do provide the user with optional information that the publisher wishes to present to the end user.

- **Off-site download/purchase.** Off-site download or purchase indicates that there is a facility provided to either take the user to another web site to download or purchase a related item. In some repositories, such as the NASA Repository Base Software Engineering (RBSE) [13] tool, or freshmeat.net [7], the whole purpose of the site is the management of metadata, and no artifacts are stored. Hence the need to go to another web site to acquire the artifact of interest.
- **Workflow.** Workflow is generally not a visible mechanism, but can be inferred by the presence of processes that may not be included in an application or website. Workflow is characterized by the ability to organize tasks, assign them to non-native processes either internal or external to a system, and to manage the tasks as they progress through the task list. IntraNet Solutions [11] provides an explicit workflow capability, such as sending e-mail with instructions upon uploads or downloads.
- **Revision control.** Such software as RCS [18], CVS [4] or PVCS [10] best exemplifies revision control. In essence, a user can evolve a version of a piece of software, as might be necessary in case of problems or bug fixes. Revision control does differ from historical versions. Historical versions generally mean that there is more than one version of an artifact present in the repository, normally due to the need to support existing software in the field. Revision control supports incremental changes made to an existing artifact, which normally occur during the development phase. Further, historical versions do not prevent

things from breaking, as an artifact can be removed and a required dependency can be broken. Revision control governs the process through which multiple versions of an artifact are stored in a repository. In particular, it ensures that required dependencies are not broken. Both Intranet Solutions [11] and SRM [17] provide revision control facilities.

- **Distribution.** Distribution means that the repository itself may be distributed over several different sites. This should not be confused with replication, in which the contents are replicated at several different sites. In essence, each member repository server of a site would contain some subset of release artifacts, while replication means that each server contains a complete set of release artifacts. SRM [17] is the only system that deliberately indicates that it provides distribution capability.
- **Internal consistency.** Internal consistency is evidenced by link correctness. The state of the repository is validated regularly, where all items and links are tested for correctness. NASA's RBSE [13] performs regular link validation, since it provides only metadata about external artifacts.
- **Auto web page generation.** Auto web page generation is a producer operation, designed to ease the burden on the artifact producer. This process results in a consistent site look and feel. SRM [17] provides a consistent download page that is automatically generated when the user provides the needed metadata describing the artifact. IntraNet Solutions [11] takes it a step further and converts documents to web pages.

- **Internal/external publication.** Released software can be published in one of two different directions, internally or externally. Internal publication means that software is released to only to an organization or group, while external publication releases the software to third parties. Internal publication versus external publication is not constrained to mean intranet or internet, although that may be one configuration, but can also be internal to a particular group within an umbrella organization, or an organization than spans other organizations. It should be kept in mind that internal or external publication is a publishing scheme, and individual access permissions can be applied to either internal or external publication, or both. IntraNet Solutions [11] provides a distinct separation for internal versus external publications as part of its repository mechanism, while others can be used either as an internal or an external tool.
- **External submissions/uploads.** Each repository owner will likely face the problem of controlling uploads into the system. Many different policies exist in which release items can be added to a repository, and a repository should allow the repository owner to use a policy of their own choice. For example, TUCOWS [19] primary reason for existence is to provide software for downloads, primarily from developers who are not commercial producers. Therefore, they provide a system where the user submits candidate software that is evaluated prior to placement in the repository.
- **Security or access control.** Security and access control are always issues when it comes to owning a repository or server. Managing or limiting access can be applied at several different levels, such as directly to the repository, to a group or

organization within the repository, or to artifacts within the repository. Access control can also apply discrete actions, such as uploading or downloading an item. IntraNet Solutions [11], SRM [17] and freshmeat.net appear to have the only defined security policy in place with respect to access control of a repository.

- **Bundled downloads.** Bundled downloads is the packaging of a selected artifact and its related strong dependencies and off-site dependencies into a single bundle for downloading. SRM [17] packages artifacts in this manner, while Gamelan [8] bundles all selected items in the shopping cart. Gamelan [8] downloads each item individually to the end user.
- **Bundled licenses.** Bundled licensing relates to bundled downloads. Some sites provide the user with the capability of bundling downloads, and provides the ability to bundle licenses for each item in the download bundle. Gamelan [8] and SRM [17] both provide bundled licenses to users that download bundled artifacts.

2.2 Survey Results

Base on the presented set of capabilities, one dozen sites were evaluated, as shown in Tables 1 and 2. These twelve websites are: GNU [9], AT&T Research Laboratories [3], AT&T Laboratories [2], the World Wide Web Consortium (W3C) [23], TUCOWS [19], Netlib [14], NASA [13], IntraNet Solutions [11], Alexander Wolf's publication pages [1], the existing Software Release Manager (SRM) [17], the Software Engineering Institute (SEI) [16], Gamelan [8], freshmeat.net[7] and Red Hat Packaging Manager (RPM) [15]. The results of the survey are shown in Tables 1 and

2, indicating the sites visited and their individual contributions based on inspecting visible characteristics of each site.

Table 2.1 Survey Results of System Capabilities.

Search Characteristics	GNU	AT&T Res.	AT&T Labs	W3C	TUCOWS	Netlib	NASA
Weak relationships	No	Yes	Yes	Yes	Yes	No	Yes
Strong relationships	No	No	Yes	~	No	?	No
Passive licensing	Yes	Yes	Yes	Yes	No	No	No
Intrusive licensing	No	Yes	No	No	No	No	No
Passive registration	No	No	No	No	No	No	No
Intrusive registration	No	Yes	Yes	No	No	No	No
Historical versions	No	No	No	Yes~	No	No	No
Metadata presence	Yes	No	Yes~	Yes~	Yes	No	Yes
Info beyond bare essentials	Yes	Yes	Yes	Yes	Yes	No	No
Off-site download/purchase	Yes	No	No	Yes	Yes~	No	Yes
Workflow	No	No	No	Yes	No	Yes	No
Revision control	No	No	No	No	No	No	No
Distributed	No	No	No	No	No	No	Yes~
Internal consistency	No	Yes	No	No	No	Yes	Yes
Auto web page generation	Yes~	Yes	?	?	No	?	Yes
Internal/external publication	No	Yes	Yes	No	No	Yes	Yes
External submissions/uploads	Yes	No	No	Yes	No	No	~
Security or access control	No	No	No	No	No	No	No
Bundled downloads	No	Yes~	No	No~	No	No	No
Bundled licenses	No	No	No	No	No	No	No

Table 2.2 Survey Results of System Capabilities.

Search Characteristics	IntraNet Solutions	Alex's Docs	SRM	SEI	Gamelan	freshmeat.net	Red Hat RPM
Weak relationships	?	Yes	No	Yes	Yes	Yes	Yes
Strong relationships	?	No	Yes	No	No~	No	Yes
Passive licensing	No	Yes	No	Yes	No	Yes	No
Intrusive licensing	No	No	Yes	No	Yes	No	No
Passive registration	No	No	Yes	No	No	No	No
Intrusive registration	No	No	No	No	Yes	No	No
Historical versions	Yes	No	Yes	No	No	No	Yes
Metadata presence	Yes	No	Yes	Yes~	Yes	Yes	Yes
Info beyond bare essentials	Yes	No	No	No	Yes~	Yes	No
Off-site download/purchase	No	No	No	Yes	No	Yes	Yes
Workflow	Yes	No	No	No	Yes~	Yes~	Yes~
Revision control	Yes	No	Yes	No	No	?	Yes
Distributed	No	No	Yes	No	No	No	No
Internal consistency	Yes	No	Yes	No	Yes~	No~	Yes
Auto web page generation	Yes	No	Yes	?	Yes~	Yes	No
Internal/external publication	Yes	Yes	Yes	?	Yes~	No	No
External submissions/uploads	Yes~	No	No	?	No	Yes	No
Security or access control	Yes	No	No	No	Yes~	Yes	No
Bundled downloads	No~	No	Yes	No	Yes	No	Yes
Bundled licenses	No~	No	Yes	No	Yes	No	No

The survey keys are as follows:

- No – no evidence was found that supports the capability.
- Yes – evidence found.
- Yes~ - evidence inferred that the capability exists.
- No~ - inferred that the capability does not exist.
- ? – Unable to determine whether the capability exists.
- ~ - Possible evidence of existence, based on inference.

In conclusion, the sites that were investigated showed that some manage only documentation, while others manage software or metadata about software external to their site. Investigating the documentation sites (e.g. SEI and Alexander Wolf's publications) shows a very similar process in releasing documents as compared to releasing software. Metadata is used to manage documents and software alike, and both can have access controls placed on uploads and downloads. As can be seen from the survey results, no one site provides all capabilities, but each site has a particular focus that come from providing some combination of the capabilities. A flexible and configurable software release management system should be able to provide the capabilities shown in the survey, and provide key functionality to sites with similar operations to those surveyed.

Chapter 3

Capabilities

From the survey results, it becomes clear that many of the capabilities that were observed or inferred can be grouped into various functional categories. A re-evaluation was made of the discovered capabilities, abstracting each and categorizing them into groups. Some of the discovered characteristics were found to be unrelated to the role of a software release management system. However, based on further analysis, additional capabilities were discovered and added to the list of twenty defined in Chapter 2. These additional capabilities are mostly intangible, or deal with issues related to distribution.

Five categories are distinguished: metadata, access control, consumer operations, producer operations, and server functionality. Metadata is any data that relates directly to an artifact. It can vary based on need or policy, and is used for identification, description, association, and location. Access control consists of processes that govern access to repositories and the artifacts within the repository. It can be applied to uploads or downloads independently. Consumer operations support the repository consumer with ease of use functionality, such as bundled downloads and search facilities. They can be applied to off-site items as well as items stored directly in the repository. Producer operations are operations supporting the repository producer or software author. They ease the workload in distributing and making available releases in a variety of ways. Server operations provide

functionality that is unrelated to consumers or producers. These operations are primarily used for repository management and operational support.

These categories form the basis for the design of the next generation software release manager. Below, each category is discussed in detail.

3.1 Metadata

Metadata is descriptive data about an artifact. Of the seven defined characteristics, the first four focus on defining an artifact in the repository, while the last three focus on retaining relationship or association information. The metadata that each repository retains must be defined at the time of repository creation, essentially forming a schema for the repository. Metadata definition can vary based on the needs of the repository owner or fit a defined policy of an organization. Overall, metadata can be used not only for identification and association, but also for description and location.

Table 3.1 illustrates the metadata-related characteristics identified in the survey or discovered during the evaluation process. Each capability is described below.

- **Key Metadata.** Key metadata is the foundation on which a generic software release management system is built. Key metadata is similar, in many ways, to primary keys in a database, and uniquely identifies an artifact in the repository, and must be capable of being flexibly defined at the time that the repository is created. For example, a repository owner may decide that a name and a version number describe an artifact, whereas a different repository owner may require that

an artifact is defined by a name, a version number, and an operating system. Therefore, a generic software release management system must understand how to manage flexible key metadata in order to support a repository owner's metadata requirement. Closely related to understanding key metadata are historical versions, strong and weak relationships, and off-site dependencies. The metadata associated with these capabilities are not just descriptive metadata. Each of these capabilities has special operating considerations that have to be kept in mind, particularly with respect to metadata. For example, an off-site dependency may have to be retrieved by the repository via the web, rather than retrieved from the repository. A software release management system must be able to handle these semantics.

- **Required metadata.** Required metadata is any metadata that is required by the repository owner to be provided when an artifact is added to the repository. Essentially, required metadata is defined by the policy of the owning organization. By definition, key metadata is included in the set of required metadata. For example, SRM [17] requires a large set of metadata that includes, among other things, the name and e-mail address of the person producing the software.
- **Optional metadata.** Optional metadata is just as the name describes, optional. The metadata is not required from the software producer at the time when the artifact is added to the repository. SRM [17] does not require that a producer add any comments to the "What's New" field.

- **Historical versions.** Historical versions occur when a repository retains multiple, discrete versions of a software item. Java [tm] is an excellent example. Sun Microsystems, Inc. maintains all released versions of Java for backward compatibility for their end user needs.
- **Strong relationships.** Strong relationships are dependencies between two pieces of software, without which one would not operate. For example, the current version of SRM [17] requires NUCM [22] to manage the repository.
- **Weak relationships.** Weak relationships are associations between items that do not otherwise exhibit a direct or strong dependency on each other. TUCOWS [19] groups all Windows 98 software together, and further groups all Windows 98 HTML editors together. Freshmeat.net [7] also groups related items together, such as Web applications, which have further sub-groups such as administration or on-line shopping tools.
- **Off-site dependencies.** An off-site dependency occurs when a dependency exists between an item in an SRM repository and an item in a non-SRM repository. An example of this type of relationship would be a piece of software that is retained in the repository and has a dependency on Java. Upon release, a URL to a specific version of Java needs to be provided. Since it is unlikely that all software repositories will be SRM compatible repositories, this capability becomes more important as the numbers of software artifacts grows. Furthermore, with many repositories becoming web-enabled, it is clear that an off-site dependency should be URL-based to ensure a consistent access capability.

3.2 Access Control

Access control is the combination of processes that govern access to the repository, artifacts within the repository, or groups. Access control can be applied to either uploads or downloads, independent of each other, and can be applied at different locations in the repository, based on the policy of the repository owner.

Table 3.2 describes the various characteristics of access control, as found in the survey. Based on the evaluation process, licensing and registration processes were grouped together with security related processes, as they jointly affect producers and consumers of the repository. The following expands on the definitions given in Table 3.2.

- **Passive licensing.** Passive licensing, essentially, is a licensing process that the user can ignore. Although there are license terms that the software producer wants the user to read and agree to, the user can bypass license acceptance easily. As with GNU, a user does not need to read the GNU public license, or signify acceptance of the license terms, in order to download any item within the GNU repository.
- **Intrusive licensing.** Intrusive licensing is a licensing process that requires that the user actively signify acceptance of the licensing terms. If the user tries to bypass the licensing terms, the software cannot be downloaded. Typically, intrusive licensing is accomplished by inserting the license terms into the download process chain prior to retrieving the artifact, with accept and reject choices given to the user.

- **Passive registration.** Passive registration places no requirement on the user to provide registration information in order to download the requested item. The user can send blank information in, or in the case of a GUI, dismiss the dialog box without providing the information.
- **Intrusive registration.** Intrusive registration requires that the user provide all required registration information to the system prior to downloading the artifact. If the user fails or refuses to provide the information, the system will not allow the download to continue. As with intrusive licensing, the registration process is inserted into the download process chain prior to retrieving an item. Potentially, the server, prior to downloading the artifact to the consumer, can validate the information to a certain degree.
- **Access control – downloads.** Downloads can be restricted through the use of permissions, or through the use of an access control list. As a matter of policy, the repository owner may want to restrict downloads of artifacts to only members of a specific group, and may use an access control list to restrict the availability of a particular release of an artifact.
- **Access control – uploads.** Uploading of software may be restricted to members of a specific group associated with a repository, and modifications of existing artifacts can be limited to the artifact owner or producer. Access would then be controlled by requiring a producer to supply a user id and password.

3.3 Consumer Operations

Consumer operations, as shown in Table 3.3, are operational aids to ease the work of a user that downloads software from a repository. While web-based systems are the model that much of these operations are derived from, the operations do apply to non-web-based systems, as well. The following describes the consumer operations in more detail.

- **Bundled downloads.** Bundled downloads are a service that the repository server provides. As provided by SRM [17], the user can choose to download a released item and all of the items it depends on, and SRM will bundle all of the items into a single tar file, which is then passed to the user.
- **Off-site download.** A foreign system, or off-site dependency, is one that the release management system recognizes as a dependency of an artifact. The dependency is, however, not managed by SRM but by some foreign Web site. In this situation, the release management system must also have a link to the location where the off-site dependency can be obtained. When a user requests an item, he or she can request that the system retrieve the item as part of the download process for the given artifact.
- **On-site purchase.** A logical use of a software release manager would be to make use of one as a web-commerce repository. For example, Gamelan has a developer specific site in which a developer can download source code or sample applications. In addition to the developer specific site, Gamelan also operates a web commerce site for the purchase of software. A joint repository that supports both operations would, by definition, be part of an on-site purchasing system.

- **Shopping cart.** A shopping cart is a mechanism by which a server aggregates choices made by a consumer, with respect to items that a user wishes to download or purchase. The items chosen are normally unrelated to each other and do not exhibit any strong dependencies with each other, except incidentally. Typically, shopping carts are associated with web commerce sites, but can be extended for use with a release management server. For example, a shopping cart can be used as a mechanism to collect all items that a user requests. When the user has completed shopping in the repository, the server could then place all requested items, and their related dependencies, into a single bundle for download to the user.
- **Searching the repository.** This capability provides a consumer with the ability to search the metadata retained by a repository for information. A typical example would be the situation where a consumer remembers certain information about an artifact, but is unable to locate the artifact directly. The search function locates the set of potentially matching artifacts based on the supplied information.

3.4 Producer Operations

Producer operations are operations that aid the software producer in making available artifacts, or information about the release, to interested parties. Since the web survey discloses only visible clues to software release management, the producer operations were derived during the post-survey evaluations.

Table 3.4 lists the derived producer operations. Each of these producer operations deserves some additional detail.

- **Mailed confirmation.** Mail confirmation is a step that can be part of the uploading process in which the server mails a confirmation to the producer, and perhaps to the repository owner. For example, a repository owner may want to keep an eye on how much software is uploaded to make sure enough space is available.
- **Release channels.** As an example of the definition in Table 3.4, the repository may have a configured list of third parties or news groups that are interested in receiving notification when software of a particular type is available, such as configuration management tools. The repository server would distribute the information through the release channels on behalf of the producer, based on direction from that user.
- **Release to multiple repositories.** A software producer may want to release an application to several repositories that are not members of a single federated repository. The producer would upload software to an initial repository, and the client software would repeat the actions with all of the other repositories. This would require that the client understand the uploading processes at the other repositories.

3.5 Server Operations

Server operations predominantly support the repository owner, providing repository management functionality and operational support. Table 3.5 describes server operations, which, like producer operations, were discovered during the post-survey evaluation. Server operations are discussed in greater detail below.

- **Distribution.** Distribution allows multiple repositories to act together in a federated manner. For example, a repository could exist on a server located in Boulder, CO while another repository could be located in New York, NY. The repositories could then work together, acting as a single unit, while residing in different locations.
- **Mirroring.** Mirroring is the ability to replicate the entire contents of one repository to another repository that is completely independent of the first. As an example, TUCOWS [19] has mirror repositories located around the world. This helps users locate and use the server nearest them, thereby reducing network traffic.
- **Internal consistency.** This is a process through which a repository ensures that its metadata is in a valid state, as well as any links exist and are valid. The NASA [13] Repository Based Software Engineering system performs link validation as it contains only metadata, and does not store software.
- **Logging.** Logging is a facility that records all operations within a repository, as they take place. The logs would then provide the repository owner with the means to determine what occurred as operations take place. In the situation where a producer experiences a problem during an upload, the repository owner would then view the log as part of the troubleshooting process.
- **Statistics.** Statistics can be derived from the activities that take place within a repository. Generally, the statistics are derived from the activity logs, and can give the repository owner insight into system usage. A typical use for statistics

would be to show uploads, downloads and inspections for a given set of artifacts over a period of time.

- **Dynamically add or remove servers.** Related to distribution is the ability to add or remove a server dynamically. A server may need to leave a federation due to policy change, or for maintenance. Building on the distribution example, the server located in New York may be removed due to an alliance change. As a result of the change, it would be undesirable to completely rebuild the repository when a server leaves. This capability would manage that process for the user.

3.6 Prioritization

The survey results show that most of the characteristics described in the Metadata, Access Control and Consumer Operations topics were directly visible from the websites, and represent either information that most end users understand, or operations that end users are familiar with. The capabilities characterized as Producer and Server operations were derived from familiarity with the current SRM or other areas of computer science.

In preparation for the design phase, each of the items presented was reviewed with the aim of making an effective prototype that addresses the goals of the thesis. Towards that end, the following items were prioritized as directly affecting distribution, metadata and relationships: key metadata, required metadata, optional metadata, historical versions, strong relationships, weak relationships, off-site dependencies, licensing, registration, access control, distribution, and dynamically

add or remove servers. Licensing will address the needs of both passive and intrusive licensing, as will registration.

With regards to the remaining capabilities, they represent capabilities that are already demonstrated in other systems, or represent capabilities that do not apply directly to a software release management repository but can be built on top of the repository. Those capabilities are: mirroring, internal consistency, logging, statistics, mailed confirmation, release channels, release to multiple repositories, bundled downloads, off-site download, on-site purchase, shopping cart, and searching the repository. These capabilities will be ignored throughout the remainder of this thesis.

Table 3.1 Metadata Related Capability Definitions.

Capability	Definition
Key metadata	Metadata used to uniquely identify any given artifact within a repository.
Required metadata	Metadata required to be completed for an object in the repository. This includes key metadata.
Optional metadata	Metadata not required to be completed for an object in the repository.
Historical versions	Maintaining previous releases of an artifact in the same repository.
Strong relationships	Dependencies, or maintaining and identifying those objects which another object is dependent on for correct operation.
Weak relationships	Grouping, or associating related items that do not have a dependency relationship
Off-site dependencies	A dependency on an artifact in a non-SRM repository.

Table 3.2 Access Control Capability Definitions.

Capability	Definition
Passive licensing	Does not interfere with or prohibit downloading of an artifact
Intrusive licensing	User must signify agreement with the license terms prior to download
Passive registration	Places no constraint on the user to register prior to download
Intrusive registration	User must register to receive a download
Access control – downloads	Downloads are restricted by applying an access control list to a group
Access control – uploads	The ability to upload, modify or remove is restricted to those users that supply a valid user id and password

Table 3.3 Consumer Operations Capability Definitions.

Capability	Definition
Bundled downloads	An artifact and any related dependencies are bundled together into a single package for downloading to the user.
Off-site download	Download of a foreign system dependency.
On-site purchase	Depending on the purpose of a repository, web commerce may play a role. The purchasing system would be a large system by itself, and does not necessarily interact with the repository directly.
Shopping cart	In essence, a web specific operation, which bundles unrelated items for downloading, possibly for purchase.
Searching the repository	Searching the repository metadata for specific artifacts related to the search criteria.

Table 3.4 Producer Operations Capability Definitions.

Capability	Definition
Mailed confirmation	Mailed confirmation to the developer, and possibly the repository owner, indicating the success of the upload operation.
Release channels	Software is often released to groups of interested third parties. In addition, notification of new releases can be posted to news groups and user lists. Release channels would support that operation, by providing a method of distributing artifacts, notification or both to interested user.
Release to multiple repositories	Releasing a single artifact to multiple, unrelated repositories.

Table 3.5 Server Operations Capability Definitions.

Capability	Definition
Distribution	Having multiple storage locations act in a concerted manner as part of a single repository.
Mirroring	Replicating the contents of an entire repository to another, independent repository.
Internal consistency	Validating the contents of a repository, along with the related metadata.
Logging	Recording the activities of a repository.
Statistics	Deriving statistical data about each activity in a repository.
Dynamically add or remove servers	For distributed systems, the ability to dynamically add or remove a member server, either for maintenance, or as alliances change.

Chapter 4

Design

Having categorized the capabilities required of the new software release manager, the next step is creating the system design. The design focuses on the informational areas affecting SRM: metadata, relationships, and logging. While many related capabilities were identified as relating directly to software release management, for the purposes of this thesis, the candidates are reduced to those factors directly involved with the base repository functionality.

This chapter introduces the key entities and relationships that exist to support relating metadata, relationships and logging within the repository. This introduction is based on an entity-relationship diagram (ERD) that not only identifies each entity and relationship, but also illustrates the associated attribute related to each entity. Below, we first introduce the ERD and then discuss each of its parts in more detail.

4.1 Key Entities and Relationships

From the list of capabilities defined in Chapter 3, we have identified the key entities and relationships that a software release management system should incorporate. From the information described in Section 3.2, an Entity Relationship Diagram (ERD) can be drawn. Figures 4.1, 4.2, and 4.3 show the ERD as it defines the system. Figure 4.1 depicts the top-level diagram, showing all major entities and relationships. Figures 4.2 and 4.3 depict attribution for each of the entities defined.

The following describes the list of entities and relationships of the entity relationship diagram as it defines the repository.

- **Repository.** The repository retains and manages all items related to software release management. The logical repository is the focal point of the software release manager. It manages all information and key relationships, and retains all items in one or more physical repositories. A physical repository contains attributes that describe its host name, port number, home directory and URL. A logical repository is defined by the schema, and is organized by the groups contained within. The logical repository can also retain licenses, which are available throughout the repository. Further, the logical repository maintains a log of events that take place within the repository. The logical repository is described by attributes that retain information such as the repository name, the owner name, contact email, and any presentation text. The remaining information in this section focuses on the entities and relationships within the logical repository.
- **Groups.** Groups are the nucleus around which release items are managed. As can be seen from TUCOWS [19], groups can also have subgroups, which, in turn, can be nested to an arbitrary level. Each group, regardless of level, contains and manages release items. A group is described by the following attributes: name, owner, password, date created, and presentation text. A group can also be protected by an access control list.

SRM E-R Diagram

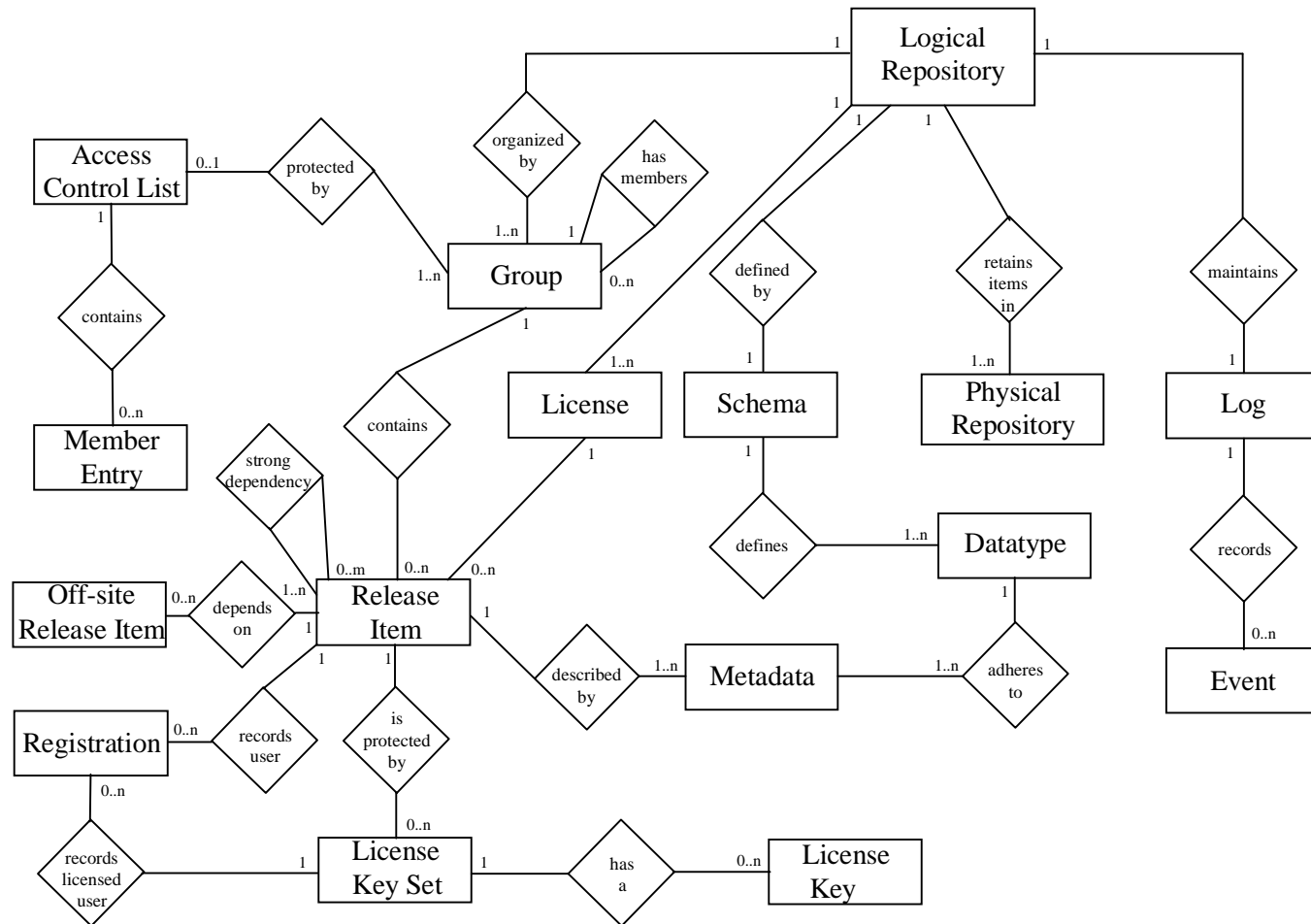


Figure 4.1 Entity Relationship Top Level Diagram.

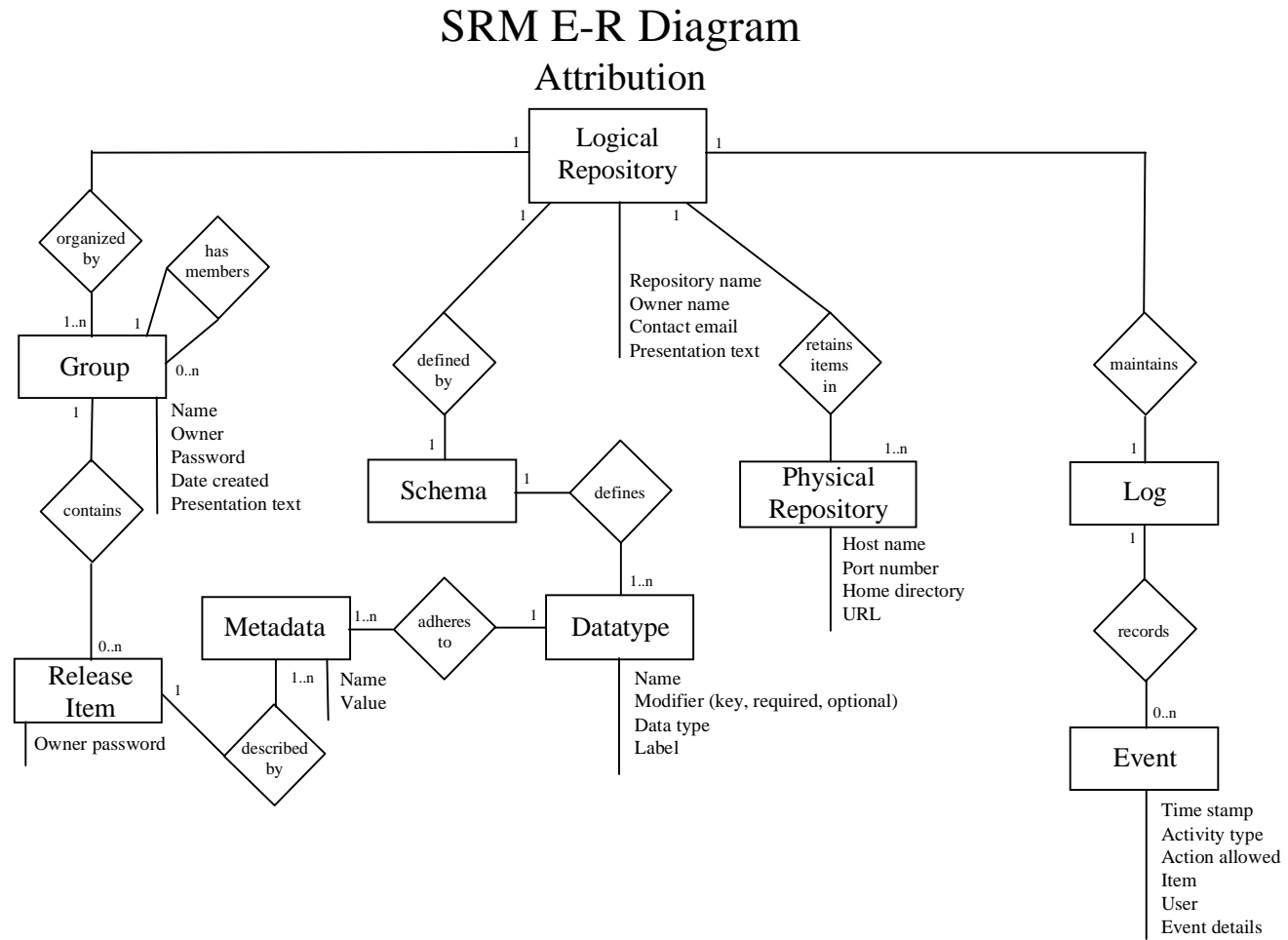


Figure 4.2 Entity Relationship Diagram – Attributes Centered on Repository Level Facilities.

SRM E-R Diagram

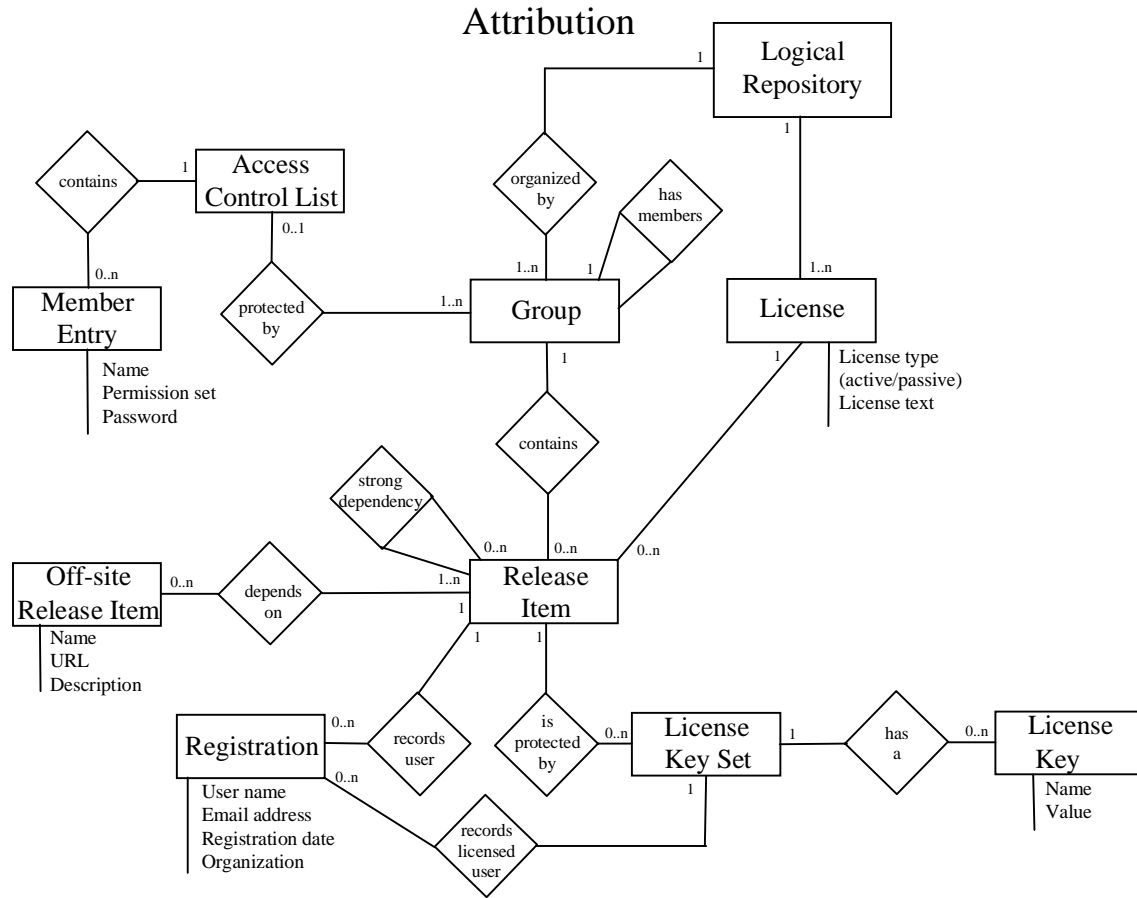


Figure 4.3 Entity Relationship Diagram – Attribution Centered on Release Item Facilities.

- **Access control lists.** An access control list provides protection to a group. An access control list contains a list of the users who are allowed access to the repository contents, and each member has a name, password and permission set associated with them. Multiple groups may use a single access control list.
- **Schema.** A single schema defines the logical repository. The schema is defined by a number of data types, or attributes. Chapter 6 discusses the role of schema and its relationship to release items.
- **Data types.** Data types are described with a name, a label, a modifier, and a type. A modifier can be one of three values: key, required, or optional. Data types are used to define a schema, and are the constraint to which metadata adheres.
- **Metadata.** Metadata describes a release item and adheres to the data types used to define the schema. Metadata also contain name value pairs used to store data.
- **Release items.** A release item belongs to the group to which it is released, and an individual release item may belong to multiple groups. A release item is described by metadata, which adheres to the data types that are defined by the schema. A release item may have strong dependencies on other release items that are contained within the repository. A release item may also depend on an off-site dependency. In addition to the metadata that describes a release item, a release item also has the owner's password as an attribute. A release item may be protected by a license key set. Further, a release item can record any number of user registrations.

- **Off-site dependencies.** A release item may also have one or more foreign system dependencies, which the repository does not manage. The release item metadata will need to retain a reference to the off-site dependency, and the repository will need to understand the off-site dependency to aid producers in managing their releases. A name, a URL, and a description describe a off-site dependency.
- **Registrations.** Registrations apply directly to a single release item. A registration records the following consumer information for a release item: user name, e-mail address, registration date, and organization. In the situation where a user provides a license key, a registration is also recorded for the licensed user.
- **Licenses.** Many licenses can be available throughout a repository, and a release item can be protected by a given license, based on the requirements of the producer. A release item will be associated with at most one license.
- **License key set.** A license key set can protect a release item, and may contain a number of license keys. While a release item is protected by a single license key set, multiple license keys can be applied to a release item through the key set. License key sets may also record any number of licensed user registrations.
- **License key.** A license key set can contain multiple license keys. License keys contain a Name and Value pair as its attributes. A license key is similar in nature to keys that are required to unlock a CD-ROM based software installation mechanism or server based software.
- **Logs.** A repository retains a log that records the events of the system. An event contains a time stamp, activity type, whether the action was allowed, the name of

the item, the user identifier, and any event details that the repository supplies for that event.

In conclusion, the major entities and relationships have now been defined for the new version of a software release manager. From the top-level diagram, as shown in Figure 4.1, we can see the hierarchical structure used to manage release items. A repository is defined by a schema, which is defined by basic data types. A repository is organized by the groups defined within it, and can have subsidiary member groups. Each group, and subgroup, can be protected by an access control list, which contains entries for each member of the list. Each group or subgroup contains a list of release items, which are stored in a file. A release item is described by metadata, which adheres to data types that define the schema. Each release item may have off-site dependencies, and may also have strong dependencies on other release items in the repository. Release items can be protected by license key sets, and registrations may be recorded for that release item. Finally, a repository can log all events that take place within the system.

Chapter 5

Mapping the Design to a Repository

Having created the logical design for the system, the next step is to map some of the entities specified in the Entity Relationship Diagram (ERD) to a physical repository. The Network Unified Configuration Manager (NUCM) [22] is chosen as the underlying repository, since it provides both distribution and configuration management capabilities.

This chapter describes the entities that are mapped to the physical repository, how the physical repository is structured, and how the metadata is stored. Also presented is a solution to the problem of dynamically adding and removing servers from the repository federation.

5.1 Defining the Physical Repository

From the ER Diagram, as described in Chapter 4, certain entities present themselves immediately as candidates for either directories or files to be created or stored in the repository. In addition to the entities and relationships defined in the ERD, distribution is an identified capability. Therefore, another consideration in mapping entities to the physical repository is the concept of parent and child repositories.

A parent repository is the lead repository in a federation of servers, and performs all management functions. The schema definition of the parent repository is the defining schema for the repository federation. A child repository, on the other

hand, is a member repository that provides release items to be made available through the parent repository.

In order to define the repository structure, it is necessary to take into account any structure necessary to separate the contents owned by a repository from those items shared by the repository federation. To do so, the concept of physical and logical repository is extended to the structure of the repository. For the purposes of the repository, the physical repository is the area in which items locally owned by a member repository are retained, whether the repository is a parent or child. The logical repository is the area where items are shared by the member repositories.

Each physical repository, whether a parent or child repository, will have the same directory structure, as shown in Figure 5.1. To differentiate the physical from the logical repositories, all physical directory names start with the prefix My. Directory names are capitalized, while file names are not.

- `/MyGroups/Groups/SubGroups/attributes`
 | `/metadata`
 | `/releaselist`
 `/attributes`
 `/metadata`
 `/releaselist`
- `/MyReleases/ReleaseItem/Version/archive`
 `/dependency_counter`
 `/registrations`
- `/MyAccessControl/access_control_list`
- `/MyKeySets/keyset`
- `/MyLicenses/license`

Figure 5.1 Physical Repository Directory Structure.

The following discussion details how the physical repository, as described in Figure 5.1, maps back to the Entity Relationship Diagram of Figure 4.1

- **Groups.** Groups translate to directories and sub-directories. A group contains files that retain metadata about the group's release items, and about the group itself. Since groups can contain sub-groups, each sub-group will then be contained in a sub-directory of the parent group, and will contain its own metadata and group attributes files. All groups stored in the physical repository map are located within the MyGroups directory.
- **Release items.** Release items will be stored in a parent directory, with each version of a release item going into its own sub-directory. In addition to the archived release file, each version directory will contain its own dependency counter file and the list of registrations for that version of the release item. The dependency counter allows the logical repository to ensure that no release item is removed while there are still dependencies on the item. All release item directories are stored in the MyReleases directory.
- **Access control list.** Access control lists are applied to groups, and multiple groups may share a single access control list. Thus, the access control lists will be stored as separate files in a directory dedicated to access control lists. All access control lists are stored in the MyAccessControl directory of the physical repository.
- **Key set.** Key sets are similar to access control lists, in terms of availability. Therefore, each key set will be contained in its own file located in a directory dedicated to key sets. Key set files are stored in the MyKeySets directory.

- **License.** License agreements are similar to access control lists and key sets, in terms of availability. As with both, each license will be contained in its own file located in a directory dedicated to licenses. Licenses agreement files are stored in the MyLicenses directory.

5.2 Defining the Logical Repository

The logical repository is the part of SRM where all repository components and files are visible, making access to these components transparent to all federation member repositories. Each directory within the logical repository is a link to a file or directory in the physical repository, and is not an actual object. NUCM provides the capability of maintaining pointers to artifacts or directories in other repositories, effectively providing distribution. As a repository is added to the federation, a repository links its items into the parent logical repository, and then points its own logical links to the parent logical repository.

To create a logical repository, each repository will have the following directory structure, as shown in Figure 5.2. Logical directory names start with All.

- `/AllGroups/group_ptr`
- `/AllReleases/release_item_ptr`
- `/AllAccessControl/access_control_list_ptr`
- `/AllKeySets/keyset_ptr`
- `/AllLicenses/license_ptr`
- `/AllLogs/log_ptr`

Figure 5.2 Logical Repository Directory Structure.

As a matter of convention, all logical repository member names are prefixed with the word All. For a stand-alone or parent repository, after the directory structure has been created, the next step is to create the links from the logical repository to the physical repository. A pointer is created from AllGroups to contents of the MyGroups directory, as is similarly done for AllReleases, AllAccessControl, AllKeySets and AllLicenses. AllLogs will contain pointers to the log files of each physical repository. AllSchemas and AllAttributes point to the schema and attribute file of the parent repository. When new contents are placed at the root level of a My directory, a link is drawn from the corresponding All directory to the newly added item. Figure 5.3 shows an example of an item added to the MyReleases directory and the link from AllReleases.

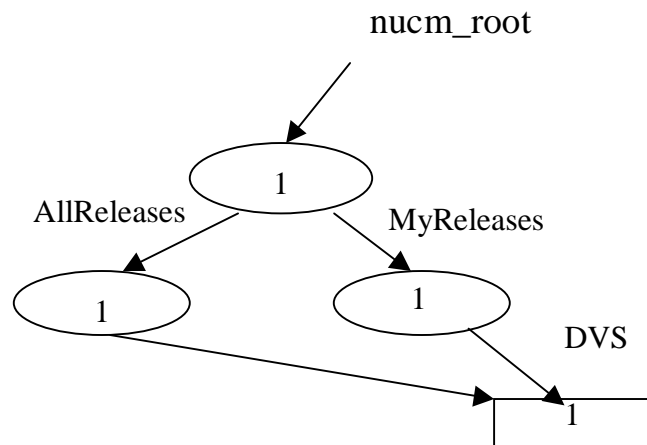


Figure 5.3 Logical Repository Link to a Release Item.

When a child repository is added to a parent repository, it creates the same directory structure as the parent repository, as shown in Figures 5.1 and 5.2. For the physical repository, the directory structure remains the same. However, the creation

of a logical repository for a child repository requires different handling. As the child logical repository is created, each All directory has a link created that point from the individual child. All directory to the corresponding parent repository All directory. Later, as items are added to a child repository, links are created from the parent logical repository to the item in the child physical repository. For example, suppose that a parent repository is created in Boulder, CO and child repository is created in New York, NY. Using release items as an example, Figure 5.4 shows that the parent repository is created as shown in Figure 5.3, then the links are created from the child AllReleases to the parent AllReleases. At some indeterminate time later, a release item is added to the child repository and a link is made from the parent logical repository to the release item. Once the links are complete, both release items are visible from either logical repository, transparently.

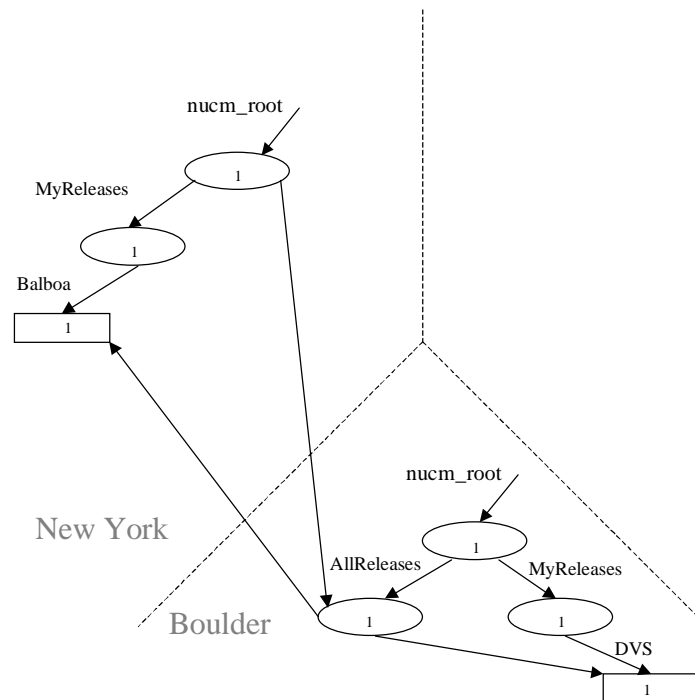


Figure 5.4 Logical Repository Links between Parent and Child Repositories.

5.3 Schema, Attributes and Logs

There are three classes of files that are important to the overall operation of the repository: the schema file, the attributes file and the log files. The following discussion describes their creation and their individual roles.

- Schema.** The schema will be defined at the creation of the logical repository, and dictates the structure of the release item metadata for all member repositories. The schema file, whose naming convention is *schema*, will be located in the root directory of the logical repository. Each parent repository owns the schema file. The schema file is made available to all member

repositories as the single source of reference for repository metadata definition, and all member repositories must point to the parent schema.

- **Attributes.** As shown in the ER Diagram of Figure 4.2, each repository has its own metadata that forms the attributes for the logical repository. Those attributes will be contained in a file, using a naming convention of *attributes*, located in the root directory of the logical repository. The parent repository owns the attributes files. As with the schema, the attributes that are defined for the parent repository are the reference set of attributes for all repository members, and all members of the federation must point to the parent attributes.
- **Log.** There will be only one log file per physical repository, which will be located in the root directory of the physical repository. Since both parent and child repositories own and manage their log files, the log file naming convention is *log.server_name.port_id*, providing a ready method for identification. The log files are maintained and managed by the individual physical repositories, and are made available to all federation members through the AllLogs directory.

5.4 Groups

Groups are worth some additional explanation, since they, alone of the other directories in the repository, can have an arbitrary nesting of subdirectories. As would be expected from Figure 5.4, a group is created as a subdirectory of MyGroups, and then a link is created from AllGroups to the newly created group. Figure 5.5 shows an example of a group named SERL created as a directory in the

MyGroups directory, with the corresponding link from the AllGroups directory to the SERL group.

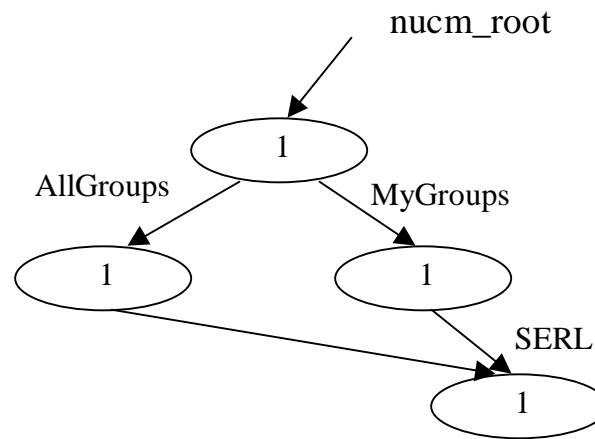


Figure 5.5 Logical Repository Links Showing a Group.

As can be seen, there is no significant difference between the link connections of Figure 5.3 and 5.5. The SERL group would be visible to all consumers from the AllGroups directory.

When a child repository is added to a parent repository, groups can be added to the child repository, as well as the parent repository, in exactly the same manner as is done with release items. A link is created from the child repository AllGroups directory to the parent repository AllGroups directory, giving immediate visibility to all parent repository groups. As groups are added to the MyGroups directory of the child repository, links are created from the parent repository AllGroups directory to the new group.

However, when subgroups are created, no additional links are needed, as the logical repository already sees the parent group in the physical repository. Figure 5.6 shows a sample repository using the same conditions as the example given for Figure 5.4.

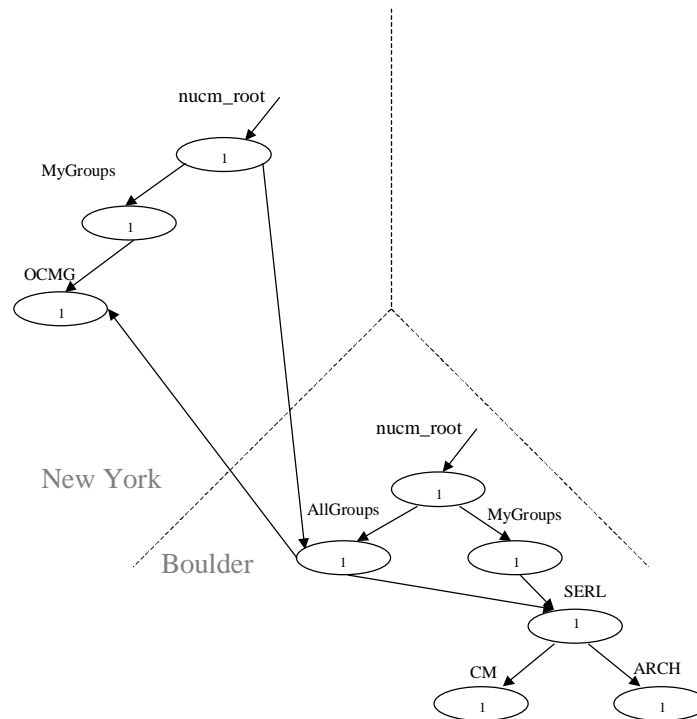


Figure 5.6 Logical Repository Links Showing Groups and SubGroups.

As can be seen in Figure 5.6, each top-level group in the logical repository is made visible to all members through the use of the remote pointers. Once a top-level group is created, any subgroup becomes automatically visible, with no further links required.

Based on this design, removing a server from the federation of repositories becomes a matter of removing links from child repositories to the parent repository and from the parent repository to the child repositories. However, the repository software must ensure that there are no outstanding links to any release items. By checking dependency counters, the repository software can warn the server owner of problems in removing the server. After that, the server owner must make the determination how to proceed to best serve the consumers and the producers dependent on the release item.

In conclusion, a brief inspection of the ER Diagram will show that each of the major top level entities in the ER Diagram relates to a directory or to a file within the repository. Groups contain files with describe themselves and contain the metadata of the software artifacts released to that group. Access control lists are stored together in their own directory, as are licenses and key sets. Release items are stored in directories by name and by version, and maintain dependency counters and registration information specific to that version. Log files, repository attributes and schema files are available from the root of the logical repository.

Chapter 6

Schema, Metadata and Management

At this point, entities and relationships have been defined, as shown in the Entity Relationship Diagram of Figure 4.1. Attributes have also been defined for each of the entities, and from the repository structural definition of Chapter 5, files and storage locations have been identified. The next step in the process is designing a method for managing metadata throughout the repository.

The survey and analysis, as well as the problems with the current version of SRM, helped identify a requirement for flexible definition of key metadata that defines a release item. While some flexibility is still required for presentation text, such as for a group, the need is not as extensive as for release items. This chapter describes the approach to managing both flexible and static metadata throughout the site.

6.1 Metadata Management

The primary metadata requirement revolves around the issue of defining, managing, and maintaining the metadata that describes a release item. The current version of SRM has a fixed metadata scheme, which inhibits the development of dependent systems with differing metadata requirements. Further analysis from the survey shows a wide variability in metadata needs. The need for flexibility makes managing release items more difficult, since a repository must understand the metadata in order to manage the release items. This implies that the schema must be

fixed at the time that the repository is created, and must have sufficient recognizable structure to allow the repository to manage the information surrounding each artifact. Thus, the schema that the repository owner defines at repository creation must be capable of flexible definition, while at the same time having sufficient structure that the repository can understand what is defined.

The Extensible Markup Language (XML) offers a solution to the metadata problem. XML is a structured language that allows the user to define a grammar, which can then be understood by parsers and by applications designed to understand the grammar. The grammar is defined in a Document Type Definition (DTD), which the parser uses to validate an XML document. XML documents can be one of two types: well formed or valid. A well-formed XML document is one in which the XML content is structurally correct according to the rules of XML. A valid document is both well-formed, and conforms to the grammar defined in a DTD [12]. As a result, the XML combined with a DTD provides for the ability to define a grammar to which a repository schema must adhere.

6.1.1 Base Data Types

Prior to defining the schema grammar, a group of basic data types must be defined, which forms the working vocabulary of the system. The repository will use these data types as a basis for interpreting the data stored in the repository and for interpreting input from the user. All of the following data types are defined.

- **STRING.** A string is a collection of character text, as is input via a text box. A string will not be interpreted by the repository, other than to pass the contents on to the requesting function.
- **TEXTFIELD.** A textfield indicates that multiple lines of text make up the content of this field. Repositories will interpret this as multi-line text, as is input by a text area. No other special interpretation is performed.
- **BOOLEAN.** An integral value meaning true or false, as is used in a programming language. The repository can interpret the boolean as a check box. No other interpretation is performed.
- **STRONG_DEPENDENCY.** This indicates to the repository that the attribute is a strong dependency. The repository can use the information contained by the attribute to package a release item with its strong dependencies.
- **OFFSITE_DEPENDENCY.** This indicates to the repository that the attribute is an offsite dependency. The repository can use the information contained by the attribute to bring the user to the dependency, or possibly bundle the dependency with its related release item.
- **EMAIL.** This data type is a character text string that can be input via a text box, and that the repository can interpret the data to be an email address.
- **LICENSE.** This data type is a character text string that can be input via a text box, which the repository interprets to be a pointer to a license retained by the repository in a separate file, and having its own grammar.
- **DATE.** This data type is a character text string that can be input via a text box or drop down control, and that the repository can interpret the data to be a date.

- **URL.** This data type is a character text string that can be input via a text box, and that the repository can interpret the data to be a Uniform Resource Locator (URL).
- **KEYSET.** This data type is a character text string that can be input via a text box, and that the repository can interpret the data to be a pointer to a key set retained by the repository.

6.1.2 Schema Definition

The schema file, which the repository owner creates, defines the structure of the metadata of each release item, and is stored as a structured XML document. The schema file structure is constrained by the Schema grammar definition contained in the schema DTD file. At the time that the repository is created, the schema file is validated against the DTD, and must correspond to the grammar contained within. The schema file then defines how the release item metadata is structured.

Figure 6.1 shows the fragment of the complete repository DTD that contains the Schema DTD. The Schema DTD defines the grammar of the repository schema. The complete repository Document Type Definition is contained in Appendix A. In this chapter, we discuss the repository DTD piecewise, and present only fragments related to the topic at hand.


```

<!ELEMENT Schema (Attribute)+ >
<!ELEMENT Attribute (Label,(Value|OffsiteDependency|StrongDependency)?)>
<!ELEMENT Label (#PCDATA) >
<!ELEMENT Value (#PCDATA) >
<!ATTLIST Attribute Name ID #REQUIRED >
<!ATTLIST Attribute Type(String|TEXTFIELD|BOOLEAN|STRONG_DEPENDENCY|
OFFSITE_DEPENDENCY|EMAIL|LICENSE|DATE|URL|KEYSET) #REQUIRED >
<!ATTLIST Attribute Modifier (key|required|optional) #REQUIRED >

```

Figure 6.1 Repository Schema Definition.

This DTD defines a Schema to contain one or more Attributes, and an Attribute contains a Label and may contain a Value or an OffsiteDependency or a StrongDependency. An Attribute also has three required attributes: a Name, a Type, and a Modifier. The Name attribute tells the repository what the name of the Attribute is, such as ReleaseName or ReleaseVersion. The Type of the Attribute indicates which of the base data types the Attribute conforms to, e.g. STRING or OFFSITE_DEPENDENCY. The Attribute Modifier tells the repository what kind of metadata this Attribute is: key, required, or optional. A key attribute is metadata that, in combination with any other key attributes, uniquely identifies a release item. Required attributes are those attributes that the repository policy requires be completed by the producer when an item is released to the repository. Optional attributes are not required to be complete when the release item is added to the repository. Labels and values are #PCDATA fields, which tells the XML parser to not parse the contents of the defined element. In essence, these fields are strings. For the purposes of defining a repository, only the Label is used. The Label field is used by the repository as the presentation text on a GUI control. For example, if the name of an Attribute is "ReleaseName" and the Label element contains the string "System Release," then the repository will display the name System Release next to the GUI control that the user uses to enter the System Name.

For the purposes of defining the repository schema, no further definition is required. Strong and offsite dependencies apply to the storage of release-item specific metadata.

Figure 6.2 contains a small example of a schema definition file that meets the format requirements of a valid XML document, and meets the grammar defined for the repository.

```
<?xml version="1.0"?>
<!DOCTYPE Schema SYSTEM "./schema.dtd" >
<Schema>
  <Attribute Name="ReleaseName" Type="STRING" Modifier="key">
    <Label>System Name</Label></Attribute>
  <Attribute Name="ReleaseVersion" Type="STRING" Modifier="key">
    <Label>System Version</Label></Attribute>
  <Attribute Name="ReleaseDate" Type="DATE" Modifier="required">
    <Label>Release Date</Label></Attribute>
  <Attribute Name="WhatsNew" Type="TEXTFIELD" Modifier="optional">
    <Label>What's New</Label></Attribute>
  <Attribute Name="StrongDependency" Type="STRONG_DEPENDENCY"
  Modifier="optional">
    <Label>System Dependencies</Label></Attribute>
</Schema>
```

Figure 6.2 Sample Schema Definition.

The first line of Figure 6.2 contains the XML document processing directive indicating that the document is an XML document. The second line contains the document type definition reference indicating to the XML parser that the document type Schema has been defined, and that its definition can be found in the file “schema.dtd” [12].

The remainder of Figure 6.2 contains a definition for a repository schema. Five attributes are defined, showing examples of all modifiers and a strong dependency. In this example, there are two key attributes, ReleaseName and ReleaseVersion, which the repository can then interpret. One required field was specified, ReleaseDate, which the repository will not interpret but will ensure that

data is supplied for it. Two optional fields are specified, WhatsNew and StrongDependency. The WhatsNew field is multi-line text, and is not otherwise inspected. The StrongDependency is defined to be optional metadata, since not every software item has a strong dependency.

6.1.3 Release Item Definition

Each group retains a file in the repository that contains the list of release items managed by that group. The file content structure is governed by the repository schema definition, as defined in the schema file. The release item Document Type Definition shares elements of the Schema DTD. Figure 6.3 shows the fragments of the repository DTD that apply directly to release items.

```

<!ELEMENT ReleaseList (ReleaseItem)* >
<!ELEMENT ReleaseItem (Attribute+, LicenseFile?) >
<!ATTLIST ReleaseItem RID ID #REQUIRED >
<!ELEMENT LicenseFile (#PCDATA) >

<!ELEMENT Attribute (Label,(Value|OffsiteDependency|StrongDependency)?) >
<!ELEMENT Label (#PCDATA) >
<!ELEMENT Value (#PCDATA) >
<!ATTLIST Attribute Name ID #REQUIRED >
<!ATTLIST Attribute Type (STRING|TEXTFIELD|BOOLEAN|STRONG_DEPENDENCY|
OFFSITE_DEPENDENCY|EMAIL|LICENSE|DATE|URL|KEYSET) #REQUIRED >
<!ATTLIST Attribute Modifier (key|required|optional) #REQUIRED >

<!ELEMENT OffsiteDependency (NamedURL)+ >
<!ELEMENT NamedURL (Url,Name,Description) >
<!ELEMENT Url (#PCDATA) >
<!ELEMENT Name (#PCDATA) >
<!ELEMENT Description (#PCDATA) >

<!ELEMENT StrongDependency (Dependency)+ >
<!ELEMENT Dependency (SRMAttr)+ >
<!ELEMENT SRMAttr (Name,Value) >

```

Figure 6.3 Release Item Definitions.

Due to an XML constraint, an XML document can contain only one root element, which corresponds directly to a document type. Since many release items

can be stored in the release item file, the release items are grouped under a release list. This corresponds directly to the notion that Groups manage a list of release items. Therefore, a Release List can contain zero or more Release Items. Each ReleaseItem has a mandatory RID, or Release ID, attribute.

Each ReleaseItem, as is the case with the Schema, consists of at least one Attribute. In addition to the list of Attributes, a ReleaseItem can contain a LicenseFile. A LicenseFile is character text that acts a pointer to a license file, which relates to a ReleaseItem. An Attribute contains a Label, and in the case of a Release Item, contains a Value, an OffSiteDependency, or a StrongDependency. The Attribute element attribution is the same for a Release Item as it is for a Schema, as described in 6.1.2. Further, Release Items, unlike Schema, make use of the Value, OffsiteDependency and StrongDependency elements, as defined in the Release Item DTD.

An OffsiteDependency is essentially a list, consisting of one or more NamedURL's. A NamedURL is a triple containing a Url, a Name, and a Description. All elements of the NamedURL are character text, as indicated by the (#PCDATA). The NamedURL contains the attributes defined for an Offsite Dependency, as shown in the ERD of Chapter 4.

A StrongDependency contains a list of one or more Dependency elements. Each Dependency contains one or more SRMAttr elements, each of which contains a Name-Value pair. Name and Value were defined elsewhere in the DTD as character text. Strong dependencies, as defined in the DTD, require a closer inspection. Each Dependency, as defined in the DTD, corresponds to a release item, as shown in the

"has dependencies" relationship in the ER Diagram of Figure 4.1. From there, the DTD differs from the ERD due to the complexity of providing flexible metadata. In essence, for each strong dependency that release item has, the dependency must provide the key metadata necessary to uniquely identify the strong dependency. The Schema DTD defines a release item to be described by a list of attributes, and since the attribute list can vary, each Dependency must contain a list of key attributes. Each of the key attributes are retained in an SRMAttr, and the Name and Value pair is a pointer to one Attribute. The SRMAttr Name element contains the Name attribute of an Attribute element. The Value element contains the Value contained by the key Attribute.

Figure 6.4 contains an example of a release list containing two release items, one with a strong dependency on the other. The ReleaseItem definition matches the Schema definition as shown in Figure 6.2.

```

<?xml version="1.0"?>
<!DOCTYPE ReleaseList SYSTEM "./schema.dtd" >
<ReleaseList>
  <ReleaseItem RID=" DVS_1.3.1">
    <Attribute Name="ReleaseName" Type="STRING" Modifier="key">
      <Label>System Name</Label>
      <Value>DVS</Value>
    </Attribute>
    <Attribute Name="ReleaseVersion" Type="STRING" Modifier="key">
      <Label>System Version</Label>
      <Value>1.3.1</Value>
    </Attribute>
    <Attribute Name="ReleaseDate" Type="DATE" Modifier="required">
      <Label>Release Date</Label>
      <Value>Sept. 1998</Value>
    </Attribute>
    <Attribute Name="StrongDependency" Type="STRONG_DEPENDENCY"
Modifier="optional">
      <Label>System Dependencies</Label>
      <StrongDependency>
        <Dependency>
          <SRMAttr>
            <Name>ReleaseName</Name>
            <Value>NUCM</Value>
          </SRMAttr>
          <SRMAttr>
            <Name>ReleaseVersion</Name>
            <Value>2.1e</Value>
          </SRMAttr>
        </Dependency>
      </StrongDependency>
    </Attribute>
  </ReleaseItem>
  <ReleaseItem RID="NUCM_2.1e">
    <Attribute Name="ReleaseName" Type="STRING" Modifier="key">
      <Label>System Name</Label>
      <Value>NUCM</Value>
    </Attribute>
    <Attribute Name="ReleaseVersion" Type="STRING" Modifier="key">
      <Label>System Version</Label>
      <Value>2.1e</Value>
    </Attribute>
    <Attribute Name="ReleaseDate" Type="DATE" Modifier="required">
      <Label>Release Date</Label>
      <Value>17-02-1998</Value>
    </Attribute>
  </ReleaseItem>
</ReleaseList>

```

Figure 6.4 Example Release List.

From the example XML file, it can be seen that the ReleaseList contains two ReleaseItems, which have a RID attribute that is the concatenation of key attribute values for the release item. Building on the earlier Schema and ReleaseList discussion, there are two ReleaseItems in the list, each of which has two key attributes, ReleaseName and ReleaseVersion. Each ReleaseItem has a required ReleaseDate attribute, and neither uses the optional WhatsNew attribute. The second ReleaseItem contains the metadata for NUCM version 2.1e, which was released on 17-02-1998. As can be seen from the metadata, NUCM has no off-site dependencies and no strong dependencies.

The first ReleaseItem contains the release metadata for DVS version 1.3.1, which shows a release date of Sept. 1998. DVS does have a strong dependency on NUCM version 2.1e, as shown by the information contained in the StrongDependency element grouping. In reading the Dependency elements, it can be seen that one SRMAAttr points to the ReleaseName key metadata of NUCM, and that the second SRMAAttr points to the second key metadata field, ReleaseVersion. The repository uses the information contained in those fields to search the key attributes for matching values for the given attribute name. In this manner, larger lists can be generated and managed programmatically.

6.1.4 Remaining Metadata

The remaining metadata for the repository provides attribute storage for groups, access control lists, licenses, and key sets. These DTDs are fixed, as there is

no flexibility requirement for those entities. The following sections describe the four remaining metadata types used by the repository.

6.1.4.1 Access Control Lists

Figure 6.5 shows the definitions for an access control list. Following Figure 6.5 is a brief discussion of the DTD and an example XML document containing an access control list.

```
<!ELEMENT AccessControlList (MemberEntry)+ >
<!ELEMENT MemberEntry (MemberEntryName, Password ) >
<!ELEMENT MemberEntryName (#PCDATA) >
<!ELEMENT Password (#PCDATA) >
<!ATTLIST MemberEntryName PermissionSet
(ALL|READONLY|UPDATEONLY|READANDUPDATE|ADMINISTER|EXCLUDED)
#REQUIRED >
```

Figure 6.5 Access Control List DTD.

An AccessControlList contains the attribute set for an Access Control List, as described in Figure 4.3. Each AccessControlList contains at least one MemberEntry, which consists of a MemberEntryName and a Password. Both MemberEntryName and Password are character text strings. A MemberEntryName also has a set of permissions as an attribute. The permissions are as follows:

- **ALL.** All permissions indicate that a user has all other access permissions, except EXCLUDED.
- **READONLY.** The user has read only access to the repository. As such, the user can perform downloads.

- **UPDATEONLY.** The user has upload only access to the repository, and cannot perform a download.
- **READANDUPDATE.** The user can perform both uploads and downloads.
- **ADMINISTER.** The user has permission to manage the repository.
- **EXCLUDED.** The user is specifically excluded from any access to the repository.

Figure 6.6 contains a sample XML file containing an access control list.

```
<?xml version="1.0"?>
<!DOCTYPE AccessControlList SYSTEM "./schema.dtd" >
  <AccessControlList>
    <MemberEntry>
      <MemberEntryName PermissionSet="ALL">Alexander Wolf</MemberEntryName>
      <Password>Milan</Password>
    </MemberEntry>
    <MemberEntry>
      <MemberEntryName>PermissionSet="READONLY">Dennis
Heimbigner</MemberEntryName>
      <Password>Who_Are_You</Password>
    </MemberEntry>
    <MemberEntry>
      <MemberEntryName PermissionSet="EXCLUDED">Bob Smith</MemberEntryName>
      <Password/>
    </MemberEntry>
  </AccessControlList>
```

Figure 6.6 Example Access Control List.

Of the three users shown, one has ALL permissions, one has read only, and one is excluded from the system. For the user that is excluded from the system, no password needs to be provided, but due to the document type definition, the password element must be included. Therefore, the password is shown as an empty XML element (<Password/>).

6.1.4.2 Licenses

Licenses contain a minimal amount of fixed information in each document file. The license DTD contains the information defined in the ERD of Figure 4.3. Figure 6.7 shows the License DTD.

```
<!ELEMENT License (LicenseDomain,LicenseText) >
<!ELEMENT LicenseDomain (#PCDATA) >
<!ELEMENT LicenseText (#PCDATA) >
<!ATTLIST License LicenseType (INTRUSIVE|PASSIVE) #REQUIRED >
```

Figure 6.7 License DTD.

A License contains a LicenseDomain and LicenseText pair. The license domain and license text are both character text. In addition, a License has a LicenseType attribute, which can be either INTRUSIVE or PASSIVE. This attribute tells the repository which process to impose during the download of a system. Figure 6.8 shows an example license file.

```
<?xml version="1.0"?>
<!DOCTYPE License SYSTEM "./schema.dtd" >
<License LicenseType="INTRUSIVE">
  <LicenseDomain>GNU General Public License</LicenseDomain>
  <LicenseText>
    GNU GENERAL PUBLIC LICENSE
    Version 2, June 1991

    Copyright (C) 1989, 1991 Free Software Foundation, Inc.
    .... (example elided for brevity)
    END OF TERMS AND CONDITIONS

  </LicenseText>
</License>
```

Figure 6.8 An Example License XML File.

As shown in the License XML file, the license domain is the GNU General Public License, and the license text contains a fragment of the GNU public license as formatted text. The LicenseType attribute is set to INTRUSIVE, telling the

repository to make presentation of the license an intrusive process during download of the artifact.

6.1.4.3 Key Sets

License key sets are also defined in the DTD file, and follow the attribute requirements described by the ERD in Figure 4.3. Figure 6.9 shows the KeySet DTD.

```
<!ELEMENT KeySet (Key)+ >
<!ELEMENT Key (Name,Value) >
```

Figure 6.9 KeySet DTD.

A KeySet can contain one or more Keys. Each Key is a Name-Value pair, and reuse definitions in earlier DTDs as character text. Figure 6.10 shows an example KeySet file.

```
<?xml version="1.0"?>
<!DOCTYPE KeySet SYSTEM "./schema.dtd" >
<KeySet>
  <Key>
    <Value>12345-98743-871204</Value>
    <Name>Andre van der Hoek</Name>
  </Key>
  <Key>
    <Value>cnowe08u234kmn0uwef</Value>
    <Name>Ken Anderson</Name>
  </Key>
</KeySet>
```

Figure 6.10 KeySet DTD.

As can be seen from the example in Figure 6.10, each KeySet contains a Key, which has a Value that contains the actual key value, and a Name, which associates a user with the Value for any given key.

6.1.4.4 Groups

Groups are the last of the fixed information metadata that will be managed and verified using a DTD. The Group DTD addresses all of the attribute and relationship requirements for the Groups entity, as described in the DTD of Figure 4.1. Figure 6.11 contains the Group document type definition.

```

<!ELEMENT Group (GroupData, ACLFile?, SubGroup* ) >
<!ELEMENT GroupData
(GroupName, GroupOwner, GroupPassword, DateCreated, PresentationText ) >
<!ELEMENT GroupName (#PCDATA) >
<!ELEMENT GroupOwner (#PCDATA) >
<!ELEMENT GroupPassword (#PCDATA) >
<!ELEMENT DateCreated (#PCDATA) >
<!ELEMENT PresentationText (#PCDATA) >
<!ELEMENT ACLFile (#PCDATA) >
<!ELEMENT SubGroup (#PCDATA) >

```

Figure 6.11 Group DTD.

The Group DTD defines a Group to contain a GroupData element, and that it can contain an ACLFile element and a list of SubGroups. A GroupData element contains a GroupName, GroupOwner, GroupPassword, the DateCreated and PresentationText. The GroupData information meets the attribute requirements for the Group entity, as described in the ERD of Figure 4.2. Each of the GroupData sub-elements contains character text. An ACLFile contains character text, and is a pointer to the access control list file that the repository will use to protect the group, as shown in Figure 4.1.

Each SubGroup element contains character text. A SubGroup is a listing of each group that the parent group contains, as shown by the “has members” relationship in Figure 4.1. While XML and the XML parsers will support a recursive all, it does not provide a programmatic method to easily relate a parent with a child.

Therefore, each group maintains a list of subgroups. An example of a Groups XML file is contained in Figure 6.12.

```
<?xml version="1.0"?>

<!DOCTYPE Group SYSTEM "./schema.dtd" >
<Group>
  <GroupData>
    <GroupName>SERL</GroupName>
    <GroupOwner>Alexander Wolf</GroupOwner>
    <GroupPassword>something_good</GroupPassword>
    <DateCreated>1/1/70</DateCreated>
    <PresentationText>
      ...(example elided for brevity)
    </PresentationText>
  </GroupData>
  <ACLFile>acl.xml</ACLFile>
  <SubGroup>SERL/CM</SubGroup>
  <SubGroup>SERL/ARCH</SubGroup>
</Group>
```

Figure 6.12 Sample Group XML File.

This example shows a sample Group definition for the SERL group, and describes the relationship between GroupData, ACLFile and SubGroups. As can be seen from the SubGroup example, each SubGroup is identified with its parent GroupName. This naming convention will continue down the group hierarchy.

Chapter 7

System Prototype

The final step in this thesis is to prototype the next generation software release management system. The goal of the prototype is to demonstrate that the design decisions made address the shortcomings in the current version of SRM, namely flexible metadata and better distribution support. This chapter describes the approach taken, starting with the creation of a parent repository and ending with examples of producer and user screens. Following the prototype examples is a brief discussion on the use of SRM as a document management system.

7.1 Creating a Repository

The first step that a repository owner must perform is the creation of a repository. Since SRM uses NUCM to manage the repository and handle distribution, a NUCM [22] server must be executing for SRM to work correctly. Figure 7.1 shows the prototype application developed to create the parent repository and its directory structure.

```

[smithr@test2 bin]$ java CreateParent

CreateParent - (c) 1999 by SERL
                Department of Computer Science
                University of Colorado - Boulder

Usage:  java CreateParent flags

Required flags:
    -mh parenthost - hostname of parent repository
    -mp parentport - port number of parent repository
    -sf schema - [path/]name of schema file
    -af attrs - [path/]name of attributes file
    -wd workingdir - path/name of working directory
[smithr@test2 bin]$

```

Figure 7.1 CreateParent Application Interface.

The CreateParent prototype has four required flags that must be set when creating the repository, along with one optional flag. The first two flags, “**-mh**” and “**-mp**”, specify the NUCM host name and port number that SRM will use as a parent repository. The third flag, “**-sf**”, specifies the name and location of the XML based schema file that defines the metadata surrounding a release item. The “**-af**” flag specifies the name of the repository attributes file. The last flag, “**-wd**”, is optional and specifies a working directory, which NUCM uses for its operation. Otherwise, the current working directory is used as the working directory. When CreateParent is run, the first item checked is whether the supplied schema file is a valid file, which is performed by validating the schema file against the Schema Document Type Definition (DTD). If the attributes or elements are not correct, the application will return a failure message, indicating the source of failure. CreateParent then checks the NUCM server to ensure that there is no other installation of SRM on that host name and port number combination. If there is, CreateParent also returns a failure message indicating the cause of failure.

Assuming that all preconditions are correct, CreateParent will create the directory structure for both the logical and physical repository, as described in Chapter 5. Figure 7.2 shows the contents of the parent repository after construction by CreateParent.

```
[smithr@test2 smithr]$ nucmclient list //test2:6947/nucm_root $PWD
0
name: MyKeySets
version: 1
name: MyReleases
version: 1
name: MyGroups
version: 1
name: MyAccessControl
version: 1
name: MyLicenses
version: 1
name: AllLicenses
version: 1
name: AllAccessControl
version: 1
name: AllGroups
version: 1
name: AllReleases
version: 1
name: AllKeySets
version: 1
name: AllLogs
version: 1
name: schema
version: 1
name: log.test2.6947
version: 1
name: attributes
version: 1
[smithr@test2 smithr]$
```

Figure 7.2 Parent Repository Directory Structure.

The following are the directories that are created in support of the physical repository: MyKeySets, MyReleases, MyGroups, MyAccessControl, and MyLicenses. Each of these directories corresponds to the directories described in Figure 5.1. The logical repository directories, as depicted in Figure 5.2, are also present: AllLicenses, AllAccessControl, AllGroups, AllReleases, AllKeySets, and

AllLogs. The last three files shown in Figure 7.2 are the schema file, repository log file, and the repository attributes file, respectively. Each of the files has the host name and port number appended to the file name to identify the instances of SRM to which they apply.

On completion of the work performed by CreateParent, the logical and physical repositories have been correctly constructed, and match the directory structure specified in Chapter 5. At this time, the logical repository just created exists with only one physical repository. From this point forward, the SRM repository is ready to add child repositories, to add or remove groups, or to add or release software.

7.2 Adding a Child Repository

A child repository, as described in Chapter 5, is one that depends on the existence of a parent repository. In order for a child repository to be created correctly, two NUCM servers must be running, one which serves the SRM parent repository, and one which serves the SRM child repository. Figure 7.3 shows the command line application that creates the child repository.

```

[smithr@test2 bin]$ java CreateChild

CreateChild - (c) 1999 by SERL
    Department of Computer Science
    University of Colorado - Boulder

Usage:  java CreateChild flags

Required flags:
    -mh parenthost - hostname of parent repository
    -mp parentport - port number of parent repository
    -sh childhost  - hostname of child repository
    -sp childport  - port number of child repository
    -wd workingdir - path/name of working directory
[smithr@test2 bin]$

```

Figure 7.3 CreateChild Application Interface.

The CreateChild application requires five flags for correct operation. As with CreateParent, the “**-mh**” and “**-mp**” flags specify the parent repository host name and port number. The “**-sh**” and “**-sp**” flags are specific to the child repository, and specify the child host name and port number, respectively. As with CreateParent, the “**-wd**” flag specifies the working directory.

It is worth noting here the difference in the flags between the two systems. There are no CreateChild flags to indicate the schema file or the attributes file. Since the repository being created is the child repository, the child repository must use the schema defined for the parent repository to be able to interoperate correctly. The attributes of the parent repository are the defining attributes for the repository federation. Figure 7.4 shows the repository structure and contents for a child repository.

```

[smithr@test2 smithr]$ nucmclient list //test2:3721/nucm_root $PWD
0
name: MyKeySets
version: 1
name: MyReleases
version: 1
name: MyGroups
version: 1
name: MyAccessControl
version: 1
name: MyLicenses
version: 1
name: AllLicenses
version: 1
name: AllAccessControl
version: 1
name: AllGroups
version: 1
name: AllReleases
version: 1
name: AllKeySets
version: 1
name: AllLogs
version: 1
name: schema
version: 1
name: log.test2.3721
version: 1
name: attributes
version: 1
[smithr@test2 smithr]$

```

Figure 7.4 Child Repository Directory Structure.

As can be expected based on the discussion of Chapter 5, the physical and logical repository top-level directory structure is the same for the child as for the parent. The difference to be noted between Figures 7.2 and 7.4 is in the files. The schema file name is the same for both repositories, as is the attributes file. It is important to note here that the Allxxx directory names, as well as the schema and attribute files, are links to the actual physical files or directories that reside on the parent repository. In both examples, the NUCM host name is test2 and the port number is 6947. However, for the child repository, the log file is named log.test2.3721, which indicates that the NUCM server, while on the same host,

operated at a different port number. The difference in file names shows that the log file is specific to the child repository, and indicates that the child will retain its log entries in that file. The functionality shown here ensures that the repository meets the structure as defined in Chapter 5.

The `CreateChild` repository function shows support for more flexible distribution, as can be seen from the examples, in that a child repository can be added at some undetermined time after the creation of the parent repository. At this point in time, the ability to remove a child repository has not been prototyped. However, the activities to remove a child repository from a parent are fairly straight forward, but is contingent on there being no dependencies on any item that exists as a member of the child repository. Should there be software within the child repository that has dependencies on it, then the removal process would have to stop. Upon such a failure, the actions that could be taken to remedy the situation are primarily the policy of the repository owner. One option is that the system at hand is transferred to one of the repositories that remain in the federation. Another option would be to establish the dependency as an off-site dependency, if the child repository owner does continue to make the child repository available in a different venue.

7.3 Adding Groups

The next step that a repository owner must perform is adding a group to the repository. The repository does not have a default group assigned, so the repository owner must create one group to operate the repository, at the very least. Groups can

be added at any time during the repository's lifetime, as each group is created and managed independently.

Figure 7.5 shows the AddGroup application. The operational flags for the AddGroup application are as follows: “-h” specifies the host name of the NUCM server, “-p” specifies the port number of the NUCM server, “-n” specifies the name of the new group to add, “-g” specifies the parent group to add the new group to, “-a” specifies the name of the group attributes file as shown in Figure 6.12, “-r” specifies the release list file for the group as shown in Figure 6.4, and “-w” specifies the name of a working directory that NUCM uses. When specifying that a group is to be added to the root groups directory of the repository, the repository owner must use “-g .” as the group specifier flag.

```
[smithr@test2 bin]$ java AddGroup

AddGroup - (c) 1999 by SERL
          Department of Computer Science
          University of Colorado - Boulder

Usage:  java AddGroup flags

Required flags:
  -h hostname - hostname of repository
  -p port     - port number of repository
  -n newgroup - name of new group
  -g parentgroup - name of parent group
  -a groupattrs - group attributes file
  -r releaselist - release list filename
  -w workingdir - [path/]name of working directory
[smithr@test2 bin]$
```

Figure 7.5 AddGroup Application Interface.

Figure 7.6 shows the commands used to create the set of groups that form the group directory structure to match the subgroups listed in Figure 6.12.

```
java AddGroup -h test2 -p 3721 -n SERL -g . -w $PWD/ws -a
    serlgroupfile -r releaselist
java AddGroup -h test2 -p 3721 -n CM -g SERL -w $PWD/ws -a
    cmgroupfile
    -r releaselist
java AddGroup -h test2 -p 3721 -n ARCH -g SERL -w $PWD/ws -a
    archgroupfile -r releaselist
```

Figure 7.6 Creating Groups.

The first entry creates a group in the root group directory on NUCM server test2 at port 3721, with an attribute file named serlgroupfile and a release list file named releaselist. The second line adds the CM group to SERL at the same NUCM server, using a different group attribute file, but the same release list file. The last line adds a second group to SERL named ARCH, which also uses the same release file but a different attributes file. For the initial addition to the repository, the release list is an empty list having no ReleaseList elements.

Figure 7.7 shows the structure of the parent repository AllGroups directory, as well as the subgroup directories directly related to the SERL group.

```

[smithr@test2 bin]$ nucmclient list //test2:6947/nucm_root/AllGroups
$PWD
0
name: SERL
version: 1
[smithr@test2 bin]$ nucmclient list //test2:6947/nucm_root/
  AllGroups/SERL $PWD
0
name: ARCH
version: 1
name: CM
version: 1
name: releaselist
version: 1
name: groupattributes
version: 1
[smithr@test2 bin]$ nucmclient list //test2:6947/nucm_root/
  AllGroups/SERL/ARCH $PWD
0
name: releaselist
version: 1
name: groupattributes
version: 1
[smithr@test2 bin]$ nucmclient list //test2:6947/nucm_root/
  AllGroups/SERL/CM $PWD
0
name: releaselist
version: 1
name: groupattributes
version: 1
[smithr@test2 bin]$

```

Figure 7.7 Parent Repository AllGroups Directory Contents.

As can be seen from the repository, AllGroups contains a single directory entry, SERL, and contained within the SERL directory is the other two groups, CM and ARCH, along with the attributes file and the release list for that group. Further inspection of the CM and ARCH directories show the attributes file and the release file for each group. These directories are consistent with the directory structure discussed in Chapter 5.

Worth noting here are the actual contents of the MyGroups directory of both SRM repositories. Figure 7.8 shows these contents.

```
[smithr@test2 bin]$ nucmclient list //test2:6947/nucm_root/MyGroups
$PWD
0
[smithr@test2 bin]$ nucmclient list //test2:3721/nucm_root/MyGroups
$PWD
0
name: SERL
version: 1
[smithr@test2 bin]$
```

Figure 7.8 MyGroups Directory Contents.

As can be seen in Figure 7.8, the MyGroups directory of the parent repository is empty, because the groups actually exist on the child repository. Furthermore, had the directory contents of AllGroups been viewed from the child repository, there would appear to be no difference between that listing and the listing shown in Figure 7.7. This shows further support for distribution by SRM.

7.3 Managing Access Control List, Key Sets and Licenses

Figure 7.9 shows the ManageItem application, which manages the addition and removal of access control lists (ACL), key sets, and licenses from the repository.

```
[smithr@test2 bin]$ java ManageItem

ManageItem - (c) 1999 by SERL
    Department of Computer Science
    University of Colorado - Boulder

Usage:  java ManageItem flags

Required flags:
    -a action - add, getrefs, remove or destroy
    -h hostname - hostname of repository
    -p port - port number of repository
    -f filename - file name
    -t itemtype - in [acl|keyset|license]
    -w workingdir - [path/]name of working directory
[smithr@test2 bin]$
```

Figure 7.9 ManageItem Application Interface.

The required flags, as shown in Figure 7.9 are as follows: “**-a**” specifies the type of action to take, “**-h**” specifies the NUCM server host name, “**-p**” specifies the NUCM server port number, “**-f**” specifies the file name to manage, “**-t**” specifies whether the item is an ACL, key set or license, and “**-w**” specifies the NUCM working directory.

The action command requires some additional explanation. The add command adds the item to the repository based on the item type defined with the `-t` flag, and in accordance with the directory structure specified in Figure 5.1. The remove command will remove the file from the repository provided that there are no outstanding references to the specified item. The `getrefs` command allows the repository owner to view the number of references to the specified items.

7.4 Adding Release Items to the Repository

The next important capability for SRM is adding a release item to the repository, particularly given the requirement of managing flexible metadata. To accomplish this, the repository must be able to interpret the schema file provided at repository creation, and then build a graphical user interface (GUI) which will adjust automatically to meet the schema definition. This section describes how SRM manages dynamic construction of an upload GUI and then stores the metadata in the appropriate release item file.

Using the schema definition shown in Figure 6.2 as an example, the repository will open the schema file when an upload is requested. The repository then parses the XML document and extracts from the Schema definition all metadata definitions and

sorts them by key metadata, required metadata, and optional metadata. Then as the upload GUI is constructed, SRM builds each section based on the type definition for the Attribute. Figure 7.10 shows the resulting upload GUI built from the schema definition shown in Figure 6.2.

The screenshot displays the SRM - Software Release Manager interface. At the top, there is a window title bar with the SRM logo and the text "SRM Release Software". Below the title bar, the main heading is "SRM - Software Release Manager". A button labeled "Select Release Group" is positioned below the heading. The interface is divided into three main sections: "Key Attributes", "Required Attributes", and "Optional Attributes".

Key Attributes: This section contains two input fields: "Release Name:" and "Release Version:", each with a corresponding text box.

Required Attributes: This section contains one input field: "Release Date:" with a corresponding text box.

Optional Attributes: This section contains a text area labeled "What's New" with a scroll bar.

At the bottom of the interface, there is a "Release File:" label followed by a text box. Below this, there are two buttons: "Set Dependencies" and "Off-site Dependencies". At the very bottom, there are four buttons: "Dismiss", "Clear", "Help", and "Submit".

Figure 7.10 Sample Upload Graphical User Interface.

As can be seen in Figure 7.10, the GUI has three distinct visual cues for the user, indicating what information is classified as key attributes, which are required, and which are optional. There is a section to name the release file, a button to locate groups to release the software to, buttons for setting dependencies, and several buttons at the bottom line to aid the user during operation. Off-site Dependencies are included on the screen to show the full functionality of the system.

When the software producer is ready to upload a software release, he or she must select a group to release the software to. When the producer selects the button, SRM presents the user with the group selection dialog shown in Figure 7.11.

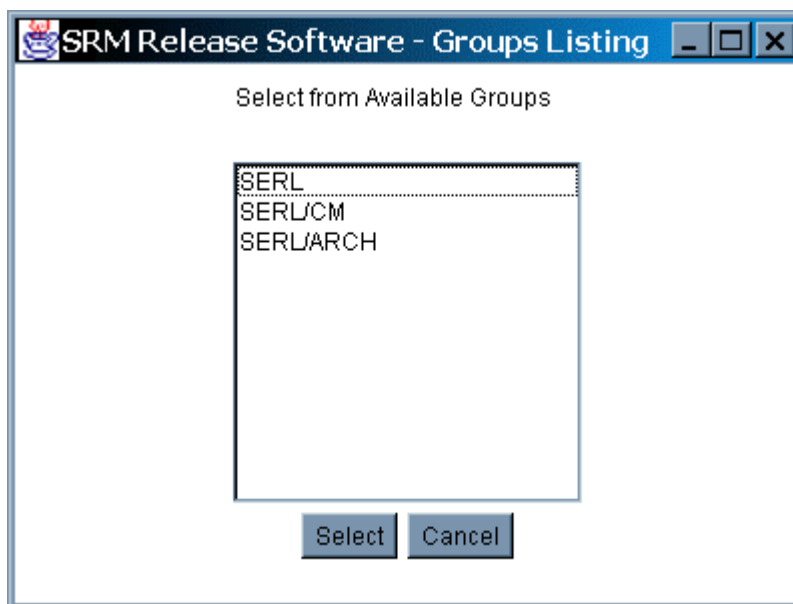


Figure 7.11 Group Selection Dialog.

The producer simply selects the group(s) of choice, and is returned to the main upload screen.

If the software producer wants to include other items on the system as strong dependencies, he or she will select the button titled Set Dependencies. SRM will

bring up another dialog box, as shown in Figure 7.12, which indicates those items within the group that the user has selected that a software producer may include as a strong dependency.

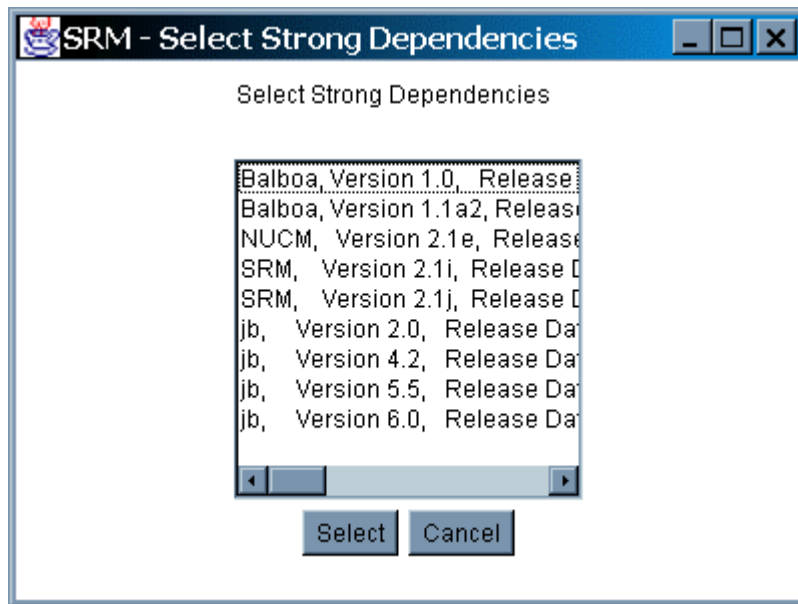
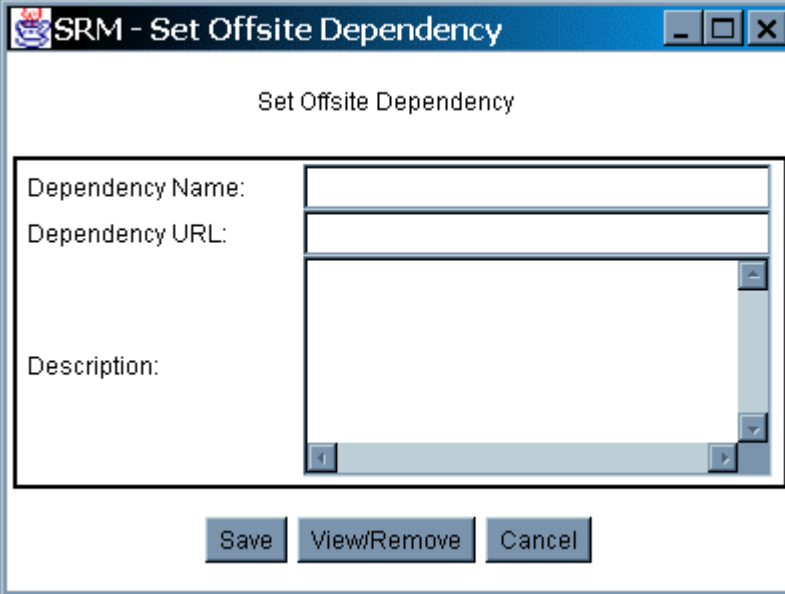


Figure 7.12 Strong Dependency Selection Dialog.

The user selects as many of the listed releases as are necessary for the released software to operate correctly.

In those situations where a software producer wants to include off-site dependencies, selection of the button titled Off-site Dependencies will bring the producer to the Off-Site Dependency dialog, as shown in Figure 7.13.



The image shows a Windows-style dialog box titled "SRM - Set Offsite Dependency". The dialog has a title bar with standard minimize, maximize, and close buttons. The main content area is titled "Set Offsite Dependency" and contains three input fields: "Dependency Name:" (a single-line text box), "Dependency URL:" (a single-line text box), and "Description:" (a multi-line text area with scrollbars). At the bottom of the dialog, there are three buttons: "Save", "View/Remove", and "Cancel".

Figure 7.13 Off-site Dependency Selection Dialog.

The producer must enter all information into the dialog. The information saved from this form meets the schema definition for an off-site dependency, as shown in Figure 6.3.

Using the DVS example shown in Figure 6.4, when the software producer adds the item to the repository, SRM first checks the repository in the MyReleases directory for any prior version of the artifact. If a previous version is not found, then a directory entry is created based on the name of the artifact, and the initial directory version is established. Otherwise, the next directory version of the artifact is created. Figure 7.14 shows the directory structure for an initial check in of DVS.

```

[smithr@test2 bin]$ nucmclient list
//test2:6947/nucm_root/MyReleases $PWD
0
name: DVS
version: 1
name: version_1
version: 1
[smithr@test2 bin]$ nucmclient list //test2:6947/nucm_root/
MyReleases/DVS $PWD
0
name: 1
version: 1
[smithr@test2 bin]$ nucmclient list //test2:6947/nucm_root/
MyReleases/DVS/1 $PWD
0
name: DVS1.3.1-linux-i386
version: 1
name: dependency_counter
version: 1
name: registrations
version: 1
[smithr@test2 bin]$

```

Figure 7.14 Directory Structure after DVS Check In.

As can be seen by the directory structure, this follows the physical repository directory design from Figure 5.1, which also addresses historical versions. At the completion of the activity, the repository will have a new entry in the release group's release list that describes the new release item. Further, the release item will have a new version committed to the local MyReleases, which will be immediately visible in the AllReleases directory. Default dependency counters and registration files are created and initialized to zero and empty respectively, and the release item is available to all member repositories of the federation.

7.5 Retrieving Software from the Repository

Software retrieval has not been prototyped. However, GUI prototypes have been developed that illustrate the options and processes that a consumer has available to retrieve software from a repository. This section first describes the software

retrieval process and then follows with a brief discussion of web-based software retrieval.

7.5.1 Software Retrieval Process

The software retrieval process begins when the user is presented with the repository software retrieval dialog. The GUI prototype, in this case a Java applet, is shown in Figure 7.15.

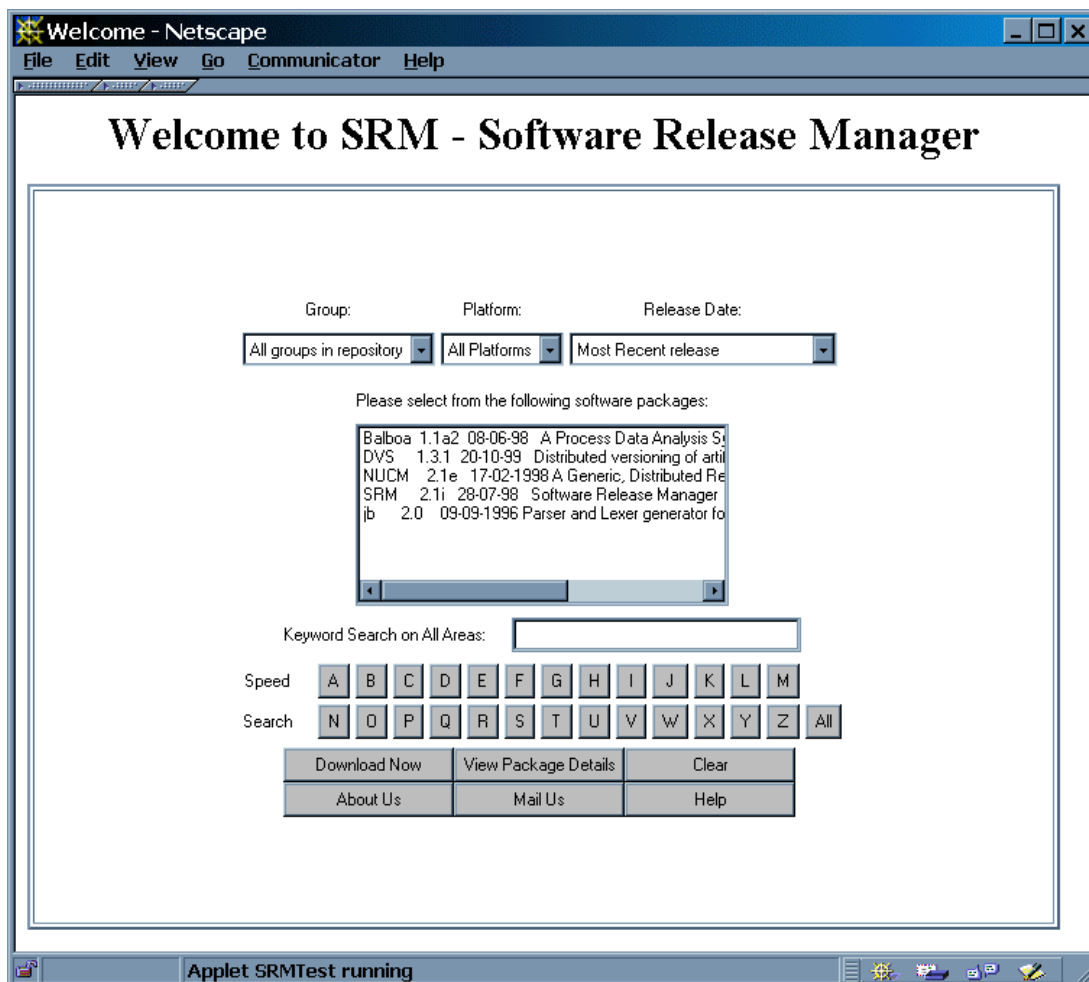


Figure 7.15 Software Retrieval Dialog GUI Prototype.

It should be noted that this GUI prototype was developed for the current version of SRM and would require a repository schema definition that supports the information shown.

The retrieval application can provide the consumer with several capabilities. The consumer can select a group to view its releases, and depending on the structure of the repository schema, view releases by platform and release date. As Figure 7.15 shows, immediately underneath the top-level selection controls is a text box that lists the releases for the selections made from the Group, Platform and Release Date controls. Following the selection box is a text area where the user can engage in a keyword search of release items contained in the site, or make use of a speed search where the buttons represent the first letter of the release item name. At the bottom of the page are several buttons which provide the consumer with the following functionality: Help provides context specific help based on the current dialog or page in the site, Mail Us provides the consumer with the ability to mail the repository owner, About Us presents the presentation text associated with the repository, the Clear button clears all selections made and returns the page to its default state, Download Now takes the consumer's selection from the software list box and delivers it immediately to the consumer, and View Package Details brings the consumer to a detailed information page for the selected software release item.

Typically, a consumer will select a release item and then decide to view the details about a release item, or may choose to simply download the artifact. In the case where the consumer decides to view the package details, the user is brought to a

page that contains the known information supporting the release item. Figure 7.16 shows a sample release information page.

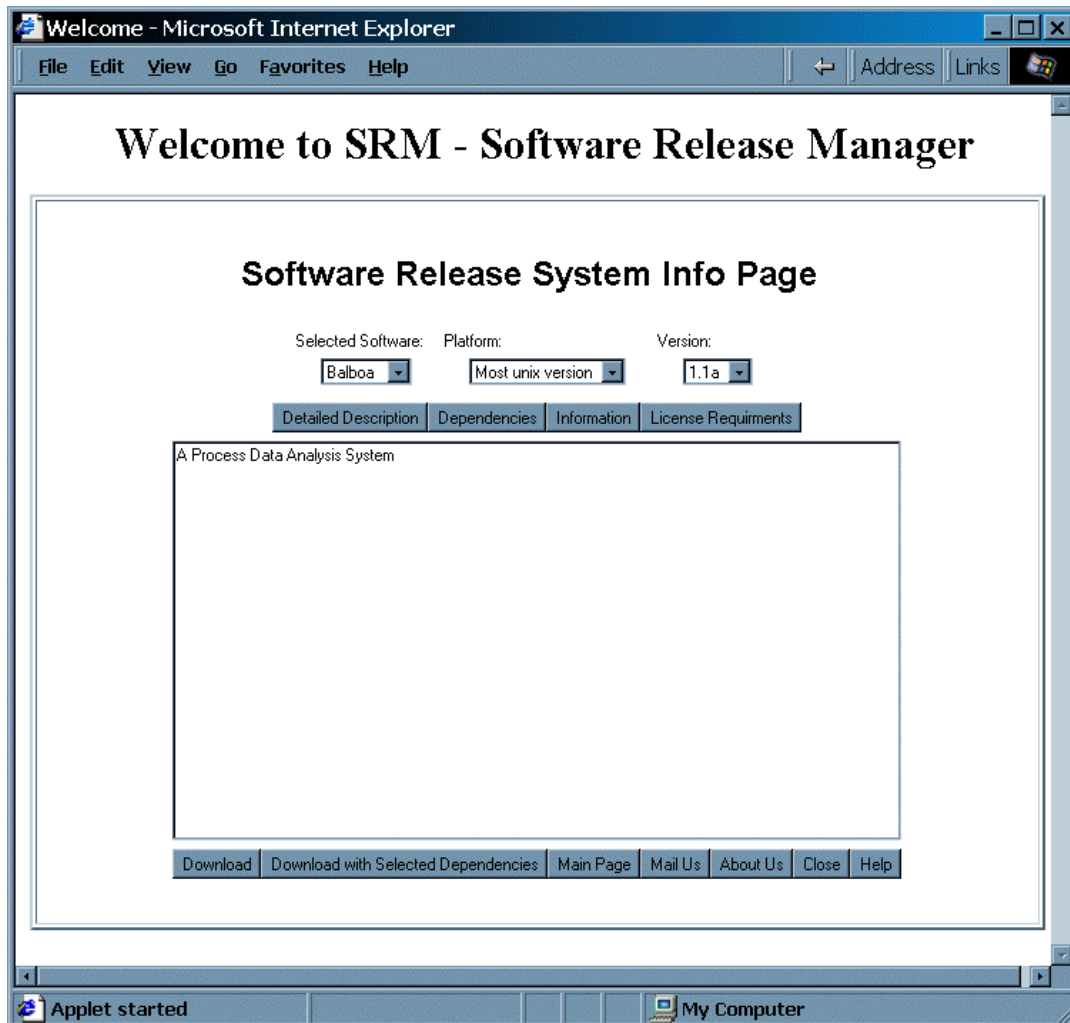


Figure 7.16 Release Item Information Dialog GUI Prototype.

From Figure 7.16, three sets of functionality selections can be made. At the top level is the list of related release, platforms and versions available for the selected release item. Immediately underneath the release item information is a set of buttons that the consumer can use to get more detail about the release item. All requested information is displayed in the text box directly below the top row of buttons. The

Detailed Description button would show the detailed description information. The Dependencies button would list the dependencies for the release item. Information would show any information associated with the release item that is not the detailed description. The License Requirements button would show any license agreement text associated with the release item.

The bottom row of buttons provide very similar functionality as the buttons shown in Figure 7.15. Mail Us, About Us, and Help function in the same was as the buttons described for Figure 7.15. Main Page brings the consumer back to the entry screen for SRM, and Close closes the current screen. The Download button downloads only the current artifact, whereas the Download with Selected Dependencies button downloads the artifact along with the items the consumer selected from the Dependencies screen.

7.5.2 Web-based Software Retrieval Process

From the discussion in section 7.5.1, it can be shown that a repository consumer has the ability to retrieve software release items as is currently performed by SRM, and that there is a fair degree of flexibility in the information that is shown. However, this is a constrained approach to providing release item information in that it requires a certain information base to support the functionality described. A retrieval GUI can be developed which is dynamic in its presentation of release item information, and that supports the retrieval of release items and their dependencies as is done with the current version of SRM. However, a Java applet based approach is not easily promoted to the Web, as browsers support Java differently, and that support

varies between versions of the browser. Therefore, a better way is to take advantage of the basic browser capability and of the information contained within the new version of SRM as XML.

XML is a mark up language for creating structured documents, and provides for the creation of a grammar that allows users to agree on the definition of the information contained in the document. As part of this thesis, a particular type of XML document was created for specifying information about a particular software release. Having this information in a structured document enables the creation of tools that can process the SRM-related XML documents and perform some task. Since one of the requirements for the next generation SRM is the ability to customize the user interface, the information within the document can be used to create a customized web-based presentation that presents the information to the user in a correct manner. The World Wide Web Consortium (W3C) has defined a presentation language based on XML called the Extensible Stylesheet Language (XSL) [6]. With a set of appropriately defined XSL style sheets, presentation pages could be generated that show the information the consumer needs directly from the XML contents of the repository. Therefore, the problem would be to design an appropriate site navigation that would support the consumer while at the same time support correctly the presentation of flexible metadata.

Figure 7.17 shows an example XSL style sheet that was used to generate the license agreement page shown in Figure 7.18. The XSL style sheet shown in Figure 7.17 supports the License DTD shown in Figure 6.7 and uses the repository license XML file shown in Figure 6.8 as its source of content.

```

<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl"
  xmlns="http://www.w3.org/TR/REC-html40" result-ns="">

<xsl:template match="/">
  <xsl:invoke macro="head"/>
  <xsl:apply-templates/>
  <xsl:invoke macro="tail"/>
</xsl:template>

<xsl:template match="License">
  <h2>License Terms</h2>
  <p>
    <b>Licensing Domain:</b>
    <![CDATA[&nbsp;&nbsp;&nbsp;]]><xsl:value-of select="LicenseDomain"/>
  </p>
  <center>
    <form>
      <textarea cols="70" rows="20">
        <xsl:value-of select="LicenseText"/>
      </textarea>
      <![CDATA[<br>
        <input type="submit" name="accept" value="Accept">
        <input type="submit" name="reject" value="Reject">
        ]]>
      </form>
    </center>
  </xsl:template>

<xsl:macro name="head">
<![CDATA[
<HTML>
  <HEAD>
    </HEAD>

  <BODY BGCOLOR="#FFFFFF">
    ]]>
</xsl:macro>

<xsl:macro name="tail">
<![CDATA[
  </BODY>
</HTML>
]]>
</xsl:macro>

</xsl:stylesheet>

```

Figure 7.17 Sample XSL Style Sheet for License DTD.

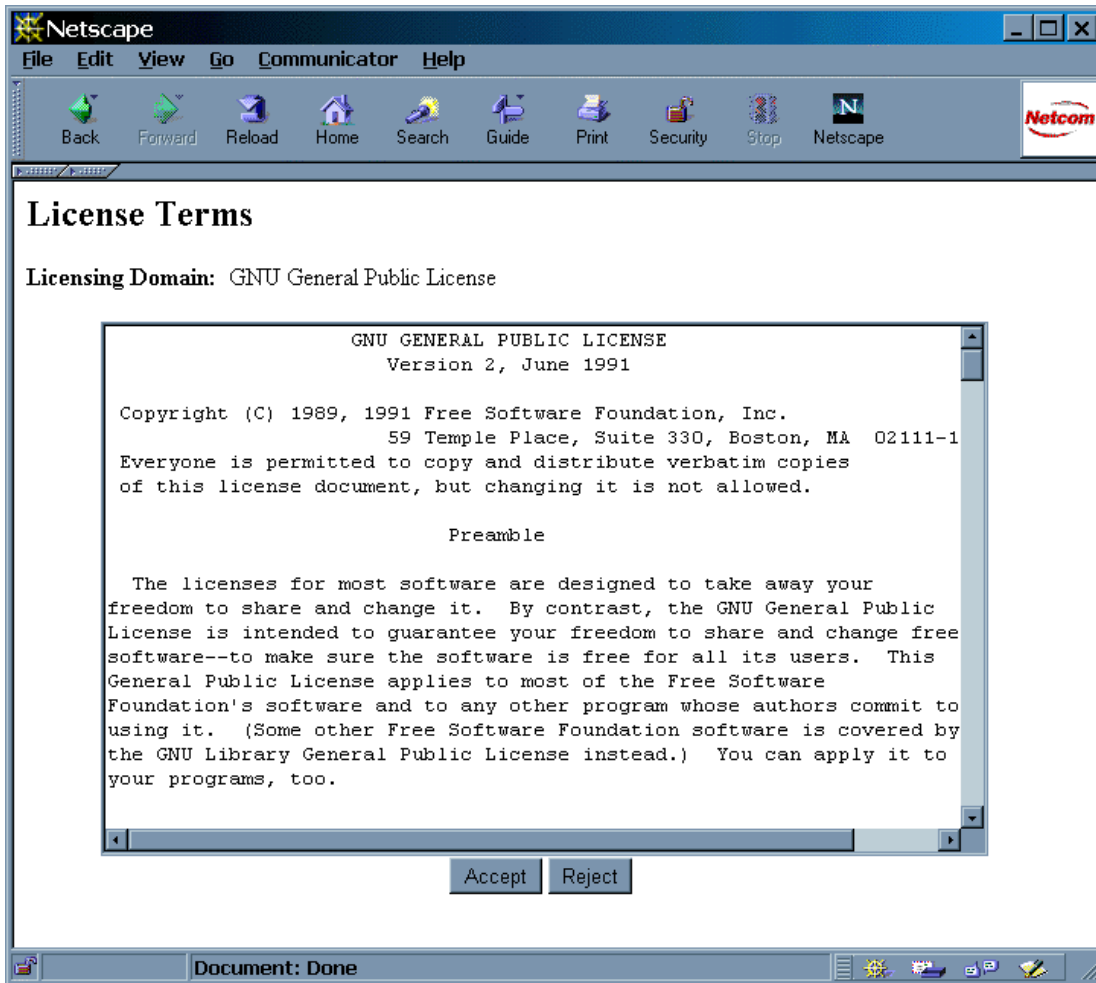


Figure 7.18 Sample License Page Generated by XSL.

As can be seen from the examples contained within this section, XSL support is a logical and powerful extension of the system capabilities for use in software retrieval. Further, XSL can be used to generate not only web-based software retrieval pages, but web-base software upload pages, as well.

7.6 Document Repository

The capabilities defined and prototyped to date can be applied to documents as well as software. Documents are often organized around a topic area, and can be described by title, author, publication date, an abstract, and possibly a document identifier. To use the next generation SRM as a document management system requires no changes to the system, only the definition of an appropriate schema, as shown in Figure 7.19.

```
<?xml version="1.0"?>
<!DOCTYPE Schema SYSTEM "./schema.dtd" >
<Schema>
  <Attribute Name="Author" Type="STRING" Modifier="key">
    <Label>Author Name</Label></Attribute>
  <Attribute Name="Title" Type="STRING" Modifier="key">
    <Label>Document Title</Label></Attribute>
  <Attribute Name="Topic" Type="STRING" Modifier="required">
    <Label>Subject Topic</Label></Attribute>
  <Attribute Name="PublicationDate" Type="DATE"
Modifier="required">
    <Label>Publication Date</Label></Attribute>
  <Attribute Name="DocumentID" Type="STRING" Modifier="required">
    <Label>Document Id</Label></Attribute>
  <Attribute Name="Abstract" Type="TEXTAREA" Modifier="required">
    <Label>Abstract</Label></Attribute>
</Schema>
```

Figure 7.19 Sample Schema Definition for a Document Repository.

As can be seen, the schema definition in Figure 7.19 is very similar to the schema definition of Figure 6.2. Both have key and required metadata, while the document definition example does not contain any optional metadata definition. Based on the schema definition in Figure 7.18, each document is identified by an Author Name and a Document Title, and supplies a Subject Topic, a Publication Date, a Document Id, and an Abstract.

From the schema definition of Figure 7.19, a matching release item file would eventually be generated. Figure 7.20 shows a release list file containing one document.

```
<?xml version="1.0"?>
<!DOCTYPE ReleaseList SYSTEM "../schema.dtd" >
<ReleaseList>
  <ReleaseItem RID="CU-CS-857-98_1.0">
    <Attribute Name="Author" Type="STRING" Modifier="key">
      <Label>Author Name</Label>
      <Value>Antonio Carzaniga, Alfonso Fuggetta, Richard S. Hall,
Dennis Heimbigner, André van der Hoek, and Alexander L. Wolf </Value>
    </Attribute>
    <Attribute Name="Title" Type="STRING" Modifier="key">
      <Label>Document Title</Label>
      <Value>A Characterization Framework for Software Deployment
Technologies </Value>
    </Attribute>
    <Attribute Name="Topic" Type="STRING" Modifier="required">
      <Label>Subject Topic</Label>
      <Value>Configuration Management</Value>
    </Attribute>
    <Attribute Name="PublicationDate" Type="DATE" Modifier="required">
      <Label>Publication Date</Label>
      <Value>April 1998</Value>
    </Attribute>
    <Attribute Name="DocumentID" Type="STRING" Modifier="required">
      <Label>Document Id</Label>
      <Value>CU-CS-857-98</Value>
    </Attribute>
    <Attribute Name="Abstract" Type="TEXTAREA" Modifier="required">
      <Label>Abstract</Label>
      <Value>(elided for brevity)</Value>
    </Attribute>
  </ReleaseItem>
</ReleaseList>
```

Figure 7.20 Sample Document Release List.

From the schema definition of Figure 7.19, the upload graphical user interface shown in Figure 7.10 will change, as depicted in Figure 7.21. Figure 7.21 is built using an XSL Style Sheet and the document repository schema file definition as defined in Figure 7.19. The XSL Style Sheet is shown in Appendix B. The intended functionality is the same for both screens.

The screenshot shows a Netscape browser window displaying the SRM - Software Release Manager interface. The browser's menu bar includes File, Edit, View, Go, Communicator, and Help. The page title is "SRM - Software Release Manager".

At the top of the page is a button labeled "Select Release Group".

The "Key Attributes" section contains two text input fields: "Author Name:" and "Document Title:". Below this is the "Required Attributes" section, which includes three text input fields: "Subject Topic:", "Publication Date:", and "Document Id:". Below these is a large text area for the "Abstract:". At the bottom of the form is a "Document File:" text input field.

At the bottom of the page are several buttons: "Set Dependencies", "Off-site Dependencies", "Dismiss", "Clear", "Help", and "Submit".

The browser's status bar at the bottom shows "Document: Done" and various icons.

Figure 7.21 Document Repository Upload Screen.

As with the example in Figure 7.10, Key Attributes still has two text boxes, the first labeled Author Name and the second labeled Document Title. Required

Attributes is expanded to contain three text boxes and a text area, labeled Subject Topic, Publication Date, Document Id, and Abstract, respectively. All other repository functionality would remain unchanged. Once the information was completed and the software producer commits the data to the repository, then the release item information is store in the release list file for the group, as shown in Figure 7.20. Once the information is committed to the repository, an XSL Style Sheet can be applied to the release list file, generating an HTML file for display on the web. Figure 7.22 shows the resulting display file. The XSL Style Sheet that created this image is available in Appendix C.

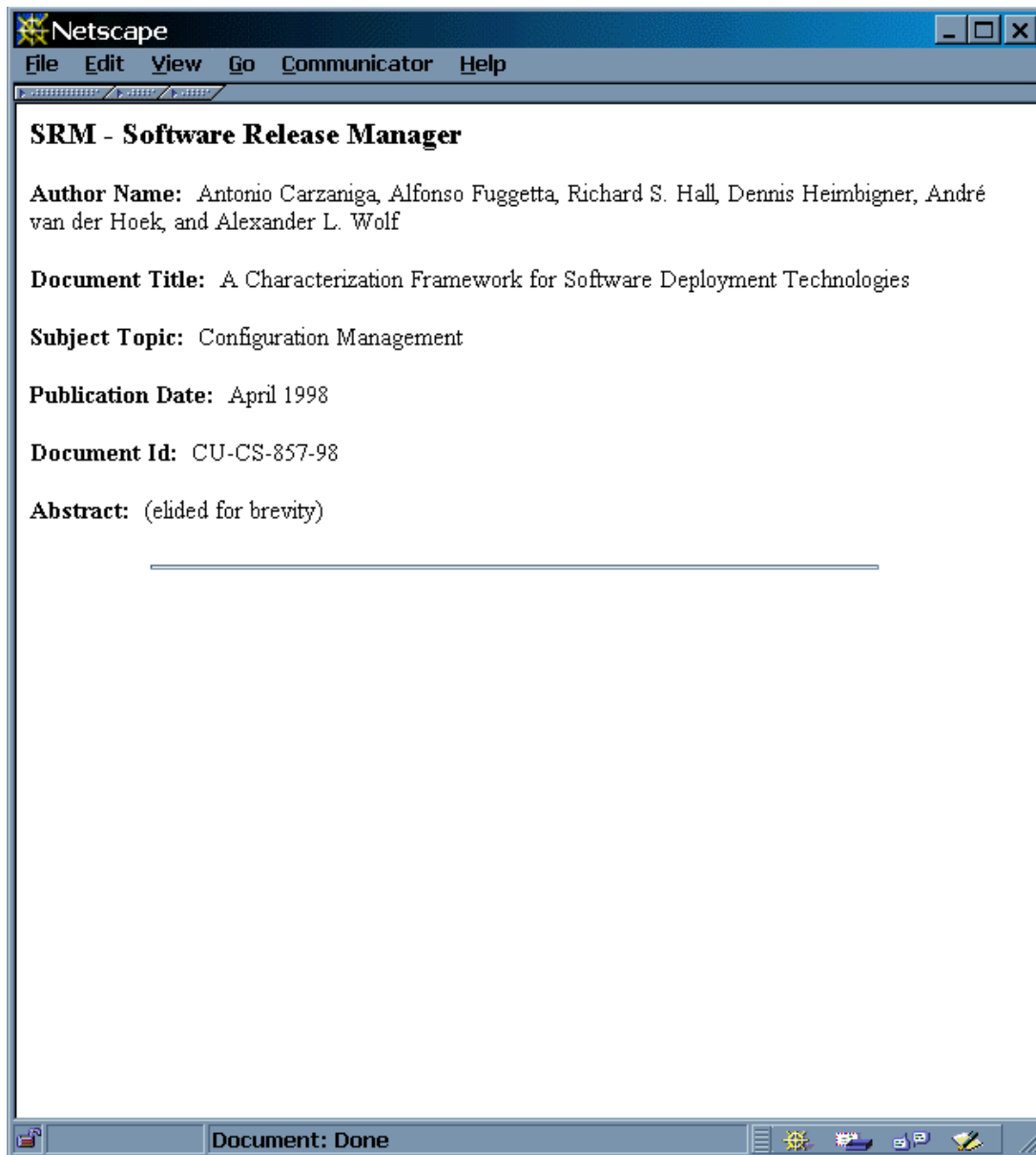


Figure 7.22 Document Repository Availability Screen.

Chapter 8

Conclusion

In this thesis, we have addressed two of the shortcomings of the current version of SRM, namely inflexibility in distribution and inflexibility in appearance and functionality. In particular, this thesis contributes a novel design that leverages the distribution capability of NUCM and the flexible data definition and presentation techniques of XML in solving the two problems. NUCM is used to allow physical repositories to join or leave a federation of repositories at will. XML is used to support the flexible definition of the functionality of the repository, along with its associated presentation style. A prototype was developed to demonstrate these new capabilities. In particular, the flexibility is demonstrated by the modeling of the current version of SRM and the modeling of a document management system within the new SRM.

Future work on the system includes completing the prototype of the system, refining the application graphical user interface for the download capability, and extending the use of XML and XSL to develop web-based access to the system. In addition, other types of release items could be evaluated for deployment by SRM, such as the type of electronic distribution that would be found in such systems as online library or course delivery systems. Finally, SRM can be extended to the field of e-commerce by evaluating its use in conjunction with a web-based purchasing system, or for use as the main functionality base of a web-based application service provider.

Bibliography

1. Alexander Wolf's publication pages. Publications Available on the Web. June 1, 1998. <http://www.cs.colorado.edu/users/alw/doc/AvailablePubs.html>
2. AT&T Laboratories. Download the DjVu Plugin. June 1, 1998. http://djvu.research.att.com/home_mstr.htm
3. AT&T Laboratories Research. Strudel Web-site Management System. June 2, 1998. <http://www.research.att.com/sw/tools/strudel/>
4. Berliner, B. CVS II: Parallelizing Software Development. In Proceedings of 1990 Winter USENIX Conference, Washington, D.C., 1990.
5. DVS. Distributed Versioning System, Carnaziga, A. Nov, 5.1999. <http://www.cs.colorado.edu/serl/cm/dvs.html>
6. Extensible Stylesheet Language, World Wide Web Consortium. Nov. 11, 1999. <http://www.w3.org/Style/XSL>.
7. freshmeat.net. Freshmeat.net. Nov. 15, 1999. <http://www.freshmeat.net>.
8. Gamelan. Gamelan, The Official Directory for Java. June 2, 1998. <http://www.developer.com/directories/pages/dir.java.html>
9. GNU. GNU's Not Unix! Feb. 6, 1998. <http://www.gnu.org/>
10. INTERSOLV, Rockville, Maryland. Using PVCS for Enterprise Distributed Development, 1998.
11. IntraNet Solutions. IntraNet Solutions, Inc. June 1, 1998. <http://www.intranetsol.com>
12. Megginson, D., Structuring XML Documents. 1998
13. NASA. Repository Based Software Engineering. June 2, 1998. <http://rbse.jsc.nasa.gov/eichmann/rbse.html>
14. NetLib. Netlib Repository at UTK and ORNL. June 1, 1998. <http://www.netlib.org/>
15. Red Hat. Red Hat Packaging Manager. Dec. 3, 1999. <http://www.redhat.com/developer>
16. Software Engineering Institute (SEI). Software Engineering Institute. June 2, 1998. <http://www.sei.cmu.edu>
17. Software Release Manager (SRM). June 2, 1998. <http://www.cs.colorado.edu/serl>
18. Tichy, W.F. RCS, A System for Version Control. Software Practice and Experience, 15(7), pp. 637-654, July 1985.
19. TUCOWS. The Ultimate Collection of Winsock Software. June 1, 1998. <http://www.tucows.com>
20. van der Hoek, A., Hall, R.S., Carnaziga, A., Heimbigner, D., and Wolf, A.L. Software Deployment: Extending Configuration Management Support into the Field. 1998
21. van der Hoek, A., Hall, R.S., Heimbigner, D., and Wolf, A.L. Software Release Management. 1997
22. van der Hoek, A., Heimbigner, D., and Wolf, A.L. A Generic, Peer-to-Peer Repository for Distributed Configuration Management. 1996
23. World Wide Web Consortium. Leading the Web to its Full Potential. June 2, 1998. <http://www.w3.org/>

Appendix A.

Document Type Definitions

```
<!-- ***** -->
<!-- SRM Schema definition. This definition allows the -->
<!-- repository owner to flexibly define a repository at -->
<!-- the time of creation. -->
<!-- ***** -->
<!ELEMENT Schema (Attribute)+ >
<!ELEMENT Attribute
(Label,(Value|OffsiteDependency|StrongDependency)?) >
<!ELEMENT Label (#PCDATA) >
<!ELEMENT Value (#PCDATA) >
<!ATTLIST Attribute Name ID #REQUIRED >
<!ATTLIST Attribute Type
(String|TEXTFIELD|BOOLEAN|STRONG_DEPENDENCY|OFFSITE_DEPENDENCY|EMAIL
|LICENSE|DATE|URL|KEYSET) #REQUIRED >
<!ATTLIST Attribute Modifier (key|required|optional) #REQUIRED >

<!-- ***** -->
<!-- A Offsite Dependency is a dependency on an item that -->
<!-- is not found within SRM. Hence it is a pointer to -->
<!-- that item. -->
<!-- ***** -->
<!ELEMENT OffsiteDependency (NamedURL)+ >
<!ELEMENT NamedURL (Url,Name,Description) >
<!ELEMENT Url (#PCDATA) >
<!ELEMENT Name (#PCDATA) >
<!ELEMENT Description (#PCDATA) >

<!-- ***** -->
<!-- Strong dependencies are release items which can be -->
<!-- found within SRM. This definition allows the -->
<!-- application to handle the flexible attribution of -->
<!-- the key attributes defining a release item. -->
<!-- ***** -->
<!ELEMENT StrongDependency (Dependency)+ >
<!ELEMENT Dependency (SRMAttr)+ >
<!ELEMENT SRMAttr (Name,Value) >

<!-- ***** -->
<!-- A Release List is defined to contain Release items, -->
<!-- which are made up of the same attributes which -->
<!-- define the schema. -->
<!-- ***** -->
<!ELEMENT ReleaseList (ReleaseItem)* >
<!ELEMENT ReleaseItem (Attribute+, LicenseFile?) >
<!ELEMENT ReleaseItem (Attribute)+ >
<!ATTLIST ReleaseItem RID ID #REQUIRED >
<!ELEMENT LicenseFile (#PCDATA) >
```

```

<!-- ***** -->
<!-- Access Control List.  Each Group will point to an -->
<!-- ACL file, rather than keep all ACLs together. -->
<!-- ***** -->
<!ELEMENT AccessControlList (MemberEntry)+ >
<!ELEMENT MemberEntry (MemberEntryName, Password ) >
<!ELEMENT MemberEntryName (#PCDATA) >
<!ELEMENT Password (#PCDATA) >
<!ATTLIST MemberEntryName PermissionSet
(ALL|READONLY|UPDATEONLY|READANDUPDATE|ADMINISTER|EXCLUDED)
#REQUIRED >

<!-- ***** -->
<!-- License.  Each license will exist in its own file. -->
<!-- License presentation is either INTRUSIVE or PASSIVE. -->
<!-- License domain can typically be shareware, public -->
<!-- domain, GPL, or user defined, and has text. -->
<!-- ***** -->
<!ELEMENT License (LicenseDomain,LicenseText) >
<!ELEMENT LicenseDomain (#PCDATA) >
<!ELEMENT LicenseText (#PCDATA) >
<!ATTLIST License LicenseType (INTRUSIVE|PASSIVE) #REQUIRED >

<!-- ***** -->
<!-- License Key Set.  A license key set is a set of -->
<!-- permissible keys which will allow a user access to -->
<!-- an item in the repository. -->
<!-- ***** -->
<!ELEMENT KeySet (Key)+ >
<!ELEMENT Key (KeyValue) >
<!ELEMENT KeyValue (#PCDATA) >

<!-- ***** -->
<!-- Groups. -->
<!-- permissible keys which will allow a user access to -->
<!-- an item in the repository. -->
<!-- ***** -->
<!ELEMENT Group (GroupData, ACLFile?,SubGroup* ) >
<!ELEMENT GroupData (GroupName, GroupOwner, GroupPassword,
DateCreated, PresentationText ) >
<!ELEMENT GroupName (#PCDATA) >
<!ELEMENT GroupOwner (#PCDATA) >
<!ELEMENT GroupPassword (#PCDATA) >
<!ELEMENT DateCreated (#PCDATA) >
<!ELEMENT PresentationText (#PCDATA) >
<!ELEMENT ACLFile (#PCDATA) >
<!ELEMENT SubGroup (#PCDATA) >

```

Appendix B.

Document Repository Upload Screen XSL Style Sheet

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl"
  xmlns="http://www.w3.org/TR/REC-html40" result-ns="">

<xsl:template match="/">
  <xsl:invoke macro="head"/>
  <xsl:apply-templates/>
  <xsl:invoke macro="tail"/>
</xsl:template>

<xsl:template match="Schema">
  <TABLE WIDTH="100%" BORDER="1">
    <TR><TD><center>
      <h4>Key Attributes</h4>
      <Table width="100%" cellpadding="0" cellspacing="0">
        <xsl:apply-templates select="Attribute[@Modifier='key']"/>
      </Table> </center></TD></TR>
    </TABLE>
    <TABLE WIDTH="100%" BORDER="1">
      <TR><TD><center>
        <h4>Required Attributes</h4>
        <Table width="100%" cellpadding="0" cellspacing="0">
          <xsl:apply-templates select="Attribute[@Modifier='required']"/>
        </Table> </center></TD></TR> </TABLE>
    </xsl:template>

<xsl:template match="Attribute[@Modifier='key']">
  <TR>
    <td align="right" width="50%"><xsl:value-of select="Label"/>:
  </td>
    <td align="left"><input type="TEXT" size="40"/></td>
  </TR>
</xsl:template>

<xsl:template match="Attribute[@Modifier='required']">
  <TR>
    <td align="right" width="50%"><xsl:value-of select="Label"/>:
  </td>
    <td align="left"><input type="TEXT" size="40"/></td>
  </TR>
</xsl:template>

<xsl:template match="Attribute[@Type='TEXTFIELD']">
  <TR>
    <td align="right" width="50%"><xsl:value-of select="Label"/>:
  </td>
    <td align="left"><TEXTAREA rows="6" cols="40"></TEXTAREA></td>
  </TR>
</xsl:template>
```

```

<xsl:template match="Attribute[@Modifier='optional']">
  <TR>
    <td align="right" width="50%"><xsl:value-of select="Label"/>:
  </td>
    <td align="left"><input type="TEXT" size="40"/></td>
  </TR>
</xsl:template>

<xsl:macro name="head">
<![CDATA[
<HTML>
  <HEAD>
  </HEAD>

  <BODY BGCOLOR="#FFFFFF">
  <center>
  <h3>SRM - Software Release Manager</h3>
    <form>
      <input type="submit" value="Select Release Group">
    </form>

    <form>
  ]]>
</xsl:macro>

<xsl:macro name="tail">
<![CDATA[
  <Table width="100%" border="1">
    <TR><TD>
      <Table width="100%" border="0">
        <TR>
          <td align="right" width="50%">Release File: </td>
          <td align="left"><input type="TEXT" size="40"/></td>
        </TR>
      </Table>
    </TD></TR>
  </Table>
    <input type="submit" value="Set Dependencies">
    <input type="submit" value="Off-site Dependencies"><br>

    <input type="submit" value="Dismiss">
    <input type="submit" value="Clear">
    <input type="submit" value="Help">
    <input type="submit" value="Submit"><br>
  </form>
</center>
</BODY>
</HTML>
]]>
</xsl:macro>

</xsl:stylesheet>

```


Appendix C.

Document Repository Availability Screen XSL Style Sheet

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl"
  xmlns="http://www.w3.org/TR/REC-html40" result-ns="">

<xsl:template match="/">
  <xsl:invoke macro="head"/>
  <xsl:apply-templates/>
  <xsl:invoke macro="tail"/>
</xsl:template>

<xsl:template match="ReleaseItem">
  <xsl:apply-templates/>
  <![CDATA[<hr width="75%" ALIGN="CENTER">]]>
</xsl:template>

<xsl:template match="Attribute[@Modifier='key']">
  <p><b><xsl:value-of select="Label"/>:</b>
  <![CDATA[&nbsp;&nbsp;&nbsp;]]>
  <xsl:value-of select="Value"/></p>
</xsl:template>

<xsl:template match="Attribute[@Modifier='required']">
  <p><b><xsl:value-of select="Label"/>:</b>
  <![CDATA[&nbsp;&nbsp;&nbsp;]]>
  <xsl:value-of select="Value"/></p>
</xsl:template>

<xsl:template match="Attribute[@Type='TEXTFIELD']">
  <p><b><xsl:value-of select="Label"/>:</b>
  <![CDATA[&nbsp;&nbsp;&nbsp;]]>
  <xsl:value-of select="Value"/></p>
</xsl:template>

<xsl:template match="Attribute[@Modifier='optional']">
  <p><b><xsl:value-of select="Label"/>:</b>
  <![CDATA[&nbsp;&nbsp;&nbsp;]]>
  <xsl:value-of select="Value"/></p>
</xsl:template>

<xsl:macro name="head">
  <![CDATA[
<HTML>
  <HEAD>
  </HEAD>

  <BODY BGCOLOR="#FFFFFF">
  <h3>SRM - Software Release Manager</h3>

  ]]>
</xsl:macro>
```

```
<xsl:macro name="tail">
  <![CDATA[
    </BODY>
  </HTML>
  ]]>
</xsl:macro>

</xsl:stylesheet>
```