DRAFT

ScmEngine

A Distributed Program Management Environment on X.500

James X. Ci, Mustafa Poonawala, Wei-Tek Tsai

University of Minnesota,

Minneapolis,

Minnesota.

Akira K. Onoma, Hiroshi Suganuma

Hitachi Software Engineering,

Yokohama,

Japan.

Abstract

This paper discusses an integrated software program management environment, ScmEngine, being built at the University of Minnesota. Industrial software is usually large, has many versions, undergoes frequent changes, and is developed concurrently by multiple programmers. In ScmEngine all information needed for program management is stored using an uniform representation in a distributed repository built on top of X.500, and the various documentation and views of the software artifacts can be generated automatically using the tool repository. The innovative capabilities of this tool are 1) Uniform software artifacts representation 2) Inter-relation and traceability maintenance among software artifacts 3) Tools repository and integration using tool composition scenarios 4) Automatic documentation and versioning control.

1. Introduction

The importance of Software Configuration Management (SCM) is increasing rapidly in modern software development, maintenance and management. It is believed that the market for configuration management software is expected to grow to $716 million by 1996 [Tho95]. With the increasing use of distributed collaborative software development and maintenance, the importance of distributed software configuration management has also been increasing [AFK+95, HHW96]. In order to effectively support software process modeling, software process

has to be as configurable as possible. This paper presents a new approach using X.500 model for distributed intelligent software configuration management and control.

There are literally hundreds of software tools available in market built for configuration and program management [ECMA 95, Feiler 91, Obendorf 88, Ramamoorthy 90]. But, ScmEngine, in addition to supporting these configuration and version control features, has the following unique features:

- ***Uniform Representation:*** ScmEngine stores all the software artifacts using an uniform object-based representation called software objects. This uniform high-level representation of information in a distributed repository provides a common accessible interface to all other components in the software and tool repository and supports addition of new objects and reuse of existing ones.
- ***Framework for Tool Integration***: ScmEngine provides a framework for integrating various software tools. This integration helps in harnessing the power and functionality of the various existing tools to generate new tools with added functionality than its component tools.
- ***Reverse Engineering Capability:*** The software object repository can also be populated from existing software applications using the built-in reverse engineering capability. Given a software application's code ScmEngine can automatically identify and reverse engineer the code into the object-based representation of ScmEngine. This feature makes it easy to go from the actual source of the software to its high-level object-based representation and vice versa.

We reports our experience for a distributed intelligent SCM system, ScmEngine, an on-going project at the computer science department of the University of Minnesota. ScmEngine focuses on distributed version control, process modeling support, optimization, network and integration [AFK+95, Est95], uniform object representation and tool composition. The experiment work done in this area has shown the potential of using X.500 model for SCM processes.

2. The X.500 Model for SCM

In this section we describe the overall architecture of ScmEngine using the X.500 model. Section 2.1 gives a brief introduction to the X.500 technology. Section 2.2 decribes the architecture of ScmEngine

2.1 The X.500 Technology

X.500 technology primarily includes X.500 Directory techniques. X.500 Directory is a

standardized tool with the capability of joining, integrating and organizing information from disparate sources and manipulating that information for use by end-users and applications. The X.500 directory is a collection of open systems which cooperate to hold a logical database of information about a set of objects in the real world. The users of the Directory, including people and computer programs, can read or modify the information, or parts of it, subject to having permission to do so. These capabilities provide a well-defined mechanism to access widely-disparate types of information sharing a single interface, while also offering extensive capabilities for its administration.

This Directory is used by ScmEngine processes to store and manipulate object information for various services such as object management, configuration control, version control, tool composition and reverse engineering. ScmEngine also uses this Directory to implement a distributed database of objects (see section 2.2).

Originally, X.500 was perceived as a generalized telephone or e-mail directory. That service, as now envisioned by the standards, has become a general, global information service. In fact, X.500 provides a cataloging service by which various pieces of information can be arranged systematically across distributed sites, potentially in a global framework [Rad94]. The contents of X.500 Directory describes objects and services separate from the Directory; which means that the X.500 Directory is the information holder of abstractions of objects or services obtained elsewhere.

This feature is used in maintaining abstractions of objects used for tool composition and also to provide a cebtralized view to the distributed database. This means that these abstractions are used to define a schema for the distributed database that is then used by ScmEngine processes and users to perfrom SCM functions (see section 2.2).

In a single X.500 Directory, one could include such information as managed configuration data, product information and catalogs, document abstracts, guides and procedure forms, personnel records, customer lists, corporate directories, and E-mail directories. The purpose of the Directory is to be a catalog rather than a general purpose database. One can compare the Directory to the catalog in a library. A library catalog shows information such as a book's title, its author, and where to find the book.

This concept of a catalog is used to store and retrieve object attributes.

The Directory is organized in a tree structure known as the Directory information tree (DIT) and the entire structure is called the Directory Information Base (DIB). Servers are called Directory System Agents(DSA) and their clients are called Directory User Agents (DUA). Although it is logically a single tree, it can be distributed physically across many machines. The DIT contains

entries that hold the information on the Directory. The data stored in X.500 is organized in a tree structure with named nodes. A wide range of attributes is stored at each node and access is not just by name but also by attribute combinations. The managing Directory software locates and returns entries requested from the DIT in a manner transparent to the end user or application.

The ability to access attributes by name and attribute combinations allows the user and the ScmEngine processes to query the distributed repository for information that it requires to provide various SCm activities. In addition constaints can also be specified on the attributes which help in building security, version management and access control for the ScmEngine artifacts.

Basically X.500 defines the structure, model and addressing syntax of the directory, a set of Directory services to it's users, and the Directory protocols. X500 does not define the user interfaces nor does it define the implementation technology. Although an X.500 Directory may not be a database, it does share many capabilities with a database. Like a database, an X.500 Directory supports a schema by which a user can define the structure and content of information. The information that can be defined by an X.500 schema is unlimited. X.500 also includes the operations needed to maintain that information.

X.500 provides requests to read entries, list children of entries, compare attribute values, search for entries matching a filter, add new entries, modify existing entries, and delete entries. All requests originate from DUAs and are processed by one or more DSAs. For each request, the DSAs will produce a single response: either the result of the request, or an error or failure indication. The DSA to which the request was sent always produces the final response. That response can include an accumulation of responses from other DSAs. The DSAs cooperate to authenticate users and to perform the user requests. If a request targets an entry not held by the DSA to which the user's DUA sent the request, the DSA passes the request toward the DSA holding the entry. This process is called chaining. The DSA can chain requests because the distinguished name of the target provides the context by which the path to the entry can be calculated. A request could pass through several DSAs as the distinguished name of the target is resolved. The response to the request retraces the path followed through the DSAs back to the originating DUA. For a list and search request, a DSA may divide the request into subrequests that are then sent to other DSAs. This process, called multichaining, allows a list or search request access to parts of the DIT that are held by different DSAs. The DSA that subdivided the request then combines the results into a single response.

Establishing an X.500 for SCM process involves a number of tasks. These tasks include designing and obtaining the contents for the X.500, defining administrative policies and manual

or automatic procedures for maintaining X.500 contents, and assigning roles and responsibilities for that maintenance, which corresponds to a SCM process involving the SCM repository and policy of its use.

2.2 ScmEngine Requirements

SCM includes practices for evaluating proposed changes, tracking changes, handling multiple versions, and keeping copies of project artifacts as they existed at various times. The project artifact managed the most often is source code, in ScmEngine we also extend and apply SCM to broad software objects, to name a few, class, modules, components, requirements documentation, plans, designs papers, test cases, bug or problem reports, user documentation, data, and any other multimedia work(video, audio, graph and image etc.) used in software process of building such software system or product. SCM environment should also supports software process modeling. SCM process has becoming more and more critical in large software projects in order to achieve maximum development speed and remain system well maintainable.

The purpose to use X.500 model for SCM process is to utilize the X.500 techniques to better solve current SCM issues and in the same time retain the existing SCM capabilities.

- Requirements management is one of key process for requirement engineering [DT94, WCS94]. Requirements are available from reverse engineering, customer request or contract. In most cases, requirements keep changing and revising. Irrespective of the software artifacts managed, requirements management must be consistent with the corresponding source code control. Various steps with their outcomes in the software life cycle have to be managed consistently in the manner the corresponding software source code is managed and controlled. X.500 model provides a sufficient facilities for managed data access.
- As object-oriented models for design and implementation are becoming popular, SCM tools for managing software entity definitions are investigated [Est95]. In a typical object oriented software development process, configuration management and control is challenged more by the variety of object types and their differing behaviors and dependencies than by the total number. It has to effectively deal with objects, classes, functions, shared procedures, modules, tables, files, packages, application constraints, libraries, schema, scripts files, make files, patches, releases, requirement specification documents, design papers with data-flow/control-flow charts and views, test cases, maintenance documentation, installation and configuration procedures, helps files, and system administration on-line reference manual, and product white papers and so on. As software system becomes bigger and larger, the variety of object, components and unit types and their differing behaviors and dependencies can be very complex. A managed software configuration data access is greatly in need

[Bro96, Est95].

- In software development process, it often requires a creation of multiple application systems with given version numbers so that specific snapshots of the system can be generated and viewed. Software configuration control needs to provide revision management on an object-by-object basis with the ability to lock and unlock specific objects or call up specific versions of an object subset. These are still considered hard to handle due to their large value. Currently Unix system and most Window systems can only handle version control on file-by-file basis while X.500 model's catalog mechanics accommodate variety of software artifacts as well as software process components and elements.
- Developers want to be able to include problem tracking, build management, remote-site linkage, and team development support in a unified distributed SCM environment. It requires version management of software code, problem detection and tracking, and remote site management, build processes and release controls. It has to offer concurrent update protection, security, audit trails, reuse across projects, synchronization with remote sites, and integration with the rest of the development environment.
- Distributed SCM is a relatively new and evolving feature in software configuration management systems. Today's SCM process still suffers performance bottleneck problems at the server repository and scaling problems, especially if the SCM system is augmented with a simple client/server interface (in a client-server setting, the client provides the user interfaces of the configuration tools and the server provides the access and repository of the configuration system). The remote and local configuration, administration and maintenance have to be supported. The distributed SCM system has to be accessible in different and distant sites. The configuration process have to be implemented not only in a client-server architecture but also in a distributed environment, for instance, server to server. X.500 model integrates a multitude of different work elements, so that software process can be able to interact on a distributed environment.
- Distributed SCM currently lacks cohesive strategies that will enable it to efficiently manage the configuration of their client-server and distributed environments. SCM system should be able to be responsible for keeping software artifacts consistent across remote and different sites. Client-server architectures involve a multitude of resources and complex deployment strategies. SCM tool vendors have yet to develop a consistent definition and standard for configuration management. Distributed configuration management can cover a broad range of disciplines, including problem tracking, access control, workstation inventory, remote-site management, software distribution, distributed version control and build management. In

order to handle such a diverse environment in a distributed context, distributed SCM needs end-to-end configuration management tools. Although there are a number of products available for end-to-end management, they are often aimed at different parts of the environment. Whereas X.500 model provides strong access control and management among DSAs.

- As computer networking becomes ever popular and important for people's life, security issues have been critical problems for all kinds of software systems. SCM process authorization and authentication facilities have to be enforced in various software configuration levels. X.500 model has been broadly used for authentication facilities over a distributed context [Rad94].
- SCM tool should provide the ability as an intelligent agent for most configuration items of software artifacts. User should be able to review the possible configuration options and selections with proper interpretations and explanations of the options and selections of items and software artifacts. It includes configuration policy verification and validation of how software artifacts are distributed, managed, and in what SCM policies can be employed. X.500 administrative policies and automatic procedures for maintaining X.500 contents facilitates the configuration policy verification and validation.
- For large software project, software configuration management is also complicated. Software configuration automation, specification and explanation of configuration steps and carrying out the system configuration tasks with minimal user interactions are in great need [Est95]. It should be able to optimize the system configuration to obtain customized performances for specific system or user needs.
- The system software development is viewed as an ongoing process. The SCM process, or change management software, must be integrated into existing processes in application development systems. From this new way of viewing development, it requires the admittance and recognition that target system source code and all software development artifacts are business asset, and it should bend to business rules in order to make parallel development efforts more smoothly and enables better communications between various teams as well as software engineers in a distributed context.

2.3 ScmEngine Architectural Design

Each ScmEngine process running on a machine consists of five primary components which support the features described in the previous section; they are, a) A local software object

repository, b) A tool manager, c) A program management module, and d) A object manager. The figure shows the overall design of the tool and e) the X.500 communication and directory interface.

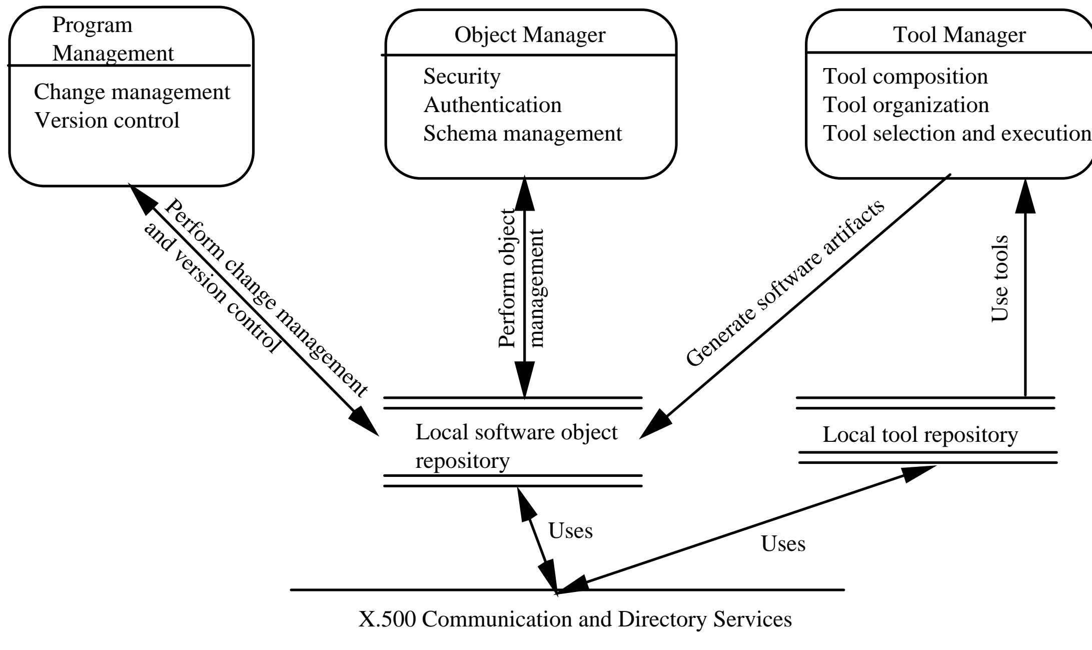


Figure 1: ScmEngine Node

ScmEngine Objects

The ScmEngine Object Repository (OOR) is a data store that can store various software information at different levels of granularity. All this information is treated at the same level and is called *software objects*.

> ***Software Objects (SO)***: All artifacts that are used in the development, maintenance and management of the software, with a predefined high-level specification format and attributes.

Motivation: By treating all software information at the same level in ScmEngine gives a variety of advantages.

1. ***Uniformity****:* To manage the complexity involved in maintaining a variety of software information that one deals with in large projects, a common representation to store all software artifacts is used. This uniform high-level representation of information in a central repository provides a common interface to access all the components in the repository. Each software object has a type and a set of attribute. Objects can be organized and selected based on their type and attribute values.

2. ***Extensibility:*** Treating all software information as software objects give the tool the flexibility to handle new format or kind of software information easily. For example, if the tool can recognize text and PS file format documents, it can be easily extended to handle graphic documents, say in GIF format. This is possible because ScmEngine is built to handle all document objects in the same manner. So, if a new format document is added in a form that complies with the others software document objects then ScmEngine can easily handle it (see Software Object Format). Thus having an uniform high-level representation helps the tool adapt itself easily to new/changing software information.

The repository contains SOs like software program components and constructs like requirements specification, design documents, source file, makefiles, modules, procedures, variables. Thus SOs can be high-level program entities such as modules and procedures, but also low-level information such as statements, basic blocks and even individual variables within a basic block. For example, modules could be "main.c", procedures could be "main()", and high-level representations of a program such as callgraph documents, and decomposition diagrams are also SOs for ScmEngine.

Software Object Format

All Software Objects are ***typed*** and hold pre-defined ***attributes***. The attributes for a given SO depend on the types of the SO. Further the SOs, can be related to one another through various ***relationships*** that ScmEngine recognizes. Thus, the SOs have three characteristic a) type, b) attributes and c) relationships.

- **Attributes:** SOs have a set of pre-defined attributes associated with them. So a SO having a type T, will have values assigned to the corresponding attribute of type T. The attributes belong to the categories: *string, file, time, a*nd *sequence*. These types are used to classify and select SOs belonging to a particular type or having certain attributes. For example, for version control, the creation and versioning attributes (which are attributes of type *time),* of the various C-Program SOs are used to select to generate a specific version of an application that was developed using ScmEngine. Explanations and examples of these attributes and attribute types are shown in Appendix C.
- **Relationships:** An entity-relationship model used to represent relationships between software objects. This model is similar to the Entity-Relationship attribute model [Chen 83].

  The reasons for storing the relationships among software objects are to:

  1. Generate versions of the software (See Version Control).
  2. Generate makefiles, source code. (See Configuration Management).

3. Generate documentation.
4. Maintain traceability and consistency among the various software objects.

Relationships are expressed as a three tuple. *R(So, Sd, N)* where, *R* is the relationship definition tuple, *So* is the source object type, *Sd* is the destination object type, and *N* is the name of the relation. This means that any instance of type *So* could be associated with the instances of the type *Sd* via the relationship named *N*. The relationship is uni-directional, which means *R(Sa, Sb, N)* has a different interpretation than *R(Sb, Sa, N).*

For example, the call relationship between two C functions is defined as *R(C-function, C-function, refer).* If function *A* calls function *B*, the relationship will be *R(A, B, refer),* and if *B* calls *A* the relationship will be *R(B, A, refer).* Unlike conventional ERA model, our model does not specify the types of associations such as one-to-one or one-to-many. Instead, the relationship implies many-to-many association. Some examples of these relationships for C programs are shown in the Appendix B.

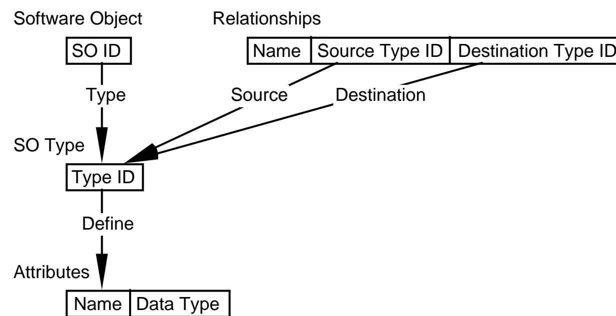
This relationship model is being extended to have association type checking, which include checks for mutual exclusion, mutual inclusion, and subset associations.

The SO, types, attributes and relationships can also be extended by the programmer. The tool allows the user to define and store the relationships needed for software development and maintenance in the repository as these definitions may differ across development organizations, languages used, lifecycles to be supported, and documentation needed to be produced. So database interface is flexible enough to allow definitions of new SOs.

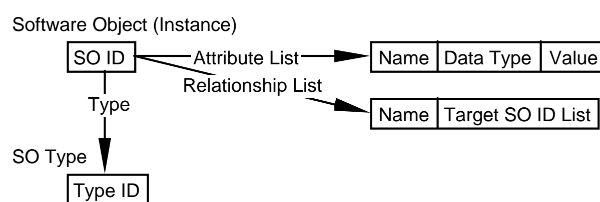
Components of OOR

OOR helps in organizing all software information, in the form of SO, which can then be used by the tools and components of ScmEngine for a variety of applications (as discussed in section 3.3). The OOR consists of two parts: 1) the Object database, and 2) the Access module.

Object Database: SOs created and/or used by the programmers are stored in the *Object database*. All these objects are typed and hold pre-defined attributes. The figure shows the relational schemas that are used in the database to store the various software objects, their attributes and relationships.



(a) Four basic schemes



(b) Instance of software object

Figure 2: (a) Four basic schemes used in the database to store the various artifacts and their relationships, and (b) representation of instantiated software object.

Database Access Module: Objects can be retrieved using their identifiers. The access module contains more than sixty library functions to access the repository. These include retrieving and storing the values of attributes, tracing the relationships, creating and deleting software objects. For example, to obtain the name of the given C function, the function *GetAtr(SO-ID, "name")* can be used; to set the name, the function *SetAtr(SO-ID, "name", "main")* can be used*;* and to open the *file* of the source code of a function, *OpenFile(SO-ID, "src", "r")* can be used.

The ScmEngine Tool Manager

ScmEngine has a tool manager, ScmEngine tool manager, to organize various software development and maintenance tools that can be accessed by software engineers.

Motivation: The advantages of having an integrated and flexible tool manager are:

1. ***Tool Integration*** Conventional software engineering CASE tools have mainly focused on developing single software representation or solution to a given software engineering problem. However, real-world software construction involves so many different complex problems that one solution can not solve all the problems [Tsai 95, Onoma 95]. Various software representations and tools proposed to address individual parts of a problem have to be integrated to solve the more difficult problems.

For example, many tools that reverse engineer information from the code have been in use for program understanding. These tools take software code written in COBOL as input, and produce various representations of the code, like, Call Graphs, Control Flow Graphs, Decomposition diagrams, Problem Analysis Diagram, Chapin chart, and HIPO charts. The reverse engineering tools that generate these representations are useful for program understanding, but they are not as useful as data cross reference tools. Most of the time the engineers try to understand software by understanding the role of data items in a data division or by identifying which paragraphs modify a particular table of a COBOL application. To generate this information we need a tool that is centered around the data flow rather than control flow of the program. It would benefit the software maintainer if the desired functionality can be achieved integrating the existing tools rather than looking for a new tool (see Tool Composition Scenarios).

2. ***Plug and Play***: ScmEngine treats all the tools in its software tool repository as tool components. So when a new tool is added in the tool component format, it is immediately available to the other tools in the repository to interact with.

The tools are the components of the ScmEngine's tool repository, and are called *Tool Components* (TC). The TCs are stored in the form of *Tool Component Abstractions* (TCA) which give an high-level interface description of the tool. The tool manager uses these TCAs to retrieve and invoke the various functionality of the tools.

In addition to storing and organizing the tools, the tool manager helps *compose* and *combine* the various software tools to generate different *scenarios* of tool usage to generate new artifacts needed for software development and maintenance. These scenarios are called *Tool Composition Scenarios* (TCS). This integration helps in harnessing the power and functionality of the various existing tools to generate a new tools with added functionality than its component tools.

Thus the tool manager has two functions a) Tool Selection and Execution (using TCAs) and b) Inter-communication (Using TCS). ScmEngine tool manager has two major components, namely, Action Control and Inter-Communication components to handle these two functions respectively.

The figure shows the architecture of the ScmEngine tool manager. The tools are stored in the form of TCAs in the repository which is accessed by ScmEngine tool manager. The Action Control component of ScmEngine tool manager uses the action commands of the tool to execute the tool. The Inter-Communication component parses the TCS and sends the appropriate commands to the Action Control to invoke the tools in the correct sequence as given in the scenarios.

DRAFT

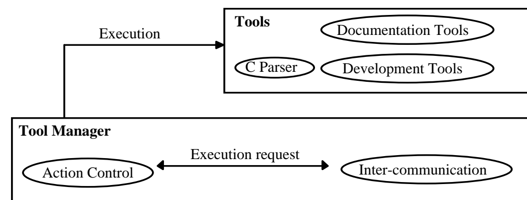


Figure 3: **ScmEngine Tool Manager**. The Inter-Communication component sends an execution request based on the TCS, to the Action Control, which invokes the appropriate tools.

ScmEngine Tool Repository: This repository is used to store the tools that are used by the tool manager. New tools can be added to the tool repository by the user. The tool repository consists of two parts: 1) the tool database, and 2) the access module.

The Tool Database: All the tools defined by users are stored in the *Tool database*. All the tools stored in the repository are specified as TCAs.

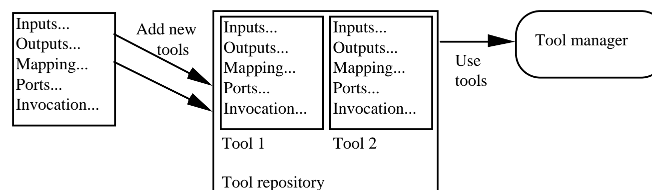


Figure 4: The Tool repository

The Access Module: The access module contains library functions to access the tool repository. These include retrieving and modifying the input, output or mapping definitions for the tools, adding new tools and deleting tools from the database. For example, to obtain the input definition of a call-graph generator, the function *GetIdef(TID, storage)* can be used; to set the output definition, the function *SetOdef(TID, OutputDef)* can be used.

ScmEngine Program Management

The program management module provides support for a) Project management and b) Version and configuration control.

Project Management

Motivation: In large organizations, several programmers are working on a single project. Due to this concurrent development, project files and artifacts need a controlled access to maintain coherency. Two important features of group development are:

a) *Exclusive access to the software objects* - only authorized programmers can access and make changes to the software objects assigned to them,

b) *Authorization of objects* - before any objects are made available to other programmers

authorization must be obtained from the project manager.

Implementation: ScmEngine implements change management by maintaining different relationships between objects. The design of change management in ScmEngine is described below:

a) ***Exclusive access****:* Every software object stored in the repository is connected to the programmer who is in charge of the object. For every project, information about these programmers is stored in the repository as a software object of type "User". When a software object is created, its *user* attribute is assigned the *Id* of the developer responsible for the object. This is stored as the *in-charge-of* relationship:

   A ***in-charge-of*** *B - which relates developers to the software artifacts they are responsible for.*

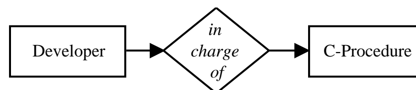


Figure 5: Example of the *in-charge-of* relationship

   Only assigned programmers can modify the software object. Other users can not make any changes to the object unless the ownership is transferred to that user.

b) ***Authorization****:* In group development, the object owner usually does not want the object to be referred to by other users until it becomes fairly stable. Once it becomes stable the owner will copy it to the public environment. In the ScmEngine repository, all software objects have two copies, a) the *volatile* (dynamic) copy and b) the *static* (controlled) copy (precise definition of the words "dynamic" and "controlled" can be found in [IEEE 87]). The copy of the object used by the programmer to make changes is referred to as the volatile copy and is stored in the *volatile* workspace. The copy that can be seen by the other users is called the static copy and is stored in the *static* workspace. ScmEngine controls authorization by means of the *make-changes* relationship.

   A ***make-changes*** *B - which relates developers to all artifacts that they can make changes to.*

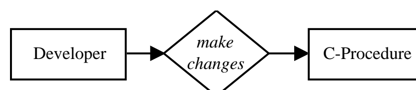


Figure 6: Example of the *make-changes* relationship

   The developer who is related to software artifacts by a *make-change* relationship can move artifacts between the static and volatile libraries. All others can access the copies in both libraries but only for reading.

DRAFT

Version and Configuration Control

Motivation: In industries, software undergoes frequent changes due to a variety of reasons [Onoma 95]. Versioning refers to creating a snapshot of the system at varying granularity so that it can be retrieved at a later time. Versions are created for a variety of reasons such as a) release deadlines, b) movement between the volatile and static library, c) changes in the specification or design due to changes in customer requirements, d) changes in the hardware or software architecture. The following example describes a typical maintenance scenario that results in versioning.

ScmEngine provides version management for different system components at a uniform level, and automatic construction of system configurations for different versions from existing program source.

Implementation: In ScmEngine, versioning is accomplished using the *have* relationship:

> *A **have** B* - this relates a project to all the artifacts that it contains. This relationship is also maintained between any object which contains a number of different items. For example, between a C-program and C-module, C-module and C-procedure.

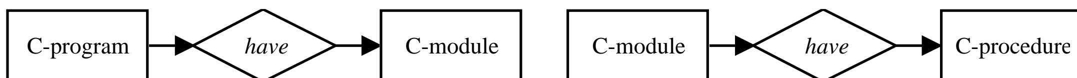


Figure 7: Example of the *have* relationship

Using this relationship, ScmEngine can obtain all the artifacts that need to be backed up for versioning. These artifacts will then be copied to the appropriate destination whenever versioning is performed. The uniform object representation provides a uniform version management implementation.

The configuration state of the system can be easily generated for different versions by means of the *have* relationship between software objects. Figure 6 shows an example of the generation of the makefile and the source for a particular software version.

DRAFT

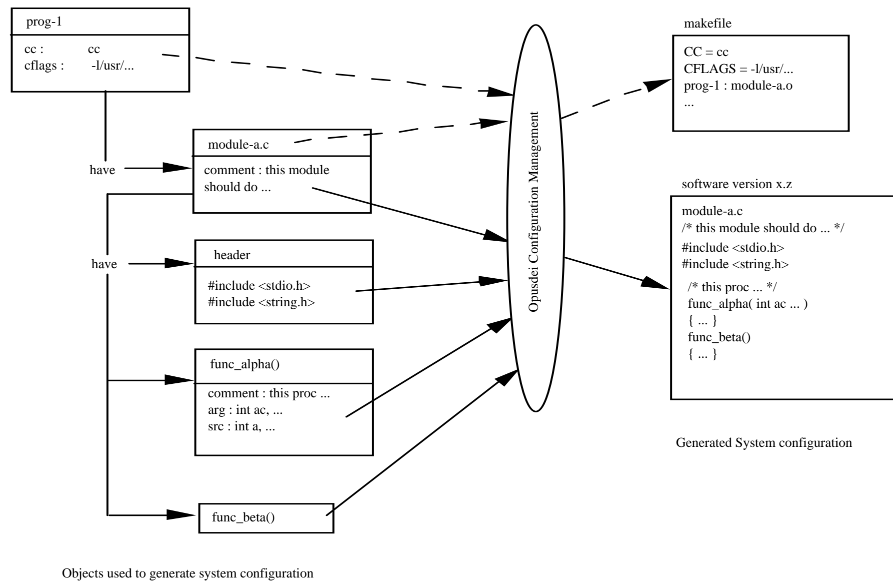


Figure 8: **Automatic generation of system configuration**. The dashed lines indicate the generation of the makefile from the software objects and the full lines indicate the generation of the program source.

ScmEngine X.500 Communication Interface and Directory Services

The figure shows how different ScmEngine nodes communicate and use the X.500 communication and directory services. The X.500 interface is used to provide the services listed in section 2.1 and 2.2 such as, a) to maintain a distributed database of ScmObjects, b) to provide a uniform schema that is visible to the individual ScmEngine processes, c) to provide services to maintain the schema, d) to provide access functions to access the objects in the distributed database. We describe in brief how different features of X.500 help in ScmEngine implementation and coordination. The details this of implementation are beyond the scope of this paper.

DRAFT

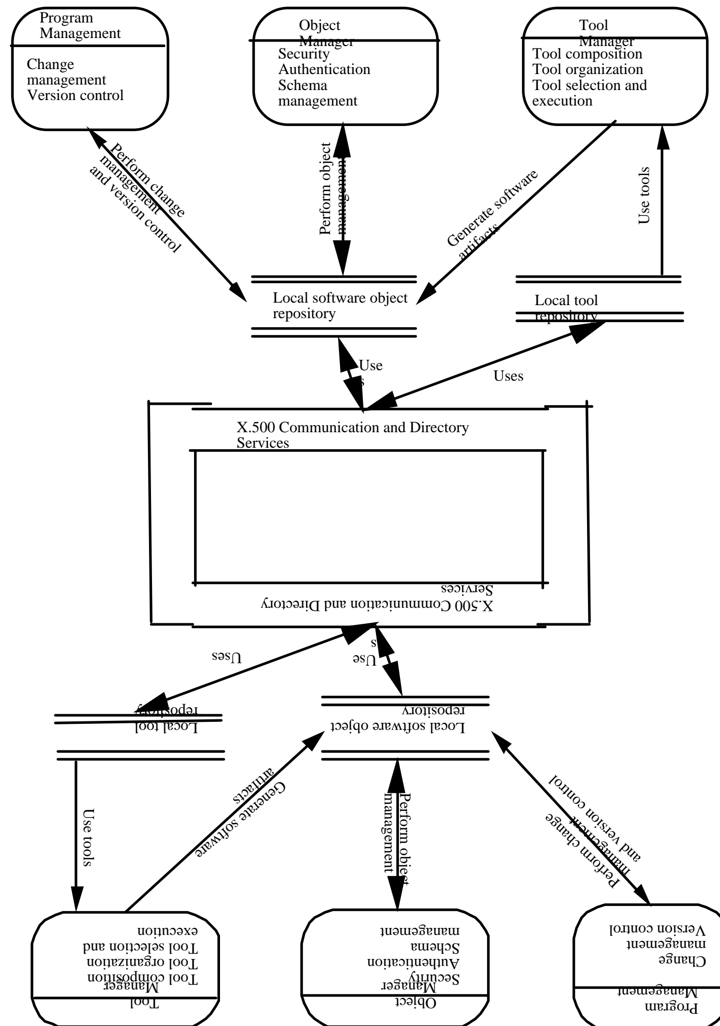


Figure 9: Inter Node Communication

WWW forms which search X.500 for software basic elements, components along with its glossaries of technical terms, encyclopedia of applications, software system online helps and reference manual, requirement specification documents, design papers and other types of information, leverages the current usage about WWW (and at the same time validate the role of the Web as a standard user interfaces and as an information resource). Ties can be established with both the managed configuration data consumers and the data producer (who are looking for better ways to disseminate information, for instance, the better way to communicate with other software engineers in a group and/or in a distributed enterprise-wide setting, with regarding of software source code, basic software components and requirement specification documents, design papers and data-flow/control flow graphs, all in a unified environment, the X.500 backbone). The underlying data store for ScmEngine can be SQL-based, and queries can be built graphically as well as textually.

The naming mechanisms of X.500 model is able to label each software development artifacts version during software development process and by using these labels versions of different designs, implementation and elements of software processes can be managed and coordinated automatically.

ScmEngine defines the infrastructure (DSAs, multi-platform DUAs, distributed software development process support, etc.) required to its distributed heterogeneous user platforms. By using Web forms as the common standard user interfaces, ScmEngine provides a means to access the managed configuration data and support software development process modeling in a unified environment for its distributed heterogeneous platforms. ScmEngine remedies several common problems, such as lack of communication and coordination between developers, and can provide more visibility of the current state of development at any given time during the process. It acts as a groupware enabler, and provides for automatic notifications of status changes. Particularly it helps coordinate efforts between multiple software engineers, serving as a conduit of vital information between those involved in a development project.

ScmEngine also serves as a type of distributed DBMS to collect, maintain, and disseminate software configuration management data in a corporate security architecture. It uses a custom DUA to authenticate a system-initiated transaction updating an entity's security profile in the X.500 directory. When processing a query to a computing resource, a master security server authenticates the query and requests the X.500 directory for the query’s individual security profile. Permission links to connect to all computing resources appropriate for that security profile, or denying are then issued to that particular request. Due to X.500 distributed nature, it does not require immediate propagation of updates amongst DSAs (hours latency is acceptable), and the ratio of updates to accesses is expected to be relatively small.

2.4 ScmEngine Implementation and Experiments

We are currently exploring an ScmEngine X.500 framework with prototype implementation that captures the key concepts and the solutions where proof-of-concept activities can be conducted. The experiments work has shown the big potential of using X.500 model for SCM process. The /c=us/o=umn-cs/ou=ScmEngine branch have sub-branches which contain descriptions of software system organizations, document libraries such as requirement specification documents, data-flow/control flow design papers, software source code files, objects, modules and components for each building blocks and units. As managed software artifacts are added as branches subordinate to /c=us/o=umn-cs/ou=ScmEngine/ou=proj1, /c=us/o=umn-cs/ou=ScmEngine/ou=proj2, … project5 and so on , they are responsible for their own schema

management such as ~/ou=project1/ou=javaApplet, ~/ou=project1/ou=com. Even with up to 800+ entries, the branch can be completely held in memory on a single DSA and provide good performance.

At various times, the prototype included from one to six participating DSAs running system routines from up to four different software development divisions in a virtual geographically distributed sites. The DSAs carried information about 105 software configuration elements, such as Java applets, objects, functions, modules and files. DUAs are distributed to an estimated 15 users. The /c=us/o=umn-cs/ou=ScmEngine/ou=proj1/ou=javaApplet/source branch includes sub-branches which hold source code with information pertinent to Java Applets for Project1 such as version control and security authentication specific schemas.

In order to avoid collisions in naming issue, ScmEngine uses commonName as a multi-valued attribute; the first attribute is an uniqueness identifier for a specific object, applet, class, a function or a file and the second value is the entity's name. The /c=us/o=umn-cs/ou=ScmEngine/ou=project1/ou= javaApplet/ou=source/ou=hello-world applet includes the unique ID, name, version control status, physical repository location, attributes about inheritances, dependencies and interoperability relations of other modules, functions or files. An entity's distinguished ID keeps stable.

An entity's name, physical storage location, and software components or sub-systems affiliation are stored as attributes for that entity; changes to these attributes do not affect the entity's usage or location in the X.500 tree. Since X.500 retains strict tree architect, ScmEngine uses aliases to deal with one-many and many-many relations among entities but the aliases are kept in a hierarchy structure consistent with the X.500 tree architect. And it also serve as pointers to entries in other branches.

3. Conclusion and Future Work

X.500 technology are typically provided by messaging vendors and are marketed primarily to messaging organizations. However, its solution and functionality can well fit into the SCM issues and problems. ScmEngine pilot prototype has been developed to demonstrate the feasibility of using X.500 model to access managed configuration data and to explore the issues of distributed software configuration management. Because the vast majority of X.500 literature describes its application to electronic messaging, X.500 is commonly misconstrued as solely a messaging technology. As a result, there are obstacles to apply the X.500 technology for a broad applications, especially the non-messaging applications.

X.500 currently is used mainly as the exchange of non-critical interpersonal messages and communication messages. It seems that it is confined only for non-critical applications. Our experiments demonstrate the advantages of X.500 model over the SCM issues, such as distributed version control, software development process modeling support. The amount of attention given to the X.500/messaging relationship tends to stifle creative non-messaging applications of X.500.

The lack of X.500/non-messaging success stories is intimidating to anyone considering a non-messaging application. In this paper, we first propose the use of X.500 model for SCM process and further we are considering X.500 as an enabler of other technologies (virtually X.500 technology could be well applied to many fields of intelligent applications such as distributed knowledge based systems.) Either messaging user agents or non-messaging users should be able to retrieve managed data from an X.500. We believe that changing technical environment efforts is key factor of using X.500 for a new generation of SCM tools. It includes using CORBA, COM and OLE techniques, distributed knowledge-based approach, HTML, and Web as a standard user interfaces for access of managed configuration data, such as distributed software configuration control and software development process modeling support.

Our search for infrastructure components to support these technologies has focused much attention on the X.500 technology. The experimental work shows that X.500 model provides a unified environment for such infrastructure toward more integrated development framework.

4. References

[ABG+91] R. Agrawal, S. Buroff, N. H. Gehani, and D. Shasha. Object versioning in Ode. In Proceedings of the IEEE 7th International Conference on Data Engineering, pages 446-455, Kobe, Japan, April 1991.

[AFK+95] Larry Allen, Gary Fernandez, Kenneth Kane, David Leblang, Debra Minard, and John Posner. ClearCase MultiSite: Supporting geographically-distributed software development. In Jacky Estublier, editor, Software Configuration Management: Selected Papers of the ICSE SCM-4 and SCM-5 Workshops, number 1005 in Lecture Notes in Computer Science, pages 194-214. Springer-Verlag, October 1995.

[AS95] Paul Adams and Marvin Solomon. An overview of the CAPITL software development environment. In Jacky Estublier, editor, Software Configuration Management: Selected Papers of the ICSE SCM-4 and SCM-5 Workshops, number 1005 in Lecture Notes in Computer Science, pages 1-34. Springer-Verlag, October 1995.

[BB95] Don Bolinger and Tan Bronson. Applying RCS and SCCS. O'Reilly & Associates, Inc. 1995.

[Ben94] Mordechai Ben-Menachem. Software Configuration Management Guidebook. McGraw-Hill, London, England, 1994.

[Bro96] Michael L. Brodie. Putting Objects to Work on a Massive Scale. In Zbigniew W. Ras and Maciek Michalewicz(Eds.), Foundations of Intelligent Systems, Proceedings of 9th International Symposium, ISMIS'96, number 1079 in Lecture Notes in Artificial Intelligence, pages 1-18, Springer-Verlag, June 1996.

[Dav93] Alan M. Davis. Software Requirements, Objects, Functions, and States. PTR Prentice Hall, 1993.

[DT94] Janet M. Drake and W.T. Tsai. System Bounding Issues for Analysis. In Proceedings of the First International Conference on Requirements Engineering, pages 24-31, IEEE Computer Society Press, 1994

[Est95] Jacky Estublier, editor. Software Configuration Management: Selected Papers of the ICSE SCM-4 and SCM-5 Workshops, number 1005 in Lecture Notes in Computer Science. Springer-Verlag, September 1995.

[HHW96] Andre van der Hoek, Dennis Heimbigner, and Alexander L. Wolf. A Generic, Peer-to-Peer Repository for Distributed Configuration Management. Proceedings of the 18th International Conference on Software Engineering (ICSE-18, '96), IEEE Computer Society Press, 1996.

[Jaz95] Mehdi Jazayeri. Component Programming--- a Fresh Look at Software Components. In Wilhelm Schafer and Pere Botella (Eds.) Software Engineering- ESEC '95, Proceedings of 5th European Software Engineering Conference, number 989 in Lecture Notes in Computer Science. Springer-Verlag, September 1995.

[KM93] David H. Kitson and Stephen Masters. An Analysis of SEI Software Process Assessment Results, 1987-1991. In Proceedings of the Fifteenth International Conference on Software Engineering, pages 68-77, Washington, D. C. IEEE Computer Society Press, 1993.

[Mil94] Organizing chaos. Database Programming & Design: Oct 1994. Miller Freeman Publications 1994.

[Rad94] Sara Radicati. X.500 Directory Services, Technology and Deployment. International Thomson Computer Press, 1994.

[Tho95] Thompson, George A. Ch-ch-ch-changes. HP Professional: Oct 1995.

[WCS94] David P. Wood, Michael G. Christel and Scott M. Stevens. A Multimedia Approach to Requirements Capture and Modeling. In Proceedings of the First International Conference on Requirements Engineering, pages 53-56, IEEE Computer Society Press, 1994

[WG95] Tim A. Wagner and Susan L. Graham. Integrating Incremental Analysis with Version Management. In Wilhelm Schafer and Pere Botella (Eds.) Software Engineering- ESEC '95, Proceedings of 5th European Software Engineering Conference, number 989 in Lecture Notes in Computer Science. Springer-Verlag, September 1995.

[BD91] Edward H. Bersoff and Alan M. Davis. Impacts of Life Cycle Models on Software Configuration Management. Communications of the ACM 34, no.8 (August 1991): 104-118.

[Chen 83] P. S. Chen. *The entity-relationship approach to information modeling and analysis.* Amsterdam, The Netherlands: North-Holland, 1983.

[ECMA 95] ECMA. *Standard ECMA-149 Portable Common Tool Environment (PCTE) abstract specification.* European Computer Manufacturers Association, draft version 3, March 1995.

[IEEE 87] IEEE. *An American National Standard, IEEE Guide to Software Configuration Management*. Technical Committee on Software Engineering of the Computer Society of IEEE, approved September 10, 1987 by IEEE Standards Board and approved March 10, 1988 by American National Standards Institute. (ANSI/IEEE Std 1042-1987).

[Joiner 94] J. K. Joiner, W. T. Tsai, X. P. Chen, S. Subramanian, C. Boddu and J. Sun, "An Integrated Data-Centered Model for Software Maintenance", in Proc. IEEE International Conference on Software Maintenance, Sep, 1994, pp 272-281.

[Oberndorf 88] P. A. Oberndorf. The common Ada programming support environment (APSE) interface set (CAIS). *IEEE Trans. Software Eng.*, 14(6): 742--748, June 1988.

[Onoma 95] A. K. Onoma, W. T. Tsai, F. Tsunoda, H. Suganuma, and S. Subramanian. Software maintenance -- an industrial experience. *Journal of Software Maintenance*, 7(12): 333--375, December 1995.

[Onoma 95a] A. K. Onoma, W. T. Tsai, M. Poonawala, H. Suganuma. Regression testing in an industrial environment. TR., University of Minnesota, December 1995.

[Ramamoorthy 90] C. V. Ramamoorthy, Y. Usuda, A. Prakash, and W. T. Tsai. The evolution support environment system. *IEEE Trans. Software Eng.*, 16(11): 1225--1234, November 1989.

[Tsai 95] W. T. Tsai, "Joint Industry-University Projects in Software Engineering - Perspective and Experience", TR., University of Minnesota, January 1994.

[Feiler 91] P. H. Feiler, "Configuration Management Models in Commercial Environments", TR. CMU/SEI-91-TR-7, March 1991.