Lecture 19
Configuration Management

Kenneth M. Anderson

Foundations of Software Engineering

CSCI 5828 - Spring Semester, 2000

These slides taken from...

A Reusable, Distributed Repository for Configuration Management Policy Programming

Dissertation Defense

André van der Hoek

Software Engineering Research Laboratory

University of Colorado at Boulder

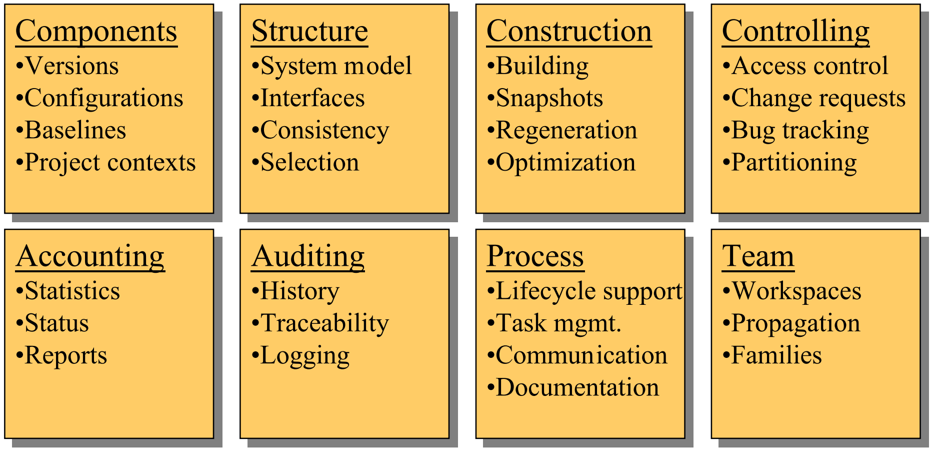
Dissertation Advisor: Alexander L. Wolf

Configuration Management

- “Configuration management (CM) is a discipline whose goal is to *control changes* to large software through the functions of: component identification, change tracking, version selection and baselining, software manufacture, and managing simultaneous updates (team work).”

Walter Tichy, SCM-1, 1988

CM Functionality



Components
•Versions
•Configurations
•Baselines
•Project contexts

Structure
•System model
•Interfaces
•Consistency
•Selection

Construction
•Building
•Snapshots
•Regeneration
•Optimization

Controlling
•Access control
•Change requests
•Bug tracking
•Partitioning

Accounting
•Statistics
•Status
•Reports

Auditing
•History
•Traceability
•Logging

Process
•Lifecycle support
•Task mgmt.
•Communication
•Documentation

Team
•Workspaces
•Propagation
•Families

Susan Dart, SCM-3, 1991

Existing CM Systems

- Process-based configuration management
  - ClearCase, Continuus, Razor, TrueChange, …
- Version control
  - CVS, Perforce, RCS, SourceSafe, StarTeam, …
- Build
  - dmake, imake, Jam, make, nmake, Openmake, …
- Miscellaneous
  - Merge Right, .RTPatch, WebKeeper, …

Challenges and Pressures

- Manage artifacts other than source code
  - Web sites, software architectures, legal databases
- Obtain customized solutions
  - comply with company standards, synchronize via e-mail, trace fine-grained artifacts
- Research and develop new approaches
  - feature logic, module-based CM, software deployment

All in a distributed setting!

Problem

- Difficult to adapt/extend existing CM systems
  - strongly geared towards source code
  - inflexible
  - rigid architecture
- Difficult to build from scratch
  - several rounds of prototyping
  - large amount of infrastructure
  - distribution

Goal

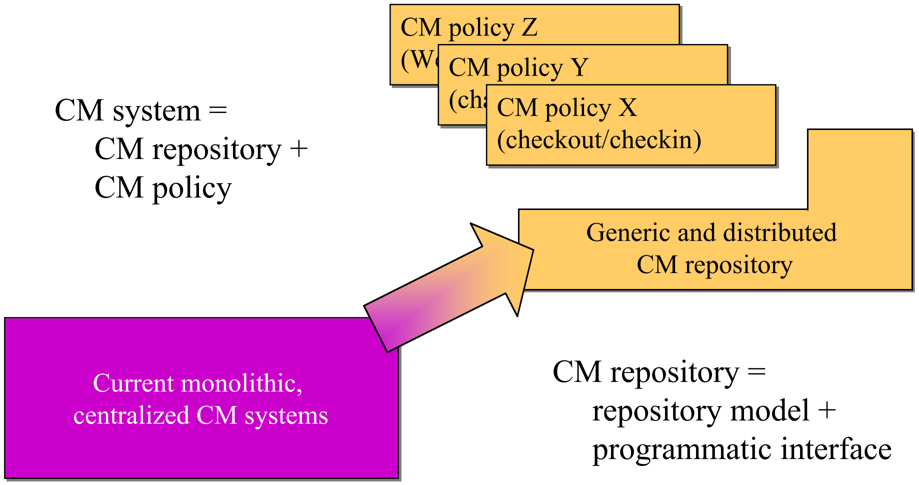
- Define and develop an abstraction layer that provides a testbed for CM policy programming
  - rapid development of new, *prototype* CM systems
  - rapid experimentation with new CM policies
  - inherent distributed operation
- Focus: storage, versioning, distribution, and access
- Out of scope: CM policy integration

Roadmap

- Abstraction layer
  - key observation
  - CM repository versus CM policy
  - repository model
  - programmatic interface
- Evaluation
- Conclusions

Key Observation: Separation of CM Repository from CM Policy

CM system =
CM repository +
CM policy


CM policy Z
(W
CM policy Y
(ch
CM policy X
(checkout/checkin)
Generic and distributed CM repository
Current monolithic, centralized CM systems

CM repository =
repository model +
programmatic interface

CM Repository versus CM Policy

CM Repository

- store for versions of software artifacts and information about these artifacts
- knows about versions
- supports distribution

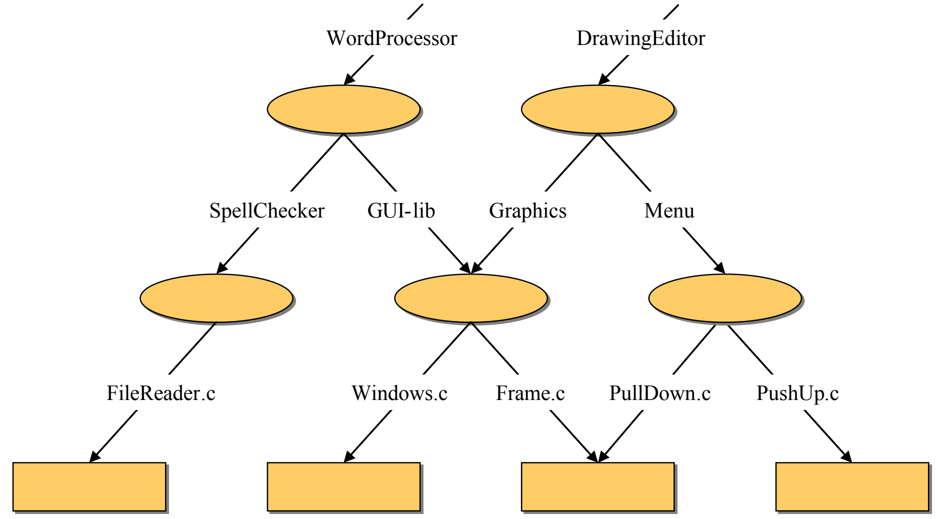
CM Policy

- specific procedures for creating, evolving, and assembling versions of artifacts
- maintains relationships among versions
- places artifacts in specific locations

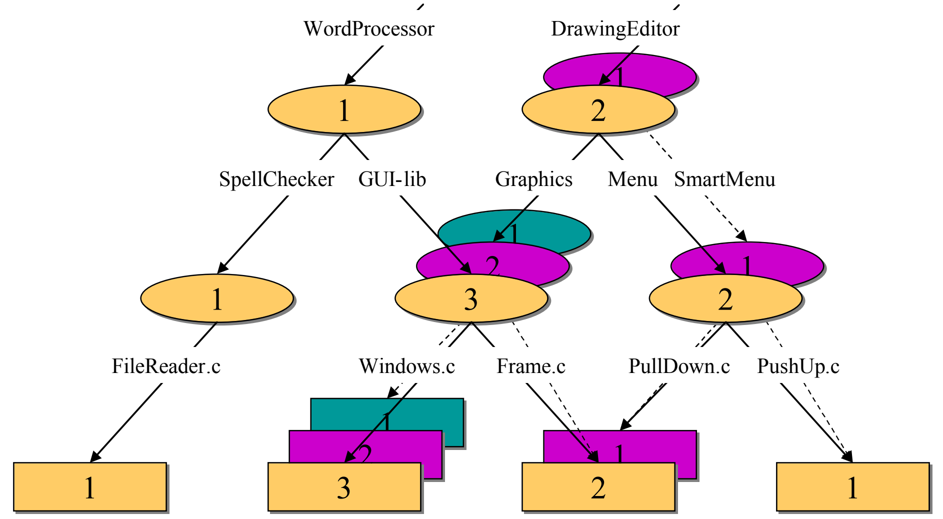
Repository Model

- Five submodels are defined
  - storage model
  - distribution model
  - naming model
  - access model
  - attribute model
- Others could be added
  - security model

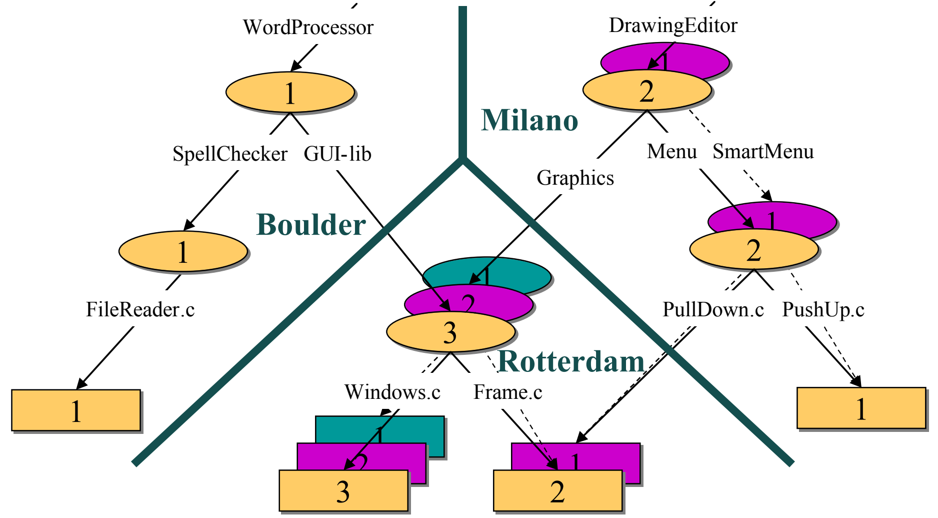
Basic Storage Model



Versioning in the Storage Model



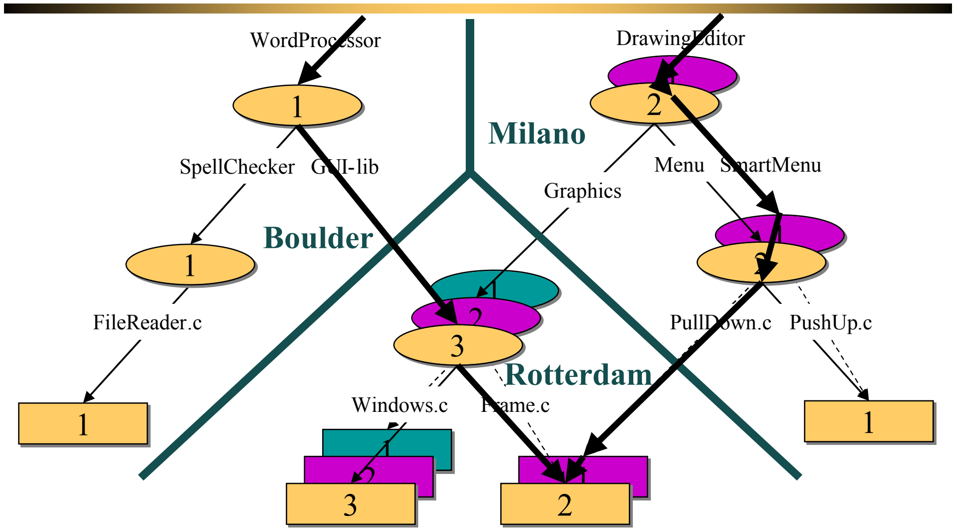
Distribution Model



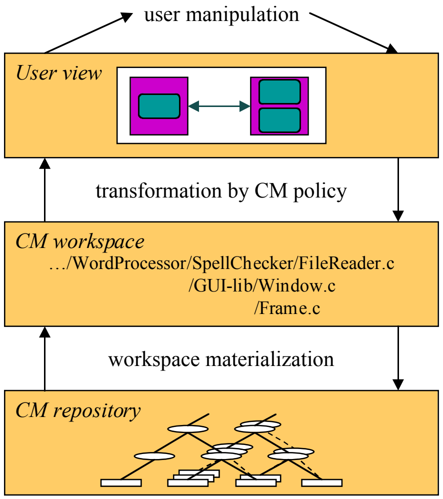
Naming Model

- ✦ Versioned path name
- ✦ Crosses distribution boundaries
- ✦ Examples
  - //Boulder/WordProcessor/SpellChecker/FileReader.c
  - //Boulder/WordProcessor/GUI-lib/Frame.c
  - //Milano/DrawingEditor/Graphics:3/Frame.c
  - //Milano/DrawingEditor:1/SmartMenu:2/PullDown.c:2

Examples



Access Model


user manipulation
User view
transformation by CM policy
CM workspace
…/WordProcessor/SpellChecker/FileReader.c
/GUI-lib/Window.c
/Frame.c
workspace materialization
CM repository

— User manipulation

— User view

— Transformation by CM policy

— CM workspace

— Workspace materialization

— CM repository

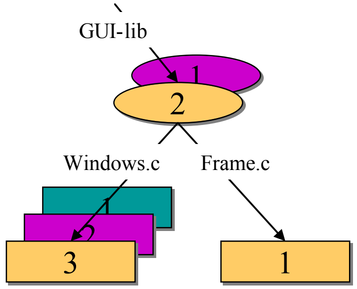
Attribute Model

1	—	**Revision = 1.0** **Author = Andre** **Change comment = initial version**
2	—	**Revision = 1.1** **Author = Antonio** **Change comment = adapted to HP**
3	—	**Revision = 1.2** **Author = Andre** **Lock = andre@cs.colorado.edu**

Programmatic Interface

Access	Versioning	Collection	Distribution
•open •close	•initiateChange •abortChange •commitChange •commitChange-AndReplace	•add •remove •rename •replaceVersion •copy •list	•setmyServer •getLocation •move
Deletion	**Query**	**Attribute**	
•destroyVersion	•getType •version •lastVersion •existsVersion •isInitiated •isOpen	•testAndSet •set •get •remove •selectVersions	

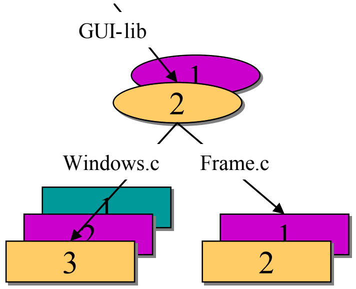
Example



…/GUI-lib/Windows.c
/Frame.c

1. nc_open(GUI-lib)
2. nc_open(GUI-lib/Windows.c)
3. nc_open(GUI-lib/Frame.c)
4. nc_initiatechange(GUI-lib)
5. nc_initiatechange(GUI-lib/Frame.c)

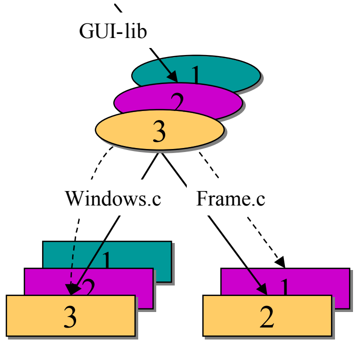
Example (continued)



…/GUI-lib/Windows.c
/Frame.c

6. nc_commitChange(GUI-lib/Frame.c)

Example (continued)



…/GUI-lib/Windows.c
/Frame.c

7. nc_replaceVersion(GUI-lib, Frame.c, 2)
8. nc_commitChange(GUI-lib)

Key Principles underlying the Abstraction Layer

- Policy independent
- Simple yet precise
- Inherently distributed
- Orthogonal
  - isolation of distribution

Roadmap

- Abstraction Layer
- Evaluation
  - expressiveness
  - feasibility
  - utility & validity
- Conclusions

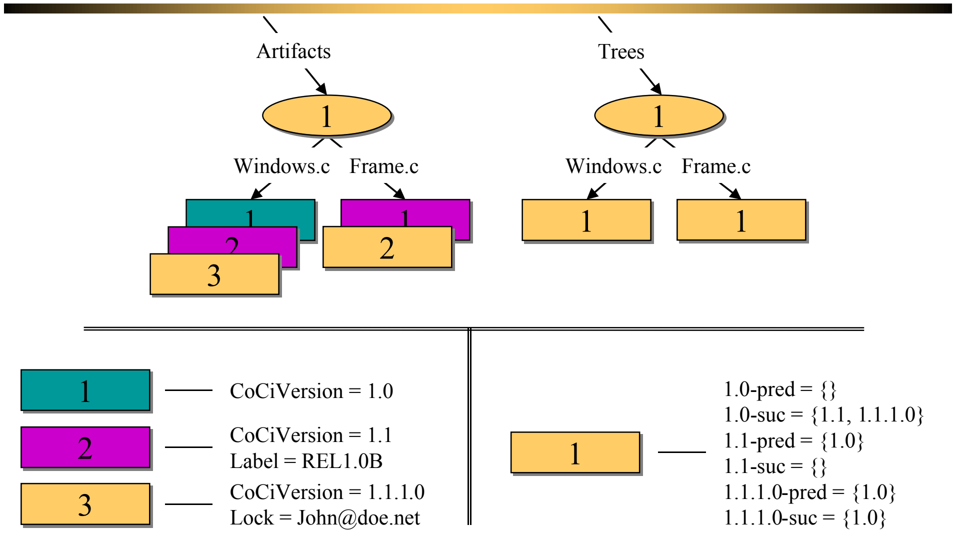
Expressiveness

- Versioning aspects of existing CM policies
  - checkout/checkin, composition, long transaction, change set
- Distribution aspects of existing CM policies
  - client-server workspaces, peer-to-peer repositories, distributed long transaction, repository replication
- Non-traditional CM policies
  - movement upon checkout, product family architectures

Checkout/Checkin Policy

- Pattern
  - check out an artifact version into a workspace
  - manipulate its contents in the workspace
  - check in the new contents to a repository as a new revision or new variant
- Individual artifacts
- Revisions and variants form a version tree
- Checked out artifacts are locked

Repository Design



CoCiVersion = 1.0

CoCiVersion = 1.1
Label = REL1.0B

CoCiVersion = 1.1.1.0
Lock = John@doe.net

1.0-pred = {}
1.0-suc = {1.1, 1.1.1.0}
1.1-pred = {1.0}
1.1-suc = {}
1.1.1.0-pred = {1.0}
1.1.1.0-suc = {1.0}

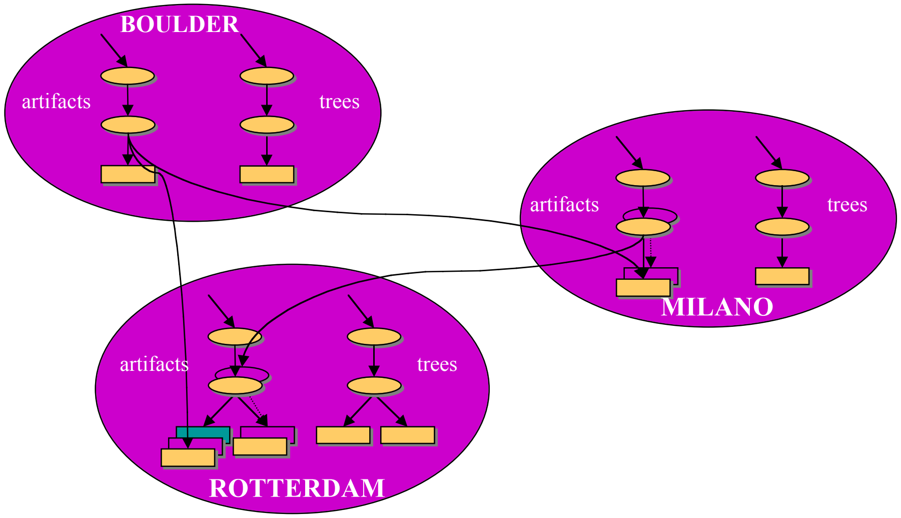
Core Policy Design

```
proc lock { artifact user } {
        if { [nc_testandsetattribute $artifact "Lock" $user] == "false" } {
                set lockuser [nc_getattributevalue $artifact "Lock"]
                puts "$artifact is locked by user $lockuser"
                exit
        }
}
proc checkout { workspace content version } {
        set user $env(USER)
        set host $env(REPOSITORYHOME)
        set artifact "//$host/Artifacts/$content"
        set filename [file tail $content]
        set wsartifact "$workspace/$filename"
        set storageversion [lindex [nc_selectversions $artifact "PolicyVersion" $version] 0]
        set artifact "$artifact:$storageversion"
        lock $artifact $user
        nc_open $artifact $workspace
        nc_initiatechange $wsartifact
}
```

Peer-to-Peer Repositories Policy

- Pattern
  - checkout/checkin
- Manages compound artifacts
- Each artifact can be stored in a different location
  - cross-repository membership

Repository Design



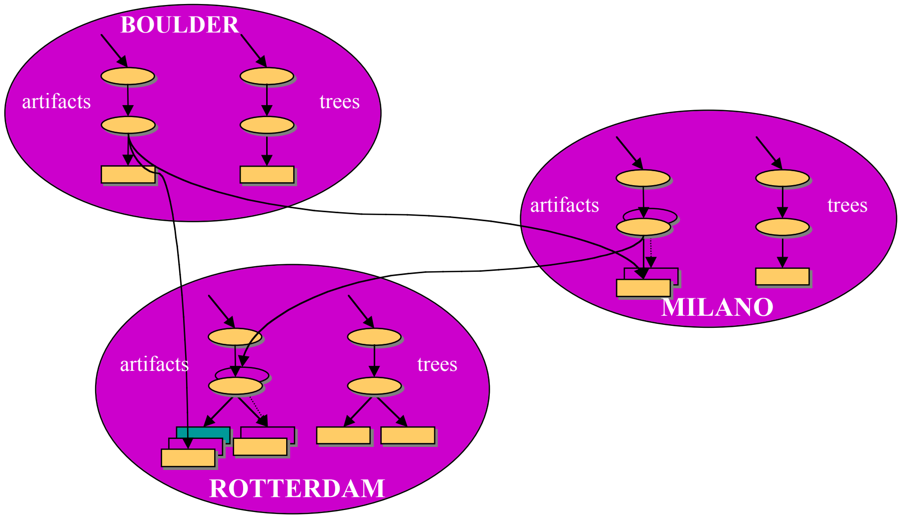
Core Policy Design

```
proc createfederation { myhost collection itshost theartifact } {
        set user $env(USER)
        set workspace "/tmp/workws"
        set filename [file tail $collection]
        set artifact "//myhost/Artifacts/$collection"
        set wsartifact "$workspace/$filename"

        lock $artifact $user

        nc_open $artifact $workspace
        nc_initiatechange $wsartifact
        nc_add //$itshost/theartifact
        nc_commitchange $wsartifact
        nc_close $wsartifact
        nc_removeattribute $artifact "Lock"
}
```

Movement upon Checkout Policy

- Pattern
  - peer-to-peer repositories
- Artifacts move from physical repository to physical repository
  - move is triggered by checkout

Repository Design



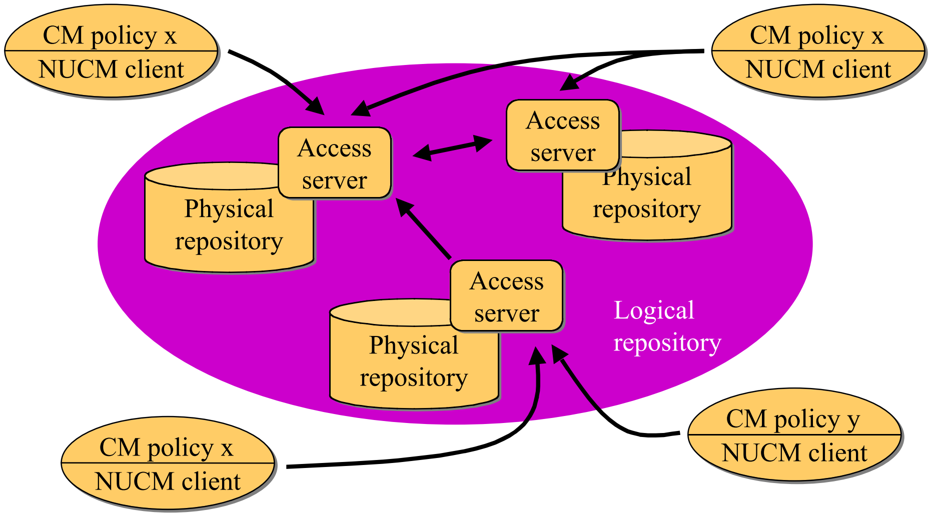
Core Policy Design

```
proc movingcheckout { workspace content version } {
        set user $env(USER)
        set host $env(REPOSITORYHOME)
        set artifact "//$host/Artifacts/$content"
        set tree "//$host/Trees/$content"
        set filename [file tail $content]
        set wsartifact "$workspace/$filename"
        set storageversion [lindex [nc_selectversions $artifact "PolicyVersion" $version] 0]
        set artifact "$artifact:$storageversion"
        set locked [nc_testandsetattribute $artifact "Lock" $user]

        lock $artifact $user

        nc_open $artifact $workspace
        nc_initiatechange $wsartifact
        nc_move $artifact $host
        nc_move $tree $host
}
```

Feasibility

- Abstraction layer is implemented and in use
  - NUCM (Network-Unified Configuration Management)
- Internal separation of concerns
  - incremental layering
  - low impact of changes to models & interface classes
- Limitations in functionality
  - no caching, compression, or delta storage

High-Level Architecture



Utility & Validity

- Three novel prototype CM systems
  - DVS -- distributed, collaborative document authoring
  - SRM -- distributed, coordinated software release management
  - WebDAV -- standard extension to HTTP for distributed authoring and versioning
- Little effort required in the implementation
- Rapid experimentation with CM policies

DVS Goal

- Support asynchronous collaborative document authoring
  - centered around workspaces and locking
  - assumes linear evolution of artifacts
- Seamless support for distribution

CM policy:
peer-to-peer repositories with (modified) composition

DVS Experience

- In use for over two years
  - grant proposals (CU, UCI, Northrup, Aerospace)
  - daily paper writing (Colorado, Italy, disconnected)
- No code was written to deal with distribution
  - relies entirely on NUCM
- Only 3,000 lines of source code
- Policy has been adjusted while in use

SRM Goal

- Simplify release process
  - multiple versions
  - dependency specification
  - multiple release repositories
- Simplify retrieval process
  - deliver a system and its dependencies
  - transparent distribution

CM policy:
linear versioning with controlled peer-to-peer repositories

SRM Experience

- In use for over three years
  - DARPA EDCS program
  - CU Software Engineering Research Laboratory
- Retrieved over 350 times
  - Boeing, Raytheon, AT&T, Dallas Cowboys, …
- NUCM-oriented code: about 10 percent
- Distribution-oriented code: about 2 percent
  - join and leave

WebDAV Goal

- Extend HTTP protocol
  - metadata
  - collections
  - name space management
  - locking
  - version management

CM policy:
checkout/checkin with client-server workspaces

WebDAV Experience

- Limited to being a *partial* prototype
- Rapid implementation
  - 4 hours for checkout/checkin policy
  - one week total, including UI development
- Core of the checkout/checkin policy is a reuse of an earlier, unrelated prototype
- Shows potential for rapid prototyping

Additional, Unexpected Characteristics

- Evolution
  - CM policies can be changed relatively easy
  - limited impact on repository design from changes to policies
- Reuse
  - CM policies incorporate parts of repository and core policy designs from other CM policies

Both need to be further investigated!

Evaluation Summary

- Expressiveness
  - many different CM policies
  - many different distribution policies
  - wide variety of different kinds of artifacts
- Feasibility
  - actual implementation that is in use
- Utility
  - actual (prototype) CM systems that are in use

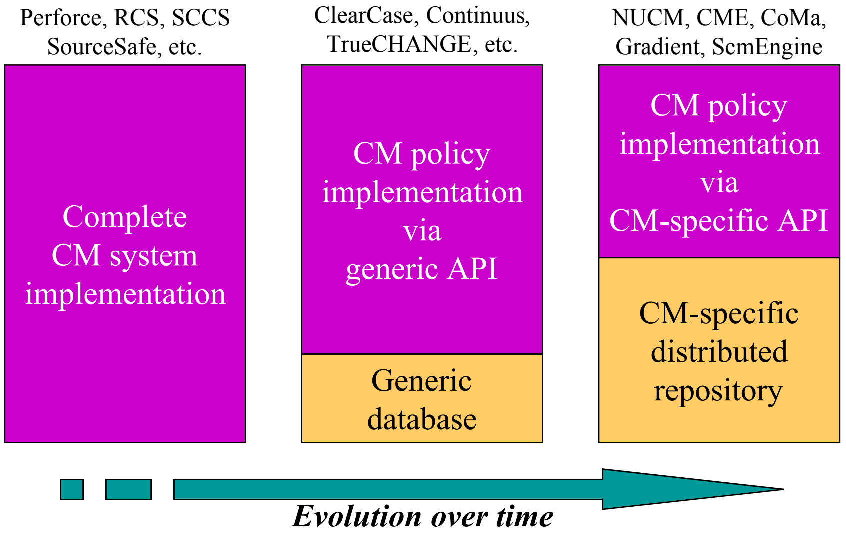
Evaluation Summary (continued)

- Validity
  - rapid construction of prototype CM systems
  - rapid experimentation with CM policies
  - inherent distributed operation
- Additional, unexpected characteristics
  - evolution of CM prototypes build with NUCM
  - incremental nature of CM policies

Roadmap

- Abstraction layer
- Evaluation
- Conclusions
  - related work
  - contributions
  - limitations
  - research impact
  - future work

Related Work -- Architectural Evolution


Perforce, RCS, SCCS
SourceSafe, etc.
ClearCase, Continuus,
TrueCHANGE, etc.
NUCM, CME, CoMa,
Gradient, ScmEngine
Complete CM system implementation
CM policy implementation via generic API
Generic database
CM policy implementation via CM-specific API
CM-specific distributed repository
Evolution over time

Related Work -- Alternative Platforms

- CME (Xcc Software, 1997)
  - limited to composition policy; not distributed
- CoMa (Westfechtel, 1996)
  - limited to composition policy; not distributed
- Gradient (AT&T Bell Laboratories, 1996)
  - limited to checkout/checkin policy; replicated repositories
- ScmEngine (Ci et al., 1997)
  - limited to distributed checkout/checkin policy

Related Work -- Other Domains

- Groupware
  - collaborative workspaces, not isolated workspaces
  - very different issues, especially in a distributed setting
- Versioned databases
  - focus on generality, not on a specific domain
  - abstraction layer can be viewed as a specific schema with a number of standard views

Contribution

- Abstraction layer that provides a reusable testbed for CM policy programming
  - model of a generic CM repository
  - programmatic interface
- Intended to lead to...
  - ...new design methods for CM systems
  - ...complete platform for constructing CM systems

Limitations

- ✦ Abstraction layer
  - inefficient in managing fine-grained artifacts
  - at times leads to heavy-weight solutions
- ✦ Implementation
  - currently not scaleable
  - currently not reliable

Research Impact

- ✦ NUCM has been downloaded over 250 times
  - many CM organizations
- ✦ Circumstantial evidence
  - Perforce -- old distribution model
  - TrueCHANGE -- release management
  - WebDAV -- collection mechanism

Future Work

- ✦ Can we futher raise the level of abstraction?
  - high-level CM policy programming language
- ✦ Can we broaden the functionality of the testbed?
  - include merge, build, and process interfaces
- ✦ Can we apply the testbed to other domains?
  - groupware
- ✦ Can we improve the functionality without changing the external interface?
  - smart caching, compression, delta storage