UTA

Today's Agenda

- Quiz 3
- HW 4 Posted
- Version Control

UTA

Outline

- Introduction
- Product & Version Space
- Interplay of Product and Version Space
- Intensional Versioning
- Conclusion

UTA

SCM

- The discipline of managing the evolution of large and complex software systems
  - A key element in achieving process maturity
- Management support: Change control, status accounting, audit and review
- Development support: Recording configurations, maintaining consistency, building derived objects, reconstructing previously recorded configurations, constructing new configurations

UTA

Version Model

- Defines the objects to be versioned, version identification and organization, as well as operations for retrieving existing versions and constructing new versions.
  - Product Space: software objects and their relationships
  - Version Space: different versions of software objects
  - Versioned object space: combines product and version space

UTA

Product Space

- Describes the structure of a software product without taking versioning into account
- Can be represented by a product graph
  - Nodes – Software objects
  - Edges – Relationship between software objects

UTA

Software Objects

- Artifacts created as the result of a development or maintenance activity
  - Examples of software objects?
- Source object vs derived object
- Object identification: each software object carries a unique identifier
- May or may not have internal structure
  - For example, a program file can be stored as a text file or a syntax tree

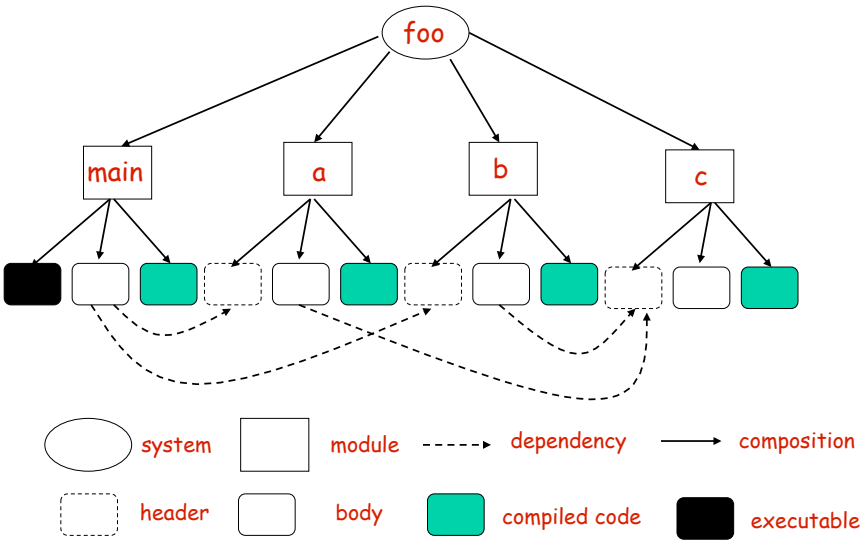
UTA

Relationship

- Composite relationship
  - Atomic objects, composite objects, and composite hierarchy
- Dependency relationship
  - Lifecycle dependencies between requirements spec, designs, and implementations
  - Import/include dependencies between modules
  - Build dependencies between compiled code and source code

UTA

Representation



UTA

Version Space

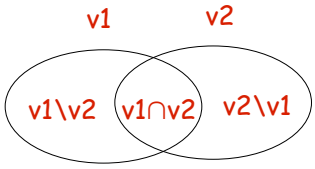
- Represents the various versions of a single item, abstracting from the product space
  - Version: a state of an evolving item
  - Versioned item: an item that is put under version control
- Versioning can be applied at any level of granularity, ranging from a software product down to text lines

UTA

Versions

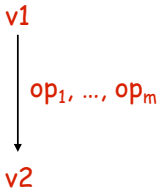
- Version identification: How to uniquely identify a version?
  - Object identifier (OID): determines whether two versions belong to the same item
  - Version identifier (VID): uniquely identifies a version within the same versioned item
- Version invariants: Properties that are shared by all the versions
  - For example, all versions must realize the same shared interface

Version Delta



$\Delta(v1, v2) = (v1 \setminus v2) \cup (v2 \setminus v1)$

Symmetric delta



$\Delta(v1, v2) = op_1 \ldots op_m$

Directed delta

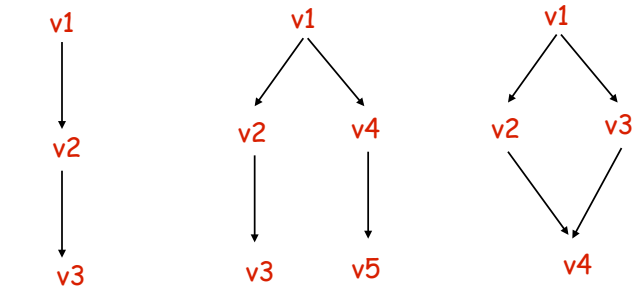
Versioning

- Extensional versioning: Versions are explicitly enumerated
  - V = {v1, v2, …, vn}
  - Versions can be made immutable automatically or on demand
- Intensional versioning: Versions are implicit and constructed on demand
  - V = {v | c(v)}
  - For example, conditional compilation can construct different versions of a source file based on certain attributes

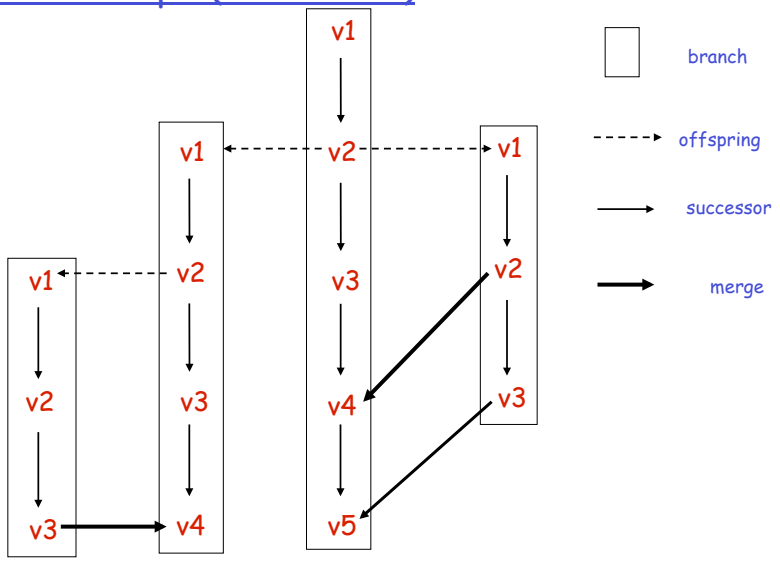
Revisions and Variants

- Revision: a version intended to supersede its predecessor
  - Bug fixes, enhancements, adaptive changes
- Variant: a version intended to co-exist with its predecessor
  - For example, a software product may support multiple operating systems

Version Graph (One level)



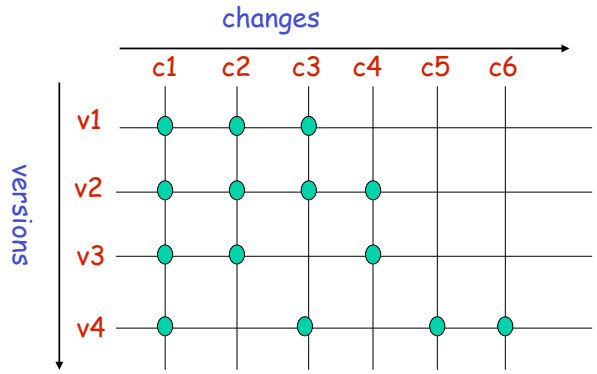
Version Graph (Two levels)



State-based vs change-based

- State-based: Versions are described in terms of revisions and variants
- Change-based: Versions are described in terms of changes applied to some baselines
  - Extensional versioning: changes are only used for documentation
  - Intensional versioning: changes can be combined freely

Change Space

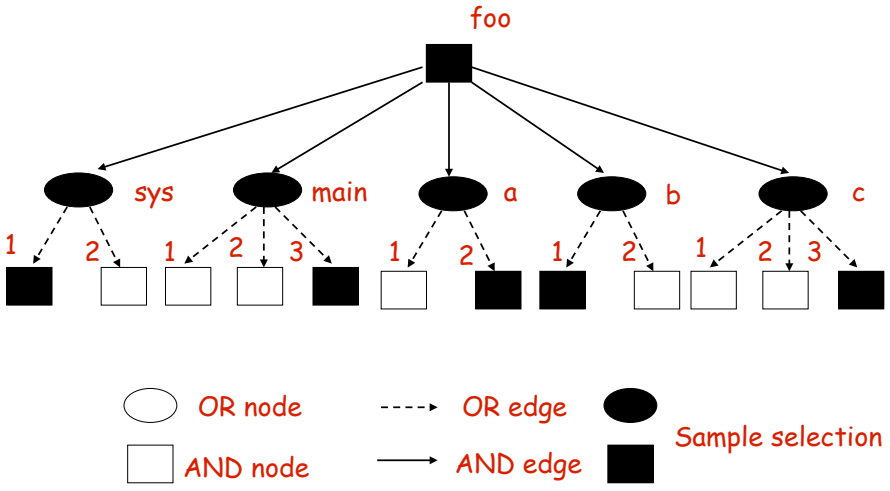


AND/OR Graph (1)

- A general model for integrating product space and version space
- AND nodes represent composition, and OR nodes represent versioning.
- Both objects and configurations can be versioned.

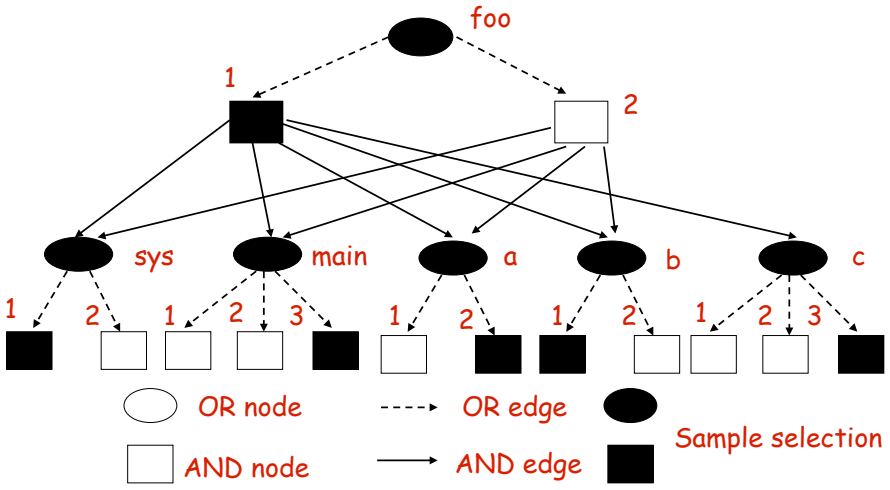
UTA

AND/OR Graph (2)



UTA

AND/OR Graph (3)



Version Granularity

- Component versioning: Only atomic objects are put under version control
  - Select the product structure first, and then the versions of the atomic objects
- Total versioning: all levels of the composition hierarchy are versioned
  - Product structures and versions can be selected in an intertwined manner
- Product versioning: Arranging versions of all objects in a uniform, global version space
  - Provides a single-version view that hides internally maintained versions of objects and relationships

Deltas

- Directed deltas: a version is constructed by applying a sequence of changes to some base version
- Embedded deltas: all versions are stored in an overlapping manner so that common fragments are shared

UTA

Data Model

- Version model on top of data model: version management considered to be an ordinary database application
  - No support for storing versions efficiently
- Version model built into the data model: data model is extended with versioning
  - Allows customized support for storing/querying versions efficiently

UTA

Intensional versioning

- Constructs new versions from property-based descriptions
  - Difference from extensional versioning?
- Combinability: versions are constructed on demand by combining different changes
- Consistency control: a constructed version must meet certain constraints
- Configuration rules: rules to configure consistent versions from a versioned object base

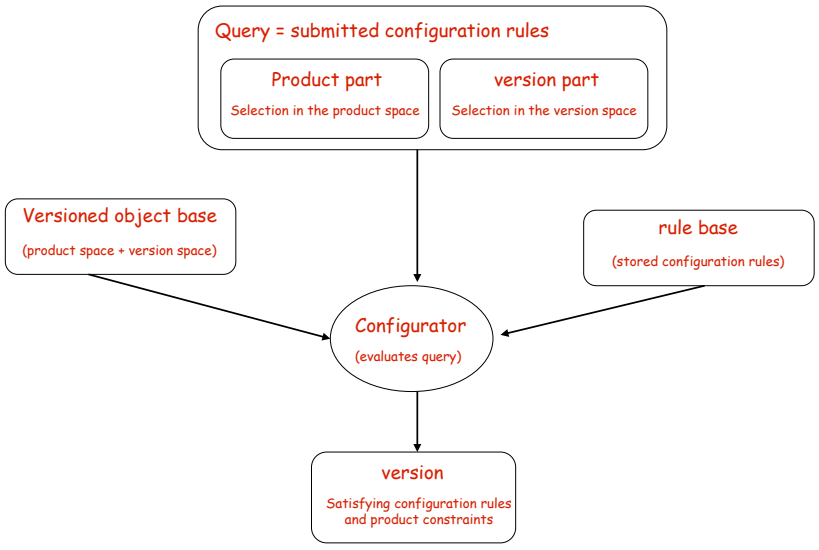
UTA

Combinability

- Assume that there are m components and each has v version. How many possible configurations?
- Configuration rules are used to rule out inconsistent combinations
- The user must be warned if a new version is created that has never been configured before
  - Quality assurance, and changes may need to make corrections

UTA

Conceptual framework



Configuration Rules (1)

- Built-in rules: hardwired into the SCM system and cannot be changed by the user
  - E.g.: At most one version of a software object is contained in any constructed configuration
- User-defined rules: supplied by the user
  - E.g.: select the latest version before Nov. 10th

Configuration Rules (2)

revision space
(1) t = max
(2) no = 1.1.1.1
variant space
(3) os = Unix && ws = X11 && db = oracle
(4) ! (os = DOS && ws = X11)
change space
(5) c1 c2 c4
(6) c2 => c1
(7) c1 ⊗ c2 ⊗ c3

Version Space

Configuration Rules (3)

1. a : status = checked_out
2. * : os = Unix && ws = X11 && db = Oracle
3. b.DependsOn* : name = alpha_release

Product Space

Strictness Class

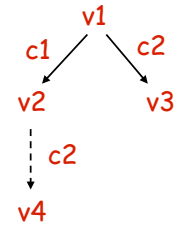
- Constraint: A mandatory rule that must be satisfied
- Preference: An optional rule that is applied only when it can be satisfied
- Default: An optional rule that is applied only when no unique selection could be performed otherwise
- Constraints -> preferences -> defaults

Configurators

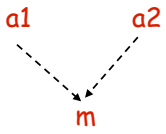
- Constructs a version by evaluating configuration rules against a versioned object base
- Functional: Apply a function to its arguments a_1, …, a_n, i.e., evaluating $q(a_1, …, a_n)$
  - Version selection is assumed to be unique.
- Rule-based: Evaluate a query against a deductive database, which consists of a versioned object base and a rule base
  - A search space of potential solutions is explored in a dept-first or breadth-first manner

Merging

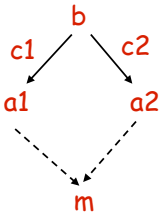
a) raw merging



b) 2-way merging



c) 3-way merging



UTA

Merge Tools

- Textual merging: applied to text files that are considered to be flat
  - Works pretty well in practice
- Syntactic merging: exploits the syntax of the versions to be merge
  - Guarantees a syntactically correct result
- Semantic merging: performs sophisticated analyses to detect semantic conflicts
  - Difficult to define semantic conflict that is neither too strong nor too weak

UTA

Major SCMs

- SCCS: Source Code Control System, Bell Labs, 1972, the first revision control system
- RCS: Revision Control System, 1980, Purdue U.
- CVS: Concurrent Versions System, 1985, client-server architecture, allows multiple developers to work together, open-Source,
- Subversion: Meant to be a better CVS, 2000, open source, http://subversion.tigris.org/
- ClearCase: Commercial, large-scale, distributed software development, Rational Software

UTA

Conclusion

- Version control is at the core of any SCM, which is further at the core of any software development organization's toolset.
- The product and version space are from the user's and tool's perspective, respectively.
- The core issues in version control are: (1) how to represent the two spaces; (2) how to store them efficiently; (3) how to present the user a consistent view?