A method for formal version numbering & distributed software configuration management

This method involves the use of Object Identifiers to designate every revision of the software code. The inherent nature of Object Identifier allows individual namespace for every object. This concept allows parallel development and multiple control mechanisms. The software tree thus generated by this method may be visualized or stored as graph of graphs type data structure. Any node in a graph can be a graph by itself.

In this method every "check-out" of the document basically creates a new "branch/fork" and a new object identifier is associated with the checked out code. Since each "check-out" creates a new "branch" file locking is not required. Each merge also creates a new branch and a new object identifier. The checked-out code based may be visualized as a new child "node" of the parent document node. Multiple parallel checkouts create children who are all siblings. This structure is shown diagrammatically as follows:

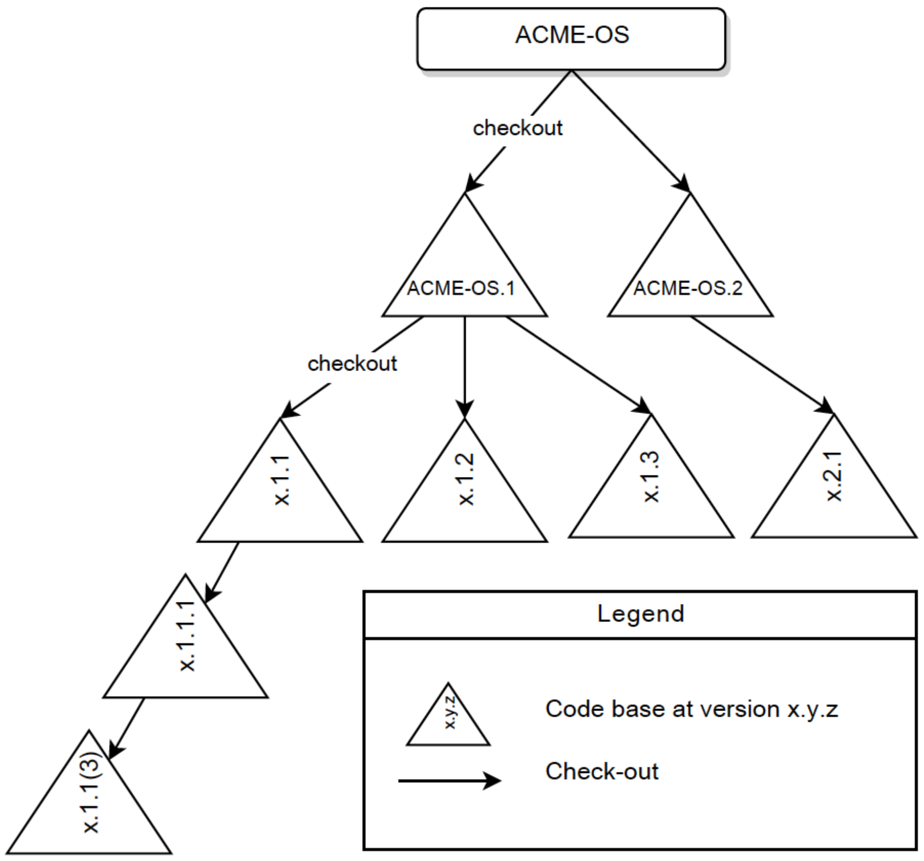


Figure 1

Certain rules that maybe followed for easier representation:

1. Repeated adjacent numbers can be clubbed together
   2.1.1.1.1.3 can be written as 2.1(4).3
2. Version numbers start from always start from natural number and 0 has special meaning.
   Versions 2.1.3.4 or 1.3.1(43).2 represent nodes in a graph
3. A zero represent a sub graph
   Version 2.1.0.1.1 represents 1.1 node in sub graph at node 2.1
4. A zero represent the node itself
   Version 1.3.4.0 is same as 1.3.4 if there is no sub graph
5. Labels can to a particular node or branch of nodes

The method of software management allows great flexibility for agile software development.

Usage Example

To illustrate a use of such technique we use a hypothetical company called Acme Inc that has released just 'Acme OS' version 1.2. The software source code is already in a version control that follows such hierarchical distributed versioning.

Parallel check-outs

There is a new requirement to add support IPv6 and 'Alice' wants is the new feature called "ipv6" to the Acme OS Version 1.2, when she checks out the code a new object ID 1.2.1 is created and assigned to her copy of code. The code 1.2.1 which is also labeled as 1.2.IPV6 to mark the branch. In parallel development 'Bob' checks out his Acme OS version 1.2 to develop another feature his version is 1.2.2, his code base is a sibling to Alice's code base as follows:

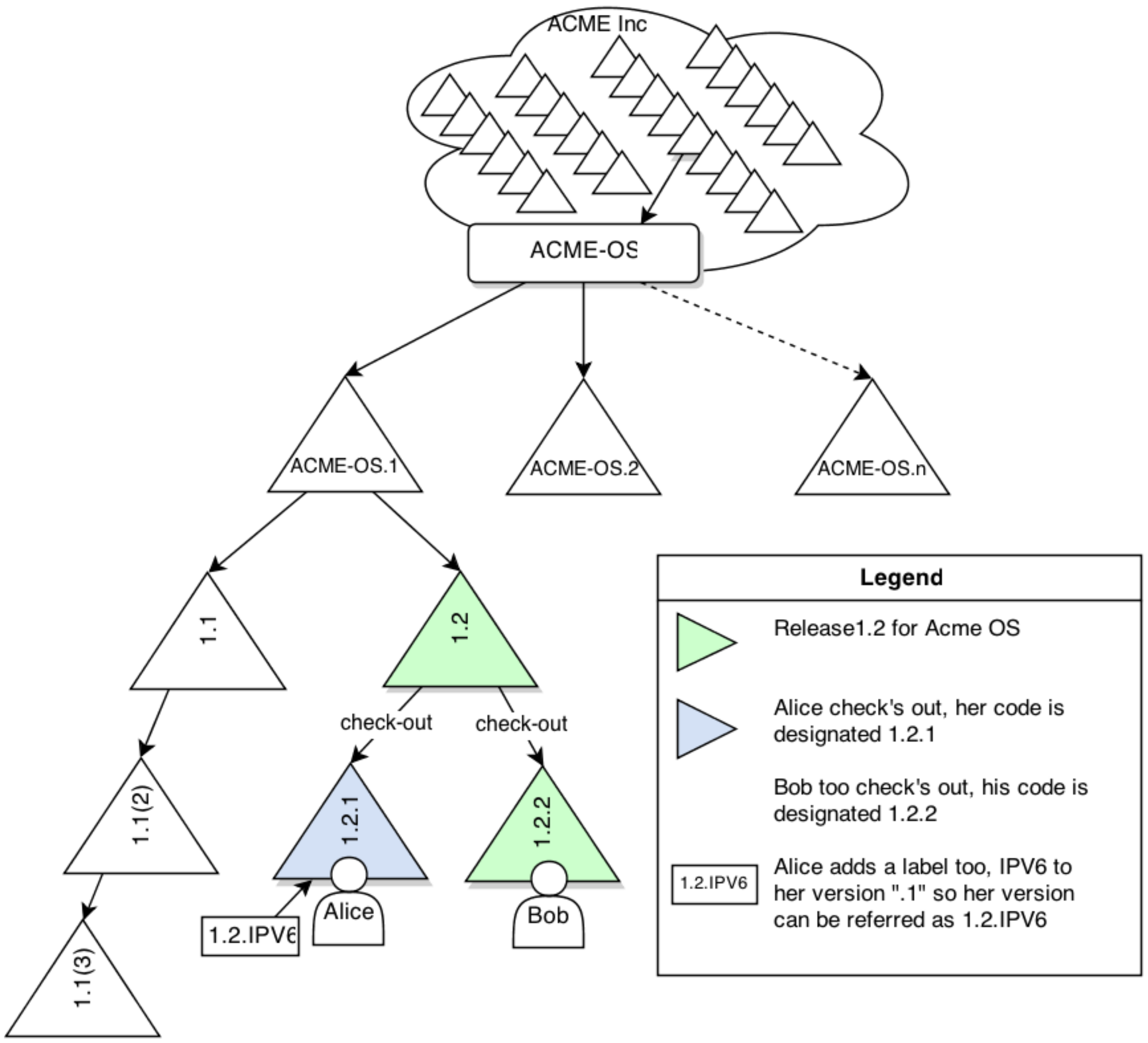

ACME Inc
ACME-OS
ACME-OS.1
ACME-OS.2
ACME-OS.n
1.1
1.2
1.1(2)
1.1(3)
check-out
check-out
1.2.1
1.2.2
Alice
Bob
1.2.IPV6
Legend
Release1.2 for Acme OS
Alice check's out, her code is designated 1.2.1
Bob too check's out, his code is designated 1.2.2
1.2.IPV6
Alice adds a label too, IPV6 to her version ".1" so her version can be referred as 1.2.IPV6

Figure 3

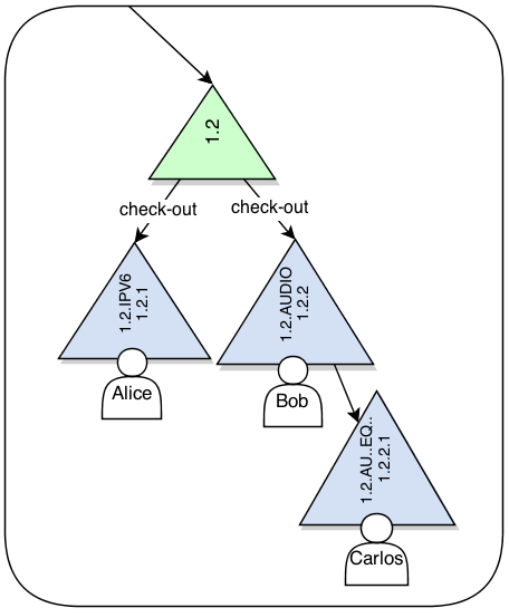


Figure 2

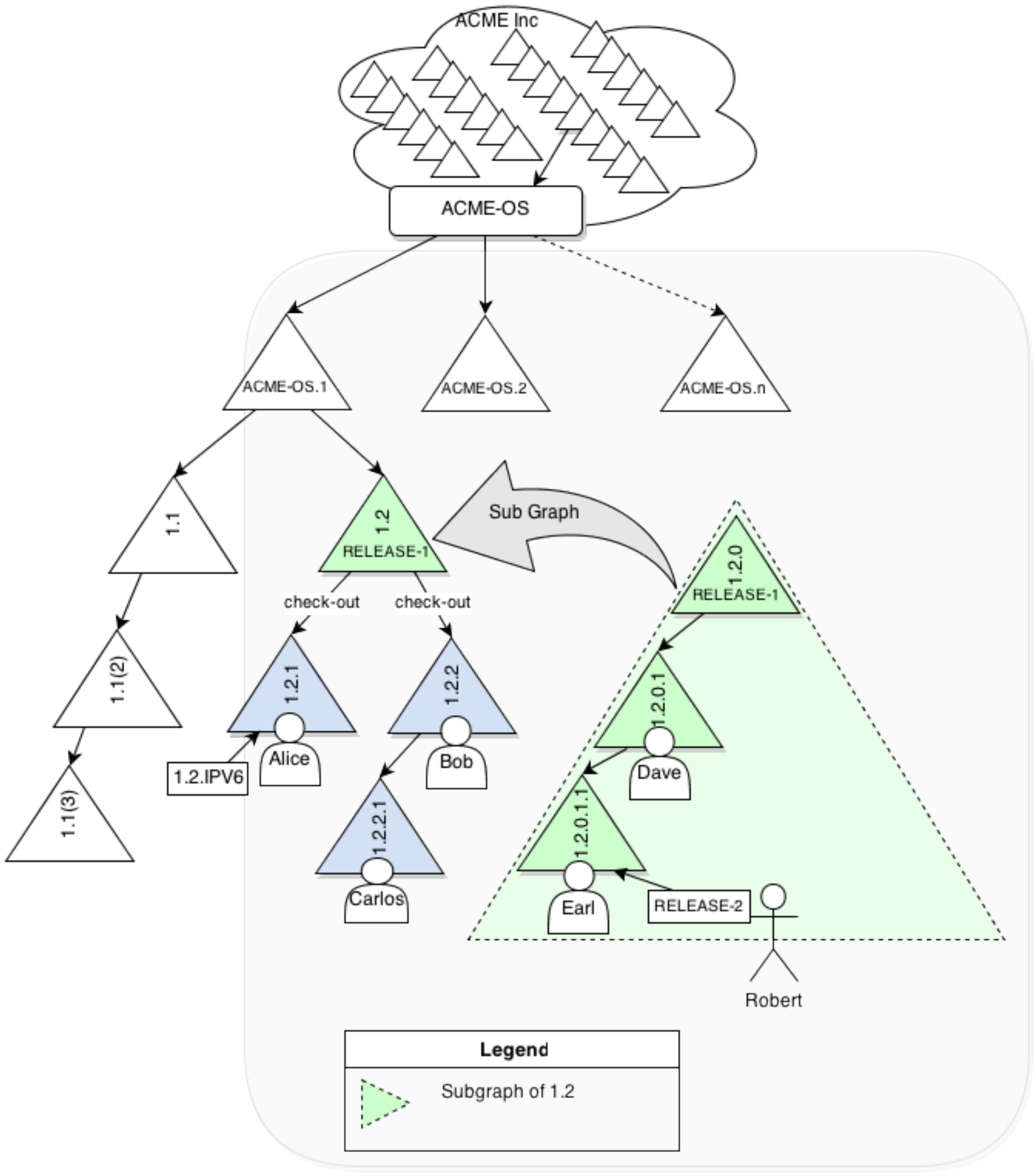
Labeling

Bob is going to implement Audio feature for the OS, an intern ‘Carlos' is asked to implement audio equalizer for Bob’s audio feature. Bob can label his version 1.2.2 as 1.2.AUDIO and Carlos’s code is labeled as 1.2.AUDIO.EQUALIZER. The tree now looks as shown in figure 3.

Bug fixing

Soon a critical bug is discovered in Acme OS version 1.2, and "Dave" is assigned to fix it, without creating a new major version, he checkout Acme OS Version 1.2.0 as a "sub graph" and his code is designated as 1.2.0.1, while 1.2.0 is the original code. He checks in quickly but after testing his code does not fix all cases. Earl is asked ensure all the cases are fixed, he checks out Dave's code 1.2.0.1 and his code is assigned as 1.2.0.1.1, he implements the fix and checks it in.

The release manager Robert then label version 1.2.0.1.1 as Release 2 or more formally it can be represented as 1.2.RELEASE-2 (RELEASE-2 in place of 0.1.1)



Merging new features

Robert' manager then wants him to add Alice's new feature in Release-3, so he merges Alice's code base 1.2.1 (1.2.IPV6) to 1.2.0.1.1 this will also ensure Earl's bug fix is also merged. Once merged it creates a new version 1.2.0.1.1.1

Now Robert can label 1.20.1(3) as RELEASE-3. But his manager thinks it's intuitive to use 1.2.0.2 instead. So he checks out 1.2.0 again creating 1.2.0.2 and merges 1.2.0.1(3) to it. Robert then also lock the branch 1.2.0.2.* to prevent any commits inside 1.2.0.2.0.* namespace. The 1.2.0.2 is labeled as 1.2.RELEASE-3) If a bug is found in Release 3, Robert will check out 1.2.0.2, which will create 1.2.0.2.1. After the above software tree is as follows:

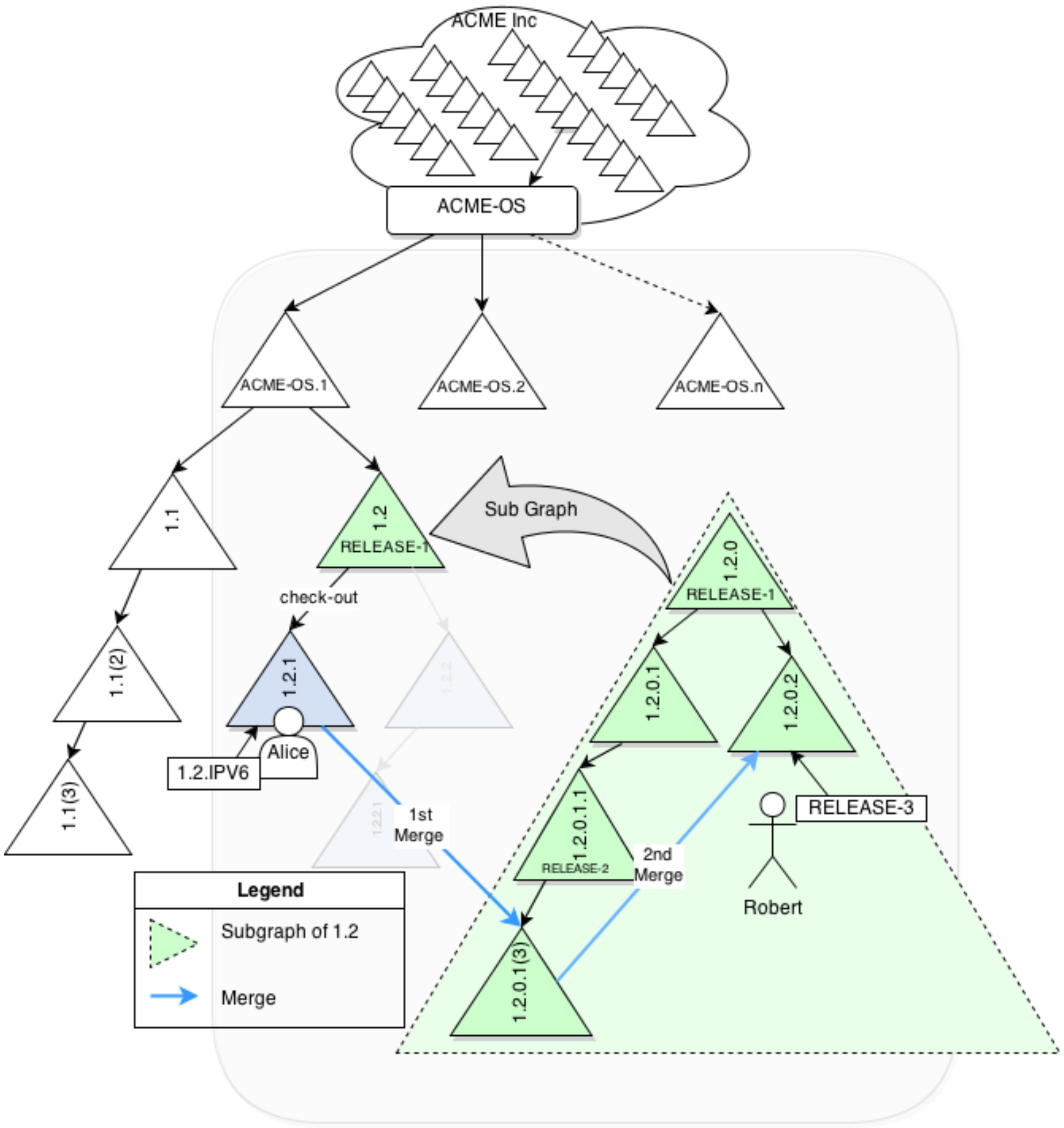


Figure 4

Advantages

For large distributed Software management only part of the name space maybe replicated for local access to Software source tree. And after development maybe merged back.

Individual namespace and labeling allows better control over source code management.

E.g. Acme OS 1.2 has following features 1.2.NETWORKING, 1.2.MULTIMEDIA, 1.2.FILESYSTEMS, 1.2.USB, 1.2.SEC etc. And Networking components are developed in NYC, while File-System components are in London and Security components in Bangalore. The software tree is thus split and replicated to repository accordingly. 1.2.SEC can be hosted locally in Bangalore repository and 1.2.FILESYSTEMS at London. Users will usage local repositories.

Hierarchy also allows fine-grained control, e.g. 1.2.SEC can have sub branches as 1.2.SEC.USERS, 1.2.SEC.AUTHNZ, 1.2.SEC.ENCRYPTION. This allows individual release managers for each feature who only have control of their own namespace.

Offline development is also possible as users will check out the code to their laptops and commit in it creating local source revision tree. When online it's merged back to repository, all history of changes done locally is preserved.

High flexibility using graphs of graphs allows a version number to represent a snapshot of source code (a node) or collection of snapshots (a sub graph). If a number version number represents a sub-graph, and it is to be checked out, the node which was last checked-in is used. This flexibility opens several possibilities for agile development, testing branches, integration branches, bug-fixes, new releases within this single framework.