

The Timesafe ® Property - A Formal Statement of Immutability in CM

Damon B. Poole
AccuRev Inc.
Lexington, Massachusetts, USA
d.poole@accurev.com
[Unpublished. Last revised: April, 2002]

Abstract. A basic function of Configuration Management (CM) is to accurately reproduce past and present information. Many other aspects of CM depend on this function. Flaws in this function affect and are magnified by other aspects of the system. Ideally, a CM system should perform this basic function automatically and flawlessly without any special knowledge or effort on the part of the users. This paper introduces the Timesafe ® property to describe CM models that possess the capability of accurately and automatically preserving and reproducing past and present information. As a first step towards enabling the proof or disproof that a CM model is Timesafe, this paper provides a formal description of the Timesafe property and the currently known list of counter-examples.

1 Introduction

Anybody who has ever used or maintained a CM system knows that it is no easy task. It requires a great deal of discipline, special knowledge, caution, and experience. Those hardy souls who have worked in environments involving hundreds of users, millions of lines of code, and many projects spread across the globe know that it can be a demanding, brutal, and practically overwhelming job.

Why is CM so hard? The author believes a major reason is that it takes a great deal of discipline, special knowledge, caution, and experience to make sure that configurations are accurately created and reproduced. In section 4, there are twenty-seven examples of typical problems, most of which can be found in most CM systems available today.

This paper introduces the Timesafe property to describe CM models and systems that accurately and automatically reproduce configurations. In the Adele Configuration Manager, *historic objects* "...should not be modified, since they correspond to a past reality and can not be changed!" [EC95]. This is a good example of the Timesafe concept. Hypothetically, Adele could be used to create a Timesafe CM model (and system) using historic objects.

A CM system based on a Timesafe model should relieve a great deal of the burden of performing CM activities and allow resources to be freed up to focus on higher level activities such as raising an organization's Capability Maturity Model level to level 2 or higher.

After a brief description of the CM concepts used in this paper, the Timesafe property is informally introduced and then developed into a formal statement expressed in first-order logic. This is followed by a section containing counter-examples that can be used to prove that a CM model or system is not Timesafe. Next there are high level requirements for constructing a Timesafe system and an example of a simple system which is hypothesized to be Timesafe. Finally, there is a short discussion of future work.

The scope of this paper is limited to information stored directly within a CM system and reproduced via retrieval from that system. No attempt is made to apply the Timesafe property to derived information. For example, the retrieval of files used to build a program is within the scope of this paper and the actual building of that program is outside the scope of this paper.

2 Anatomy of a CM System

Although this paper concentrates on CM as implemented in software, a CM model can also be implemented by other means and the material related to CM models can be applied to any CM model, whether it is implemented in software or not.

It is common to refer to the discipline of software-based CM as Software Configuration Management (SCM). This term has been used to mean both CM systems that are implemented using software and also CM systems that are used to manage software. Generally, CM systems that manage software systems are implemented in software. Since a software-based CM system can also be used to manage things other than software, the term SCM is ambiguous. In this paper, the term SCM is used to refer to a CM system which is implemented in software but may be used to manage things other than software such as web sites.

To support the function of reproducing configurations, CM models use a number of special concepts. The following CM concepts are used in this paper:

- **Object.**
Typically files and directories.
- **Object Name.**
Typically file and directory names, but can be anything used to uniquely identify objects in a repository.
- **Version.**
Usually a monotonically increasing positive integer starting with 1 used to designate the different contents of an object over time. Any designation which uniquely identifies a version will suffice.

- **Timestamp**
The time at which an action was performed. Usually, each version is timestamped in addition to being numbered. Timestamps are often the time at which a transaction was performed.
- **Label.**
A mnemonic name (alias) used to refer to a particular version.
- **Branch.**
A branch is a point of divergence. It starts a new, uniquely named series of versions from an existing version. Different systems use different naming schemes. In RCS [Tich85], a new numbering sequence is added. Creating a branch off of 1.1 in RCS might create a new sequence such as 1.1.5.1, 1.1.5.2, 1.1.5.3, etc. In ClearCase [Rati96], there is an initial branch, /main, which begins a sequence that looks like this: /main/1, /main/2, etc. Creating a branch off of /main/2 in ClearCase would record internally that the branch was rooted at /main/2 and the new sequence would look like this: /main/gizmo/1, /main/gizmo/2, etc.
- **Attribute.**
An attribute is similar to a label. An attribute is a named value that is associated with a version. One use for attributes is to tag a version with a change request number to identify which change it is a part of. Some systems allow attributes to be applied to other objects, such as branches, as well.
- **Configuration.**
CM is typically applied to logical groupings of objects. For instance, you might want to use CM to manage all of the files that are used to build a product or all of the files contained in a web site. A configuration is one instantiation of whatever it is you are managing. You might have a configuration that represents version 1.0 of a product, or a configuration that represents new product development.
- **Change-set.**
A change-set represents a logical change. It is the set of all changes that must be applied to a previous configuration to derive a new configuration.
- **Repository.**
A repository provides storage of all names, versions, labels, branches, attributes, change-sets, and configuration definitions.
- **Replication.**
For efficiency, it is often desirable to make a copy of all of or part of a repository at one or more geographically separated sites.
- **Object mastership.**
When using replication, each object that has a copy at multiple sites is

mastered at one and only one site. That is, that site owns the object and updates to that object may only be made at that site.

- **Synchronization.**
Periodically, it is necessary to update replicated repositories with new information. This process is called synchronization.
- **Configuration definition.**
A configuration definition is a recipe that is used to specify the exact contents of a configuration. The definition is constructed from the names, versions, timestamps, labels, branches, attributes, and/or change-sets from one or more repositories.
- **Configuration name.**
A configuration name is a reference that is directly understood by the CM system and can be used to specify the configuration definition. The system uses its configuration name dictionary to automatically translate configuration names into configuration definitions.

3 The Timesafe Property

A CM system is Timesafe if and only if it can automatically and accurately reproduce configurations. This means that the results of a query for the contents of a configuration at a particular point in time should not vary with the time of the query. The result of the query "what was gizmo1.0 on Monday" performed on Tuesday should be the same as the result of the query "what was gizmo 1.0 on Monday" performed on Friday. Furthermore, the Timesafe property should not depend on any special effort or knowledge on the part of the user. The user should not have to use caution to avoid changing a configuration after it is created. It should be absolutely impossible to change the past by any means whatsoever.

Let's formalize this a bit by attempting to model RCS's behavior. Building a complete model of RCS's behavior is beyond the scope of this paper, so the behavior will be restricted to small examples.

```
% echo "I am the contents" > foo.c
% ci foo.c
% rcs -Ngizmo_990314:1.1
```

Consider a function `Config (cdef_name)` that takes the name of a configuration definition as an argument and returns the configuration corresponding to it. In this example, the configuration is named `gizmo_990314` and is defined as all versions in all files in the repository which match the label `gizmo_990314`. `Config (gizmo_990314)` evaluates to:

```
foo.c:1.1:"I am the contents"
```

If I then create a CD with foo.c,v on it, and I only ever use the CD to reproduce Config (gizmo_990314) I will always be able to reproduce Config (gizmo_990314) exactly.

But RCS and a CD do not go well together. It is reasonable to expect to be able to create new versions *and* still be able to reproduce Config (gizmo_990314) exactly.

```
% co -l foo.c
% echo "I am the new contents" > foo.c
% ci foo.c
```

I've created a new version, and Config (gizmo_990314) = foo.c:1.1:"I am the contents" is still valid.

This example didn't change the past. If it is possible to change the past in RCS, we need to be able to model that too. Here's an example of changing the past in RCS which can be used to further develop the model. In RCS, labels can move with no record of their previous location. Doing so effectively changes the past.

```
% rcs -Ngizmo_990314:1.2
```

Config (gizmo_990314) now produces:

```
foo.c:1.2:"I am the new contents"
```

We have now changed the configuration of gizmo_990314 and our simple function Config (cdef_name) is unable to model the behavior of RCS.

To accommodate RCS's behavior, we need to add another argument to the Config function to indicate the time of the query.

```
Config ( cdef_name, query_tm )
```

Assuming the previous queries were performed at time 990314 and 990315 respectively, we get Table 1.

cdef_name	query_tm	Config (cdef_name, query_tm)
gizmo_990314	990314	foo.c:1.1:"I am the contents"
gizmo_990314	990315	foo.c:1.2:"I am the new contents"

Table 1.

Since Config (gizmo_990314, 990314) != Config (gizmo_990314, 990315), the results of Config (cdef_name, query_tm) vary with query_tm and therefore RCS is not Timesafe. In RCS we have no way of automatically knowing what the

previous contents were or even if there were previous contents. RCS does not provide a way to prevent this situation from happening, let alone prevent it automatically. Thus, RCS cannot be relied upon to *automatically* preserve the past without any special knowledge or effort on the part of the user.

The next thing we need to take into account is the fact that configuration definitions can change over time, just like the version a label is associated with can change over time. RCS has no concept of a configuration definition, so we'll need to switch to a more powerful system. ClearCase supports configuration definition with something called a configuration specification, also known as a 'config spec.'

In ClearCase, a configuration definition is set to the contents of a text file supplied by the user. The configuration definition is simply a list of rules to use when selecting versions of files and directories. A simple configuration definition might look like this:

```
element * /main/LATEST
```

This means: select all elements (files or directories) in all currently available repositories. Then, for each element, select the highest version (LATEST) available on the /main branch.

Assume that only one repository exists and it currently contains the following versions:

```
foo.c:/main/1:"I am the main contents"  
foo.c:/main/gizmo/1:"I am the gizmo contents"
```

The configuration definition "element * /main/LATEST" will select foo.c:/main/1:"I am the main contents".

The configuration definition can be changed at any time. For instance, I can set it to "element * /main/gizmo/LATEST" which will select the highest version on the /main/gizmo branch. Given the current contents of the repository, this definition will select foo.c:/main/gizmo/1 .

But what if I want to know the contents of the configuration at some point in the past? ClearCase allows time-based queries too. Just add the -time option to the configuration definition:

```
element * /main/gizmo/LATEST -time 14-Mar.16:00
```

The only problem is, on 14-Mar.16:00, the definition was "element * /main/LATEST", so this configuration definition will not give me what I want

automatically. In ClearCase, only the most recent configuration definition is known to the system. Therefore, ClearCase is not Timesafe.

To express this, let's create a function ConfigDef (cdef_name, trans_tm) which gives the configuration definition in effect for configuration cdef_name at time trans_tm. Just as with the Config function, the result may vary with the query time, so we also need to add the query time to ConfigDef which gives ConfigDef (cdef_name, trans_tm, query_tm). The value of this function for values of trans_tm and/or query_tm less than the time at which a configuration definition has been created is NULL.

The Config function can be thought of as the combination of two functions, ConfigDef and a function which translates configuration definitions into configurations. Since ConfigDef has a trans_tm argument, it will also have to be added to the Config function.

Note that in a Timesafe system, rows 4 and 5 in Table 2 would be identical because the only thing that changed is the time at which the query is being performed. The values of Config (gizmo, ...) and ConfigDef (gizmo, ...) depend entirely on the values of trans_tm and query_tm.

row	trans_tm	query_tm	Config (gizmo, trans_tm, query_tm)	ConfigDef (gizmo, trans_tm, query_tm)
1	0	0	NULL	NULL
2	0	14-Mar.16:00	NULL	element * /main
3	0	14-Mar.16:01	NULL	element * /main/gizmo/LATEST
4	14-Mar.16:00	14-Mar.16:00	"I am the main contents"	element * /main
5	14-Mar.16:00	14-Mar.16:01	"I am the gizmo contents"	element * /main/gizmo/LATEST
6	14-Mar.16:01	14-Mar.16:01	"I am the gizmo contents"	element * /main/gizmo/LATEST

Table 2.

We now have the ability to formally define the Timesafe property.

Given:

a function Config (cdef_name, trans_tm, query_tm)

where:

cdef_name is the name of a configuration definition,
trans_tm is a transaction time,
query_tm is a query time

a CM model is Timesafe IFF:

```
for all cdef_name, trans_tm, query_tm1, query_tm2
  if trans_tm <= query_tm1 <= query_tm2 then
    Config ( cdef_name, trans_tm, query_tm1 )
  = Config ( cdef_name, trans_tm, query_tm2 )
```

Figure 1.

In a Timesafe model, you will always get the same result for a given query regardless of when you ask. Note that this does not preclude a Timesafe model from allowing an operation that removes a label, it only requires it to remember where the label was prior to the removal operation. This allows you to change the state of the system in the present and affect the future while preserving the past. In the past, the label is still there.

The easiest way to identify a non-Timesafe model or system is by counter-example. A large number of common counter-examples exist, but it only takes one to render a model or system non-Timesafe. One of the biggest offenders is the concept of a movable label. Interestingly, this is also one of the most heavily used features of modern CM systems.

4 Currently Known Counter-Examples

Many of the counter-examples do have workarounds, some of them obvious. Some of the workarounds involve scripting, policy, or user education. All of them require manual intervention, special knowledge, and work done outside of the control of the CM system. The focus of this paper, and of Timesafe in general, is on built-in capability, accuracy, and safety, not bolted-on workarounds.

Any CM model that allows one or more of the following operations violates the Timesafe property. Most CM systems allow most of these operations. It is possible to generalize these counter-examples and group them together, but it is harder to visualize them in a more general form.

In most cases, only quick justifications are necessary. More detailed justifications are provided for the more subtle counter-examples.

- **No enforcement of monotonic time.**
If the system cannot detect and prevent out-of-order timestamping, then query results depend on when the query is performed. A common cause for timestamp problems is server clock drift. A CM system doesn't have to

run on a system with a perfect clock, it just has to prevent the creation of timewarps in its database.

- **Permanent version removal.**
The existence of a version depends on the time the query is performed.
- **Non permanent version removal followed by recreation of the same version.**
The contents of a version depend on the time of the query.
- **Non-versioned object name changes.**
The object associated with a name depends on the time a query is performed. For instance, in rcs:

```
% mv foo.c,v bar.c,v  
% mv other.c,v foo.c,v
```
- **Non-versioned branch removal.**
The contents of a configuration that is bound to a particular time can change over time.
- **Branch removal followed by recreation of branch of the same name.**
This can produce duplicate namespace entries. Example: in ClearCase, a branch is created, and a version is created on that branch, say version /main/gizmo/1 . It has some contents. The branch is removed and recreated, again producing /main/gizmo/1 but this time with different contents. A configuration based on /main/gizmo/1 can change over time.
- **Non-versioned, non-timestamped branch addition.**
The contents of a configuration that is bound to a particular branch at a particular time can change over time.
- **Non-versioned branch renaming.**
Creating a time-based definition after the rename using the old branch name will fail.
- **Non-versioned, non-timestamped attribute removal.**
The contents of a configuration based on an attribute can change over time.
- **Non-versioned, non-timestamped attribute value change.**
The contents of a configuration based on an attribute value can change over time.
- **Non-versioned, non-timestamped attribute renaming.**
Creating a time-based definition after the rename using the old attribute name will fail.

- **Non-versioned attribute addition.**
The contents of a configuration that is bound to a particular attribute can change over time.
- **Non-versioned label removal.**
The contents of a configuration based on a label can change over time.
- **Non-versioned label movement.**
The contents of a configuration based on a label can change over time.
- **Non-versioned label renaming.**
Creating a time-based definition after the rename using the old label name will fail.
- **Non-versioned, non-timestamped label addition.**
The contents of a configuration based on a label can change over time.
- **Permanent non-versioned change-set removal**
A configuration that depends on a removed change-set can change over time.
- **Non-versioned change-set renaming.**
Creating a time-based definition after the rename using the old change-set name will fail.
- **Configuration definition not directly maintained by system.**
If a configuration definition is not directly maintained by the system, it is possible for two users to have two different definitions of the same thing at the same time. Example 1: rcs does not maintain configuration definitions, each user must know which branch or label to check-out against. Example 2: ClearCase does not maintain configuration definitions. The user must manually set them. Although they are just text and may be stored as text objects 'within' the system, they must be manually retrieved and set by the user before being used by the system directly.
- **Configuration definition not directly versioned by system.**
It is not enough for a system to maintain the current definition of a configuration. It must maintain all past definitions as well. Otherwise, time-based queries will use the current definition instead of the definition in effect at the time included in the query. Example: the current definition of gizmo is 'the highest version on branch foo' and it used to be 'the highest version on branch bar.' The definition was changed at time 2. A query of the current contents of gizmo will return the correct results. A query of the contents of gizmo at time 1 will return the highest version on branch foo at time 1 instead of the highest version on branch bar at time 1.
- **Some configuration definition done manually.**
Example: manually applying labels to capture the state of a configuration

at a point in time (snapshotting). If an unnoticed error occurs during the process or the labeling command is issued incorrectly, the contents of the configuration will not match the definition of the configuration.

- **Configuration definition tied to objects by name, not unique internal identifier.**

If an object's name changes, the references in configuration definitions including the name will be wrong.

- **Configuration definitions which include information by implication.**

For instance, in ClearCase, configuration definitions (configuration specifications or config specs) are not directly tied to a repository (vob) or repositories. The complete contents of a configuration are implied by whatever vobs are available at the time of the query. Thus, the contents of the configuration depend on the time the query is performed.

- **Allowing query or definition beyond the time of the last update using non-locally mastered replicated data.**

Replicated non-locally mastered information is only as up to date as the time of the last update. If the system allows time-based queries or 'LATEST' queries, the query may return different results as subsequent updates make more information available. Example: foo.c is not mastered locally. Version 1 was created at time 1 and version 2 at time 2. The non-local information is only updated to time 1, so only version 1 is available locally. The query 'what is the version of foo.c corresponding to time 2' will produce version 1 since it appears locally that at time 2, version 1 was available, although the correct answer is 2. The repository is then synchronized with the repository that has mastery of foo.c. The query will now produce the correct result, version 2. The result of the query depends on the time the query is performed.

- **Ambiguous mastery of replicated objects.**

If anybody can produce a new version of an object, conflicts may arise which will require resolution. After the conflicts are resolved, the contents of a particular version may now be different for one of the conflicting sites. The results of a query involving that object depend on the time at which the query is performed.

- **Disaster recovery which does not automatically return repositories to internally consistent states.**

- **Disaster recovery that does not return all repositories related by configuration definition to the state they were in at the same point in time.**

Example: the definition of configuration 'gizmo_990314' is time-based, was created at time 990314, and was recovered in the restoration process. gizmo_990314 depends on repository A and B. Repository A is

restored to time 990313 and repository B is restored to time 990314. The contents of gizmo_990314 will now be different than originally intended.

5 High-level Requirements for Constructing a Timesafe Model

Many counter examples have been given to aid in the identification of non-Timesafe models and systems. If a model or system does not have any of the counter-examples, it does not make it Timesafe. So how can a Timesafe model or system be constructed?

Producing a Timesafe model and proving that it has the Timesafe property are beyond the scope of this paper. However, some broad guidelines are provided here. Even if following these guidelines does not result in a Timesafe model, the resulting model will be better than the original model.

A necessary step in producing a Timesafe model is to provide both tracking and versioning for all of the objects that participate in the reproduction of a configuration.

Typically, the following objects are tracked: repositories, namespace (file and directory names), labels, branches, attributes, change-sets, and configuration definitions. Sadly, the only one of these that are typically versioned is files.

As an example, RCS tracks files, labels, and branches. There is no concept of repositories, directories, or configuration definitions. Configuration definitions, when they exist, are generally posted on a web page or sent around via e-mail. A first step in making RCS Timesafe would be to track repositories, directories, and configuration definitions. The next step would be to provide versioning for directories, labels, branches, and configuration definitions.

While these are necessary steps, they are not sufficient to produce a Timesafe model. There is no sure-fire method for constructing a Timesafe model. The only way to show that a candidate model is Timesafe is to transform the model into a theorem which includes the Timesafe property and then prove the resulting theorem.

6 Simple Candidate Timesafe System Example

The example system severely restricts the capabilities of the CM system in order to make a point. Assume a single-user system that will support one and only one project and one and only one stream of development. To make it easy, we'll say that the entire contents of a user's hard disk is the current state of the project.

Each and every write operation will be monitored and written to tape with a timestamp. To view an old configuration, all operations on the PC are halted and

all of the write operations are replayed on a second PC up to the timestamp corresponding to the configuration desired.

At this point, the tape can be put back in the first PC and fast-forwarded to the end of the current record. Changes to the development stream may now resume. None of the counter-examples apply to this system, but it is only a hypothesis that this system is Timesafe.

A product that is very similar to this exists on the market today. It is called Flashback and is sold by Aladdin Systems.

7 Future Work

A formal model that exhibits the Timesafe property and may be used to construct CM systems that are Timesafe is in progress. A proof that the model is Timesafe is also underway. [AccuRev](#) [ACC02] is based on this work and is hypothesized to be Timesafe.

Acknowledgements.

Mario Moreira, Andre van der Hoek, Ted Ede, Josh MacDonald, Steve Harris, and Brad Appleton provided valuable feedback on previous versions of this paper.

References

[ACC00] AccuRev Inc., Brookline, MA, USA. [AccuRev](#) User's Guide, 2002

[EC95] Estublier, J. and Casallas, R. Three dimensional versioning. In J. Estublier, editor, Software Configuration Management: Selected Papers SCM-4 and SCM-5, LNCS 1005, pages 118-135. Springer Verlag 1995.

[Rati96] Rational, Lexington, Mass, USA. ClearCase User's Manual, 1996

[Tich85] Tichy, Walter F., RCS - A System for Version Control, Software Practice and Experience, Vol. 15 No. 7, July 1985.

AccuRev ® and Timesafe ® are registered trademarks of AccuRev Inc.