IBM Rational ClearCase 7.0: Master the Tools That Monitor, Analyze, and Manage Software Configurations

Take a deep dive into extending ClearCase 7.0 to ensure the consistency and reproducibility of your software configurations

Foreword by Lars Bendix, Ph. D., ETP, Lund University, Sweden

Marc Girod **Tatiana Shpichko**

[PACKT] PUBLISHING enterprise professional expertise distilled

Table of Contents

5
MultiSite Concerns

Not everybody is concerned with distributed development. However, it is wise to anticipate the move to a multisite environment, and to ensure that it would not result in a major discontinuity. There are many scenarios, most of which unpredictable from the developers' point of view, in which a **MultiSite** environment may become relevant: outsourcing, acquisitions, mergers, splits. It is important to note that some of them are incompatible with a central point of control. It is thus wise to design for flexibility.

Other concerns may lead to partition one's network, and to design a MultiSite solution, even on the same physical site. We'll see some issues related to performance in the next paragraph.

Imagine at any rate that a need arises suddenly: will it break the original single-site setup? Will this need to be redesigned completely? Or can the users continue working just as they have got used to so far? If the environment remains stable over such a change in development scale, we have a strong indication of a proper original setup, optimized for collaboration.

Our agenda for this chapter is:

- Push or pull?
- Replicas: avoid depending on mastership.
- Global types: yes. Admin vobs: no!
- Shortcomings of ClearCase MultiSite.

Distribution model

There is a common debate concerning the pros and cons of centralized versus distributed configuration management systems. Beyond the fact that it concerns mostly version control systems, this dichotomy actually misses the case of ClearCase, which is often wrongly considered as a representative of the former camp. The incorrect assumption, which adds to the common ignorance of MultiSite, is to tie distribution to a *pull* model (such as in *git* or *Mercurial*: users have to get, i.e. explicitly download the software they are interested in from remote sites), whereas the push model is associated with the centralized systems (such as CVS or *subversion*: users *commit*, i.e. explicitly upload their changes to the central server).

ClearCase MultiSite is a distributed system with an implicit *push* model: remote changes are made available to the user in the background, **asynchronously**.

This is a departure from the common culture of *synchronous* communications, and is probably felt as a loss of *control*, while the gain of *management* is not clearly perceived.

Synchronization, that is, reaching the *same state* in the *same time* (or meeting the same states in the same order), is only ever obtained by *waiting*, that is, adjusting to unavoidable time differences. This is inescapably a loss of efficiency and of scalability. The sophistication of the interactions between the ClearCase clients and servers (especially during builds: advertising the creation of derived objects, promoting local ones for remote winkin) made it clear to the architects of ClearCase that synchronized behavior was not an option.

Now, vobs are databases, the consistency of which is guaranteed by journaled transactions. There is a clear mismatch! Clearly every vob server has its own time, but this time is *local*, and there is no *global time*: the situation is one of relativity, similar to this of Einstein's mechanics. Propagation times are significant compared to CPU speeds, and cannot be ignored.

In fact, if we consider the normal behavior of ClearCase, let's say during a build, we are writing to a vob database, and all these events have to be serialized, that is, we are constantly acquiring database locks, executing transactions, and releasing the locks. This happens between the client and the server, so that the duration of the process includes the round-trip time of the IP packets between the two hosts, and this time is incompressible! It is bound to the speed of the signal and the physical distance (even if in practice, the propagation of the signal is the slowest due to its treatment in switches, routers, and firewalls). The most important fact in the communications is thus not the bandwidth, which has been

increasing steadily, but the *latency*. In fact, measured in clock cycles (and thus in opportunities lost while waiting), the impact of latency is continuously getting more serious.

This is why ClearCase is optimally used within a 0-hop LAN, and any activity involving remote hosts should happen in distinct sites, and thus involve MultiSite. The trade-offs unfortunately involve a loss of functionality over MultiSite, and of MultiSite scalability.

Multitool, and MultiSite Licenses

ClearCase MultiSite is presented as an add-on product, with distinct licenses: one needs a MultiSite license as soon as one accesses a replicated vob. This is of course a major discontinuity and a questionable commercial decision, which may have limited the historical use of ClearCase.

Being a distinct product, it came with its own tool: **multitool**, (which we'll alias as `mt`) for running MultiSite specific commands. Over time, several multitool commands have migrated to cleartool as well (and backwards such as `cd` and `pwd`), as using both is often inconvenient (especially when using a background session to which to pipe commands: then one would need two such sessions, and need to keep the two in sync). There remain however some commands that require the use of multitool. We'll use some below.

Replicas and mastership

The solution to the dilemma is for every site to have, not an identical copy of the vobs, but a **replica** of it: this replica is a workspace which can only be modified locally, and is prevented by the system from diverging from the other replicas. In the absence of a global time, it would be meaningless to say that the data is the same: by the time one would have asserted the state on one site and brought this information to another site for comparison, it would be unavoidable that this state would have changed in either or both replicas. What is important is that the differences do not grow, and that the *past*, up to a recent date, is common.

This is implemented in the concept of **epochs**: ever growing figures that represent high-water marks: the state reached from the point of view of every replica.

We'll review here the commands that may concern users, interested in tracing the origin or the status of changes. We'll have a later *Chapter 11*, *MultiSite Administration*, on administrative aspects.

First, know who you are and who are the partners in communication, both by name and by virtue of the servers:

```
$ ct des -fmt "%[replica_name]p\n" vob:.
wonderland
$ ct lsrep -fmt "%n %[replica_host]p\n"
wonderland beyond.lookingglass.uk
sky alpha.centauri.org
```

Then find out when was the last import and what is the resulting status in terms of the epoch number of the remote replica.

```
$ ct lshis -fmt "%d %o\n%c" -last replica:wonderland
2010-06-11T16:10:40+01 importsync
Imported synchronization information from replica "sky".
Row at import was: sky=377 bigbrother.deleted=565 wonderland=1206
$ mt lsepoch wonderland | grep '(sky)'
 oid:6ba49d09.011621db.8ab1.00:01:93:10:fe:84=378 (sky)
```

If we have connectivity to the remote server, we can compare with its own values:

```
$ albd_list alpha.centauri.org > /dev/null
albd_server addr = 123.1.2.3, port= 371

$ multitool lsepoch -actual sky | grep '(sky)'
 oid:6ba49d09.011621db.8ab1.00:01:93:10:fe:84=378 (sky)
```

This is however unlikely. You have a better chance from your vob server (it must be connected to some ClearCase server, but it may not be directly to the remote vob server, rather an intermediate shipping server). How it is connected, at least from the point of view of shipping its own packets, will show with:

```
$ cd /var/adm/rational/clearcase/config
$ egrep '^[^#]*ROUTE' shipping.conf
```

If this shows nothing, then the connection is direct—to any hosts one tries to ship to, which may only be a subset of the servers hosting replicas of your vob (in case a delegation scheme is in place, using one kind of *hub*). This is, assuming the setup uses the ClearCase **shipping_server** mechanism, which is likely, but not necessary. This setup guarantees a high degree of transparency and orthogonality between the ClearCase servers having the direct connections to one another (the prerequisite is that the above mentioned `albd_list` command succeeds). The following commands will allow you to investigate the history and the schedules:

```
$ ct getlog -host alpha.centauri.org -inquire
vob ClearCase vob_server log
view ClearCase view_server log
sync_import MultiSite import synchronization log (unformatted)
```

```
shipping_receipt Multisite shipping receipt log
shipping MultiSite shipping_server log
...
```

This command shows all the various ClearCase logs that are accessible to you from the remote host (`alpha.centauri.org`). All these logs can be fetched by the same `cleartool getlog` command, and each particular log can be referenced by its name specified in the output of the `-inquire` command above, e.g.:

```
$ ct getlog -host alpha.centauri.org shipping
============================================================
Log Name: shipping Hostname: alpha.centauri.org Date: #######
                                    2010-06-23T10:00:37+01:00
Selection: Last 10 lines of log displayed
------------------------------------------------------------
```

This command returns a list of all the scheduled ClearCase jobs on the remote host:

```
$ ct sched -get -host alpha.centauri.org
```

The concept of **mastership** guarantees that inside every object, there is an area exclusively assigned to the local replica, in which local users may write without consideration for anything that might happen on other sites. The price of considering this would be exorbitant: it would be synchronization. Note that mastership works like a dimension orthogonal to any structure of the software configuration.

```
$ ct des -fmt "%[master]p\n" brtype:main
sky@/vob/apps
```

This preceding example command line shows that the mastership of the main branch type belongs to the `sky` replica of the `/vob/apps` vob.

We wrote inside *every object*, and this works well for elements, because it suits the concept of branches, which may be used for this purpose (the summary will wait until next chapter). But it doesn't work that well for **metadata**: for metadata, we need to decide whether to resort to conventions, or whether the item (usually a type) is stable enough (remember that the concern is only relevant for modifications) to be shared between the replicas.

Avoid depending on mastership

It is possible to change the mastership of objects. As administrator, it is always possible to grant one's mastership to some other replica, and to *send* it. Note that it takes time, and as such, is irreversible. Once you have sent the mastership, you depend on the other site to send it back to you. Or it is also possible to configure the system so that one may request the mastership of certain objects. This does also require time. In both cases, the underlying technology is this of creating a

sync packet, containing all the **oplogs** with numbers below the event one is interested in, to **ship** this packet, and to **import** it at the destination. The `reqmaster` command tries to take some shortcuts to make it faster, but whether it is possible depends on the network configuration, and ultimately is not always safer (one still needs that all the previous oplogs sent previously have been delivered and imported, in order to be able to import *the* oplog...).

We are not trying to paint things black: this is possible, and even works most of the time. And when it doesn't, it is possible to fix. But, it is always better not to depend on it at all! Even more: suppose some amount of mastership transfers happens, one is better to avoid by all means depending on anything related to it, just because it opens the door to surprises and instability: it introduces a global state of the system.

So, let's not change mastership. The remaining danger is unfairness: we may still easily be left with some sites *more equal than others*. One site owns *the* main branch type (do you remember? We insisted that *topologically*, there is only one in the version tree). If the process mandates that the official versions must be found on *main* branches, then one site is structurally in a different position than others! The situation doesn't change an iota by selecting any other type for the *integration* branches.

So, our goal will be to answer the simple question: how to avoid depending on mastership?

The answer is surprisingly simple: read from anywhere, write only to objects (that is, to branches in the case of elements) you know you master, because you created them. Share and publish **in-place**.

In this radical simplicity, this common sense rule defeats 90% of the processes usually presented as best practices, including the infamous UCM.

This rule rejects all concepts of *main* or *integration* branches: never write there, especially for publication!

Branches

Create branches: this is always possible, if you use a type you created. So, in order to shoot yourself in the foot, decide to use a predefined type. As long as you don't, you are free. Branch types should thus not be bound (hardwired) to an intention: keep them a commodity. Stick to this from the beginning, and neither MultiSite nor mastership can force you to change your way.

Do not design a hierarchy: not everybody would be allowed to create the lower branches in elements where they wouldn't exist.
Avoid cascading: prefer branching off main branches (or whatever the root branch type is named). One cannot cascade forever: sooner or later, one meets system limitations. Choose a policy which preserves the topology of the system for the next user: the *bush* model.

Labels

If branch types should be kept free from meaning, it is that semantics are better held by label types. The cause of the fundamental asymmetry in ClearCase between branches and labels is simple: labels can be aliased, whereas branches cannot:

```
$ ct des -s foo@@/AAA foo@@/BBB | sort -u
foo@@/main/3
$ ct des -fmt "%Nl\n" foo@@/main/3
AAA BBB
```

MultiSite brings however again some far reaching constraints. The namespace of types is flat and shared among all replicas. This means that any given name may be reserved by the fastest replica, and preempt the ability of others to use it (there is a mechanism to resolve conflicts afterwards, but nobody wants the discontinuity implied by such occurrences). Furthermore, because of the *asynchrony* of replication, it is possible for this event to occur *retroactively:* every time a packet gets imported, it is the past that changes (note by the way, that this affects time rules in config spec as well as labels). Of course, the probability of such accidents is small, but it is a good idea to use conventions to make it smaller yet. Conventional namespaces, or pre-assigned families of names, play this role.

Another issue related to MultiSite is that while the same label name may be associated across several vobs to the same *baseline* (that is, reference software configuration), sync packets are per vob, and independent from others. The result is that there is no guarantee that a baseline drawn across several vobs will be synchronized atomically or even only consistently in time.

To wrap up the considerations brought by MultiSite, we advise to:

- Avoid *sharing* label types: to put it clearly, avoid `mklbtype -shared`.
- Use only *local* label types, that is, types created locally. This refers both to *applying* them (one cannot apply unshared types mastered on other replicas), and using them directly in config specs (you never know whether you imported *all* the "make label" events). Note that you may (and should) use remote labels as a basis to apply local ones, for integration purposes.

- Acquire conventional sub-domains (maybe only prefixes) and create new types within them (to avoid collisions).

A last piece of advice which only marginally relates to MultiSite: we recommend to lock label types, especially the ones you intend to move (`mklabel -replace`). Lock them, and unlock them as needed, of course: this gives a simple guarantee that they have been stable after a certain time stamp. Locks are not replicated (and it doesn't make much sense to lock a type mastered elsewhere, and which one therefore cannot apply), apart for *obsolete* locks (`lock -obs`). Obsoleting types requires mastership.

Other types

We shall deal with other metadata types in more detail in *Chapter 9*, *Secondary Metadata*, but we need to anticipate this slightly here. Similar considerations as reviewed for label and branch types apply roughly. The option of sharing types is reasonable for stable types, created once for all, and only used later. This results in less types and in smoother collaboration. The trade-offs depend however clearly on usage patterns: for example, attribute types bound to label ones (we'll give examples already in the next chapter) should clearly *not* be shared, and their name should also be taken within conventionally pre-acquired domains.

The case of element types is special and needs to be noted, when custom element type managers are defined and installed. ClearCase MultiSite does nothing whatsoever to replicate the managers (the built-in tools are installed under `/opt/rational/clearcase/lib/mgrs`, which makes it the natural place to put custom additions) to other sites: this responsibility is left entirely to administrators, and this meta-responsibility to the users... But no bother: one will be reminded of the existence of element types managers at the time of failing to import sync packets on a remote site!

```
multitool: Error: Operation "create_element" unavailable for manager ####
                                                                    "bar"
   (Operation pathname was: ##############################################
                    "/opt/rational/clearcase/lib/mgrs/bar/create_element")
multitool: Error: Unable to store new version.
multitool: Error: Unable to replay oplog entry 1800: error detected by ##
                                                      ClearCase subsystem.
1800:
op= mkelem
```

Global types and admin vobs

We saw that types are vob specific, and also that config spec rules based on them actually use their names only. This situation is clearly suboptimal, and may be improved by using **global** types, that is, creating types with the `-global` flag, which allows one to set up hyperlinks of `GlobalDefinition` types. This affects the way types are locked, renamed, and removed (together). This, however, creates a relationship between vobs:

```
# Site 1
$ ct mklbtype -nc -global FOO@/vob/server
$ ct mklbtype -nc FOO@/vob/client
$ ct mkhlink GlobalDefinition \
 lbtype:FOO@/vob/client lbtype:FOO@/vob/server
```

Describing the type in either vob will now show the same information: the one stored on the server side, i.e. about the global type:

```
$ ct des -ahl -all lbtype:FOO
FOO
 Hyperlinks:
 GlobalDefinition <- lbtype:FOO@/vob/client
```

There are, however, two different objects: on the client side, the type is a local copy, which may be checked using the `-local` flag:

```
$ ct des -ahl -all -local -fmt "%[type_scope]p" \
 lbtype:FOO@/vob/client lbtype:FOO@/vob/server
global Hyperlinks:
 GlobalDefinition <- lbtype:FOO@/vob/client
local copy Hyperlinks:
 GlobalDefinition -> lbtype:FOO@/vob/server
```

The MultiSite replication system does nothing particular to support (wasn't originally designed to support) the consistency of this inter-vob relationship. This means that one may have a replica of one vob (`/vob/server`, hosting the global definition), but not of the other (`/vob/client`), and the hyperlink will then show up as "dangling":

```
# Site 2
$ ct des -ahl -all lbtype:FOO@/vob/server
FOO
 Hyperlinks:
 ? <- <object not available>
```

One may get more information (the oid of the link type, link object, vob, and the other end object) with `ct dump`, although this amounts to what was contained in the `ct des -local` report.

As such, this doesn't cause any problem, beyond some confusion among remote users.

This may become annoying only if they decide to clean up the glitch, and use the following command as vob owner on their site:

```
# Site 2
$ ct checkvob -hlink lbtype:FOO
```

The problem being that this will succeed and thus destroy your information! Even when this happens, it leaves a trace in the history, which may be used to communicate and avoid the same problem in the future. To be precise, the information is not destroyed: the hyperlink just gets detached from the `/vob/server` label type (on both sites) and remains attached to the `/vob/client` label type:

```
# Site 1
$ ct des -ahl -all -local -fmt "%[type_scope]p %n\n" \
 lbtype:FOO@/vob/client lbtype:FOO@/vob/server
local copy FOO
 Hyperlinks:
 GlobalDefinition -> lbtype:FOO@/vob/server
global FOO
```

Note that the situation is different, if, on contrary, the non-replicated vob is the one hosting the global definition (`/vob/server`). In that case, the error report is the one as shown here:

```
# Site 2
$ ct des lbtype:BAR
cleartool: Error: Unable to find replica in registry for VOB with #######
                     object ID:"046d37bf.7ac511df.9a89.00:01:84:a9:f4:34"
cleartool: Error: Unable to locate versioned object base with object ####
                          id: "046d37bf.7ac511df.9a89.00:01:84:a9:f4:34".
cleartool: Error: Trouble finding the global definition for local type ##
                                                                    "BAR".
$ ct des -local -ahl -all lbtype:BAR
BAR
 Hyperlinks:
 GlobalDefinition -> <object not available>
```

In this case, the `checkvob` command will fail, as will any `rmhlink` one (and `lstype` will complain and return an error code, which may affect scripts):

```
$ ct checkvob -hlink lbtype:BAR
Unable to determine if the following hyperlink is intact.
GlobalDefinition@129@/vob/client lbtype:BAR@/vob/client -> <object not ##
                                                                  available>
Delete it? [no] yes
cleartool: Error: Unable to find replica in registry for VOB with #######
                     object ID:"d5b61d1f.c73211db.8b18.00:16:35:7f:04:52"
cleartool: Error: Unable to locate versioned object base with object ####
                           id: "d5b61d1f.c73211db.8b18.00:16:35:7f:04:52".
cleartool: Error: Unable to perform operation "remove hyperlink" ########
                                    in replica "sky" of VOB "/vob/client".
cleartool: Error: Master replica of hyperlink is "wonderland".
cleartool: Error: Unable to remove hyperlink ############################
  "GlobalDefinition@129@/vob/client lbtype:BAR@/vob/client -> <object not
                                                                 available>".
```

Now it actually obeys the mastership rules unlike in the previous case we saw.

One might say that this encourages the anti-social behavior of not replicating the source of one's information, but such a strategy would be cumbersome and unsustainable in the long range, if wanting to create global `lbtypes` from different server vobs.

In summary, global types are thus a useful feature, even if a low-level one, and as such supported in extremely minimal ways: even the `cptype` command would not create a client copy of a global type—one must create the `GlobalDefinition` hyperlink explicitly!

Built on top of them, there exists the more ambitious concept of *admin vobs*. Admin vobs build up a hierarchy of vobs, linked by hyperlinks of type `AdminVOB`, and offering shared definitions of global types missing in the lower vobs of the tree as they are being requested (for example, while running `ct mklabel` for a type not yet defined in the current vob). Note that for this mechanism to work, one still has to create the types explicitly as `-global`, and in the admin vob: types created in the client vobs are by default `-ordinary`.

The same issue as previously mentioned for `GlobalDefinition` hyperlinks exists with `AdminVOB` ones, but its consequences may be more severe: it is possible that packets for the replica of a client vob cannot be successfully imported in the absence of the admin vob.

Admin vobs bring though a conceptual problem: they bind all sites to one and the same hierarchy, and thus at least to a common set of vobs. In this respect, they defeat the fairness of the MultiSite design. If we remember the discussion which started this chapter, about the various scenarios along which one might get pushed to use MultiSite, one clearly sees that such uniformity doesn't suit the flexibility requirements considered. It is easy to conceive scenarios in which one would "inherit" admin vob hierarchies from various origins, which would be utterly cumbersome to merge, thereby leading to chaos.

Our recommendation concerning admin vobs is thus, based on MultiSite concerns, to avoid using them at all! On the contrary, global types may be used: some scripting will raise up the functionality to a reasonable and useful level.

Shortcomings of MultiSite

We saw in the beginning of this chapter that there could be advantages in partitioning a large local network into several *sites* (in the MultiSite sense of the word), and we alluded to trade-offs concerning the loss of functionality.

It is now time to clarify these trade-offs, that is, to give reasons not to use MultiSite in spite of the benefits this might bring.

Our reader will not be surprised if we stress here the complete lack of support for derived objects. This is sometimes worked around by naive users through *staging*, that is, by checking them in as *versioned derived objects*, partly for the purpose of replicating them; we already mentioned this in *Chapter 3, Build Auditing and Avoidance*. We believe that this is throwing the baby with the bath water. Versioned derived objects are certainly still derived objects, in that they retain their config records, but they lose what makes config records valuable: that clearmake uses them to implicitly *manage* derived objects—clearmake identifies derived objects by their config records, which allows it to avoid their useless duplication. This shift of responsibility from users, or administrators, to the tool, thus freeing human resources for more creative and more complex tasks, is what potentially makes ClearCase a next generation SCM system.

Furthermore, as we already mentioned, we do not believe there is much value in replicating the data of derived objects: the makefile system should be optimized to make it faster to build locally than to download binaries. At any rate, one shouldn't trust binaries that one cannot build oneself, or rather, that one cannot analyze locally. At the SCM level, such analysis can only be generic, that is, based on comparison. This is the road to enhancement: MultiSite should replicate the config records, as config records. This means that no derived object data would need to be replicated,

but the config records would be available for tool supported comparisons. As we already noted, at the moment one can only emulate this by explicitly checking in the config record:

```
$ ct diffcr foo .@@/main/cr/1/foo/main/mg/1
< Target foo built by mg.user
> Target foo built by mg.user
< Reference Time 2010-06-20T14:48:26+01, this audit started ##############
                                                    2010-06-20T14:48:26+01
> Reference Time 2010-06-20T14:44:32+01, this audit started ##############
                                                    2010-06-20T14:44:32+01
----------------------------
MVFS objects:
----------------------------
< /vob/test/t@@/main/mg/1 <2010-06-20T14:44:45+01>
> /vob/test/t@@/main/cr/1 <2010-06-20T14:43:28+01>
----------------------------
< /vob/test/t/foo@@--06-20T14:48.16247
> /vob/test/t/foo@@/main/mg/1
```

Note that in the above transcript, one object is a derived object and the other a derived object version, actually checked in with the `-cr` option.

Note also that we checked it in in a special branch `cr`, conventionally reserved for this purpose, of its parent directory, which allowed us to retain the exact path name, whereas avoiding to leave a read-only file in the way of our normal builds. To update this element, we use a different view, in order to avoid altering the original DO and test the effects of the `-cr` option:

```
$ ct setview v2
$ ct co -nc -nda foo
$ ct ci -nc -cr -from /view/v1$(pwd)/foo -rm foo
Checked in "foo" version "/main/mg/2".
$ ct des -fmt "%[DO_ref_count]p %[DO_kind]p\n" foo
 unshared
$ ct co -nc -nda foo
$ ct ci -nc -from /view/v1$(pwd)/foo -rm foo
Checked in "foo" version "/main/mg/3".
$ ct des -fmt "%[DO_ref_count]p %[DO_kind]p\n" foo
 shared
$ ct setview v1
$ ct des -fmt "%[DO_ref_count]p %[DO_kind]p\n" foo
1 unshared
```

The conclusion is surprising (to us): neither the checked in config record (not promoted to the *shared* status) nor the plain checked in DO (which, conforming to the documentation, *is* promoted to the shared status) has a **reference count** or affects the reference count of the original object (displayed last).

What this shows is a relatively recent change in ClearCase, which we had not noticed: until version 2003.06, the journaling record for checkin events used to have a reference to the derived object, which was only released by *scrubbing oplogs* (we'll come back to oplogs scrubbing in *Chapter 10*, *Administrative Concerns* and *Chapter 9*, *Secondary Metadata*). It used to be a common cause of problems, because users would want to remove versions of large binaries, which resulted in dangling pointers inside the oplogs and errors while importing sync packets. It seems that the versioned derived object is now a full duplicate of the original DO (with the result that it is being shared or not is rather pointless), which is a quite radical fix.

Versioned derived objects are thus simply not visible to `lsdo` and cannot (probably) be winked in:

```
$ ct lsdo foo
--06-20T15:27 "foo@@--06-20T15:27.16250"
```

On the remote site:

```
$ ct des -fmt "%n, %m, %[DO_ref_count]p, %[DO_kind]p\n" foo
foo@@/main/mg/3, derived object version, , shared
$ ct des -fmt "%n, %m, %[DO_ref_count]p, %[DO_kind]p\n" foo@@/main/mg/1
foo@@/main/mg/1, derived object version, , unshared
$ ct lsdo foo
cleartool: Error: Not a derived object: "foo"
```

These versioned derived objects are thus there only for explicit manipulation. An idea for making them more useful would be (some kind of request for enhancement) to make it possible to promote them to a status related to the real derived objects, that is, to support *lazy identification*: allowing them to work as a bound between sets of derived objects built on different sites.

Summary

We showed that MultiSite is an integral part of ClearCase, and that its design has deep roots into requirements set by the sophistication of build management. It is thus essential to take it into consideration while designing the development process.