

## Case Unclear

Marc Girod  
IONA Technologies PLC

ClearCase, originally a product of Atria, then Pure Atria, Rational, now IBM, has been a market leader in the SCM arena for 15 years. The last years' developments, especially since the adoption of the UCM extension, show however signs that it might reach the end of its life. Its most faithful users are thus looking for a replacement, especially from the Open Source. And they cannot find any.

This is both surprising, given the plethora of competitors, and worrisome. Could it be that what once made the novelty and the unique value of ClearCase has been poorly understood – first of all by IBM?

The inception of UCM marked a major turn in ClearCase history. Until then, the tool had been process agnostic; with it, it became geared toward enforcement of top-down designed practices. The move accompanied a new commercial strategy, from the relatively small world of Unix, to the mass market of Windows. The stereotypical user spent less attention to minute detail, and expected the tool to make more decisions on his behalf, at the click of a button in a Graphical User interface.

This change couldn't happen without jeopardizing the conceptual integrity of the tool, as well as the ability to further develop it in directions which had felt interesting so far. This split the user community in two, users of UCM on the one hand, and of 'Base ClearCase' on the other. A situation which ended up satisfying neither.

Many among the original or long-time ClearCase users, have thus started to look for a replacement tool. This proves however to be a surprisingly difficult task, either on the commercial market, or in the Open Source. There is indeed a profusion of version control, and of process or workflow lifecycle enforcement tools, but nothing which would build upon auditing differences via virtual file systems.

## Open Source

The Open Source looks in itself like the most valuable novel idea of the recent years, so that many would be ready to trade some amount of functionality, in exchange for the benefits expected from the sources being available, and free from the constraints and risks traditionally bound to proprietary solutions.

These concerns are especially natural in the realm of SCM, where a certain *impatience* (claimed by Larry Wall, long before the Agile manifesto, as one of the cardinal virtues of programming) is the law: nobody wants to get stuck waiting for the solution to a problem, which could come exclusively from a vendor. Free software should offer a guarantee of *continuity*, a cure to the uncertainties of business life – as was learnt the hard way with ClearCase.

In the Open Source, the hard name of the day, which merely supplanted the older CVS, is *subversion*. It has a large deployed base, a satisfying record of robustness, shows proud reports of reasonable speed, interfaces ssh, http, and WebDAV, and fixes

the most glaring historical issues with CVS (directory and binary versioning, although commonplace by now, even in modern CVS).

But in terms of SCM functionality proper, it doesn't bring anything significantly new on top of the most traditional version control. Its most characteristic feature is disputable: the grain of versioning set at the directory level...

Maybe what feels exceptionally poor to the stereotypical user of base ClearCase (and thus of *dynamic views*) is that the real work happens in a workspace which is totally invisible to other users of the same repository, and which is only managed *locally*, by the client, without any reporting to the server. In ClearCase, the workspace was a virtual file system, and continuously managed.

This aspect of ClearCase, which was the novel one 15 years ago, is largely discredited nowadays, in the name of performance, but for largely inapt reasons (usually inadequate network configuration, with respect to the intended use).

## Auditing, Build Avoidance, Winkin

There are several justifications for this virtual file system, interrelated and hiding each other as it seems, but also building up a graph of justifications if one cares to order them, be it the one originally intended or not. The *view* acts as a filter, selecting out of all the possible instances of any configuration item potentially offered in the *versioned object base*, one at most, excluding any other. It offers this selection in a *referentially transparent* way, i.e. it decouples the selection of the *item family*, from this of the exact instance suitable for the user.

But ultimately, the justification for both of the above, is to allow to audit the uses of the objects in the view, and to present the results of this auditing in a way suitable for a two-step analysis (people familiar with the *syntagmatic* and *paradigmatic* axes of linguistic analysis will recognize an analogy). *Config records* will first be identified as structured sets of configuration items (families of instances). Records matching at this level will themselves build up families. Within such families, records will differ by the instances of the items representing their own family.

Let's give a example, with a trivial event (on a unix shell command line) involving four file objects (and a shell built-in operator):

```
cat foo bar > zoo
```

The first object, `cat`, is a standard unix binary which concatenates its arguments (two here, `foo`, and `bar`), expected to be files, producing a standard output which we redirect here into a third data file, `zoo`. This last object being the only one modified (actually produced), is the one to which the *config record* will be attached. The textual representation of this event plays the role I equated above to the syntagmatic axis of linguistic, and allows us to identify a family of records. Note that it is thanks to the use of the *view* virtual file system that this representation may be stable across different similar events, run independently by different users in their respective workspaces. The paradigmatic axis, allowing to identify distinct instances of records

of such a zoo production, is built along the various versions potentially available of the four different file objects.

What is remarkable in this analysis, is the conservation at the level of *derived objects*, of the two-step identification characteristic of *configuration items*.

It is remarkable for two reasons. The first is that there are potentially much more derived objects than source elements in typical software configuration candidates, and that the derived objects are typically more meaningful and more directly valuable to the end users. The second reason is that the domain of source elements is *flat*, from an SCM, generic point of view, agnostic to the specificities of file formats, whereas the domain of derived objects is inherently structured in a generic way: through *dependencies*. Nothing prevents that one or more among our three initial (zoo didn't need to exist prior to running the command) file objects above, could have been itself a derived object. Producing it would then constitute a precondition to the event we recorded, and the records could be chained in a directed graph.

ClearCase does more than allowing the comparison of more or less arbitrary events. Performing such an analysis as the one we just depicted, before executing the command, it can detect that a similar event has already occurred, preempt the execution, and provide the result previously obtained, avoiding thus to produce a duplicate of it.

It is not clear whether the original developers were aware of the transmutation of lead into gold which they had achieved, or whether they themselves thought merely of having saved some disk space and some CPU time –an unfortunately frequent yet hasty analysis.

My point is thus the following: with winkin, what ClearCase potentially offered was the unique identification of derived objects in a way that made them eligible as Configuration Items. An SCM system based upon *derived objects* instead of upon *source elements* would be useful to end-users (who see first the executable, then the rpm, and only later, if they really insist, the sources as packaged...), not only to *developers*.

Of course, base ClearCase never fully developed according to these lines. As for now, it still lacks (I cannot give details here for every item. This would be beyond the scope of this paper) :

- Replication of config records over MultiSite, in order to make it possible to compare derived objects produced in different sites.
- Ability to group derived objects together, possibly ignoring some irrelevant aspects, and designing preferences between equivalents, to solve problems with spurious differences.
- Support for lots of sites, e.g. with hierarchical replication, in replacements for *snapshot views*. Snapshot views were a hasty reply to the challenge posed by the existence of the first laptops.
- Winkin for Java.
- Winkin independent from builds.
- Open journaling format to allow to keep in sync, databases of different tools.

This list is of course only indicative. Many optimizations could be added; some practices could be developed, pertaining to the paradigm depicted here, and which could benefit from direct support.

## **Conclusion**

There should be opportunities to design a third generation SCM tool (after the first generation based on lines of text files, and the second one, based on 'source' artefacts). ClearCase may be seen as the prototype for such a tool, a prototype which aborted too early, and never fulfilled promises that were maybe never made, yet were heard by a community among its users, as early as ten years ago. It is as for now still open whether this tool should be developed in the Open Source, or as a commercial product.