A proposal to improve performance of ATL collections

Jesús Sánchez Cuadrado

University of Murcia
jesusc@um.es
http://www.modelum.es/personal/jesus

Abstract. This paper presents a proposal to replace the current implementation of collections in the ATL-VM (based on the Java collection library) with a new implementation that supports inmutable collections natively.
A first prototype using the Clojure collection library has been implemented, and some performance results from benchmarks has been gathered. They show that the proposal has the potential to improve performance significantly.

1 Introduction

Model transformations are at the heart of Model Driven Engineering (MDE). ATL is widely used to develop model transformations, providing a good performance in most cases. However, as model transformation definitions are more complex and the size of the models is significant performance issues arises. Notably, navigating large collections could become a bottleneck.

In the context of ATL, some performance issues in the navigation of collections has been reported [2]. The problem stems from the way OCL collections are implemented in the ATL Virtual Machine (ATL-VM).

This paper presents an alternative approach to implement OCL collections in the ATL-VM, which is based on inmutable collections. We have implemented a small prototype in order to asses the potential advantages of the new implementation. We will show some performance benchmarks.

The paper is organized as follows. Next section introduces the performance issues of the ATL-VM. Section 3 presents our proposal, and Section 4 shows some benchmark results. Finally, Section 5 gives the conclusions.

2 Problem description

Current implementation of the ATL-VM provides several native collection datatypes. They correspond to the ones of the OCL specification (Sequence, Bag, Set, OrderedSet) plus a datatype for associative tables (Map).

Collection operations are inmutable, that is, they never change the original collection. Basic operations like `includes`, `including`, `union` are implemented

directly by the ATL-VM. On the contrary, operations that takes an iterator expression are implemented by the ATL compiler. For instance, the `select` operation is compiled as follows (in pseudo-code):

```
result = new Sequence
iterate o <- collection
  v = evaluate iterator-expr(o)
  if v
    result = result.including(o)
  end
enditerate
```

The performance problem arises when the actual implementation of the update operations is considered. The ATL-VM rely on the Java collection library, whose update operations are mutable, to implement inmutable update operations. This mismatch makes every update operation do a complete copy of the original collection for each call, and then modify the copy.

In the case of the `select` operation shown above, the `result` collection is copied each time an element is selected.

A solution, pointed out in [2], is to add mutable versions of update operations to the VM, and let the compiler decide when it is safe to use them. For instance, there is an important performance improvement if a mutable version of the *including* operation is used to implement the `select` operation. However, the drawback of this approach is that the ATL compiler must be changed. Some of the changes would be straightforward, but there are cases that require specialized analysis. Moreover, existing transformation definitions should be recompiled.

3 Proposal

To tackle the issues presented in the previous section, we have replaced the Java collections library (in the ATL-VM) with a library implementing inmutable collections. In particular, we have reused the implementation provided by Clojure [1], a functional language for the JVM. It efficiently deals with update operations for inmutable collections (see [4][3] for more information).

We propose the following mapping between OCL/ATL datatypes and Clojure classes:

ATL	Clojure
Sequence	PersistentVector
Bag	PersistentList
Set	PersistentHashSet
OrderedSet	PersistentTreeSet
Map	PersistentHashMap

The mapping is simple, since Clojure collection library provides datatypes similar to that of OCL. There is however an issue with sequences. At first sight, a list would be the proper mapping, but the main update operation for lists in Clojure

(like in Lisp) is `cons`, which returns a new list where the new element is the first one and the original list is the rest. This is why OCL Sequences are mapped to Clojure vectors, which add elements to the end of the collection. This has an slight impact in performance, as will be shown in next section.

Implementing a first prototype that is able to execute basic operations has been relatively straightforward. Two Java classes were modified, `OclType` and `ExecEnv`. The former establish a mapping between OCL types and Java classes implementing them, while the latter contains the implementation of the operations of each type. These were the only two classes that had to be modified.

However, there were two issues that hindered the implementation:

1. *Datatype instantiation.* In Clojure, an empty collection is actually a constant. However, the mapping between OCL types and Java classes requires setting the name of Java class that will be instantiated each time. A wrapper class for empty collections has been created, but this has required duplicating the operations code in `ExecEnv`.
2. *Operation mapping.* There is not a one-to-one mapping between OCL collection operations and Clojure operations. For instance, in Clojure collection concatenation (`union` in OCL) is not implemented in Java code but in Clojure's. Thus, only a part of the Clojure library can be reused, the rest has to be implemented from scratch in Java.

4 Benchmarks

This section presents a couple of results from the performance benchmarks we have carried out. While they are simple (more or less synthetic) benchmarks, they confirm our insight that implementing true inmutable collections in the ATL-VM will improve the performance of ATL transformations.

Each benchmark measures performance of the ATL-VM (EMF-VM) without any modification, against a modified version which uses mutable operations (it is unsafe, not usable in a real context), and against the a modified VM that uses Clojure collections.

4.1 Sequence

The benchmark for the `including` operation consists of iterating over an OCL sequence of n elements, adding the current element to another sequence in each iteration step. The execution time for different input sizes is shown in Figure 1.

As can be seen, in the the current version of the VM `including` does not scale well (i.e. time doesn't grow linearly with respect to the model size). On the contrary, execution time using the other three versions grows linearly. Interestingly, performance of *PersistentList* is comparable to using a mutable list. Anyway, as explained, this is not a proper choice to implement OCL sequences because of the semantics of "cons". Performance of *PersistentVector* is slightly worse, but still better than the current implementation.

4.2 Map

The performance benchmark for Map has consisted on implementing a simple matching algorithm. It takes as input a model containing the same number of elements A and B, randomly ordered. For each element A, the corresponding matching element B is found (comparing the value of an attribute). The source model ensures that there is one and only one matching element B for each element A. At the beginning of the transformation all matches are computed and stored in map. Applying a similar algorithm is needed in some model weaving problems.

Again, the current version of the VM does not scale well because of the need to copy the whole map for each update. Both alternative versions improve performance. Moreover the performance of *PersistentMap* is comparable to the unsafe version.

5 Conclusion

In the paper, we have presented a proposal to improve performance of collections in ATL. It has been compared to other two alternatives (current version of the ATL-VM and modifying the ATL compiler to support mutable operations). It does not require any change to the ATL compiler, and therefore will not require re-compiling existing ATL programs, because it is only a change in the VM.

The initial results shows that our prototype, based on the Clojure collection library, outperforms the current ATL-VM. This suggests that it is worth the effort of implementing a complete solution based on true inmutable collections (probably implementing a specific collection library for the ATL-VM).

References

1. Clojure programming language. http://clojure.org/.
2. J. S. Cuadrado, F. Jouault, J. Garcia-Molina, and J. Bèzivin. Optimization patterns for OCL-based model transformations. In *Proceedings of the 8th OCL Workshop*, 2008.
3. K. Krukow. Clojure hash-map implementation. http://blog.higher-order.net/2009/09/08/understanding-clojures-persistenthashmap-deftwice/.
4. K. Krukow. Clojure vector implementation. http://blog.higher-order.net/2009/02/01/understanding-clojures-persistentvector-implementation.

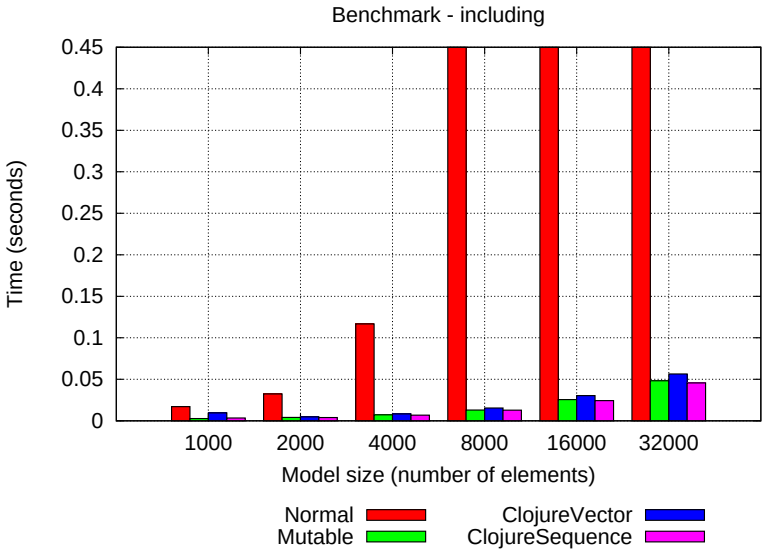

Fig. 1: Results of the *including* benchmark for sequences. For model sizes of 16000 and 32000, the red bar is out of the chart (1.7648 and 9.2986 seconds).

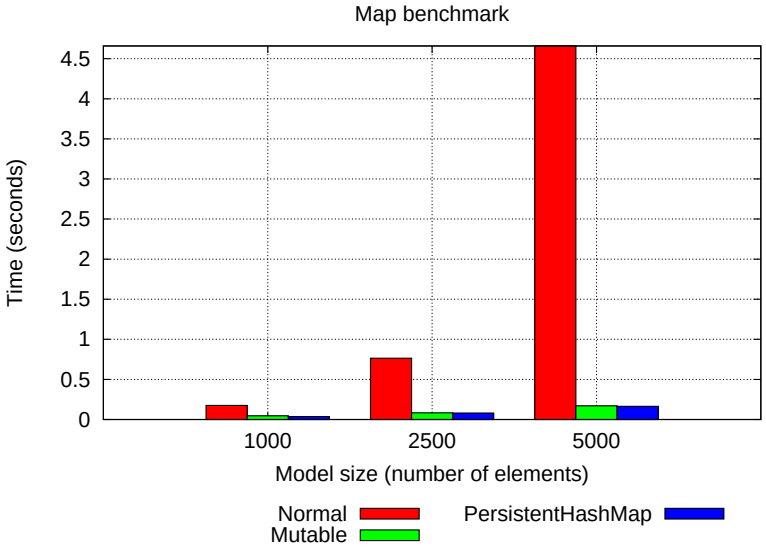

Fig. 2: Results of the *matching* benchmark.