EMF Compare

Summary :

Table des mises à jour			
Version	**Date**	**Auteur(s)**	**Mises à jour**
v1.0	06/10/11	Laurent Goubet	Initial Version

Sommaire

Ce document est la propriété de Obeo et ne saurait être communiqué à un tiers sans son autorisation.

1 - Scope

This document aims at describing the current state of the EMF Compare performance and scalability, from the points of view of both memory and time management. We will then strive to describe the potential improvements that could be made to the comparison process in order to either reduce the memory footprint of EMF Compare or reduce the overall comparison time.

Ce document est la propriété de Obeo et ne saurait être communiqué à un tiers sans son autorisation.

2 - Model Structure

The models used to perform the different performance tests are UML models randomly generated according to a set of specifications for packages, classes, operations, state machines... They are fragmented so that each package, sub-package and state machine is extracted into its own fragment.

Each of the test models are copy/pasted and their copy randomly modified so that it presents a number of changes.

The models will henceforth be referred to according to their sizes : one of « small », « nominal » or « large ». Following is the count of element for each of these three sizes.

	Small	Nominal	Large
Fragments	99	399	947
Packages	97	389	880
Classes	140	578	2169
Primitive Types	581	5370	17152
Data Types	599	5781	18637
State Machines	55	209	1311
States	202	765	10156
Dependencies	235	2522	8681
Transitions	798	3106	49805
Operations	1183	5903	46029

3 - Methodology

The methodology followed to retrieve time and memory information is the same for each of the three model sizes. All tests have been carried with the same machine, with the same environment.

3.1 - Environment

- Windows 7 home edition 64 bits
- JDK 1.5 update 22 64 bits
- Yourkit Java profiler 8.0
- Eclipse 3.7.0 64 bits
- EMF Compare 1.3.0 M3

3.2 - Procedure

- Launch Yourkit Java Profiler
- Launch a new Eclipse instance under monitoring of the Yourkit Java profiler
- Make sure that EMF Compare is using the XMI ID for the comparison
- Select both versions of the model and use the action Compare With > Each Other
- Stop the Profiling through Yourkit when the comparison process has ended

The comparisons launched for these tests are two-way comparisons. In the context of a Version Control System, the comparisons would be three-way, inducing slightly more important times and memory consumptions.

The heap status is displayed over time by Yourkit; what we record in these test is the maximum value that the "used" heap reached during the comparison process.

The comparison times recorded here are the "CPU time" as displayed by Yourkit between the time it has been launched (before we hit the "Compare With > Each Other" action) and the time the comparison ended (when the compare editor is opened and has focus).

The matching, differencing and model loading times we consider are the CPU times reported by Yourkit for each of the three distinct processes.

4 - Results

4.1 - Time and memory management

- Small model : comparison time : 6 seconds. Maximum heap : 555 Mo.
- Nominal model : comparison time : 22 seconds. Maximum heap : 1019 Mo.
- Large model : comparison time : 125 seconds. Maximum heap : 2100 Mo.

4.2 - Analysis

- Small : 52% loading resources, 19% matching, 29% diffing
- Nominal : 44% loading resources, 17% matching, 39% diffing
- Large : 24% loading resources, 28% matching, 48% diffing
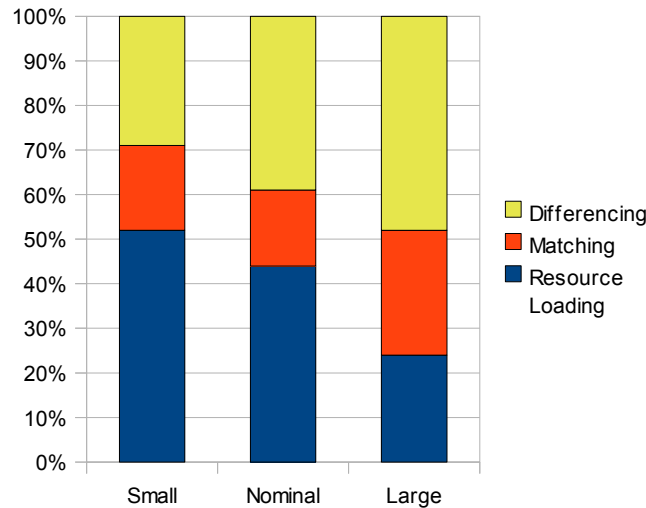
The current limitation of EMF Compare lies in its memory management, as it needs to fully load the models to compare, whether fragmented or not. Furthermore, EMF Compare loads two instances of the compare model (left and right) or up to three instances when comparing with a repository (left, right, and common ancestor of the two).

This current architecture means that not only do we fully load the *n* instances of the model, but we iterate over the whole thing at least twice during the matching and differencing process, contributing to the overall increase of the comparison time.

Moreover, comparing the whole model as EMF Compare currently does mean that the comparison result is scarcely exploitable : the comparison editor will be slow, the number of differences displayed at once too high ... and the sheer number of elements represented in the editor (the whole tree representation of the model, twice at least (left and right)) will make the editor sluggish.

Three potential axes of improvement can be explored :

1. **Thread the model loading**

   This will only serve to reduce the model loading time, and will do nothing for the memory consumption. However, this will allow us to fully use the CPU's power for multi-core machines, and will reduce the loading time by a rate depending on the number of available cores.

2. **Fragment the comparison process**

   The results of a comparison (matching and differencing) are models mimicking the comparison input's hierarchy. However, neither the match nor the diff are fragmented : they are both saved in a

Ce document est la propriété de Obeo et ne saurait être communiqué à un tiers sans son autorisation.

single resource, created on-the-fly however the fragmentation of the input models.

Fragmenting the matching and differencing process so that they reflect their input would be a way to reduce the overall memory consumed during the comparison process, allowing EMF Compare to unload the fragments it has already iterated over.

This axis would decrease the memory footprint of EMF Compare during the comparison process; however it would do nothing for the comparison time.

3. Reduce the scope of the comparison

The current architecture makes EMF Compare consider models as a whole, forcing us to load the whole models in memory every time we need to compare them. However, out of a hundred fragments, there might have only be three that really changed, and thus we *could* have obtained the desired result by only loading -and comparing- those three fragments; dramatically improving both the comparison time (we would only need to iterate on a small portion of the input models) and memory consumption (only three thousandth of the model need be loaded in memory).

By making use of the logical models, it should be possible to determine which of the *n* fragments really changed, thus enabling a meaningful partial comparison and merging.

This particular improvement axis would help the comparison time, reduce the overall memory footprint of EMF Compare by orders of magnitude **and** solve the lack of reactivity of the EMF Compare editor when large models are being compared and displayed.