Evaluating a Tree Diff Algorithm for Use in Modelica Tools

Martin Sjölund[1]

[1]Department of Computer Science, Linköping University, Sweden, martin.sjolund@liu.se

Abstract

Modelica tools change the formatting of the source code when performing operations in the graphical user interface. These unintended changes cause problems for source code management since a code review would mostly go through changes that do not change any semantics. The intent of this work is to present a workflow where edits from an interactive graphical user interface does not contain these unintended changes when using the source code management system.

A diff tool that can merge two Modelica files and produce a merged copy is presented and evaluated. The diff algorithm works by comparing syntax subtrees of Modelica code and having some domain knowledge about which subtrees belong together, speeding up the diff algorithm. The result is a merged file by taking formatting of the first file and the semantics from the second file. This works very well for smaller changes (a single edit) and scales with file size (making the user interface faster for smaller files).

To test the algorithm on a larger set of changes, a conversion script was applied to a set of libraries. The effect of applying a conversion script is a set of automated edit operations, which cause unintended changes in the formatting of the source code. The diff algorithm with applied to these changes and the performance was analyzed.

The results are very promising especially for Modelica libraries that are split into multiple files rather than a large single file. Having a single large file takes slightly longer to process and produces additional unintended formatting changes compared to a library developed as a set of smaller files.

Keywords: Modelica, diff, file comparison, conversion script, interactive user interface

1 Introduction

An important problem to handle for any software development is the management of source code. It is important to be able to see what changes are introduced in every new version of your software and one common way of showing this is with a simple text diff. However, most text diff tools are very limited in what they can do and introducing any new whitespace in a line will often flag the entire line. If you introduce a line break or move part of a line around, you can often forget about seeing what you actually changed in a diff. Most text diff algorithms will use something similar to Myers (1986) algorithm, performing the diff on lines of code since its $O(ND)$ scaling performs poorly on when words are used items instead of lines. There are algorithms available to improve the performance of Myers (1986), such as diff-match-patch (Google 2019), but they have most of the same drawbacks as the original.

When working with source code management systems such as git, there are best practices to make the history easier to read. Some of the following best practices make it easier for Myers (1986) algorithm to work since there are fewer changes to consider:

- every developer must use the same width for indentation (tabular characters and/or spaces).
- include no trailing whitespace at the ends of line.
- use text line endings (to avoid CR/LF issues).
- don't commit generated files such as binaries.
- commit only related work together.

Tools that compare source code of languages do exist, for example Diff/TS (Hashimoto and Mori 2008) and GumTree, but these need domain knowledge about the language to perform a diff on and need to be tuned to produce good results (Matsumoto, Higo, and Kusumoto 2019).

How this problem relates to the Modelica language (Modelica Association 2021) is probably apparent to anyone who has collaborated on a Modelica project. Modelica tools are graphical user interfaces where you move components around, change some value, or drag and drop components. This means that the source code needs to be added, removed, or updated. If these operations are performed internally to the Modelica tools, the internal representation needs to be unparsed in order to write these changes to file[1]. This unparsing will be slightly different in all Modelica tools, and some tools may be smart enough to at least only update the part of the class that changed. In this work, OpenModelica (Fritzson et al. 2020) and its graphical user interface OMEdit will be used to evaluate the work. OpenModelica will update the entire class and it will move something around since the internal representation for example only allows comments in certain places (compare Listings 1 and 2).

[1]The edits could also be performed directly on a concrete syntax tree, but this would require a full redesign of how Modelica code is handled in Modelica tools and would be much harder to implement.

```
model M
  MyModel m(
    // ??? Works if we do this
    x = 0,
    // Disable heat port
    y = false,
    // Forces the model into mode 2
    z = 200
  );
end M;
```

Listing 1. Example listing with comment.

```
model M
  MyModel m(x = 0, y = false, z = 200);
  // ??? Works if we do this
  // Disable heat port
  // Forces the model into mode 2
end M;
```

Listing 2. The example in Listing 1 unparsed by OpenModelica moves the comment.

What this paper tries to answer is how to perform a diff of unparsed Modelica code in a way that would produce small text diffs in a source code management system.

In order to be able to evaluate the proposed solution, a large enough test set is required. The Modelica Language Specification 3.4 (Modelica Association 2017) standardized the concept of conversion scripts. Using conversion scripts it is possible to for example rename a component in a class of a library and to automatically upgrade a model using the old version of the library with the new names. This potentially changes every single line of code in a library that is converted in this way. The Modelica Standard Library version 4.0.0[2] has a conversion script from major version 3 and there are many libraries still using one of these versions of the standard library. The diff algorithm can be evaluated by applying the conversion script to these libraries since a large set of real-world edit operations will be produced.

2 Method

The Modelica diff algorithm was created over several iterations. The current implementation will be presented.

Then the Modelica diff algorithm will be evaluated based on how it performs for common operations in the OMEdit GUI. Both quality and performance will be considered. Operations in OMEdit are mostly single edits followed by running the Modelica diff algorithm. This is what the algorithm was designed for.

Recently, support for conversion scripts was added to OpenModelica. This works by converting the internal representation of a loaded library, but it is also possible to create a script to write the changes to file, and merge these files with the original ones. Conversion scripts can potentially change every line of code, so this will serve as a stress test for the Modelica diff algorithm.

3 The diff algorithm

The basis of our Modelica diff algorithm is the classical Myers (1986) text diff algorithm with some additional optimizations based on ideas by Butler (2009) and Google (2019).

The ideas used from Butler (2009) and Google (2019) stem from the fact that you can easily check if the two compared sequences have a common prefix or suffix. Myers (1986) scales with the sum of the sizes of the two inputs, and if there are only changes local to a part of the file trimming away a common prefix and/or suffix will significantly improve performance of the algorithm. However, these optimizations do not improve performance if there are changes all over the file. In our algorithm, these checks also ignore whitespace (so unparsed text will be considered equal).[3]

The implementation in OpenModelica is a generic implementation because our diff algorithm is not based on text (lexer tokens, etc).[4] Instead of using a text diff, the full algorithm is performing a diff on concrete syntax trees. In order to start the diff algorithm, the inputs of both files go through a lexer and a hand-written recursive-descent parser which both preserve comments and whitespace. The output of the parser is a tree where nodes also contain whitespace and comments belonging to this subtree of the code. Where the Modelica grammar has nodes that can be given a name (such as classes, elements, or named modifications), this node is labelled by the parser. Modelica models are not allowed to define the same name twice in the same scope, making these labels unique.[5] Tichy (1984) considers blocks of text as units and moved them together instead of as in Myers (1986) where each modified unit of text that is moved is considered one move. However, as the algroithm presented below is a tree diff algorithm it is more similar to for example Matsumoto, Higo, and Kusumoto (2019) than Tichy (1984), as it works on units already divided into blocks using domain knowledge of Modelica and compares these instead of trying to create blocks from text.

The tree diff implemented in OpenModelica[6] works recursively for each node where the diff algorithm runs on the sequence of nodes in each subtree that is not equal to

[2] `https://github.com/modelica/ModelicaStandardLibrary`

[3] Due to the internal representation and unparsing in OpenModelica, parentheses are also considered whitespace. This is done to not have a diff when the unparsing adds or removes unnecessary parentheses.

[4] `https://github.com/OpenModelica/OpenModelica/blob/master/OMCompiler/Compiler/Util/DiffAlgorithm.mo`

[5] The labels are only unique for valid Modelica models. The quality of the diff is decreased when performed on invalid models.

[6] `https://github.com/OpenModelica/OpenModelica/blob/master/OMCompiler/Compiler/Parsers/SimpleModelicaParser.mo`

the corresponding subtree in the other sequence. When there is only 1 change in the entire sequence, most of the file will be kept the same without any possibility of adding whitespace in the wrong place. However, the algorithm is no longer scaling as $O(ND)$ since we may potentially perform this operation at each depth of the tree.

The tree diff algorithm has additional optimizations performed on the result returned by the generic diff algorithm:

1. Move operations are detected by looking only for nodes with the same label in order to improve performance. If there is a node deleted and added with identical contents, the merged result contains the text of sequence A in the position that it was moved to in sequence B. This is also performed for comments, trying to not move them.

2. Changes to whitespace are ignored unless they are needed to separate two tokens in the merged text.

3. Indentation is preserved to match the previous line in the original.

4. Nodes that are not equal to some node in the other sequence are compared to each other (this is a recursive algorithm). If the labels match, those nodes are compared to each other. If there is only one node remaining, it may have been renamed and is compared. The algorithm does not consider multiple renamed nodes at the same time and will fallback to resetting the formatting of the nodes in this case.

4 Evaluation

The diff algorithm has been extensively tested and improved since it was introduced in OpenModelica back in 2015. The first version used Myers (1986) as token-based diff, but this took several hours to perform a diff on 300 kB large files due to many whitespace changes[7].

There are two aspects of the evaluation: quality and performance. This section presents results for both of these metrics. The computer used for the performance measurements uses an AMD Ryzen 5900X CPU and has 32 GB RAM.

Listings 1 and 2 merge into the same content as Listing 1 in 633 µs. Some of the largest example models in the standard library with a diagram where components can be moved around are `ComparisonPullInStroke` and `BatchPlant_StandardWater`[8].

For `BatchPlant_StandardWater`, one component was moved in the OMEdit GUI as shown in Figure 1, causing an update to a component and two connections.

[7]https://github.com/OpenModelica/OpenModelica/commit/dc2d3ef0465e

[8]Sizes were calculated based on OpenModelica's unparsing of the example and not the size of the file it is stored in. `BatchPlant_StandardWater` is not the only class stored in the file it is defined in.

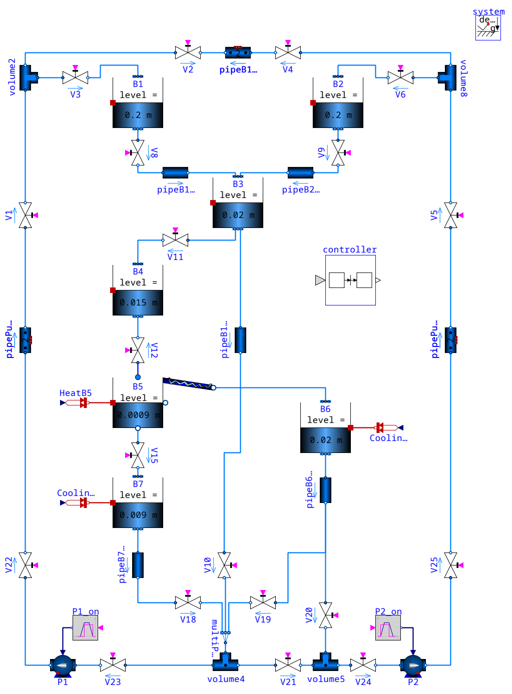


Figure 1. The diagram for BatchPlant_StandardWater. Component B2 is moved slightly down to the right in the example described in the text.

For performance reasons, OMEdit performed all 3 updates first and then called the Modelica diff algorithm, resulting in Listing 3. This causes the connections to be indented slightly incorrectly due to two adjacent nodes being updated at the same time. Note that connections are slightly more complicated to handle than components since they do not have a name and the diff algorithm does not detect that these were existing connections that were updated. When OMEdit modifies the component, the visible and rotation modifiers are added because the compiler does not keep track of which values are defaults and which had explicit modifiers. This causes the formatting of those annotation to be affected as well. The rest of the file remains the same as before, which makes the diff readable. The diff algorithm takes 0.55 s to run, which explains why OMEdit tries to perform a few edits at the same time. To illustrate that the updates look nicer, consider Listing 4 where only one connection was updated. The time it takes is slightly lower than performing 3 diffs at the same time (0.37 s), but not 3 times lower since the whole file needs to be parsed and the common prefix/suffix optimization saves some time here.

For `ComparisonPullInStroke`, a connection was added to see how the Modelica diff algorithm handles added connections. The new connection was added and changed to red color and the updated diagram can be seen in Figure 2. Listing 5 shows that the connection is added at the correct indentation level, with the same formatting as OpenModelica's default for added annotations. The diff algorithm runs faster (0.08 s) than for the larger `BatchPlant_StandardWater` file.

```
@@ -292,8 +292,7 @@
         diameter=0.011,
         height=0)},
       stiffCharacteristicForEmptyPort=false)
-                        annotation (Placement(transformation(extent={{50,180},
-              {90,220}})));
+                        annotation (Placement(visible = true, transformation(extent = {{98,
↪  124}, {138, 164}}, rotation = 0)));
     Modelica.Fluid.Examples.AST_BatchPlant.BaseClasses.TankWithTopPorts B3(
       redeclare package Medium = BatchMedium,
       height=0.5,
@@ -546,10 +545,10 @@
         points={{-130,220},{-120,220},{-120,230},{-90,230},{-90,221}}, color={0,127,255}));
     connect(volume8.port_3, V6.port_a) annotation (Line(
         points={{150,220},{130,220}}, color={0,127,255}));
-    connect(V6.port_b, B2.topPorts[1]) annotation (Line(
-        points={{110,220},{100,220},{100,230},{70,230},{70,221}}, color={0,127,255}));
-    connect(B2.ports[1], V9.port_a) annotation (Line(
-        points={{70,179},{70,175},{70,175},{70,170}}, color={0,127,255}));
+  connect(V6.port_b, B2.topPorts[1]) annotation(
+      Line(points = {{110, 220}, {118, 220}, {118, 165}}, color = {0, 127, 255}));
+  connect(B2.ports[1], V9.port_a) annotation(
+      Line(points = {{118, 123}, {118, 174.5}, {70, 174.5}, {70, 170}}, color = {0, 127, 255}));
     connect(V9.port_b, pipeB2B3.port_a) annotation (Line(
         points={{70,150},{70,144},{50,144}}, color={0,127,255}));
     connect(pipeB2B3.port_b, B3.topPorts[2]) annotation (Line(
```

Listing 3. `BatchPlant_StandardWater` where component B2 was moved, the connections to it updated, and the Modelica diff merged the changes. The text is a regular unified text diff of the files (since the whole file is 100 kB large).

```
@@ -547,7 +547,7 @@
     connect(volume8.port_3, V6.port_a) annotation (Line(
         points={{150,220},{130,220}}, color={0,127,255}));
     connect(V6.port_b, B2.topPorts[1]) annotation (Line(
-        points={{110,220},{100,220},{100,230},{70,230},{70,221}}, color={0,127,255}));
+        points={{110,220},{118, 220},{118, 165}}, color={0,127,255}));
     connect(B2.ports[1], V9.port_a) annotation (Line(
         points={{70,179},{70,175},{70,175},{70,170}}, color={0,127,255}));
     connect(V9.port_b, pipeB2B3.port_a) annotation (Line(
```

Listing 4. `BatchPlant_StandardWater` where only one connection was updated and the Modelica diff merged the changes. The text is a regular unified text diff of the files (since the whole file is 100 kB large). Note that there are fewer whitespace changes than in Listing 3 (OMEdit also removed some lines when re-routing the connection).

```
@@ -407,6 +407,8 @@
                                                  color={0,0,255}));
   connect(simpleSolenoid.flange, simpleLoad.flange_a)
     annotation (Line(points={{0,-50},{20,-50}}, color={0,127,0}));
+  connect(simpleLoad.flange_b, advancedLoad.flange_b) annotation(
+    Line(points = {{40, -50}, {66, -50}, {66, 30}, {40, 30}}, color = {170, 0, 0}));
   annotation (experiment(StopTime=0.05, Tolerance=1e-007), Documentation(
         info="<html>
 <p>
```

Listing 5. `ComparisonPullInStroke` where a connection was added, its color changed, and the Modelica diff merged the changes. The text is a regular unified text diff of the files (since the whole file is 30 kB large).

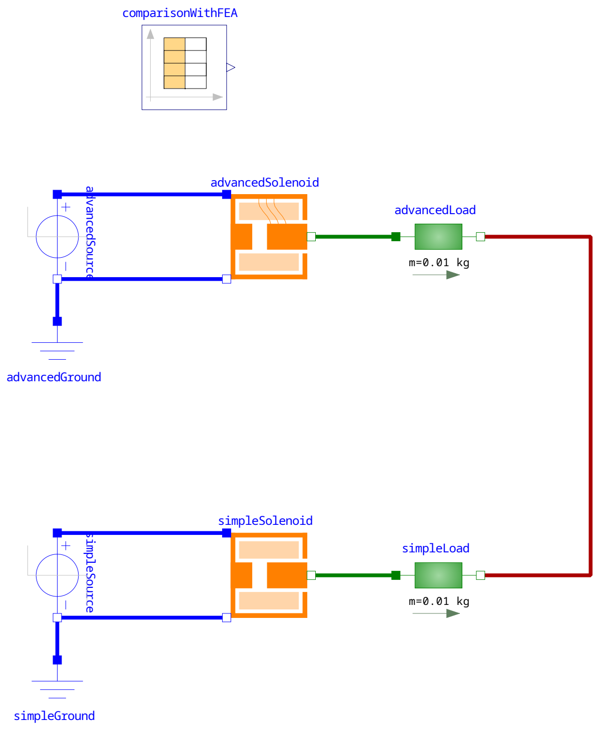


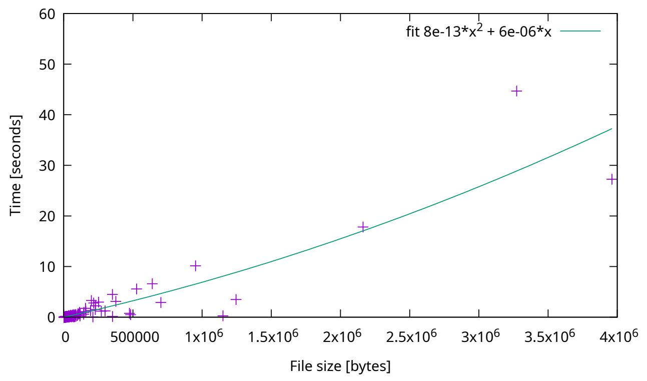
Figure 2. The diagram for `ComparisonPullInStroke`. The connection in red color was added and the examples try to merge the text before this connection with the text after the update.

The diff algorithm was also tested on libraries that had the MSL 4.0.0 conversion script applied to them.[9]. Not all files will have been updated, and the diff algorithm failed on a few files. Figures 3a and 3b show a curve fitting on the sets of files where there was a diff and where there was no diff detected after merging, comparing size to how long the diff operation takes. For smaller files, the scaling is linear (around 8x higher for files with a diff than those without). As files grow larger, they seem to have quadratic scaling. Note that the number of edits are not known for these files. If you consider all files in the same set (Figure 3c), the scaling is quadratic as the files where the content changed will dominate the overall times.

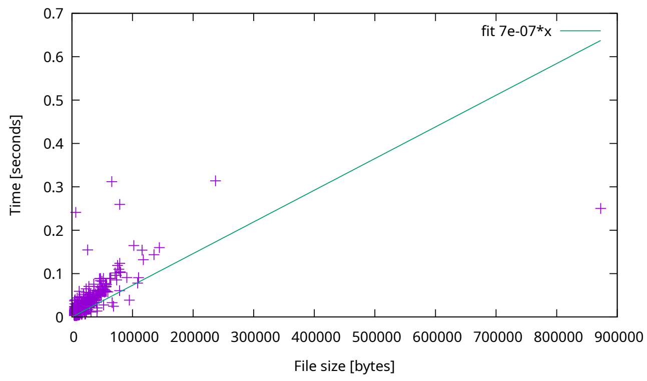
The conversion script test has also been grouped by the library that was converted and a summary can be seen in Table 1. One library that has few modified files, runs fast, and produces a very good diff[10] is the BioChem library. This is because the BioChem library mostly uses its own units based on the SI units from the standard library. The Buildings library is a much larger library, with many changed files. Since Buildings is split into over 3000 files, it does not use much memory although it takes almost a minute to complete. The quality of the diffs in Buildings

[9]A copy of the text diff compared to the original is available at `https://gist.github.com/sjoelund/b7574f7aaf052500b0835f14e4b25d95`

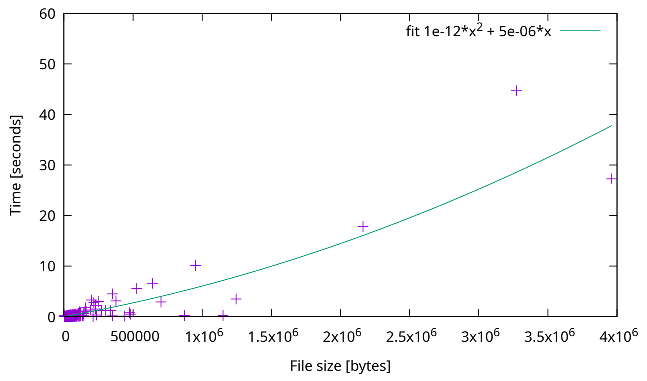
[10]https://github.com/OpenModelica/BioChem/commit/517a9962647



(a) Files where the contents changed (so there is a diff). There is a slight quadratic trend in the scaling.



(b) Files where the contents did not change (so there is no diff).



(c) All files (both with and without diff).

Figure 3. Converted files larger than 4096 bytes plotted as time it takes for the diff algorithm to handle problems of a given file size.

goes from great (Listing 6) to reasonable (Listing 7) with no very bad results. ScalableTestSuite and Physiomodel both have a single very large file in the library (and some smaller ones), which causes a lot of memory to be needed. Despite the large file, ScalableTestSuite produces a very nice diff whereas Physiomodel has whitespace changes in a lot of places. For the files where the diff algorithm failed, OpenModelica's unparsing of the internal form is used instead. This changes too many lines to be able to tell what semantic changes were actually performed.

5 Discussion

The method is limited to programmatically modified text. If a regular text editor is used and the diff algorithm is used to merge the changes, all the manual whitespace changes are lost. This means it cannot be used as an enforced hook in your source code management system since you would never be able to fix broken whitespace.

Handling diffs in connections is difficult because the connections do not have names. This means that subtrees are only compared if a single connection was updated. In practice, this does not seem to affect components in the conversion script.

There is another limitation in that parenthesis are considered whitespace – the unparsing of OpenModelica would need to be updated to preserve parenthesis where added manually (and to not output parentheses in other places either). This particular limitation sometimes causes the merging to fail with catastrophic results. These merge failures are detected by a sanity check that verifies that the semantics before and after merging are the same. Removing a parenthesis or moving parenthesis to the wrong location is thus detected. Not all of the failures may be due to this limitation of the diff algorithm – earlier versions of the algorithm also failed because the parser did not handle the full Modelica grammar.

The unparsing of text is also assumed to preserve formatting of real numbers. In OpenModelica, the parser keeps the text instead of transforming the value into floating point in order to produce something easier to perform the diff on. If another Modelica tool would read `1000000.0` and output `1e6`, the diff algorithm would assume this is a desired change.

If a Modelica tool would reorder for example modifications in annotations, this would cause the merged code to also move the modifications around (possibly with some changes to indentation and whitespace).

The algorithm has difficulty with performance and unintended edits when there are many changes at the same time. In order to improve performance and quality of the diff, Modelica libraries should be split into multiple files as this vastly improves responsiveness of the OMEdit GUI when moving objects around. This effect is also seen when applying conversion scripts to libraries where libraries with a small number of files have a lower quality in the diffs produced by the diff algorithm. The diff algorithm could easily be extended to run in parallel for libraries split into multiple files, further improving performance at the cost of memory usage. Splitting the library into multiple files also has a positive impact on load performance since it is trivial to parse multiple files in parallel[11].

Continuously improving the diff algorithm has fixed a lot of similar problems in the past, and with a bit more finetuning for things like indentation the algorithm could become even better.

The diff algorithm has not been evaluated on output from other Modelica tools, but a reasonable way to do so would be to run the diff algorithm on git commits in some of these libraries.

6 Conclusion

The diff algorithm was intended to be used in an interactive GUI with single changes between each modification and it works well for this use-case. When the diff algorithm is used on larger changes such as applying a conversion script, the quality of the diffs goes down and it works much better when the library is divided into many smaller files. The good news is that even when the indentation is changed, it is usually feasible to manually correct these since there are only local changes (`git diff` will not show you irrelevant edits). The time spent on making sure there are only small changes saves time for the code reviewer. Compared to existing tools that change formatting of the whole file even when not making any change, it is a big step in the right direction.

Given the existing diff algorithm, the recommendation for development of Modelica libraries would be to split the library into more files since you get a faster response from the user interface for each edit as well as fewer unintended changes. This workflow is something that should work well in most cases.

After fixing the remaining bugs and tuning the diff algorithm, it should be able to run the conversion script on all the tested libraries.

Detecting which connections belong together could be resolved by making the unparsing include additional information to both files that is then removed from the merged file. This might allow tools to perform more edits at the same time and still preserve formatting within each connection. Alternatives include looking for the names of the ports that are connected (but this might break if a connection is moved between ports), or perhaps by having named connections in the Modelica specification.

Changes that improve the quality of diffs after applying the conversion script would also improve the tool when running on files modified by other Modelica tools (for example if pull requests in Github need to be cleaned up to see what changes are actually proposed).

[11]Loading files in parallel was implemented in OpenModelica back in 2014 and typical speedup is 5x for 8 threads with the garbage collection as bottleneck. Loading a single large file vs. multiple files in a single thread has no impact on performance in our experience.

Table 1. The performance of running the conversion script. The exact libraries converted are either the latest release or from a master branch on Github (this has little relevance on the results, but only 1 copy of each library was used). Shown is the total file sizes of all files in the library, the largest file size (to see if libraries not split into smaller files are more problematic to handle), how many files were modified, how many the diff algorithm failed on, and how many files there were in the library. The time it takes to run the conversion script and the total time to run the Modelica diff algorithm on all files in the library as well as the maximum heap size for these phases are also provided. If the maximum heap size for the diff algorithm is the same as for the conversion script, it means that the memory reclaimed by the garbage collector was enough to run the diff algorithm.

Library	Size	Max size	Diffs	Fail	Files	Conversion		Diff	
AdvancedNoise	340 kB	39 kB	12	0	107	0.2 s	24 MB	0.7 s	24 MB
AixLib	12,585 kB	89 kB	972	3	2,731	5.0 s	490 MB	36.4 s	491 MB
BioChem	564 kB	231 kB	3	0	134	0.6 s	58 MB	1.1 s	107 MB
BuildSysPro	7,110 kB	79 kB	871	12	1,948	3.2 s	234 MB	22.5 s	235 MB
BuildingSystems	7,574 kB	132 kB	676	2	2,089	3.2 s	315 MB	22.7 s	316 MB
Buildings	13,947 kB	472 kB	1,117	0	3,341	5.8 s	474 MB	39.4 s	475 MB
ConPNlib	50 kB	6 kB	1	0	37	0.1 s	18 MB	0.2 s	18 MB
ElectricalEnergyStorage	330 kB	330 kB	1	1	1	0.2 s	33 MB	1.2 s	156 MB
ExternalMemoryLib	28 kB	28 kB	1	0	1	0.1 s	25 MB	0.1 s	25 MB
FCSys	1,121 kB	292 kB	7	0	16	0.9 s	58 MB	2.8 s	123 MB
FastBuildings	181 kB	4 kB	8	0	131	0.2 s	24 MB	0.7 s	24 MB
HanserModelica	372 kB	14 kB	58	0	137	0.2 s	33 MB	2.0 s	33 MB
HelmholtzMedia	477 kB	73 kB	24	0	242	0.5 s	33 MB	1.4 s	44 MB
IBPSA	4,901 kB	67 kB	494	0	1,382	2.4 s	186 MB	13.8 s	186 MB
IdealizedContact	625 kB	625 kB	1	0	1	0.2 s	44 MB	6.6 s	188 MB
IndustrialControlSystems	717 kB	19 kB	11	0	241	0.3 s	44 MB	1.5 s	44 MB
KeyWordIO	58 kB	7 kB	7	0	38	0.1 s	24 MB	0.2 s	24 MB
LibRAS	255 kB	14 kB	36	2	80	0.2 s	33 MB	1.1 s	33 MB
MEV	75 kB	75 kB	1	1	1	0.1 s	18 MB	0.2 s	44 MB
ModelicaByExample	292 kB	6 kB	54	0	355	0.7 s	33 MB	1.7 s	33 MB
Modelica_DeviceDrivers	634 kB	67 kB	30	0	192	0.7 s	44 MB	2.0 s	58 MB
Modelica_Noise	290 kB	23 kB	12	0	101	0.2 s	24 MB	0.6 s	24 MB
Modelica_Synchronous	827 kB	246 kB	7	0	9	0.4 s	44 MB	7.2 s	107 MB
NcDataReader2	12 kB	2 kB	1	0	12	0.1 s	18 MB	0.0 s	18 MB
ObjectStab	243 kB	44 kB	22	0	159	0.3 s	33 MB	0.9 s	44 MB
OpenHydraulics	530 kB	26 kB	30	0	162	0.2 s	44 MB	1.7 s	44 MB
OpenIPSL	1,294 kB	30 kB	52	0	402	0.6 s	90 MB	3.9 s	90 MB
PNlib	816 kB	76 kB	5	0	224	0.3 s	44 MB	1.6 s	59 MB
PhotoVoltaics	271 kB	21 kB	35	0	118	0.2 s	24 MB	1.2 s	33 MB
PhotoVoltaics_TGM	100 kB	7 kB	20	0	20	0.1 s	18 MB	0.5 s	18 MB
Physiolibrary	886 kB	265 kB	7	0	10	1.0 s	44 MB	4.7 s	123 MB
Physiomodel	3,417 kB	3,196 kB	2	0	4	0.9 s	186 MB	44.8 s	991 MB
PowerGrids	643 kB	26 kB	17	0	202	0.3 s	44 MB	1.6 s	44 MB
PowerSystems	1,973 kB	109 kB	5	0	104	1.6 s	90 MB	3.7 s	123 MB
ScalableTestSuite	6,186 kB	3,869 kB	14	0	22	0.8 s	254 MB	45.8 s	1,002 MB
SiemensPower	400 kB	16 kB	95	0	169	0.3 s	33 MB	1.0 s	33 MB
SolarTherm	1,161 kB	62 kB	264	2	534	1.2 s	74 MB	4.7 s	74 MB
Spot	1,944 kB	115 kB	6	0	90	1.4 s	90 MB	3.9 s	107 MB
SystemDynamics	1,216 kB	1,216 kB	1	0	1	0.3 s	74 MB	3.5 s	396 MB
ThermalSeparation	4,642 kB	1,123 kB	134	4	533	3.1 s	138 MB	14.7 s	270 MB
ThermoPower	2,502 kB	930 kB	8	1	10	1.2 s	106 MB	19.8 s	350 MB
ThermoSysPro	4,588 kB	343 kB	594	0	980	1.9 s	186 MB	24.1 s	300 MB
iPSL	3,068 kB	852 kB	41	0	534	0.7 s	106 MB	5.7 s	122 MB

```
@@ -3,7 +3,7 @@
   "Controller for single zone VAV system"
   extends Modelica.Blocks.Icons.Block;

-  parameter Modelica.SIunits.Temperature TSupChi_nominal
+  parameter Modelica.Units.SI.Temperature TSupChi_nominal
     "Design value for chiller leaving water temperature";
   parameter Real minAirFlo(
     final min=0,
@@ -11,10 +11,10 @@
     final unit="1")
     "Minimum airflow fraction of system"
     annotation(Dialog(group="Setpoints"));
-  parameter Modelica.SIunits.DimensionlessRatio minOAFra
+  parameter Modelica.Units.SI.DimensionlessRatio minOAFra
     "Minimum outdoor air fraction of system"
     annotation(Dialog(group="Setpoints"));
-  parameter Modelica.SIunits.Temperature TSetSupAir
+  parameter Modelica.Units.SI.Temperature TSetSupAir
     "Cooling supply air temperature setpoint"
     annotation(Dialog(group="Setpoints"));
   parameter Buildings.Controls.OBC.CDL.Types.SimpleController controllerTypeHea=
```

Listing 6. An example of a typical diff in Buildings, which looks good.

```
@@ -5,16 +5,16 @@

   Modelica.Blocks.Sources.Sine mixAirTem(
     amplitude=7.5,
-    freqHz=1/86400,
+f     =1/86400,
     offset=20 + 273.15) "Mixed air temperature"
     annotation (Placement(transformation(extent={{-100,80},{-80,100}})));
   Modelica.Blocks.Sources.Sine retAirTem(
     amplitude=10,
-    freqHz=1/86400,
+f     =1/86400,
     offset=21 + 273.15) "Return air temperature"
     annotation (Placement(transformation(extent={{-100,-70},{-80,-50}})));
   Modelica.Blocks.Sources.Sine outAirTem(
-    freqHz=1/86400,
+    f =1/86400,
     amplitude=6,
     offset=18 + 273.15) "Measured outdoor air temperature"
     annotation (Placement(transformation(extent={{-100,-40},{-80,-20}})));
@@ -51,7 +51,7 @@
     annotation (Placement(transformation(extent={{-100,-10},{-80,10}})));
   Modelica.Blocks.Sources.Sine supAirTem(
     amplitude=7,
-    freqHz=1/86400,
+f     =1/86400,
     offset=13 + 273.15) "Supply air temperature"
     annotation (Placement(transformation(extent={{-100,-100},{-80,-80}})));
 equation
```

Listing 7. An example of an uncommon diff in Buildings. The results are reasonable, with only local changes. However, the indentation has been changed except where `freqHz` was the first modification.

Future work includes not considering parentheses whitespace, which requires changing OpenModelica to never add or remove parentheses in its internal representation. This change would resolve most issues where files failed to merge. Implementing this change would make the algorithm only work in tools that preserved parentheses in its unparsing, making the diff algorithm only work well with OpenModelica.

Given this knowledge, it can be concluded that the given approach will not work to merge any arbitrary changes of Modelica code. However, the approach should work well when it is integrated with a tool that preserves as much of the original structure as possible (including positions of parentheses, the exact string representation of floating point numbers, as well as the order of modifications and connections).

There are alternative approaches that would make library development easier, such as Modelica tools making edits directly in the concrete syntax tree. However, this approach requires a much more targeted approach and needs to be considered at an early stage of development. Another approach would be to use the same version of a Modelica tool for all edits of Modelica code, or a formatter that enforces consistent formatting before making a commit to the source code management system. This approach makes it harder to make manual changes to the source code since the tool may automatically revert the changes as they do not conform to its formatting rules.

Acknowledgments

This research was funded by Vinnova in the ITEA EMPHYSIS project (grant number 2017-05121). Support from the Swedish Government has also been received through the ELLIIT project and through SSF in the LARGEDYN project (grant number ITM17-0154).

References

Butler, Nicholas (2009). *Investigating Myers' diff algorithm: Part 1 of 2*. URL: https://www.codeproject.com/Articles/42279/Investigating-Myers-diff-algorithm-Part-1-of-2 (visited on 2021-04-21).

Fritzson, Peter et al. (2020). "The OpenModelica Integrated Environment for Modeling, Simulation, and Model-Based Development". In: *Modeling, Identification and Control* 41.4, pp. 241–295. DOI: 10.4173/mic.2020.4.1.

Google (2019). *Diff Match Patch*. URL: https://code.google.com/p/google-diff-match-patch/ (visited on 2021-04-21).

Hashimoto, Masatomo and Akira Mori (2008). "Diff/TS: A Tool for Fine-Grained Structural Change Analysis". In: *2008 15th Working Conference on Reverse Engineering*, pp. 279–288. DOI: 10.1109/WCRE.2008.44.

Matsumoto, Junnosuke, Yoshiki Higo, and Shinji Kusumoto (2019). "Beyond GumTree: A Hybrid Approach to Generate Edit Scripts". In: *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pp. 550–554. DOI: 10.1109/MSR.2019.00082.

Modelica Association (2017-04). *Modelica – A Unified Object-Oriented Language for Systems Modeling. Language Specification Version 3.4*. Tech. rep. Linköping: Modelica Association. URL: https://www.modelica.org/documents/ModelicaSpec34.pdf.

Modelica Association (2021-02). *Modelica – A Unified Object-Oriented Language for Systems Modeling. Language Specification Version 3.5*. Tech. rep. Linköping: Modelica Association. URL: https://specification.modelica.org/maint/3.5/MLS.html.

Myers, Eugene (1986). "An $O(ND)$ difference algorithm and its variations". In: *Algorithmica* 1, pp. 251–266. DOI: 10.1007/BF01840446.

Tichy, Walter (1984-11). "The String-to-String Correction Problem with Block Moves". In: *ACM Trans. Comput. Syst.* 2.4, pp. 309–321. ISSN: 0734-2071. DOI: 10.1145/357401.357404.