Towards Visualising Change-based Models

Alfa Yohannis*‡, Horacio Hoyos Rodriguez*, Fiona Polack†, Dimitris Kolovos*
*Department of Computer Science, University of York
alfa.yohannis@merahputih.id, horacio_hoyos_rodriguez@ieee.org, dimitris.kolovos@york.ac.uk
†School of Computing and Maths, Keele University, United Kingdom
f.a.c.polack@keele.ac.uk
‡Department of Computer Science, Kalbis Institute, Indonesia

***Abstract*—This paper extends our previous work on change-based model persistence and demonstrates a tool that can replay the construction change-based model. The tool takes a change-based persistence file as input and plays it in the form of evolving diagrams reflecting the changes applied to the model.**

***Index Terms*—visualization, change-based persistence, model evolution, BPMN2, model-driven engineering, replay, animation**

I. Introduction

In [1], we explored the concept of change-based persistence for models conforming to object-oriented metamodelling architectures, such as MOF and Ecore. We also demonstrated a prototype (EpsilonLabs CBP[1]), which enables persisting EMF models as sequences of changes (as opposed to snapshots of their state in XMI), which was further evaluated and extended in [2]–[4].

In this paper, we extend our previous work on change-based persistence (CBP) of models conforming to the MOF/EMF metamodelling architectures [1]–[4] by contributing a tool that can visualise changes that have been applied to construct a model. Our tool takes a change-based model as input and constructs an animation of the changes it comprises.

The rest of the paper is structured as follows. Section II provides an overview of our previous work on change-based model persistence. Section III discusses our approach to using change-based persistence to visualise model construction. Section **??** presents the evaluation of our approach. Section IV provides an overview of related work, and Section V concludes with a discussion on directions for future work.

II. Change-based Persistence

Instead of persisting snapshots of models as commonly practised in model-driven engineering, change-based model persistence persists the complete editing history of changes. This means that all changes applied to a model are recorded so that they can be re-used for other purposes.

Figure 1: A simple BPMN2 model.

[1]https://github.com/epsilonlabs/emf-cbp

Let's say that Bob developed a simple BPMN2 model in Figure 1. If the model is persisted in a state-based format, the persistence produces an XMI file (simplified) as in Listing 1. This type of persistence only preserves the eventual state of the model, losing the detailed information of changes executed by Bob to construct the model. In contrast, if we record all the changes made by Bob and persist them into a CBP file, we can obtain a list of change events in Listing 2. In this list, the CBP file is pseudo-formatted to improve readability, but in implementation, the file is persisted in an XML-like format. Replaying these recorded changes produces a model with the same eventual state as in Figure 1 and Listing 1.

Listing 1: A BPMN2 model in Figure 1 persisted in simplified XMI.

```
1 <process id="e1" name="Process 1">
2  <startEvent id="e2">
3    <outgoing>e6</outgoing>
4  </startEvent>
5  <endEvent id="e4">
6    <incoming>e7</incoming>
7  </endEvent>
8  <task id="e5" name="Task A">
9    <incoming>e6</incoming>
10   <outgoing>e7</outgoing>
11 </task>
12 <sequenceFlow id="e6" sourceRef="e2" targetRef="e5"/>
13 <sequenceFlow id="e7" sourceRef="e5" targetRef="e4"/>
14 </process>
```

From Listing 2, we can know the sequence of changes made by Bob to construct the model. We can also identify that Bob made invalid changes that connect SequenceFlow `e3` from EndEvent `e4` to StartEvent `e2` (no SequenceFlow is allowed to come out from an EndEvent or enter a StartEvent) (lines 10-11) which he deleted in the following changes (lines 13-17). Such phenomenon might not be identified if we persist the model in state-based format.

Listing 2: The pseudo-formatted CBP of the model in Figure 1.

```
1  create e1 type Process
2  set e1.name to "Process 1"
3  create e2 type StartEvent
4  add e2 to e1.flowElements at 0
5  create e3 type SequenceFlow
6  set e3.name from to "Sequence Flow 1"
7  add e3 to e1.flowElements at 1
8  create e4 type EndEvent
9  add e4 to e1.flowElements at 0
10 add e3 to e2.incoming at 0
11 add e3 to e4.outgoing at 0
```

```
12 add e1 to resource at 0
13 remove e3 from e2.outgoing at 0 composite c1
14 remove e3 from e4.incoming at 0 composite c1
15 unset e3.name from "Sequence Flow 1" to null composite c1
16 remove e3 from e1.flowElements at 1 composite c1
17 delete e3 type SequenceFlow composite c1
18 create e5 type Task
19 set e5.name from to "Task 1"
20 add e5 to e1.flowElements at 2
21 create e6 type SequenceFlow
22 add e6 to e2.outgoing at 0
23 add e6 to e5.incoming at 0
24 add e6 to e1.flowElements at 3
25 create e7 type SequenceFlow
26 add e7 to e5.outgoing at 0
27 add e7 to e4.incoming at 0
28 add e7 to e1.flowElements at 4
29 set e5.name to "Task A"
```

While persisting all these changes is perceived too excessive as it requires more storage space [3], in some conditions, it is desirable especially when we want to perform model analytics, such as understanding model evolution, identifying model editing/language usage patterns, etc. [1]. Moreover, CBP can also drastically speed up model comparison [4].

Trying to understand changes executed by Bob in Listing 2 may require extra cognitive effort as one who tries to comprehend it needs to build a mental model and emulate the changes in their mind. One solution to reduce the required cognitive effort is by providing a visualisation tool. With the tool, they can replay the model construction and observe what kind of changes have been made by Bob.

III. Visualising Model Construction

For the reasons we have exposed, we have built CBP-Player, a prototype[2], that visualises the construction of change-based models. It is a JavaScript application and uses the mxGraph diagramming library [5] for displaying graphical representations of the evolving model.

A. Process

Figure 2 shows the process of drawing a model in CBP-Player. The process consists of three phases: Loading, Building, and Drawing. In the Loading phase, the CBP-Player loads a CBP file as a sequence of change events in memory. It also loads the metamodel of the change-based model. The metamodel[3] defines the model that will be constructed by the Building phase.

In the Building and Drawing phases, the CBP-Player plays the loaded change events one-by-one emulating the construction of the model in the CBP file. This construction also computes the values required to draw the model's graphical representation. For example, the shifting of elements' indexes caused by the deletion of an element in a containing feature in order to label edges, or determining the types of edges to represent features.

[2]Project and demo can be found at https://github.com/epsilonlabs/emf-cbp/tree/master/CBPPlayer and https://alfa-ryano.github.io/visualization.html

[3]For now, the metamodel is still defined from scratch in Javascript. Further extension is required to load metamodel from Ecore files.

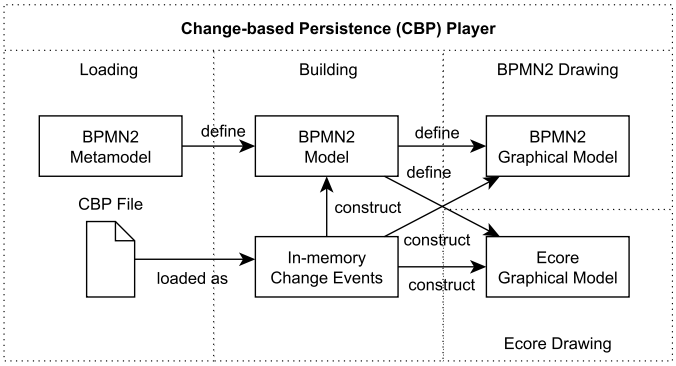


Figure 2: The process to visualise model construction using change-based persistence.

When executing a change event, besides applying the change to construct an abstract/semantic model, the change is also reflected to a graphical model. Currently, the prototype is designed to have more than one model drawer to reflect changes to different graphical models. In Figure 2, we have two drawings which allow viewing a model construction in two perspectives: here, BPMN2 and object-diagram-like notation that visualises Ecore models in a graph/tree structure (further we refer this as Ecore).

B. Implementation

Figure 3 depicts the simplified class diagram of the implementation of the CBP-Player. The main class, `CBP-Player`, has methods `load`, `play`, and `stop` to load and control the play of a CBP file.

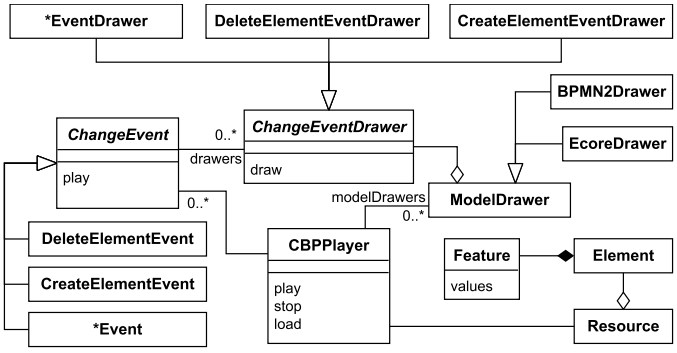


Figure 3: The simplified class diagram of the CBP-Player implementation.

When initialising, the `CBPPlayer` registers all kinds of `ModelDrawer` that are going to be used. In Figure 3, we use `EcoreDrawer` and `BPMN2Drawer` classes as examples for the model drawers. Both classes are derived from `ModelDrawer` class. This `ModelDrawer` class consists of different extension classes of the `ChangeEventDrawer` class. Each extension class determines what kind of drawing operations should be executed on the graphical model if an instance of `ChangeEvent` is executed. For example, when the play method of the `DeleteElementEventDrawer` class is

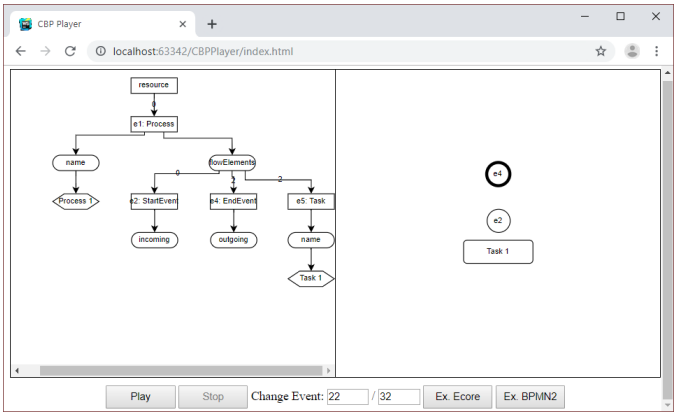


Figure 4: The application of CBP-Player.

executed, it removes an element from the graphical model. The instance of this class should be registered to its corresponding `ChangeEvent`, in this case, class `DeleteElementEvent`. If the instance of this `ChangeEvent` plays, it also executes the `draw` method of the `ChangeEventDrawer` instances registered to it. Thus, if an element is deleted from an abstract model, the corresponding element is also deleted in its graphical representation.

While executing the load method, a CBP file is loaded into memory, a list of `ChangeEvent` (changeEvents), and a corresponding `ChangeEventDrawer` is registered to each change event. When `CBP-Player` plays this `changeEvents`, it iterates throughout the list and executes the `play` method of each change event emulating the construction of a model. After executing each `play` method, the `draw` method of the corresponding registered `ChangeEventDrawer` is also executed to draw the model's graphic. Classes `Resource`, `Element`, and `Feature` are used as internal data structures for constructing the abstract model.

C. Application

To obtain the change-based model in Listing 2, firstly, we had to modify the Eclipse BPMN2 Modeler [6]. In particular, we added `ChangeEventAdapter` [7] of the EpsilonLabs CBP [1] to the `Bpmn2Resource`'s `eAdapters` in the Eclipse BPMN2 Modeler. The said adapter can capture every change made to a model through the modeller. We then created the model in Figure 1 by following the course of changes in Listing 2, and saved the model, producing XMI and CBP files such as those in Listings 1 and 2.

We have built a simple application in which we employ the CBP-Player (Figure 4). The application can visualise the construction of a BPMN2 model using two notations: an object-diagram-like notation, and the standard BPMN2 graphical syntax. To demonstrate the application, we feed it with the produced CBP file. The application loads the change events in the CBP file into memory and replays them one-by-one. Every time a change event is replayed, it is also reflected in the two diagrams.

In the object diagram perspective, as displayed in Figure 5c, elements and values are represented with rectangles, features are depicted in rounded rectangles, and containment/non-containment relationships are represented with solid/dashed directed edges. Solid directed edges also represent ownership between elements and their features. Numbers on the edges indicate the indexes of the elements in their containers.

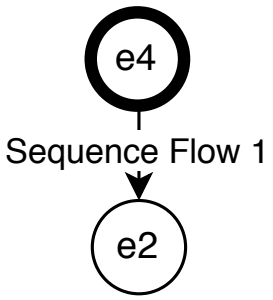
Figure 5 shows the graphical models displayed by the prototype. Figures 5c and 5a show the object diagram and BPMN2 diagram at the time when edge `e3` is about to be removed (Listing 2, line 12). In Figure 5c, we can notice there is element `e1` with type `Process` contained by the resource at index 0. The element has a single-valued feature `name` with value "Process 1". Element `e1` also has a multi-valued feature `flowElements` that contains three other elements, `e2` with type `StartEvent`, `e3` with type `SequenceFlow`, and `e4` with type `EndEvent`, each at index 0, 1, and 3 respectively. Element `e3` has a feature `name` with value "Sequence Flow 1". This element is referenced by elements `e2`'s `incoming` and `e4`'s `outgoing` features.

Figure 5a displays the model in BPMN2; the state of the model in the object diagram perspective is hidden from users. In the figure, we can notice a `SequenceFlow` with name "Sequence Flow 1" wrongly connects `EndEvent e2` to `StartEvent e1`. No `SequenceFlow` is allowed to leave an `EndEvent` or enter a `StartEvent`. Later, the `SequenceFlow` is removed from the model (Listing 2 lines 13-17).

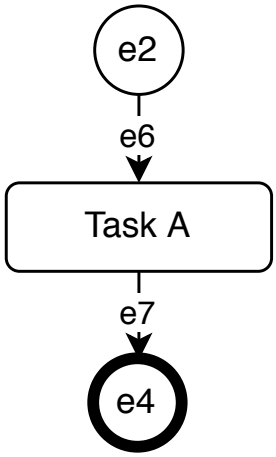
Figures 5b and 5d show the diagrams after all changes have been applied. In Figure 5b, the diagram does not contain the `SequenceFlow` with name "Sequence Flow 1" anymore. It is replaced with a Task "Task A" and two `SequenceFlow` `e6` and `e7` in the correct direction. Figure 5d shows the same model but displayed in the object diagram perspective.

IV. Related Work

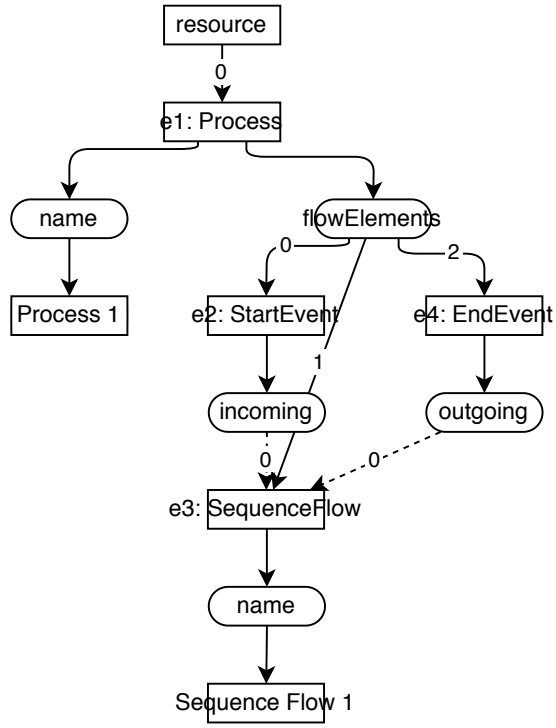
Some tools are available to visualise Ecore-based models. EcoreTools [8] is a tool to define metamodels graphically while other tools such as Eugenia [9] and Sirius [10] are used to define the visual concrete syntax of models. Nevertheless, these tools are not intended to visualise the evolution or changes of models. Visualisation of model evolution has been studied in different domain-specific modelling languages for business processes [11] and software [12]. Work on visualising changes specific to Ecore-based models was performed by Maier et al. [13]. They used a state-based approach to persist different versions of a model. Changes between versions were derived using EMF Compare [14], a model differencing tool, and then presented in a timeline format with colour-coding used to identify changes between versions, e.g. a new element is coloured green while other old elements are coloured grey. In contrast, our approach exploits change-based persistence to obtain changes and visualise them in the form of animation.



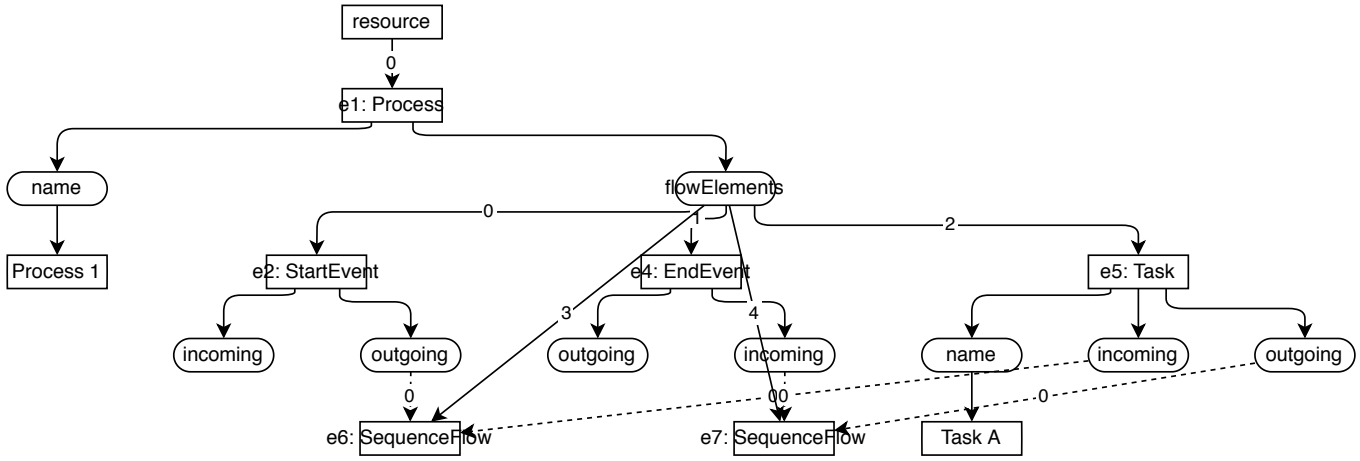
(a) BPMN2 model at line 12 in Listing 2



(b) BPMN2 model at line 29 in Listing 2



(c) Ecore model at line 12 in Listing 2



(d) Ecore model at line 29 in Listing 2

Figure 5: The graphical models produced by the prototype during the construction of the model in Figure 1.

V. Conclusions and Future Work

This paper has presented an extension to the work of change-based model persistence that aims at providing a tool, the CBP-Player, to visualise model construction. The tool takes a CBP file as input and plays it in the form of a graphical model, emulating the changes applied to the persisted model. The tool is designed to be extensible to any types of visualisation that aim to exploit the information contained in CBP. The tool itself is still at a prototype state. Some features that are planned to be added are visualising model differencing and conflict detection and model metrics (number of elements, features, etc.) throughout the evolution of a model. We have not undertaken any performance evaluation in this work since performance is not the main goal of the current prototype. To evaluate correctness, we have employed unit tests that exercise the features presented in this work. A systematic evaluation is required in later iterations to evaluate the performance, usefulness, and usability of the tool.

ACKNOWLEDGMENT

This work was part supported by the European Commission via the CROSSMINER (project number 732223) and TYPHON (project number 780251) H2020 projects and through a scholarship managed by *Lembaga Pengelola Dana Pendidikan Indonesia* (Indonesia Endowment Fund for Education).

REFERENCES

[1] A. Yohannis, D. S. Kolovos, and F. Polack, "Turning models inside out," in *Proceedings of MODELS 2017 Satellite Events co-located with ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS 2017), Austin, TX, USA, September, 17, 2017.*, 2017, pp. 430–434. [Online]. Available: http://ceur-ws.org/Vol-2019/flexmde_8.pdf

[2] A. Yohannis, H. H. Rodriguez, F. Polack, and D. S. Kolovos, "Towards efficient loading of change-based models," in *Modelling Foundations and Applications - 14th European Conference, ECMFA 2018, Held as Part of STAF 2018, Toulouse, France, June 26-28, 2018, Proceedings*, 2018, pp. 235–250. [Online]. Available: https://doi.org/10.1007/978-3-319-92997-2_15

[3] ——, "Towards hybrid model persistence," in *Proceedings of MODELS 2018 Workshops co-located with ACM/IEEE 21st International Conference on Model Driven Engineering Languages and Systems (MODELS 2018), Copenhagen, Denmark, October, 14, 2018.*, 2018, pp. 594–603. [Online]. Available: http://ceur-ws.org/Vol-2245/me_paper_3.pdf

[4] A. Yohannis, H. H. Rodriguez, F. Polack, and D. Kolovos, "Towards efficient comparison of change-based models," B. Combemale and A. Shaukat, Eds., vol. 18, no. 2, Jul. 2019, pp. 7:1–21, the 15th European Conference on Modelling Foundations and Applications. [Online]. Available: http://www.jot.fm/contents/issue_2019_02/article7.html

[5] JGraph, "mxGraph 4.0.0," https://jgraph.github.io/mxgraph/, accessed: 2019-06-05.

[6] Eclipse, "Eclipse BPMN2 Modeler," https://www.eclipse.org/bpmn2-modeler/, accessed: 2019-06-04.

[7] EpsilonLabs, "emf-cbp/ChangeEventAdapater.java," https://github.com/epsilonlabs/emf-cbp/blob/master/org.eclipse.epsilon.cbp/src/org/eclipse/epsilon/cbp/event/ChangeEventAdapter.java, accessed: 2019-06-06.

[8] Eclipse, "EcoreTools - Graphical Modeling for Ecore - Eclipse," https://www.eclipse.org/ecoretools/, accessed: 2019-06-10.

[9] D. S. Kolovos, A. García-Domínguez, L. M. Rose, and R. F. Paige, "Eugenia: towards disciplined and automated development of gmf-based graphical model editors," *Software & Systems Modeling*, vol. 16, no. 1, pp. 229–255, Feb 2017. [Online]. Available: https://doi.org/10.1007/s10270-015-0455-3

[10] Eclipse, "Sirius - Overview - Eclipse," https://www.eclipse.org/sirius/, accessed: 2019-06-10.

[11] B. Fritscher and Y. Pigneur, "Visualizing business model evolution with the business model canvas: Concept and tool," in *2014 IEEE 16th Conference on Business Informatics*, vol. 1, July 2014, pp. 151–158.

[12] A.-L. Mattila, P. Ihantola, T. Kilamo, A. Luoto, M. Nurminen, and H. Väätäjä, "Software visualization today: Systematic literature review," in *Proceedings of the 20th International Academic Mindtrek Conference*, ser. AcademicMindtrek '16. New York, NY, USA: ACM, 2016, pp. 262–271. [Online]. Available: http://doi.acm.org/10.1145/2994310.2994327

[13] S. Maier and M. Minas, "Recording, processing, and visualizing changes in diagrams," in *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, Oct 2015, pp. 131–135.

[14] Eclipse, "EMF Compare," https://www.eclipse.org/emf/compare/, accessed: 2018-01-15.