

declare-js: A Web-Based Viewer and Editor for Declarative Process Models

Sabine Nagel¹, Eric Amann¹ and Patrick Delfmann¹

¹ University of Koblenz, Universitätsstr. 1, Koblenz, Germany

Abstract

In this work, we introduce *declare-js*, a web-based viewer and editor for declarative process models (DPMs) in the modeling language Declare, built in JavaScript. This tool enables users to import and export models while providing visual and textual interfaces for creating and editing declarative constraints. Moreover, given the complexity of understanding DPMs, *declare-js* offers features to enhance comprehension by visually linking textual and visual representations. While the tool is currently optimized for smaller models, future iterations are set to incorporate features that facilitate the visualization and comprehension of larger and more complex DPMs, along with features for detecting and visualizing redundancies and inconsistencies within DPMs.

Keywords

Declare, Declarative Process Models, Declarative Process Specifications, Visualization, Editor, Viewer

1. Introduction

As part of business process management (BPM), business processes can be represented by so-called business process models [1]. This includes formal or textual specifications, as well as the corresponding graphical notations. Generally, it is distinguished between procedural and declarative process models (DPMs) [2]. While the former approach explicitly models every possible execution trace, declarative approaches implicitly model processes using a set of constraints. All constraints, i.e., the entire declarative process specification, must be satisfied during process execution [3]–[5]. For both approaches, several process modeling languages exist, common ones being BPMN (*Business Process Model and Notation*) for procedural models and *Declare* for declarative models. In *Declare*, constraints are based on linear temporal logic (LTL) but can be modeled using a set of pre-defined constraint templates. Thus, the underlying logic remains hidden [3], [4], [6], which allows modelers to work with a graphical or textual representation of DPMs without having to be familiar with the logic-based formalization [3]. While several applications include features for graphically modeling DPMs (e.g., [7], [8]), a web-based editor and/or visualizer only exists for procedural models, the most common one being *bpmn-js*². Thus, in this work, we introduce *declare-js*, a web-based viewer and editor for declarative process specifications using the modeling language *Declare*.

The remainder of the paper is structured as follows: In Section 2 we describe the tool and its functionality in more detail. This includes importing and exporting models, the interfaces to create and edit textual and visual constraints, as well as functionality to visually link both representations to enhance model comprehension. In Section 3 we explain how to use *declare-js* and briefly discuss the maturity of the tool's current version. We conclude and discuss future work in Section 4.


ICPM Doctoral Consortium and Demo Track 2023, October 23-27, 2023, Rome, Italy

✉ snagel@uni-koblenz.de (S. Nagel); amann@uni-koblenz.de (E. Amann); delfmann@uni-koblenz.de (P. Delfmann)

ORCID 0000-0003-4838-8246 (S. Nagel); 0009-0005-6241-6081 (E. Amann)



© 2023 Copyright for this paper by its authors.
Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

² <http://bpmn.io>

2. Tool Description

declare-js is fully implemented in JavaScript and does not use any external libraries to allow for maximum flexibility during development and integration. Currently, our tool supports a standard set of *Declare* templates based on [4], [5] and can handle variations of certain templates (e.g., *AtLeastOne* vs. *AtLeast1*). Figure 1 shows an overview of our tool, which includes possibilities for import and export (1), an edit menu to visually create and edit the current model (2), a canvas showing the visual representation of the model (3), and a text editor to also enable creating and editing the textual representation of the model (4). In the following sections, we will explain these components and additional functionalities in more detail, a corresponding screencast can be found here³.

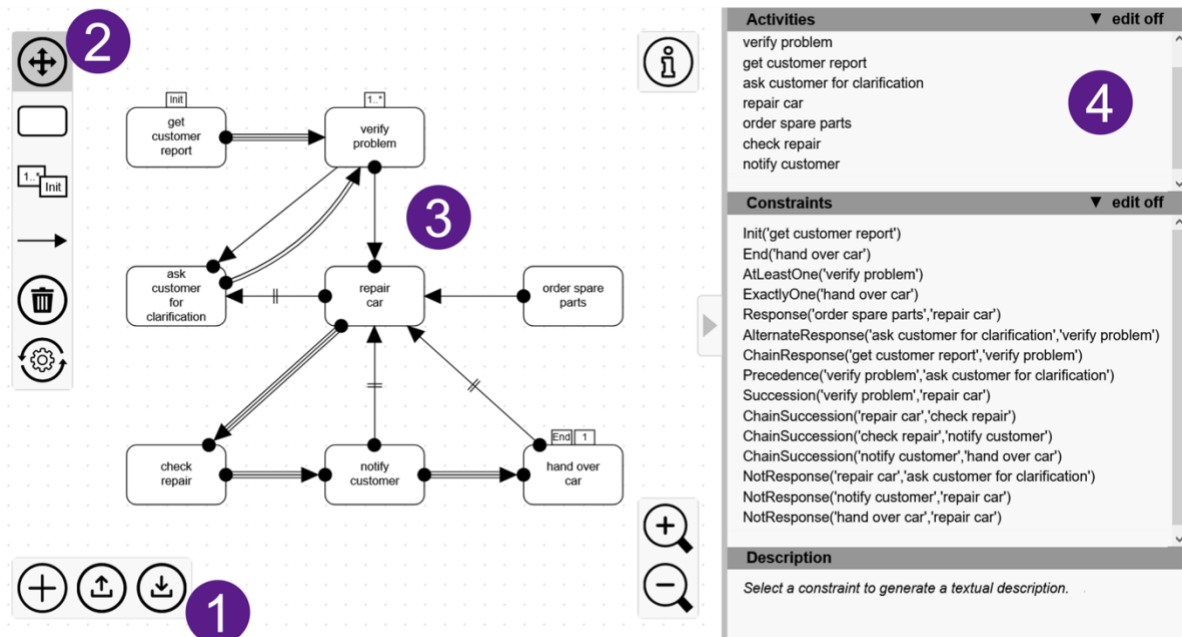


Figure 1: Overview of *declare-js* and its main components

2.1. Import and Export

In addition to modeling a specification from scratch, *declare-js* enables importing existing DPMs in various file formats (e.g., JSON, CSV) to allow seamless integration with other tools or algorithms. The tool also comes with different possibilities for exporting the current model. This includes all input formats but also allows downloading the currently visible part of the model directly as an SVG file, so the created visual DPMs can easily be integrated into other applications or documents. For testing purposes, we provide exemplary models on our website⁴.

2.2. Visual Editor

New elements can be created and added to the model using the edit menu on the left of the screen (B). The design of vertices and edges is based on related works in the area of DPMs [4], [5], [9]. While activities (rounded rectangles) can be directly placed anywhere on the canvas, existence constraints (small rectangles) must be defined first by choosing a category (position or cardinality) and then selecting the desired value. Afterward, they can be attached to any activity on the canvas.

Relation constraints can be added via the arrow button in the edit menu. Here the user can choose between the following properties, with each (valid) combination representing one of the

³ <https://uni-ko.de/declare-js-screencast>

⁴ <https://uni-ko.de/declare-js>

pre-defined templates. First, a constraint can be directed or undirected. For directed relations, it is then possible to specify if the constraint is a regular (e.g., *Response*), alternate (e.g., *Alternate Response*) or chain relation (e.g., *ChainResponse*). Next, the direction of activation can be set by defining whether the constraint should be activated by its first parameter (e.g., *Response*), second parameter (e.g., *Precedence*), or both parameters (e.g., *Succession*). Lastly, it is also possible to negate a constraint (e.g., *NotResponse*). When predetermining the properties first, the constraint can directly be placed with its final visualization. However, it is also possible to first place a default relation constraint and change the properties in the edit menu afterward. This can also be done throughout the entire modeling process to edit existing constraints.

When importing an existing textual model, all activities are auto-positioned to minimize overlapping edges. To this aim, we implemented a force-directed graph algorithm that consists of four parts: (1) a force towards the center, (2) repulsion between activities, (3) attraction of connected activities, and (4) a force to prevent collisions between existence and relation constraints of the same activity. This auto-position feature can also be applied at any time during modeling. More specifically, the user can choose between auto-positioning the entire model or locking individual activities in place via their context menu and only auto-positioning the remaining activities. This allows keeping the part of the model that has already been adjusted or modeled according to the user's needs as is.

At any point in time, users can move activities around by dragging them to the desired location. Also, single or multiple activities can be deleted, either using their context menu or the trash can button in the edit menu on the left.

2.3. Textual Editor

When the edit mode is turned on, it is also possible to edit the textual representation of the current specification by adding new or changing existing elements. As soon as a valid textual constraint is entered, the corresponding visual constraint is automatically added to the canvas as well. A constraint is considered valid if the used template and the number of parameters (one for existence constraints and two for relation constraints) match one of the pre-defined *Declare* templates and if the activities are defined for the current model. Until then, newly entered constraints are displayed in red. To organize the textual constraints, we implemented possibilities for sorting activities alphabetically and constraints based on their underlying template.

2.4. Comprehension

Many works have investigated comprehension of DPMs and concluded that – especially in contrast to procedural models – DPMs are rather hard to understand, especially in their visual form [3], [9]–[11]. Therefore, the visual and textual representation of constraints are linked as follows. When hovering over an activity or constraint in the visual model, its corresponding textual constraint is shaded in gray and vice versa, so hovering over a textual constraint directly points towards the respective part of the visual model. This linking also applies to the selection of single or multiple elements in the visual or textual editor. Additionally, when selecting a visual or textual constraint, a textual description is provided in the bottom section of the text editor.

3. Usage and Maturity

Our application can either be used directly in the browser⁴ or by cloning our git repository⁵ and integrating the viewer or editor directly into web applications. Here, the desired mode (viewer or editor) can be selected, so it is possible to just include a viewer without the full editing functionality. Further documentation on how to use or integrate *declare-js* into other applications can be found on our website⁴.

⁵ <https://uni-ko.de/declare-js-git>

As declarative constraints represent circumstantial rather than sequential information, visualizing large and highly interrelated models will always be challenging. Thus, our tool is especially useful and efficient for smaller models or subsets of specifications. This includes inconsistencies (i.e., minimally inconsistent subsets), whose comprehensibility is expected to benefit from visualizations [12].

To ensure usability for users with prior experience in process modeling, the design and main functionality were inspired by state-of-the-art web-based modeling tools (e.g., *bpmn-js*). However, *declare-js* still is at an early stage of development and has yet to be validated externally. Therefore, as an immediate next step, we will empirically investigate the usability of our tool by conducting an eye-tracking study. This includes assessing ease of use for end users and developers that aim to integrate *declare-js* into their own applications. Furthermore, we aim to validate if visually linking textual and visual model representations in its current form actually increases comprehensibility and also identify possibilities for improvement here.

4. Conclusion and Outlook

In this work, we introduced *declare-js*, a web-based viewer and editor for DPMs in the modeling language *Declare*. In its current version, *declare-js* supports several import and export formats, provides editors for both visual and textual constraints, and links both representations to facilitate DPM comprehension.

As explained in the previous section, our tool is currently optimized for smaller models, as visualizing more complex models is a rather challenging task due to the interrelated nature of activities in DPMs. However, future iterations are set to incorporate features that facilitate the visualization and comprehension of larger and more complex DPMs. To this aim and to be able to break down and/or modularize the specification, we will also implement extended sorting and filtering, which lowers complexity and can also increase understanding [13]. In addition to conducting usability studies regarding the current functionality of the tool, we also aim to identify further possibilities for improvement.

In addition to continuously improving the tool and its base functionality, we will also further extend the tool. While the current set of pre-defined templates is based on related literature [4], [5] and was chosen to allow seamless integration with other applications and declarative process mining algorithms, we plan to further extend this set in future versions, for instance by allowing higher cardinalities for existence templates and implementing additional relation templates.

Furthermore, we are currently developing extensions for model verification by detecting and visualizing redundancies and inconsistencies. Highlighting such problematic subsets of constraints directly in the textual or visual model is expected to support inconsistency understanding in general, as well as future inconsistency resolution and prevention approaches [11], [12].

References

- [1] M. Weske, *Business Process Management: Concepts, Languages, Architectures*. Springer, 2007.
- [2] D. Fahland, J. Mendling, H. A. Reijers, B. Weber, M. Weidlich, and S. Zugal, "Declarative versus Imperative Process Modeling Languages: The Issue of Maintainability," in *Business Process Management Workshops*, in Lecture Notes in Business Information Processing. Berlin, Heidelberg: Springer, 2010, pp. 477–488.
- [3] K. Figl, C. Di Ciccio, and H. A. Reijers, "Do Declarative Process Models Help to Reduce Cognitive Biases Related to Business Rules?," in *Proceedings of the International Conference on Conceptual Modeling*, in Lecture Notes in Computer Science, vol. 12400. 2020, pp. 119–133.

- [4] C. Di Ciccio, F. M. Maggi, M. Montali, and J. Mendling, "Resolving Inconsistencies and Redundancies in Declarative Process Models," *Information Systems*, vol. 64, pp. 425–446, Mar. 2017.
- [5] C. Di Ciccio and M. Montali, "Declarative Process Specifications: Reasoning, Discovery, Monitoring," in *Process Mining Handbook*, vol. 448, W. M. P. van der Aalst and J. Carmona, Eds., Cham: Springer International Publishing, 2022, pp. 108–152.
- [6] F. M. Maggi, M. Westergaard, M. Montali, and W. M. P. van der Aalst, "Runtime Verification of LTL-Based Declarative Process Models," in *Proceedings of the International Conference on Runtime Verification*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 131–146.
- [7] A. Alman, C. D. Ciccio, D. Haas, F. M. Maggi, and A. Nolte, "Rule Mining with RuM," in *2020 2nd International Conference on Process Mining (ICPM)*, Padua, Italy: IEEE, Oct. 2020, pp. 121–128.
- [8] F. M. Maggi, "Declarative Process Mining with the Declare Component of ProM," in *Proceedings of the BPM Demo sessions 2013, Co-located with 11th International Conference on Business Process Management (BPM 2013)*, Beijing, China, 2013.
- [9] C. Haisjackl *et al.*, "Understanding Declare models: strategies, pitfalls, empirical results," *Softw Syst Model*, vol. 15, no. 2, pp. 325–352, May 2016.
- [10] C. Haisjackl and S. Zugal, "Investigating Differences between Graphical and Textual Declarative Process Models," in *Advanced Information Systems Engineering Workshops*, in Lecture Notes in Business Information Processing. 2014, pp. 194–206.
- [11] S. Nagel and P. Delfmann, "Investigating Inconsistency Understanding to Support Interactive Inconsistency Resolution in Declarative Process Models," in *ECIS 2022 Research-in-Progress Papers*, 2022.
- [12] S. Nagel and P. Delfmann, "Exploring Cognitive Effects of Inconsistency Characteristics on Understanding Inconsistencies in Declarative Process Models," in *Proceedings of the 57th Hawaii International Conference on System Sciences (HICSS)*, 2024.
- [13] A. A. Andaloussi, P. Soffer, T. Slaats, A. Burattin, and B. Weber, "The Impact of Modularization on the Understandability of Declarative Process Models: A Research Model," in *Information Systems and Neuroscience*, Cham: Springer International Publishing, 2020, pp. 133–144.