

Improving the Understandability of Declarative Process Discovery Results Using EASYDECLARE

Graziano Blasilli^a, Lauren S. Ferro^b, Simone Lenti^a, Fabrizio Maria
Maggi^c, Andrea Marrella^a, Tiziana Catarci^a

^a*Sapienza University of Rome, via Ariosto 25, 00185 Rome, Italy*

^b*CSIRO's Data61, Clayton, VIC 3168, Australia*

^c*Free University of Bozen-Bolzano, Piazza Domenicani 3, 39100 Bolzano, Italy*

Abstract

Declarative process models allow us to capture the behavior of a business process through temporal constraints on the evolution of process activities. In process mining, declarative process discovery focuses on deriving these constraints from event logs. Although the semantic aspects of declarative processes have been extensively investigated, there has been less focus on designing declarative visual notations that enhance model understanding and support analysts in solving process mining tasks. To improve the human understandability of declarative process models, in this paper, we present EASYDECLARE, a novel visual notation to specify declarative process models using the DECLARE language. EASYDECLARE was developed with consideration of the well-established Moody's design principles. We conducted extensive user experiments to demonstrate that EASYDECLARE, when compared with the original graphical representation of DECLARE, reduces the cognitive load required to interpret DECLARE models of increasing complexity, making it a promising alternative to enhancing overall comprehension of declarative process discovery tasks.

1. Introduction

One of the most common ways to visualize and communicate process mining insights is to use process models that show the actual flow of ac-

Email addresses: blasilli@diag.uniroma1.it (Graziano Blasilli),
lauren.ferro@data61.csiro.au (Lauren S. Ferro), lenti@diag.uniroma1.it (Simone Lenti),
maggi@inf.unibz.it (Fabrizio Maria Maggi), marrella@diag.uniroma1.it
(Andrea Marrella), catarci@diag.uniroma1.it (Tiziana Catarci)

tivities, events, and resources involved in a business process as recorded in an event log. Process models can be represented through many languages (e.g., BPMN, Directly-Follow Graphs, Petri-Nets, etc.) that visually map the overall process structure and support process mining specialists in identifying problematic areas (e.g., congestion, bottlenecks) and improving decision-making. The processes (and their issues) that such languages map out ultimately need to be read by business stakeholders. However, the issue with using such process-oriented languages is their level of interpretability for persons who are not inherently from the field of computer science.

Despite the abundance of languages available for process models, it is widely acknowledged that most process modeling languages fall somewhere on the imperative-declarative spectrum. Imperative notations such as BPMN facilitate the description of sequential process flows, whereas declarative specifications like DECLARE describe cause-effect temporal relations between events [41]. Although the semantic aspects of declarative processes have been extensively researched, there has been comparatively less focus on designing declarative visual notations that enhance model understanding and incorporating features into these notations that facilitate comprehension as model complexity grows [26].

State-of-the-art solutions, e.g., [41, 11, 21], struggle with effectively communicating explicit concepts of how to interpret a declarative process model. Existing literature suggests that a new notation easing understandability is needed [16, 26]. This need arises because current notations can become overwhelming and confusing, especially for non-expert stakeholders who need to interpret these models to make informed business decisions. Given the importance of declarative models in accurately reflecting the flexible nature of business processes, improving their interpretability is critical for leveraging the full potential of process mining.

Based on early-stage development [17], the notation presented in this paper, called EASYDECLARE, is designed to ease the process of understanding declarative process models. The development of this language has been done with consideration of the well-established Moody’s design principles [33] and several parameters (e.g., use of shapes) to maintain a contextual fit. EASYDECLARE aims to bridge the gap between the complexity of declarative models and the practical need for clarity and interpretability by business stakeholders. It incorporates intuitive visual elements that simplify the mapping of complex relationships and constraints within process models, thereby reducing cognitive load and enhancing overall comprehension. We also developed a software tool to support the creation of DECLARE models based on our proposed graphical notation.

We conducted extensive user experiments whose results highlight that EASYDECLARE demonstrates superior effectiveness and efficiency in complex tasks and higher user satisfaction in ease of use and intention to use if compared with the original graphical representation of DECLARE. This makes it a promising tool for improving the communication of process mining insights related to declarative process discovery tasks to a broader audience, ultimately supporting better decision-making and process optimization.

The rest of the paper is organized as follows. Section 2 first introduces the required background on declarative processes and the development of visual notations necessary to understand the paper. Then, it discusses some state-of-the-art efforts aimed to improve the interpretability of DECLARE models through innovative graphical notations. Section 3 describes the rationale for the design of EASYDECLARE. Section 4 presents the results of the comparative evaluation conducted to assess EASYDECLARE against the original graphical representation of DECLARE, focusing on performance and perception metrics. Section 5 details the realization of the software tool to specify DECLARE models through EASYDECLARE, showing its usability through a dedicated user experiment. Finally, Section 6 concludes the paper.

2. Background and State of the Art

In this section, we introduce basic background concepts necessary for comprehending the content of the paper.

2.1. Declare

DECLARE is a declarative process modeling language introduced by Pestic and van der Aalst in [41]. DECLARE is qualified as “declarative” because it does not explicitly specify every possible sequence of activities leading from the start to the end of a process execution. Instead, it bases models on a set of constraints that must hold true during the execution of the process. All behaviors that respect those constraints are allowed. Constraints are applied to sets of activities and mainly pertain to their temporal ordering. In particular, DECLARE specifies an extensible set of standard templates (see Table 1) that a process analyst can use to model a process. Constraints are concrete instantiations of these templates. DECLARE is equipped with a formal semantics on Linear Temporal Logic on Finite Traces (LTL_f) [10], but the use of templates makes model comprehension independent of the logic-based formalization. Indeed, analysts can work with the graphical representation of templates while the underlying formulas remain hidden. Graphically, a

Table 1: DECLARE constraints

Constraint	Explanation	Examples	Notation
Existence constraints			
EXISTENCE(n, a)	Activity a occurs at least n times in the trace		
PARTICIPATION(a) \equiv EXISTENCE(1, a)	a occurs at least <i>once</i>	✓ bcac ✓ bcaac × bcc × c	
ABSENCE($m + 1, a$)	a occurs at most m times		
ATMOSTONE(a) \equiv ABSENCE(2, a)	a occurs at most <i>once</i>	✓ bcc ✓ bcac × bcaac × bcacaa	
INIT(a)	a is the <i>first</i> to occur	✓ acc ✓ abac × cc × bac	
END(a)	a is the <i>last</i> to occur	✓ bca ✓ baca × bc × bac	
Relation constraints			
RESPONDEDEXISTENCE(a, b)	If a occurs in the trace, then b occurs as well	✓ bcaac ✓ bcc × caac × acc	
RESPONSE(a, b)	If a occurs, then b occurs after a	✓ caacb ✓ bcc × caac × bacc	
ALTERNATERESPONSE(a, b)	Each time a occurs, then b occurs afterwards, before a recurs	✓ cacb ✓ abcab × caacb × bacacb	
CHAINRESPONSE(a, b)	Each time a occurs, then b occurs immediately afterwards	✓ cabb ✓ abcab × cacb × bca	
PRECEDENCE(a, b)	b occurs only if preceded by a	✓ cacbb ✓ acc × ccbb × bacc	
ALTERNATEPRECEDENCE(a, b)	Each time b occurs, it is preceded by a and no other b can recur in between	✓ cacba ✓ abcaacb × cacbba × abbabcb	
CHAINPRECEDENCE(a, b)	Each time b occurs, then a occurs immediately beforehand	✓ abca ✓ abaabc × bca × baacb	
Mutual relation constraints			
COEXISTENCE(a, b)	If b occurs, then a occurs, and vice-versa	✓ cacbb ✓ bcca × cac × bcc	
SUCCESSION(a, b)	a occurs if and only if it is followed by b	✓ cacbb ✓ acbb × bac × bcca	
ALTERNATESUCCESSION(a, b)	a and b if and only if the latter follows the former, and they alternate each other in the trace	✓ cacbab ✓ abcabc × caacbb × bac	
CHAINSUCCESSION(a, b)	a and b occur if and only if the latter immediately follows the former	✓ cabab ✓ ccc × cacb × cbac	
Negative relation constraints			
NOTCOEXISTENCE(a, b)	a and b never occur together	✓ cccbbb ✓ ccac × accbb × bcac	
NOTSUCCESSION(a, b)	a can never occur before b	✓ bbcaa ✓ cbca × aacbb × abb	
NOTCHAINSUCCESSION(a, b)	a and b occur if and only if the latter does not immediately follow the former	✓ acbacb ✓ bbaa × abcab × cabc	

DECLARE model is a diagram in which activities are represented as nodes (labeled rectangles) and constraints as arcs between activities.

Compared with procedural approaches, DECLARE models are more suitable for describing processes operating in unstable environments and characterized by numerous exceptional behaviors. Since anything not explicitly specified is allowed, a few constraints can specify many possible behaviors at once. DECLARE templates can be divided into four main groups: *existence templates*, *relation templates*, *mutual relation templates*, and *negative relation templates*. The first group consists of unary templates, which can be expressed as predicates over a single parameter. The remaining groups correspond to binary predicates over two parameters.

Starting from the first row of [Table 1](#), $\text{EXISTENCE}(n, a)$ is an existence template that requires the execution of a at least n times in every process instance. $\text{PARTICIPATION}(a)$ is a specific case where a is required to occur at least once in every process instance. Similarly, according to $\text{ABSENCE}(m + 1, a)$, the execution of a is allowed at most m times in every process instance. $\text{ATMOSTONE}(a)$ specifies that a is not allowed to be executed more than once in a process instance. $\text{INIT}(a)$ and $\text{END}(a)$ specify that a occurs as the first and last activity, respectively, in every process instance.

The group of relation templates comprises rules that are imposed on the occurrence of target activities when activation tasks occur. For example, $\text{RESPONDEDEXISTENCE}(a, b)$ is a relation template imposing that if a is performed at least once during the process execution, then b must also occur at least once, either before or after a . $\text{RESPONSE}(a, b)$ extends $\text{RESPONDEDEXISTENCE}(a, b)$ by requiring that b must eventually occur *after* a . $\text{ALTERNATERESPONSE}(a, b)$ further adds the condition that no other a events can occur between the execution of a and the subsequent b . Template $\text{CHAINRESPONSE}(a, b)$ is even stricter, specifying that whenever a occurs, b must occur immediately after. $\text{PRECEDENCE}(a, b)$ requires that a must occur *before* b . $\text{ALTERNATEPRECEDENCE}(a, b)$ adds to $\text{PRECEDENCE}(a, b)$ the condition that no other b events can occur between the execution of b and the preceding a . Finally, $\text{CHAINPRECEDENCE}(a, b)$ specifies that whenever b occurs, a must occur immediately before.

Two specializations of the relation templates are mutual relation templates and negative relation templates. The first group includes templates where both constrained activities are activation and target. For example, $\text{COEXISTENCE}(a, b)$ is a mutual relation template requiring that if a is executed, then b must be performed as well, and vice versa. $\text{SUCCESSION}(a, b)$ requires that both response and precedence relations hold between a and b . $\text{ALTERNATESUCCESSION}(a, b)$ strengthens $\text{SUCCESSION}(a, b)$ by specify-

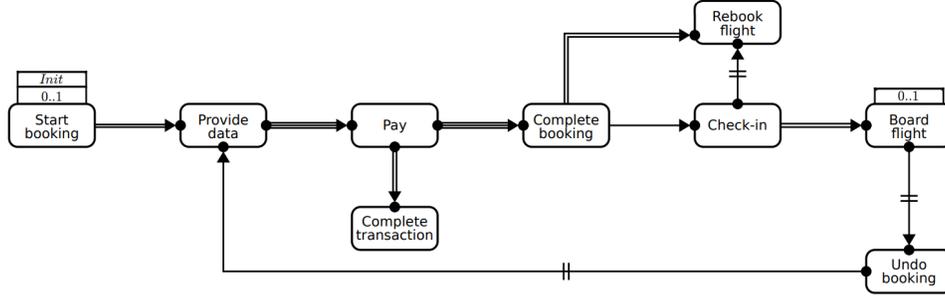


Figure 1: The DECLARE model of a flight booking process.

ing that activities must alternate without repetitions in between. Template $\text{CHAINSUCCESSION}(a, b)$ further strengthens $\text{ALTERNATESUCCESSION}(a, b)$ by requiring that occurrences of activities a and b are next to each other. In the second group, the occurrence of an activation activity excludes the occurrence of a target activity. $\text{NOTCOEXISTENCE}(a, b)$ is a negative relation template requiring that if a is executed, then b cannot be performed in the same trace, and vice versa. $\text{NOTSUCCESSION}(a, b)$ requires that no b activities occur after a (and therefore no a activities occur before b). Finally, $\text{NOTCHAINSUCCESSION}(a, b)$ requires that the activity immediately following a cannot be b .

Example 2.1 (A flight booking process model). *The scenario presented here illustrates the process a flight passenger follows from booking a ticket to boarding the aircraft. The corresponding declarative model is depicted in Fig. 1. The process begins with booking the ticket, represented in the model by activity Start booking. Passengers' personal information can then be provided and enriched with payment details. This submission is denoted as activity Provide data in the figure. The subsequent step is the authorization of the ticket payment (Pay), which triggers the completion of the booking phase (Complete booking). The authorization is eventually followed by the actual transfer of money (Complete transaction). As long as Check-in for the flight has not occurred, customers can still modify the provided data, such as changing the date of departure or amending personal information (Rebook flight). After this, only cancellation is permitted (Undo booking), in case the passenger ultimately decides not to proceed with boarding (Board flight).*

The behavioral constraints specifying how the tasks can be carried out are the following:

1. $\text{INIT}(\text{Start booking})$

2. ATMOSTONE(Start booking)
3. ALT.PRECEDENCE(Start booking, Provide data)
4. ALT.SUCCESSION(Provide data, Pay)
5. ALT.SUCCESSION(Pay, Complete transaction)
6. ALT.SUCCESSION(Pay, Complete booking)
7. ALT.PRECEDENCE(Complete booking, Rebook flight)
8. NOTSUCCESSION(Check-in, Rebook flight)
9. ALT.PRECEDENCE(Check-in, Board flight)
10. NOTSUCCESSION(Board flight, Undo booking)
11. NOTSUCCESSION(Undo booking, Provide data)

2.2. Process Mining and Event Logs

Process mining is a family of techniques used to analyze and improve business processes by extracting insights from the so-called *event logs* [40] generated during the execution of those processes [15]. An event log is a structured text file documenting the executions of a single process. Each event log contains a collection of *traces*, each representing the enactment of a unique case (a process instance). Traces are in turn sequences of *events*, i.e., single data entries related to the carry-out of an activity, within the process instance evolution. In 2010, the IEEE Task Force on Process Mining has adopted eXtensible Event Stream (XES)¹ [44] as the standard for storing, exchanging, and analyzing event logs.

Process mining involves three main types of activities: *process discovery* identifying and visualizing the actual process flows from event logs, which reveals how processes are performed in practice compared to their intended designs; *conformance checking* comparing the discovered process models against predefined models to identify discrepancies and ensure compliance with desired process standards; *model enhancement* improving existing process models based on insights gained from event logs, such as optimizing performance, reducing bottlenecks, or increasing efficiency. By leveraging data captured during process execution, process mining helps organizations understand how their processes operate, identify areas for improvement, and ensure that processes align with their goals and compliance requirements.

¹<http://www.xes-standard.org/>

2.3. Declarative Process Discovery

Declarative process discovery is a branch of process mining concerning the discovery of declarative process models from event logs. This is the group of techniques we deal with in this paper as *we propose a new visual notation that we demonstrate improves the current output of these process mining techniques, which is primarily based on the original graphical representation of DECLARE*.

In the area of declarative process discovery with models expressed using DECLARE, the work [31] first proposed an unsupervised algorithm for the discovery of declarative process models. This approach was improved in [28] using a two-phase approach. The first phase is based on an apriori algorithm used to identify frequent sets of correlated activities. A list of candidate constraints is built on the basis of the correlated activity sets. In the second phase, the constraints are checked by replaying the log on specific automata, each accepting only those traces that are compliant to one constraint. Those constraints satisfied by a percentage of traces higher than a user-defined threshold, are discovered. Other variants of the same approach are presented in [29, 27, 30]. The technique presented in [29] leverages apriori knowledge to guide the discovery task. In [27], the author extends the approach to discover metric temporal constraints, i.e., constraints taking into account the time distance between events. Finally, in [30], the authors propose mechanisms to reduce the execution times of the original approach presented in [28].

In [13], a two-step algorithm for the discovery of DECLARE constraints is presented. The first step of the approach is the building of a knowledge base, with information about temporal statistics about the (co-)occurrence of tasks within the log. Then, the validity and the support of constraints is computed by querying that knowledge base. The value assigned to support is calculated by counting the activations not leading to violations of constraints. In [12], the authors propose an extension of MINERful to discover target-branched DECLARE constraints, i.e., constraints in which the target parameter is replaced by a disjunction of actual tasks.

Another approach for the discovery of DECLARE models is described in [36]. The presented technique is based on the translation of DECLARE templates into SQL queries on a relational database instance, where the event log has previously been stored. The query answer assigns the free variables with those tasks that lead to the satisfaction of the constraint in the event log. The methodology has later been extended towards multi-perspective DECLARE discovery [35], to include data in the formulation of constraints. An Evolutionary Declare Miner that implements the discovery

task using a genetic algorithm was presented in [43]. In [2], the authors investigate how to leverage Model Learning algorithms [38] for the automated discovery of deterministic finite state automata representing DECLARE templates from event logs. Finally, an approach for the online discovery of DECLARE models has been presented in [8]. Here, the models are discovered from streams of events generated at runtime during the execution of a business process. The models are adapted on-the-fly as soon as the behavior of the underlying process changes.

The approaches mentioned above specify the discovered process models leveraging the original graphical representation of DECLARE [39]. In Section [Section 2.5](#), we discuss some literature research efforts aimed to improve the interpretability of DECLARE models through innovative (and sometime non-conventional) graphical notations.

2.4. Designing and Developing Visual Notations

The use of visual notations is extensive throughout engineering and information technology, where shapes and patterns aim to represent processes (e.g., BPMN, flowcharts, etc.). Developing a visual language’s syntax requires an informative approach that considers several key parameters of visual communication, such as the level of cognitive processing required to understand and use the notation, the shapes used to communicate the processes, and the conventions followed. By documenting this design approach, the final design is justified, and insights can be gained into aspects that effectively and efficiently communicate processes and those that do not.

Several frameworks provide principles for researchers developing or evaluating visual notations within a scientific context. For example, the Cognitive Dimensions of Notations (CDs) framework defines 13 dimensions that describe the structure of cognitive artifacts [20], while the Semiotic Quality (SEQUAL) framework proposes general qualities for models and modeling languages, organized along the semiotic ladder (i.e., the scale ‘physical’, ‘empirical’, ‘syntactic’, ‘semantic’, ‘pragmatic’, and ‘social’) [24].

However, one of the more notable examples, and one that is extensively used, is *The Physics of Notations theory* by Moody [33] (PoN). PoN provides the foundation for many studies either as a way to inform the development of new visual notations or to examine the effectiveness of existing ones. This theory presents a framework for how visual notations are constructed and processed by the human mind, drawing on theories from communication, graphic design, semiotics, visual perception, and cognition. Moody’s seminal work emphasizes the meanings of graphical symbols and how they are

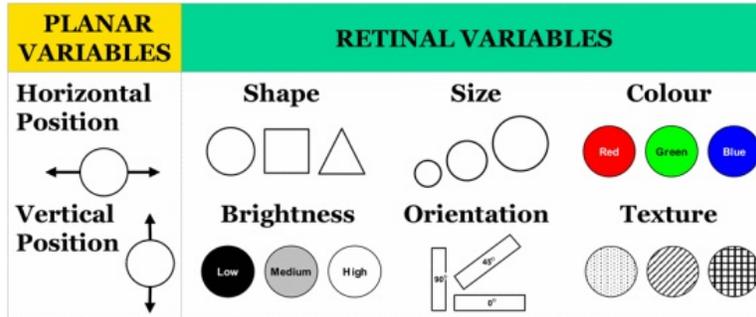


Figure 2: Moody's Visual Variables [33]

defined by mapping them to the constructs they represent [33]. Moody asserts that visual variables define a set of atomic building blocks for constructing visual notations, where these variables provide the grammar for the notation. Therefore, the choice of visual variables should not be arbitrary but should consider the conventions of symbols currently used or accepted within the field, follow a logical process, and be easily understood by those using and interpreting the notation. This consideration is particularly important because each variable has properties that can make it more suitable for encoding certain types of information over others.

Based on this information, Moody presents nine principles for designing cognitively effective visual notation, as shown in Fig. 2. The following definitions are described in [42]:

1. Semiotic clarity: There should be a 1:1 correspondence between semantic constructs and graphical symbols.
2. Perceptual discriminability: Symbols should be clearly distinguishable from one another.
3. Semantic transparency: Use symbols whose appearance suggests their meaning.
4. Complexity management: Include explicit mechanisms for dealing with complexity.
5. Cognitive integration: Include explicit mechanisms to support the integration of information from different diagrams.
6. Visual expressiveness: Use the full range and capacities of visual variables.

7. Dual coding: Use text to complement graphics.
8. Graphic economy: Keep the number of different graphical symbols cognitively manageable.
9. Cognitive fit: Use different visual dialects for different tasks and/or audiences.

Moody outlines three key properties that should be considered in the development of visual notation:

- **Level of organization:** Each variable can encode a certain level of information. For example, the use of color (e.g., red/green) can indicate success or error.
- **Capacity:** While each variable has infinite variations, only a finite number can be easily interpreted by the human mind (perceptible steps). For example, orientation has an infinite number of possible values (angles) but only four perceptible steps (its capacity or length).
- **Efficiency:** The order in which variables are processed (e.g., in parallel, sequentially, etc.).

In addition, Moody indicates approaches for dealing with excessive graphic complexity, such as reducing semantic and graphic complexity and increasing visual expressiveness.

Further emphasizing Moody’s work, both Popescu and Wegmann [34] and Diamantopoulou and Mouratidis [14] explore the use of Moody’s nine principles. Diamantopoulou and Mouratidis [14] assert that the design rationale for developing visual notation is often absent in the design of visual notations. They infer that this could be due to the fact that the description and reporting of the language are oriented towards science rather than art or design or that researchers consider visual notations as being informal and, therefore, analyze the notations based on their semantics, potentially undermining or undervaluing the role of design. Similarly, Popescu and Wegmann [34] and Genon, Heymans, and Amyot [19] also suggest using the set of nine principles defined by Moody [33] to evaluate how effectively modeling languages communicate their intended messages. Lastly, others have investigated systematic approaches to applying PoNs (e.g., [9, 42]), where such approaches, such as Van Der Linden, Zamansky, and Hadar [42], propose a systematic framework for applying the Physics of Notations that focuses on guiding designers to make their design choices explicit and grounded in evidence and requirements for their notation.

2.5. Alternative notations for declarative constraints

Efforts to improve the interpretability of declarative models have been diverse, with several innovative graphical notations being proposed. Di Ciccio et al. [11] introduced a graphical notation designed for declarative processes. Their approach includes two complementary views of a model: a local view and a global view. The local view focuses on one activity at a time, providing detailed insights. The global view offers an overview of the entire model, showcasing the interconnections and flow. This notation uses a bi-dimensional drawing where time is represented on the ordinates (y-axis) and implication on the abscissa (x-axis). The thickness of the boundaries around the boxes representing activities indicates their repeatability, cf. Fig. 3(a). This dual view and the graphical representation aim to enhance the user’s understanding of the process by visually differentiating between temporal and causal relationships.

Conversely, Hanser et al. [21] propose an alternative notation that challenges the traditional DECLARE notation by using circles instead of squares. This notation incorporates cursors to indicate sequential relations between activities, with inwards and outwards cursors at the end of arcs. Optional activities are represented by smaller circles placed in the middle of the arc, often accompanied by an asterisk, indicating that additional constraints may allow further activities to be executed in between.

Hanser’s notation represents an evolution of the declarative templates initially presented by van der Aalst et al. [39]. It offers a fresh perspective on designing process notations by revisiting and modifying existing elements, aiming to provide a clearer and more intuitive visualization of constraints and sequences within declarative models. Fig. 3(a) and Fig. 3(b) illustrate these notations, showing the $\text{RESPONSE}(a, b)$ and $\text{ALTERNATERESPONSE}(a, b)$ templates encoded using the respective styles proposed by Di Ciccio et al. and Hanser et al.

Finally, in 2018, Ferro et al. [17] presented VERTO, which included restyling the original DECLARE constraints to align them with the state-of-the-art design principles for visual notations. The list of DECLARE constraints represented with the VERTO notation is shown in Fig. 3(c). These visual representations ([11], [21], [17]) underscore the distinct approaches to modeling and interpreting declarative processes using DECLARE, each with its own set of advantages, limitations and enhancements.

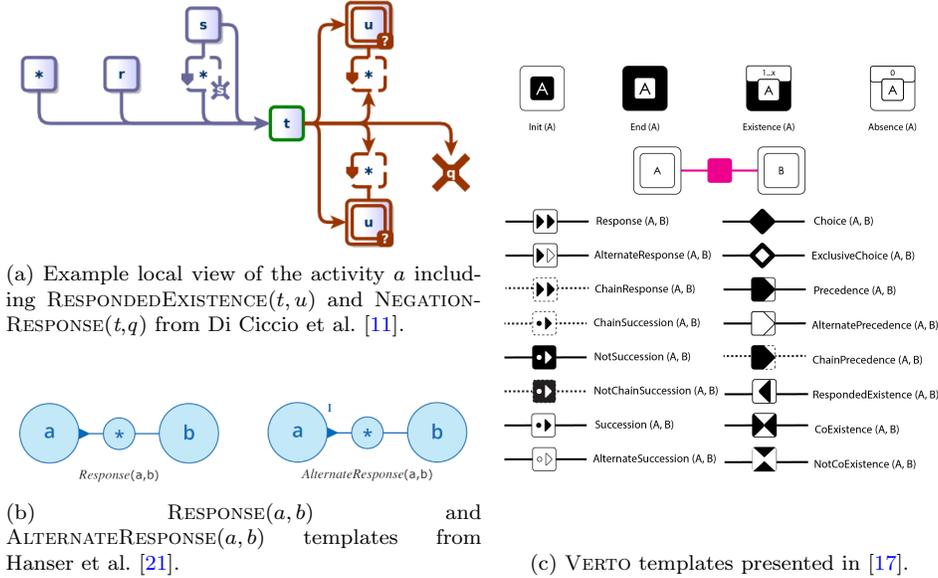


Figure 3: Alternative notations for declarative constraints

3. Design

Various elements influenced the design of `EASYDECLARE`. The first step was analyzing the three existing notations for `DECLARE`: the original one [39], the notation of Hanser et al. [21] and `VERTO` [17]. We discarded the notation in [11] from the analysis because it was neither evaluated nor fully explained by the authors, denoting its early-stage fashion.

We identified some limitations in the notations by analyzing them directly and informally collecting feedback from professionals and researchers. For `DECLARE`, two main limitations emerged. Firstly, the encodings of some concepts could be more intuitive, leading to potential misinterpretations. For example, using two lines for encoding *alternate templates* and three lines for encoding *chain templates* can be confusing. Additionally, when analyzing models with many templates, it becomes complicated to interpret the templates because the information is represented both on the line and at its ends. This increases cognitive load, especially when the entities it connects are far apart. Hanser et al. share the same approach and similar limitations. In particular, the use of inwards and outwards cursors and their combination to represent sequentiality and activations challenges perceptual discriminability. The main limitation of `VERTO`, on the other hand, lies in the high number of visual means necessary for constructing its templates.

For example, different types of arrows represent response, succession, and precedence. This can make it challenging for users to effectively utilize the notation without extensive memorization and familiarity with the various encodings.

We designed a new notation, named `EASYDECLARE`, from these limitations and informed by the nine principles of Moody [33]. The new notation addresses the identified limitations by simplifying the mapping of concepts and reducing the cognitive load required to interpret models with many templates. The templates in the new notation are constructed by composing a few elementary visual elements complying with the graphic economy. These elements are designed to be intuitive and easy to remember, reducing the need for extensive memorization. In the following, we present these elements, explaining their encoding mechanisms. We will then show how these visual means can be combined to form the resulting templates, showcasing the expected advantages of our newly designed notation.

3.1. Visual elements

The `DECLARE` and Hanser et al. relation templates are constructed by applying visual means distributed throughout the arc joining the two activities. The templates for constraints implying sequentiality (response, precedence, and succession) are shown as cursors (at the end of the arc in `DECLARE`, at both extremes in Hanser et al.), while activations are shown at the ends of the arc (as circles in `DECLARE`). In `DECLARE`, the number of lines represents the variations of the constraint (plain, alternate, chain), while negation is represented with two vertical lines in the middle of the arc. In Hanser et al., the cursors are positioned internally or externally to the activity depending on the activations and are hollow to represent the negation. The fact that information is spread can be an issue when templates are used in models that have many constraints. When information is spread across different parts of a diagram, users need to mentally connect these parts to understand the relationship. This can be cognitively demanding. For this reason, we decided to depict the binary templates with a single glyph on the arc defined according to the constraint it represents, as shown in Fig. 4. The arc is a plain line and does not contain any other visual means (e.g., arrows) except the glyph.

By consolidating all the information into a single symbol, users can interpret the information more quickly and with less mental effort. This improves the readability of complex models, as users do not need to look at multiple locations to gather the information. When information is located at the ends of arcs, there is a higher chance of misinterpreting or missing parts of

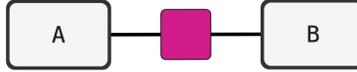


Figure 4: Binary constraint between two activities (A and B) as defined by EASYDECLARE visual notation. A single glyph over the arc represents the constraint. Each constraint primitive has its own glyph; in this figure, the purple box is a placeholder for that.

the information, especially in complex diagrams. A single symbol minimizes this risk by presenting all the relevant information in one place, thereby enhancing accuracy. This choice is consistent with the Proximity Compatibility Principle, which claims that *the elements that need to be integrated and processed together should be in close proximity*. Using a single symbol to represent the constraint makes all necessary data spatially close, making it easier to process the information.

To foster semiotic clarity, shapes are mapped 1 : 1 to different concepts: horizontal rectangles represent activities, squares represent relation templates, triangles (arrows) represent sequentially, and vertical bars represent activations. All the binary templates are shaped as squares, with the only variation occurring with `CHOICE(a, b)` and `EXCLUSIVECHOICE(a, b)`, where the shapes are diamonds. The reason for this is that the diamond is associated with choice or decision in both Flowcharts and UML.

The use of color is another area that, in many ways, offers an additional layer of implicit information. For example, the color red suggests an error, while the color green denotes success. In addition, the use of color also contains accessibility limitations in situations where users may suffer from a vision impairment (e.g., color blindness). For these reasons, the only colors that are used for EASYDECLARE are black and white. The first reason is its *ease of use* and *accessibility*. By using black and white, we mitigate accessibility issues, as the contrast between the two is enough to be seen by a majority of users. The second reason is related to printing. If a user intends to print their EASYDECLARE models, the use of black and white allows it to be clearly interpreted regardless of the printer type that is used (i.e., monochrome or color). Templates are generally encoded through a black glyph over a white background, the opposite for negative constraints. This color differentiation allows users to easily tell apart models such as `CHAINSUCCESSION(a, b)` and `NOTCHAINSUCCESSION(a, b)`, as shown in Fig. 5, where the former is white and the latter is black.

The use of arrows to indicate direction is a well-established convention in graph theory and diagramming. Adhering to this convention ensures that the graph is consistent with what users typically expect, enhancing usability.

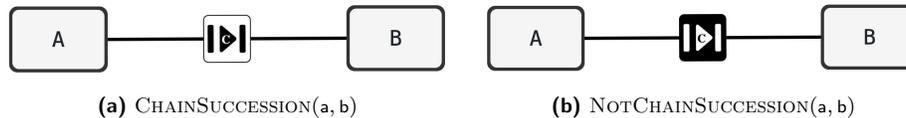


Figure 5: An example of a $\text{CHAINSUCCESION}(a, b)$ and a $\text{NOTCHAINSUCCESION}(a, b)$. The white background (a) and black background (b) on the constraint symbol discriminate between a positive and a negative relation.

ity and reducing potential confusion. For this reason, constraints expressing sequentiality are represented with an arrow (as in DECLARE) inside the template symbol. The arrow direction indicates which template precedes the other. Alternating arrow directions (as in Hanser et al.) may lead to confusion, particularly in cases where the constraint is part of complex models where it cannot be guaranteed that the template ordering corresponds to the temporal constraints. On the other hand, having an arrow explicitly indicating “who precedes who” allows templates to be interpreted correctly even in cases where they are positioned counter-intuitively (e.g., right to left, bottom to top).

The activations are represented as vertical bars inside the glyph representing the constraint. The bars are positioned according to the role of the activities; for example, $\text{RESPONSE}(a, b)$ has a bar on the left of the arrow because the activity activating it is the first one, while $\text{PRECEDENCE}(a, b)$ has a bar on the right because the template it activates is the one that follows the other one. Distinctive examples are $\text{COEXISTENCE}(a, b)$ and $\text{RESPONDEDEXISTENCE}(a, b)$; the symbol of the former consists of the two bars of the activations, while the latter has the left activation bar and a smaller bar on the right, thus recalling the direction of the template.

Finally, variations of relation templates are encoded by exploiting dual coding through letters embedded in the symbol: A for “Alternate” variations, and C for “Chain” variations.

3.2. Templates

Following the aforementioned primitives, we defined a notation for both existence and relation templates. Activities are encoded with white rectangles, although a light gray background can optionally be used to improve contrast. The name of each activity is written inside the rectangle, ensuring clear identification and easy readability. Existence constraints are represented as labels attached to the top or the bottom of the activity rectangles. These labels contain text that specifies the unary constraint. Each template has a fixed position over the activity rectangle to leverage preattentive

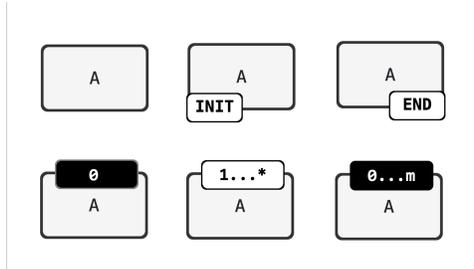


Figure 6: EASYDECLARE notation for existence constraints. First row: plain activity A , $\text{INIT}(A)$, $\text{END}(A)$. Second row: $\text{ABSENCE}(1, A)$, $\text{EXISTENCE}(1, A)$, $\text{ABSENCE}(m + 1, A)$.

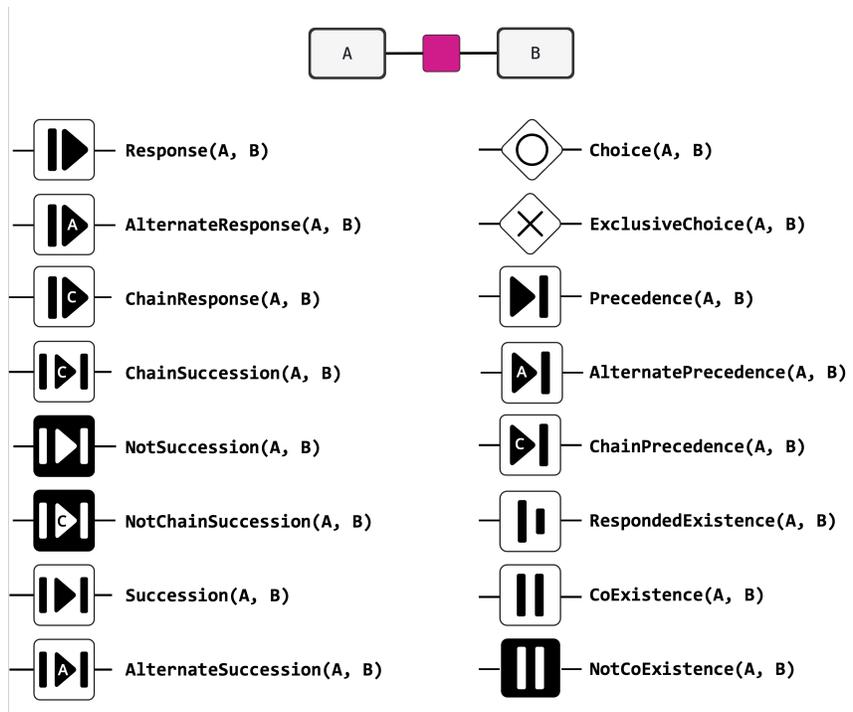


Figure 7: EASYDECLARE graphical notation for binary constraints.

visual properties. The differentiation of placement makes the constraints easily recognizable at a glance. 6 shows how existence constraints are represented.

Relation templates are represented by a glyph positioned in the middle of the arc connecting two activities. The arc is plain, without any additional visual elements (e.g., arrows), ensuring that the focus remains on the glyph

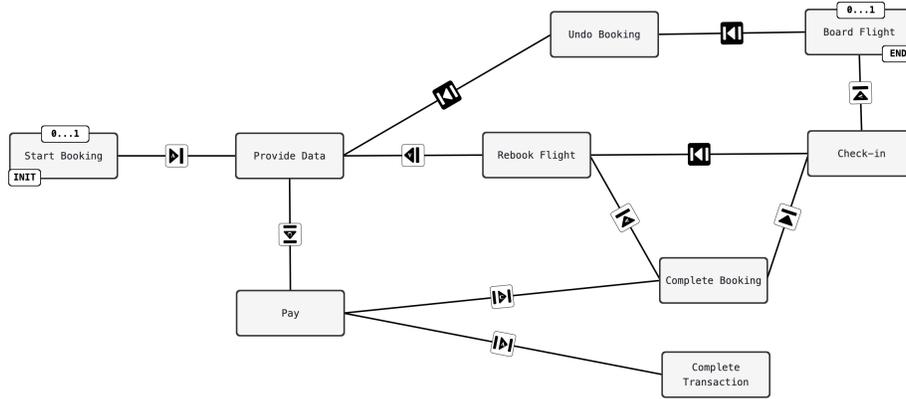


Figure 8: The EASYDECLARE model of a flight booking process, same as [Example 2.1](#)

representing the constraint. Each type of relation template is associated with a unique glyph, which is shown in [7](#).

Example 3.1 (A flight booking process model). *The scenario presented here illustrates the process a flight passenger follows from booking a ticket to boarding the aircraft. The corresponding declarative model is the same as [2.1](#) and in [8](#) is represented using the EASYDECLARE notation.*

4. Evaluation

We designed a controlled experiment to evaluate the proposed graphical notation. First, we describe the experiment’s design, then present and discuss the obtained results.

4.1. Design

The design of the experiment relies on the Method Evaluation Model (MEM) [\[32\]](#), collecting performance-based and perception-based metrics to evaluate the likelihood of acceptance of EASYDECLARE. We designed a comparative experiment in which subjects had to perform tasks using the original graphical representation of DECLARE (cf. [Table 1](#)), and the one we propose, EASYDECLARE (cf. [Fig. 6](#) and [7](#)).

A graphical notation for declarative process modeling supports two elementary tasks:

- *Template encoding* – given a template, depict it through its graphical representation;
- *Template decoding* – given the graphical representation of a template, identify it correctly.

In its practical application, it typically supports higher-level tasks built on the elementary ones:

- *Model encoding* – given a model, depict it through its graphical representation;
- *Model decoding* – given the graphical representation of a model, comprehend its meaning.

For encoding tasks, a template (or a model) was presented to the subject, who had to select its representation from five alternatives; conversely, for decoding tasks, the subject had to choose the textual representation corresponding to the graphical representation of a template (or a model) given as input.

According to the MEM, the efficacy in performing a task is defined as the quality of the results (*effectiveness*) and the effort required (*efficiency*) in achieving them. For the experiment tasks, effectiveness corresponds to the correctness of the result, while efficiency can be defined as the ratio between the effectiveness and the time spent to perform the task.

The experiment targets novice users without prior knowledge of graphical notations for declarative process modeling. Therefore, we decided to include the evaluation of the learning process of notations by subjects in the experiment. In the first phase of the experiment, the subjects had to perform elementary tasks alternated with increasingly detailed explanations of the notation. The results collected in this phase contribute to evaluating the semantic transparency and learnability of the notations.

Some empirical studies [18, 5] suggest that problem-solving tasks can be used to measure understandability. We, therefore, used model encoding and decoding to evaluate it.

Each subject experimented first with one notation and then with the other. An instance of the experiment is generated by randomly choosing the templates and the models used for the tasks. For each instance, another is created for another user in which the notations are swapped so that the order in which they are presented and the variability of the templates affect the result the least. This instance is divided into two parts, A and B, corresponding to the two notations. Each part was structured as follows:

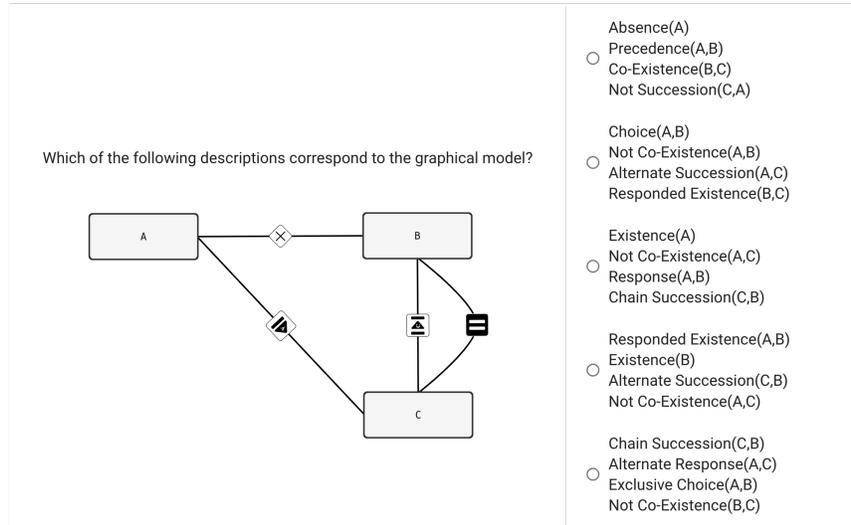


Figure 9: Example of a *Model decoding* task for the EASYDECLARE notation where the correct answer is the last one.

As the first step, the subject had to perform 4 tasks of *Template decoding* (TD_1) and 4 tasks of *Template encoding* (TE_1) without a prior explanation of the notation to evaluate its *semantic transparency*.

Afterward, subjects were provided with the list of templates and relative graphical representations (E_1), which they could consult for a maximum of 3 minutes. Then, they had to perform another 4 tasks of *Template decoding* (TD_2) and 4 tasks of *Template encoding* (TE_2).

The subjects were then provided with the list of templates and relative graphical representations and a brief explanation of the rationale behind the notation (E_2), which they could consult for as long as they deemed necessary.

Finally, they had to perform the last block of tasks composed of 4 tasks of *Template decoding* (TD_3), 4 tasks of *Template encoding* (TE_3), 4 tasks of *Model decoding* (MD_1), and 4 tasks of *Model encoding* (ME_1). Fig. 9 shows an example of a *Model decoding* task.

We collected the score (i.e., 10 if the answer is correct, 0 otherwise) and elapsed time for each performed task, and the time devoted by each subject to the two explanations.

Additionally, at the end of each part, the subjects had to answer a questionnaire regarding their perception of the graphical notation adapted from

Abrahão et al. [1] visible in Table 2.

Table 2: Questionnaire used to assess the user perception of the graphical notation, adapted from Abrahão et al. [1].

Item	Statement
PEOU1	The graphical notation is simple and easy to use.
PEOU2	Overall, it was easy to understand what the graphical templates represent (Template Decoding).
PEOU3	Overall, it was easy to understand what the graphical models represent (Model Decoding).
PEOU4	Overall, it was easy to represent the templates with the graphical notation (Template Encoding).
PEOU5	Overall, it was easy to represent the models with the graphical notation (Model Encoding).
PEOU6	The graphical notation is easy to learn.
PU1	Overall, I found the graphical notation to be useful.
PU2	Overall, I think this graphical notation provides an effective way of describing declarative templates.
PU3	I believe this graphical notation would reduce the time required to model declarative processes.
PU4	I believe this graphical notation is useful for modeling business processes in complex environments.
PU5	I believe that the models obtained with this graphical notation are organized, clear, concise and non-ambiguous.
PU6	I believe this graphical notation has enough expressiveness to represent declarative processes.
PU7	Using this graphical notation would improve my performance in describing complex business processes.
ITU1	I would use this graphical notation to specify complex business processes.
ITU2	It would be easy for me to become knowledgeable in using this graphical notation.
ITU3	I would recommend the use of this graphical notation to describe complex business processes.

The users had to answer 16 questions with a five-level Likert scale, ranging from “*strongly disagree*” to “*strongly agree*”. The answers are then mapped into a numerical value ranging from [0, 10]. The questions are intended to estimate three perception-based variables: Perceived Ease of Use (PEOU), i.e., the degree to which a person believes that using a particular technology or system will be free from effort; Perceived Usefulness (PU), i.e., the extent to which a person believes that using a specific technology or system will provide some beneficial outcome; and Intention to Use (ITU), i.e., the user’s motivation or plan to employ a particular technology or system in the future.

4.2. Hypotheses

According to the Goal-Question-Metric (GQM) template [4], the experimentation goal is to: *Analyze* the declarative process modeling graphical notations EASYDECLARE and DECLARE, *for the purpose of* evaluating their effectiveness and efficiency, *with respect to* their semantic transparency, learnability, understandability, ease of use, usefulness, and intention to use, *from the point of view of* novice and non-expert users.

The evaluation measures 3 performance-based variables (semantic transparency, learnability, and understandability) and 3 perception-based variables (PEOU, PU, and ITU). The results are analyzed with a *paired t-test*

to check whether there is a statistically significant difference between the average performance with the two notations ($p\text{-value} < 0.05$). Specifically, we formulate the following hypotheses:

Semantic transparency.

H1₀ Users without prior knowledge of EASYDECLARE and DECLARE have the same effectiveness and efficiency in decoding templates represented with the two notations.

H2₀ Users without prior knowledge of EASYDECLARE and DECLARE have the same effectiveness and efficiency in encoding templates with the two notations.

Learnability.

H3₀ Users without prior knowledge of EASYDECLARE and DECLARE have the same effectiveness and efficiency in learning to decode templates represented with the two notations.

H4₀ Users without prior knowledge of EASYDECLARE and DECLARE have the same effectiveness and efficiency in learning to encode templates with the two notations.

H5₀ Users without prior knowledge of EASYDECLARE and DECLARE take the same amount of time to memorize the two notations.

Understandability.

H6₀ Users have the same effectiveness and efficiency in decoding models represented with the two notations.

H7₀ Users have the same effectiveness and efficiency in encoding models with the two notations.

Perception-based variables.

H8₀ The Perceived Ease of Use is the same for the two notations.

H9₀ The Perceived Usefulness is the same for the two notations.

H10₀ The Intention to Use is the same for the two notations.

Subjects. The subjects who participated in the experiment are 31 students with prior knowledge of the syntax and semantics of declarative process modeling templates but not of their graphical notations. We had to discard one result due to technical problems that occurred during the experiment, reducing the number of valid experiments to 30.

The use of students as subjects might affect the experiment’s external validity; however, they are relatively close to the population of interest, being the next generation of professionals, and can be considered representative of novice and non-expert users of declarative process modeling [23, 22]. To limit the experimental bias that may affect the experiment’s internal validity, we anonymously collected the results and did not inform subjects of which notation we proposed.

Subjects were 15 males and 15 females, 29 Master students and 1 PhD student in Engineering in Computer Science.

4.3. Analysis

Semantic transparency. Regarding semantic transparency, almost no significant differences emerge between the two notations. The only difference appears in decoding effectiveness (see Fig. 10(a).T1), where using EASY-DECLARE ($\mu_V = 6.4$) is greater than using DECLARE ($\mu_D = 5.0$). This difference is not significant for efficiency (Fig. 10(b).T1) as well as for template encoding (Fig. 10(c).T1, Fig. 10(d).T1), which shows no statistically significant differences for either effectiveness or efficiency.

Results: $H1_0$ is partially rejected for the difference in the average effectiveness. However, no significant differences were observed in the efficiency of decoding. This is because the users were more effective using EASYDECLARE but at the cost of more time taken to respond. $H2_0$ is accepted, thus no differences were found between the two notations for encoding.

Learnability. Subsequently, subjects were provided with a list of templates and their relative graphical representations. The data show no statistically significant differences in the time the subjects spent on the two notations (Fig. 11(a).E1). Subsequent decoding tasks show no significant differences in efficiency and effectiveness. Conversely, template encoding shows significant differences between the two notations (Fig. 10(c).T2, Fig. 10(d).T2), with EASYDECLARE having better performance both in terms of effectiveness ($\mu_V = 9.7$, $\mu_D = 8.8$) and efficiency ($\mu_V = 1.6$, $\mu_D = 1.3$).

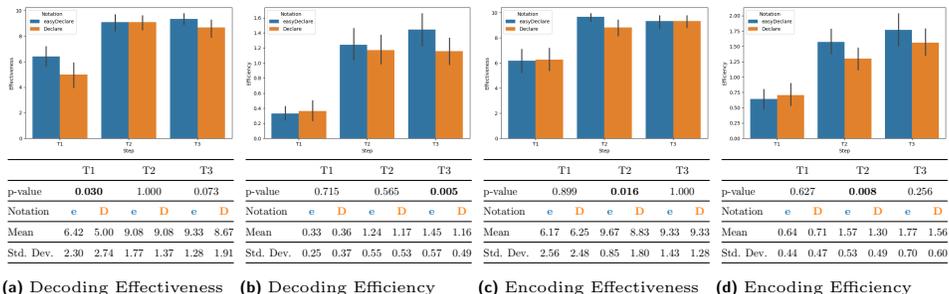


Figure 10: Templates encoding and decoding

The subjects were then provided with a list of templates and their relative graphical representations, along with a brief explanation of the rationale behind the notation. The time spent on the two notations is different in this case (Fig. 11(a).E2), with that devoted to EASYDECLARE ($\mu_V = 32.4$) significantly lower than that devoted to DECLARE ($\mu_D = 61.4$). The last block of template decoding and encoding tasks does not show significant differences. The only difference appears in decoding, with the efficiency of EASYDECLARE ($\mu_V = 1.45$) higher than the efficiency with DECLARE ($\mu_D = 1.16$).

Results: **H3₀** is almost fully accepted; however, the third time the subjects had to decode the templates, they were more efficient using EASYDECLARE. **H4₀** is partially rejected; after the first explanation of the notations, subjects showed that they were more effective and more efficient using EASYDECLARE. However, these differences are not significant after further explanation of the notations. The analysis of the time spent analyzing the notations seems to confirm these results. While the time spent on the first explanation does not show significant differences, the time spent on the second explanation is significantly higher for DECLARE, suggesting that the subjects were less confident with this notation. **H5₀** is thus partially rejected.

Understandability. The tasks dealing with the decoding and encoding of entire models are where the most significant differences emerged (see Fig. 11(b), Fig. 11(c)). Decoding performance with EASYDECLARE was better than with DECLARE both for effectiveness and efficiency. With EASYDECLARE, subjects scored on average $\mu_V = 8.2$ for effectiveness and $\mu_V = 0.29$ for efficiency, whereas with DECLARE the performance was much lower ($\mu_D = 2.9$ for effectiveness and $\mu_D = 0.11$ for efficiency). Similar performances were

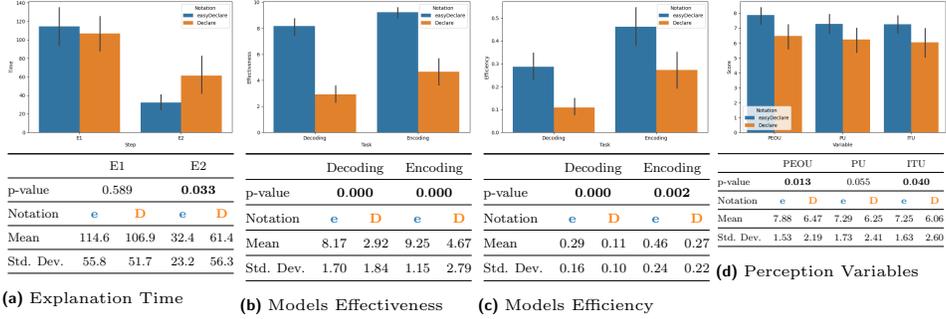


Figure 11: Explanation times, models encoding and decoding, perception-based variables

observed in model encoding. The effectiveness of EASYDECLARE ($\mu_V = 9.3$) was much higher than that of DECLARE ($\mu_D = 4.7$); similarly, the efficiency of the former ($\mu_V = 0.46$) was higher than the efficiency of the latter ($\mu_D = 0.27$).

Results: Both hypotheses on understandability are rejected. The differences shown are substantial for both decoding and encoding models in terms of effectiveness (**H6**₀) and efficiency (**H7**₀). Although they cannot be considered conclusive, these results seem to confirm our hypothesis that using DECLARE is progressively more difficult as the complexity of the task increases. EASYDECLARE, on the other hand, seemed to suffer less from this problem, with subjects maintaining consistently good performance even on these more complex tasks.

Perception-based variables. Perception-based variables show some differences between the two notations (see Fig. 11(d)). More specifically, the PEOU of EASYDECLARE ($\mu_V = 7.88$) is higher than that of DECLARE ($\mu_D = 6.47$). Similarly, the ITU of EASYDECLARE ($\mu_V = 7.25$) is higher than that of DECLARE ($\mu_D = 6.06$), while the PU does not show significant differences.

Results: **H8**₀ is rejected, suggesting that the Perceived Ease of Use of EASYDECLARE is higher than that of DECLARE. Similar considerations arise for the Intention to Use, as **H10**₀ is rejected. Conversely, the Perceived Usefulness does not show significant differences, as **H9**₀ is confirmed.

5. EDD – EasyDeclare Designer

We developed EDD – *EasyDeclare Designer* – a tool to support the creation of models based on our proposed graphical notation. EDD is an open-source web application implemented in JavaScript, freely available along with its source code.² Further details about the tool, including features and technical specifications, are reported in [Section 5.1](#).

To evaluate the usability of EDD, we conducted a user study using the System Usability Scale (SUS), a common approach for measuring the usability of systems, providing a quantitative score that reflects the overall user experience and assesses their perceived usability. The results of the SUS indicated a high usability level, achieving an A+ rating. Details about this evaluation are reported in [Section 5.2](#).

5.1. Tool Overview

The interface of EDD has been designed to be simple and efficient. As shown in [Fig. 12](#), it is composed of three panels: A) *Canvas Pane*, B) *Details Pane*, and C) *Overview Pane*.

The main pane is the *Canvas* (A) on the left side of the interface, which shows the model. This pane also offers an interactive interface for model creation and manipulation. Users can create activities by double-clicking on an empty area of the canvas. To create binary constraints between activities, users can mouse over the border of an activity and drag it to another activity. Although the tool automatically arranges activities and constraints on the canvas to maintain an organized layout, users can manually adjust the positions by dragging these elements across the canvas, allowing for a custom layout. Furthermore, the canvas supports zoom and pan functionalities, supporting users to manage models of various sizes.

On the right side of the interface is the *Details Pane* (B), which provides an overview of the existing activities and constraints in the model. This pane also shows details about each activity and constraint and allows the editing of their properties.

In the bottom right corner of the interface, there is the *Overview Pane* (C). This pane shows the overview of the entire model, highlighting which part is currently visible in the canvas, following the focus+context visual paradigm. This feature helps users maintain a view of the entire model even if they zoom in and out on the canvas to work on different areas.

² <https://github.com/blasilli/easyDeclare>

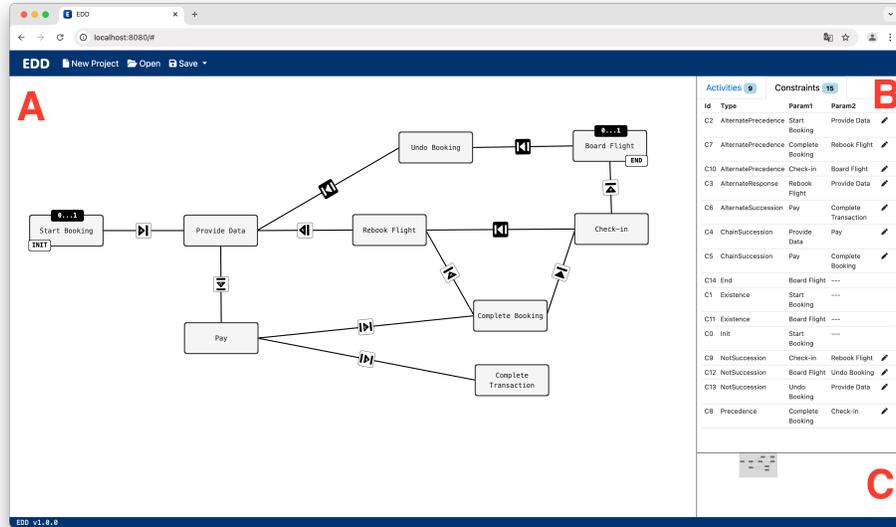


Figure 12: An overview of the EDD interface. It is composed of three panels: A) *Canvas Pane*, B) *Details Pane*, and C) *Overview Pane*.

To allow for the interoperability of the tool, we implemented existing standards for the input and output of model files, such as *decl* [37] and *RuM* [3]. Additionally, we defined a new input/output format named *edj*, which is based on JSON and contains information about activities, constraints, and layout. This format is detailed in the tool repository.

5.2. Usability Evaluation

To evaluate the usability of EDD, we conducted a user study using the System Usability Scale (SUS) [6, 7]. The SUS is a questionnaire composed of ten questions designed to assess the perceived usability of a system. The user, after having performed tasks on the system, is required to answer the 10 questions with a value on a five-level Likert scale, ranging from “*strongly disagree*” to “*strongly agree*”. The answers are then mapped into a numerical value ranging from [0, 10], and their sum becomes the SUS score assigned to the system, which ranges from [0, 100]. Additionally, Lewis and Sauro [25] defined a scale mapping the SUS score into 11 grades from higher to lower: [A+, A, A-, B+, B, B-, C+, C, C-, D, F].

The ten questions of the SUS are reported in the following:

Q1. I think that I would use this system frequently;

- Q2. I found the system unnecessarily complex;
- Q3. I thought the system was easy to use;
- Q4. I think that I would need the support of a technical person to be able to use this system;
- Q5. I found the various functions in the system were well integrated;
- Q6. I thought there was too much inconsistency in this system;
- Q7. I would imagine that most people would learn to use this system very quickly;
- Q8. I found the system very awkward to use;
- Q9. I felt very confident using this system;
- Q10. I need to learn a lot of things before I could get going with this system.

The user study involved 18 researchers and practitioners in the process mining field composed of: 1 associate professor, 3 assistant professors, 10 PhD students, 2 master students, 1 bachelor student, and 1 process mining consultant.

5.2.1. Methodology

Participants were first asked to evaluate their expertise in three areas: a) declarative process modeling and related graphical notations, b) imperative process modeling and related graphical notations, and c) BPMN and related graphical notations. This was done using a five-level Likert scale ranging from “*not competent*” to “*highly competent*”. Next, we briefly reviewed the basic concepts of Declarative Process Modeling and illustrated our proposed visual notation. We then presented EDD, explained its functionalities, and allowed the participants ten minutes to familiarize themselves with the tool.

To test the system functionalities, we designed two tasks (T1 and T2) for the users to complete. After completing the tasks, participants were asked to fill out the SUS questionnaire. Finally, they were invited to share their comments and thoughts on the tool.

T1. The first task is designed to familiarize users with the system by creating a simple model from scratch. Users were asked to model an Emergency Room procedure as follows:

1. Open the tool and create a new project.
2. Add three new activities named *Registration*, *Admission*, and *Triage*.
3. Add a INIT unary constraint to the activity *Registration*.
4. Add an EXISTENCE constraint with cardinality 0...1 to the activity *Registration*.
5. Add a binary constraint ALTERNATERESPONSE(*Registration*, *Admission*).

6. Add a binary constraint $SUCCESSION(Registration, Triage)$.
7. Add a binary constraint $PRECEDENCE(Triage, Admission)$.
8. Save the model to a local file.

T2. The second task involves modifying an existing and more complex model (shown in Fig. 12), which represents a booking procedure for flight tickets. Users were asked to:

1. Open the “flight.edj” file in the tool.
2. Add a INIT unary constraint to the activity *StartBooking*.
3. Add a END unary constraint to the activity *BoardFlight*.
4. Add a new activity named *UndoBooking*.
5. Add a binary constraint $NOTSUCCESSION(BoardFlight, UndoBooking)$.
6. Add a binary constraint $NOTSUCCESSION(UndoBooking, ProvideData)$.
7. Transform the $CHOICE(ProvideData, Pay)$ into $CHAINSUCCESSION(ProvideData, Pay)$.
8. Invert the order of activities in the $ALTERNATESUCCESSION(CompleteTransaction, Pay)$ to $ALTERNATESUCCESSION(Pay, CompleteTransaction)$.
9. Save the model to a local file.

The correct model, after the required modifications, is shown in Fig. 8.

5.2.2. Results

Participants’ expertise was assessed using a five-level Likert scale, where “*not competent*” was encoded as 0 and “*highly competent*” as 10. The expertise levels were as follows (with μ and σ being the average score and the standard deviation, respectively):

- Declarative process modeling: theory $\mu = 5.56$, $\sigma = 3.16$; related graphical notations $\mu = 5.42$, $\sigma = 3.56$.
- Imperative process modeling: theory $\mu = 5.52$, $\sigma = 3.56$; related graphical notations $\mu = 5.42$, $\sigma = 3.76$.
- BPMN: theory $\mu = 7.36$, $\sigma = 2.5$; related graphical notations $\mu = 7.08$, $\sigma = 2.88$.

Overall, the SUS results demonstrate a highly positive evaluation of the tool’s usability, with an average grade of A+ and an average score of $\mu = 87.64$ and a median of 90. This indicates that users generally found the system very usable. The standard deviation of $\sigma = 7.54$ suggests relatively low variability in scores, meaning that most users had a similar usability

experience with the tool. Detailed question and score results are shown in Fig. 13.

Interestingly, question Q1 — *I think that I would use this system frequently* — reported the lowest median among all questions. This observation suggests that the low median does not necessarily reflect the tool’s usability but may be related to the participants’ work patterns. Many participants did not need to use the tool regularly, as their roles did not require frequent creation or editing of models.

Participants provided several positive comments and valuable suggestions for improving the tool. Users generally appreciated its visual design and usability. For example, one user commented, “*Very nice visual tool, with clear feedback and high usability level*”. Another noted, “*In general, I found the tool very simple to use*”.

Despite these positive remarks, users also offered constructive feedback for enhancements. One suggestion was to “*add support for multiple projects open at the same time in different tabs*”, which would improve multitasking capabilities. Another user suggested, “*It would be helpful to double-click on an activity to rename it*”, indicating a potential improvement in interaction design. Additionally, implementing keyboard shortcuts, such as “*clicking on an activity and pressing the delete key to remove the activity*”, was also recommended.

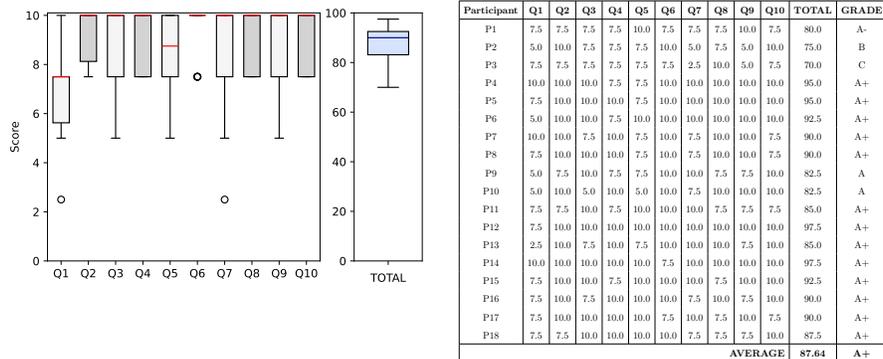


Figure 13: System Usability Scale (SUS) results for EDD. These results indicate a high usability level, with an average rating of A+ (score 87.64).

6. Conclusion

In this paper, we introduced and evaluated a new graphical notation, EASYDECLARE, designed to improve the interpretability of declarative pro-

cess discovery results. Traditional process modeling languages, while effective for specialists, often pose comprehension challenges for non-experts due to their complexity and technical nature.

Our controlled experiment evaluated the proposed graphical notation against the pre-existing standard, DECLARE, focusing on performance and perception metrics.

In decoding tasks, EASYDECLARE showed higher effectiveness than DECLARE, although it took more time, indicating better accuracy but reduced efficiency. For encoding tasks, there were no significant differences in effectiveness or efficiency between the two notations. There was no significant difference in the initial time spent learning both notations. However, EASYDECLARE required significantly less time for subsequent explanations. In template encoding tasks, EASYDECLARE showed better performance after initial explanations, though differences became insignificant after further explanations. In both decoding and encoding tasks for models, EASYDECLARE outperformed DECLARE in effectiveness and efficiency, suggesting it handles complexity better. EASYDECLARE was perceived as easier to use (PEOU) and users were more likely to use it in the future (ITU). There were no significant differences in perceived usefulness (PU) between the two notations.

In conclusion, EASYDECLARE demonstrated superior effectiveness and efficiency in complex tasks and higher user satisfaction in ease of use and intention to use, making it a promising alternative to the traditional graphical notation provided by DECLARE for declarative process modeling. As a future work, we aim to integrate EASYDECLARE in a declarative process mining suite like *RuM* [3] to widen its usage among practitioners and perform further longitudinal user tests.

References

- [1] Abrahão, S., Insfran, E., Carsí, J. A., and Genero, M. (2011). Evaluating Requirements Modeling Methods Based on User Perceptions: A Family of Experiments. *Information Sciences*, 181(16):3356–3378.
- [2] Agostinelli, S., Chiariello, F., Maggi, F. M., Marrella, A., and Patrizi, F. (2023). Process mining meets model learning: Discovering deterministic finite state automata from event logs for business process analysis. *Inf. Syst.*, 114.
- [3] Alman, A., Ciccio, C. D., Haas, D., Maggi, F. M., and Nolte, A. (2020).

- Rule mining with rum. In *2020 2nd Int. Conf. on Process Mining (ICPM)*, pages 121–128.
- [4] Basili, V. R., Caldiera, G., and Rombach, H. D. (1994). The Goal Question Metric Approach. In *Encyclopedia of Software Engineering*, pages 646–661. Wiley.
- [5] Bodart, F., Patel, A., Sim, M., and Weber, R. (2001). Should Optional Properties Be Used in Conceptual Modelling? A Theory and Three Empirical Tests. *Information Systems Research*, 12(4):384–405.
- [6] Brooke, J. (1996). SUS: a quick and dirty usability scale. *Usability evaluation in industry*, 189(3):189–194.
- [7] Brooke, J. (2013). SUS: a retrospective. *Journal of usability studies*, 8(2):29–40.
- [8] Burattin, A., Cimitile, M., Maggi, F. M., and Sperduti, A. (2015). Online discovery of declarative process models from event streams. *IEEE Transactions on Services Computing*, 8(6):833–846.
- [9] da Silva Teixeira, M. d. G., Quirino, G. K., Gailly, F., de Almeida Falbo, R., Guizzardi, G., and Barcellos, M. P. (2016). PoN-S: a systematic approach for applying the physics of notation (PoN). In *Enterprise, Business-Process and Information Systems Modeling*, pages 432–447. Springer.
- [10] De Giacomo, G. and Vardi, M. Y. (2013). Linear temporal logic and linear dynamic logic on finite traces. In *Twenty-Third Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 854–860.
- [11] Di Ciccio, C., Catarci, T., and Mecella, M. (2011). Representing and visualizing mined artful processes in MailOfMine. In *HCI-KDD*, pages 83–94. Springer.
- [12] Di Ciccio, C., Maggi, F. M., and Mendling, J. (2016). Efficient discovery of Target-Branched Declare constraints. *Inf. Syst.*, 56:258–283.
- [13] Di Ciccio, C. and Mecella, M. (2015). On the discovery of declarative control flows for artful processes. *ACM Trans. Manage. Inf. Syst.*, 5(4):24:1–24:37.
- [14] Diamantopoulou, V. and Mouratidis, H. (2018). Applying the physics of notation to the evaluation of a security and privacy requirements engineering methodology. *Information & Computer Security*, 26(4).

- [15] Dumas, M., La Rosa, M., Mendling, J., and Reijers, H. A. (2013). *Fundamentals of Business Process Management*. Springer.
- [16] Fahland, D., Mendling, J., Reijers, H. A., Weber, B., Weidlich, M., and Zugal, S. (2010). Declarative versus Imperative Process Modeling Languages: The Issue of Maintainability. In *Business Process Management Workshops*, pages 477–488.
- [17] Ferro, L. S. and Marrella, A. (2018). VERTO: A visual notation for declarative process models. In *2018 Int. Conf. on Advanced Visual Interfaces (AVI)*, page 62. ACM.
- [18] Gemino, A. and Wand, Y. (2005). Complexity and Clarity in Conceptual Modeling: Comparison of Mandatory and Optional Properties. *Data & Knowledge Engineering*, 55(3):301–326.
- [19] Genon, N., Heymans, P., and Amyot, D. (2010). Analysing the cognitive effectiveness of the bpmn 2.0 visual notation. In *Int. Conf. on Software Language Eng. (SLE)*, pages 377–396.
- [20] Green, T. R., Blandford, A. E., Church, L., Roast, C. R., and Clarke, S. (2006). Cognitive dimensions: Achievements, new directions, and open questions. *Journal of Visual Languages & Computing*, 17(4):328–365.
- [21] Hanser, M., Ciccio, C. D., and Mendling, J. (2016). A New Notational Framework for Declarative Process Modeling. *Softwaretechnik-Trends*, 36(2).
- [22] Höst, M., Regnell, B., and Wohlin, C. (2000). Using Students as Subjects—A Comparative Study of Students and Professionals in Lead-Time Impact Assessment. *Empirical Software Engineering*, 5(3):201–214.
- [23] Kitchenham, B., Pfleeger, S., Pickard, L., Jones, P., Hoaglin, D., El Emam, K., and Rosenberg, J. (2002). Preliminary Guidelines for Empirical Research in Software Engineering. *IEEE Transactions on Software Engineering*, 28(8):721–734.
- [24] Krogstie, J. (2016). Sequel specialized for business process models. In *Quality in Business Process Modeling*, pages 103–138. Springer.
- [25] Lewis, J. R. and Sauro, J. (2018). Item benchmarks for the system usability scale. *Journal of Usability Studies*, 13(3).

- [26] Lopez A., H. A. and Simon, V. (2022). How to (re)design declarative process notations? A view from the lens of cognitive effectiveness frameworks. In *15th IFIP Working Conference on the Practice of Enterprise Modeling 2022 (PoEM-Forum 2022)*.
- [27] Maggi, F. M. (2014). Discovering metric temporal business constraints from event logs. In *13th Int. Conf. on Perspectives in Business Informatics Research (BIR)*, pages 261–275.
- [28] Maggi, F. M., Bose, R. P. J. C., and van der Aalst, W. M. P. (2012). Efficient discovery of understandable declarative process models from event logs. In *24th Int. Conf. on Advanced Information Systems Engineering (CAiSE)*, pages 270–285. Springer.
- [29] Maggi, F. M., Bose, R. P. J. C., and van der Aalst, W. M. P. (2013). A knowledge-based integrated approach for discovering and repairing Declare maps. In *25th Int. Conf. on Advanced Information Systems Engineering (CAiSE)*, pages 433–448.
- [30] Maggi, F. M., Di Ciccio, C., Di Francescomarino, C., and Kala, T. (2018). Parallel algorithms for the automated discovery of declarative process models. *Information Systems*, 74:136–152.
- [31] Maggi, F. M., Mooij, A. J., and van der Aalst, W. M. P. (2011). User-guided discovery of declarative process models. In *IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*, pages 192–199.
- [32] Moody, D. (2003). The Method Evaluation Model: A Theoretical Model for Validating Information Systems Design Methods. *11th European Conf. on Information Systems (ECIS)*.
- [33] Moody, D. (2009). The “physics” of notations: Toward a scientific basis for constructing visual notations in software engineering. *IEEE Transactions on Software Engineering*, 35(6):756–779.
- [34] Popescu, G. and Wegmann, A. (2014). Using the physics of notations theory to evaluate the visual notation of seam. In *2014 IEEE 16th Conf. on Business Informatics (CBI)*, volume 2, pages 166–173.
- [35] Schönig, S., Di Ciccio, C., Maggi, F. M., and Mendling, J. (2016a). Discovery of multi-perspective declarative process models. In *14th Int. Conf. on Service Oriented Computing (ICSOC)*, pages 87–103.

- [36] Schönig, S., Rogge-Solti, A., Cabanillas, C., Jablonski, S., and Mendling, J. (2016b). Efficient and customisable declarative process mining with SQL. In *28th Int. Conf. on Advanced Information Systems Engineering (CAiSE)*, pages 290–305.
- [37] Skydaniienko, V., Di Francescomarino, C., Ghidini, C., and Maggi, F. M. (2018). A tool for generating event logs from multi-perspective declare models. volume 2196 of *CEUR Workshop Proceedings*.
- [38] Vaandrager, F. W. (2017). Model learning. *Commun. ACM*, 60(2):86–95.
- [39] van Der Aalst, W. M., Pesic, M., and Schonenberg, H. (2009). Declarative workflows: Balancing between flexibility and support. *Computer Science-Research and Development*, 23(2):99–113.
- [40] van der Aalst, W. M. P. (2011). *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer.
- [41] van der Aalst, W. M. P., Pesic, M., and Schonenberg, H. (2009). Declarative workflows: Balancing between flexibility and support. *Computer Science - R&D*, 23(2):99–113.
- [42] Van Der Linden, D., Zamansky, A., and Hadar, I. (2017). A framework for improving the verifiability of visual notation design grounded in the physics of notations. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 41–50. IEEE.
- [43] vanden Broucke, S. K. L. M., Vanthienen, J., and Baesens, B. (2014). Declarative process discovery with evolutionary computing. In *2014 IEEE Congress on Evolutionary Computation (CEC)*, pages 2412–2419.
- [44] Verbeek, H. M. W., Buijs, J. C. A. M., van Dongen, B. F., and van der Aalst, W. M. P. (2010). XES, XESame, and ProM 6. In *CAiSE Forum 2010*, pages 60–75.