A New Notational Framework for Declarative Process Modeling

Michael Hanser, Claudio Di Ciccio and Jan Mendling

Vienna University of Business and Economics, Austria
michaelhanser@gmx.net, {claudio.di.ciccio, jan.mendling}@wu.ac.at

Abstract

In order to capture flexible scenarios, a declarative approach to business process modeling describes constraints that limit a process' behavior instead of specifying all its allowed enactments. However, current graphical notations for declarative processes are tough to understand, thus hampering a widespread usage of the approach. To overcome this issue, we present a novel notational framework for representing declarative processes, devised in compliance with well-known notation design principles.

1 Introduction

Owing to the increasing necessity of business processes to retain a high degree of flexibility, a declarative process modeling approach seeks to address the issue of current modeling languages' lack of supporting flexible scenarios [13]. Contrary to the commonly used imperative paradigm of process modeling, the declarative approach does not enforce a strict order of activities, but limits their behavior by using constraints. However, since this approach is considered less intuitive [6], a modeling language and notation capable of conveying concepts in an efficient manner is necessary. Current state of the art solutions struggle with effectively communicating explicit principles of how to interpret a declarative process model. Results in existing literature [6, 7] suggest that a new notation, facilitating understandability, is needed. The notation presented in this paper is designed to ease the process of comprehending declarative process models. Being developed in compliance with respected notation design principles [12], it offers a set of consistent mechanisms to efficiently communicate semantic constructs. The framework contributes to existing literature as it builds upon, refines and extends the notation approaches presented in [8, 13, 3, 4, 5]. More specifically, this paper succinctly summarizes the findings presented in [8] and extends them by providing a comparative discussion that shows why the new notation is superior to the old one.

This paper is structured as follows. Section 2 outlines the new notation and briefly summarizes its background. Section 3 then compares and contrasts both the old and new declarative notation by means of exemplary process models. Section 4 concludes the paper.

2 Notation

A declarative process model allows any order, repetition or absence of activities that does not violate the constraints in the model. Declare [3], a well-known language for declarative process modeling, offers a predefined set of *constraint templates*, each of them consisting of a unique name, a graphical representation and a formal semantic specification in terms of Linear Temporal Logic (LTL) [15, 1, 2].

Graphic notations such as the one of Declare can be evaluated using Moody's principles for designing *cognitively effective* notations [12], which relate to the speed, ease and accuracy by which the human mind can process and interpret a notation [9]. These principles are:

1. **Semiotic Clarity**: semantic constructs have a 1:1 correspondence with respective graphical symbols.
2. **Perceptual Discriminability**: symbols can be clearly distinguished.
3. **Semantic Transparency**: graphical representations suggest their meaning.
4. **Complexity Management**: explicit mechanisms for dealing with complexity exist.
5. **Cognitive Integration**: the integration of information from different diagrams is supported.
6. **Visual Expressiveness**: full range and capacities of visual variables is used.
7. **Dual Coding**: text complements graphical symbols.
8. **Graphic Economy**: the number of symbols is cognitively manageable.
9. **Cognitive Fit**: different visual dialects exist for different purposes.

Building upon the work of Di Ciccio et al. in [4, 5], the new notation employs two corresponding views on a process: *(i)* a static, multi-level *global view*, illustrating the entire process at once and *(ii)* a *local view*, focusing on one activity and its directly related constraints and implications at a time. The static multi-level *global view* serves as a way of regarding an entire process scheme at once. Within this view, the notation provides for different levels of granularity, thus increasing readability at first sight. Note that, for the sake of conciseness, this paper only focuses on the global view and its more detailed "standard" granularity level. The interested reader is referred to [8].

The notation's rationale is based on a network topology-like alignment of activities, which are accordingly depicted by means of circular elements and complemented by full text identifiers. Declare constraints are divided into *(i) Existence constraints*, specifying the cardinality of a task or the first and last activity in a trace; *(ii) Relation constraints*, making an activity's behavior depend on the one of another task; *(iii) Mutual Relation constraints*, which build upon Relation constraints but further cover the converse behavior, i.e., both activities depend on their respective others; and *(iv) Negation constraints*, representing negated versions of Relation or Mutual Relation constraints.

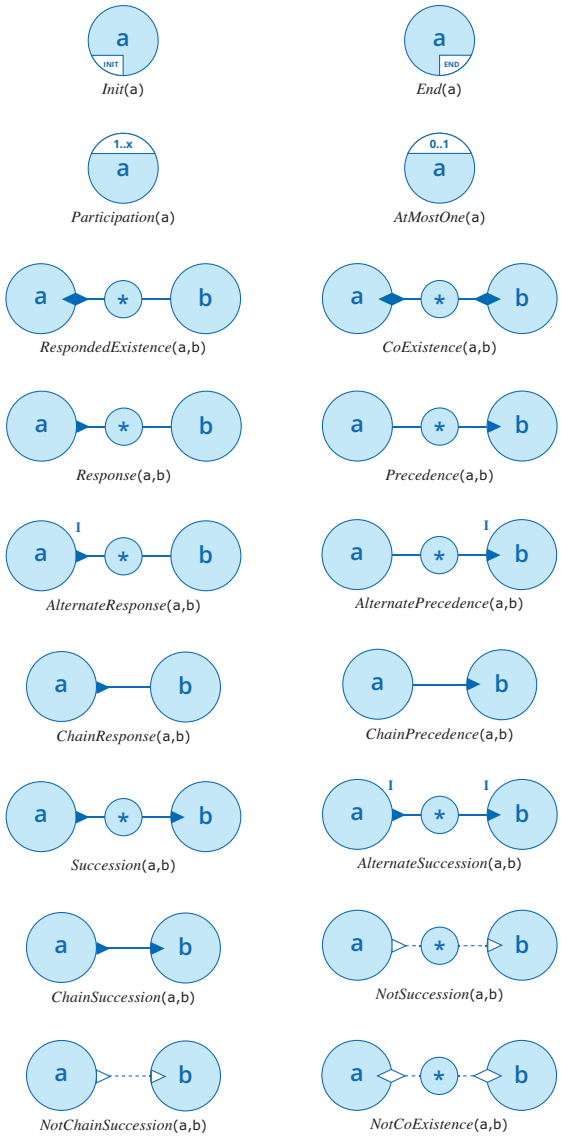


Figure 1: Declare constraints by means of the new notation.

As shown in Figure 1, Existence constraints are illustrated by placing text annotations within an activity element. *Participation*(a), for instance, is an Existence constraint specifying that activity a must be performed at least *once*. Similarly, *AtMostOne*(a) prescribes that this activity can only be performed either *zero* times or once. The notation depicts such constraints that prescribe the cardinality of a task by adding text to the upper half of the circular element. If the constraint specifies the first or last activity in a process, *Init*(a) or *End*(b) respectively, it is indicated by an annotation in the lower left or right part of the element.

Relation constraints are embodied by utilizing *solid* lines and cursors between activities. Conversely, a constraint involving a *dashed* line always implies its belonging to the group of Negation constraints. *Response*(a,b) is a Relation constraint, which prescribes that activity a must eventually be followed by activity b. Dually, *Precedence*(a,b) imposes that b must be preceded by a. *Succession*(a,b) depicts a combination of the former and the latter, i.e., every activity a must be succeeded by b and every activity b must be preceded by a, thus being a Mutual Relation constraint. These three constraints can be further strengthened by using the *Alternation* and *Chain* limitation. The concept of *Alternate* constraints indicates that the activating task can not reoccur without having the other task executed in between. Similarly, *Chain* constraints represent an even stricter limitation as they prohibit the execution of *any* other activity in between.

Relation constraints are perceived as "if-then" statements: The "if part", namely the *activation*, is complemented by a cursor, placed pointing either *inwards* or *outwards* of the activating task circle, depending on the sequence-verse of the constraint. This suggests that, if the cursor points inwards, the respective *target* activity ("then-part") must have been executed *before* the activation task can be performed. Conversely, if the cursor points outwards, the target activity must happen *after* the activation task is completed. Applying this to the *Response*(a,b) constraint consequently implies that, since a is the activation task of the constraint, the cursor is placed at this very activity. Furthermore, as it specifies that the respective target activity b must eventually be performed afterwards, the cursor is placed *outwards* on the activity border.

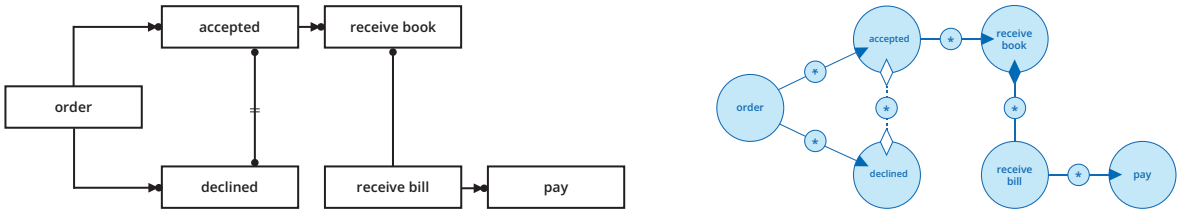
In case a constraint allows executions of further tasks in between, these optional activities are visualized by means of smaller circles and complemented by an asterisk (∗), referring to "any other activity". Moreover, to indicate an *Alternate* limitation, the Roman symbol for 1 ("I") is added to the activation part of the constraint. Acting as a counter, it states that this very activity is allowed to only happen once until the target task is performed.

Certain Relation constraints may signify the correlated execution of activities, with no restriction on their temporal order. *RespondedExistence*(a,b), e.g., specifies that the execution of activity a also requires activity b to happen at some point in the process. Similarly, *CoExistence*(a,b) also includes the converse behavior, i.e., implying that the occurrence of a or b always implies the occurrence of one another. In order to illustrate such constraints, the notation employs two connected cursors, thus forming a diamond, which is placed at the activation of the constraint.

Finally, Negation constraints are based on existing Mutual Relation constraints, depicting their respective negated form. As already outlined, these constraints are always indicated by dashed lines and empty cursors. The *NotSuccession*(a,b) constraint, e.g., states that activity a must *never* be succeeded by b and b must *never* be preceded by a – thus stating the opposite of *Succession*(a,b). Likewise, *NotChainSuccession*(a,b) expresses that a and b *cannot* occur after one another, as opposed to *ChainSuccession*(a,b). *NotCoExistence*(a,b) imposes that a and b are *not* allowed to occur in the same trace.

3 Discussion

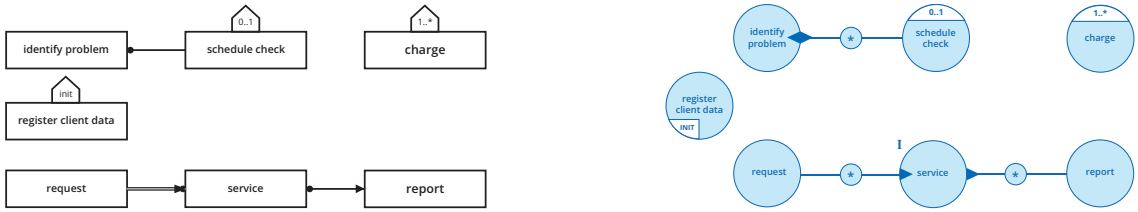
To demonstrate how the novel notational framework is applied in an existing declarative process model, Figure 2 illustrates a simplified process of ordering a book, inspired



(a) Old notation

(b) New notation

Figure 2: An order process, inspired by the work in [13]



(a) Old notation

(b) New notation

Figure 3: A car rental process, inspired by [14]

by the work in [13]. Figure 2a depicts the process by means of the old notation provided by Declare, whilst Figure 2b represents a visualization in terms of the new framework.

The order process comprises six activities: order, accepted, declined, receive book, receive bill and pay. Initially, an order must be placed to either be accepted or declined. To receive the book, the order must be accepted. Moreover, receiving the book implies that a bill must be obtained at some point. One may only pay, if he/she is in possession of a bill. Note that receive bill may occur both *before* or *after* receive book. Accordingly, the activities' behavior is limited by the constraints *Precedence*(order, accepted), *Precedence*(order, declined), *NotCoExistence*(accepted, declined), *Precedence*(accepted, receive book), *RespondedExistence*(receive book, receive bill) and *Precedence*(receive bill, pay).

Most notably, the two process models differ with respect to how each of them visualizes Relation constraints, e.g., *Precendece*(order, accepted). Indeed, both notations employ an underlying "if-then" rationale to assemble constructs. However, while the old visualization approach utilizes dots and arrows for respectively specifying the activation and the temporal order (namely, the task occurring later), the new notation accomplishes to combine both concepts into one symbol – a cursor, either placed inwards or outwards of the activating task. Consequently, when using the old notation, a user is forced to navigate through the arcs between activities to identify both concepts, as shown in Figure 2a. In contrast, the new notational framework only needs one symbol per constraint to convey this information and therefore results in a more compact and efficient way of expressing constraints. A less spatially sparse distribution of information turns out to be highly valuable, especially when diagrams increase in size and complexity. A notation's ability to provide concise and explicit mechanisms for dealing with complexity relate to Moody's *principle of complexity management* [12], which is more properly respected in the new notational framework.

The way both notations visualize Negation constraints depicts another key difference, being connected to the *principle of perceptual discriminability* [12], which strongly advises notations to clearly distinguish between symbols. As shown in Figure 2a, *NotCoExistence*(accepted, declined) only differs from *CoExistence* by crossing out the connecting line between activities, thus indicating its negated semantics. Though the underlying principle appears readily understandable, its representation is likely to be overlooked at first sight. Particularly as process models become more complex, identifying Negation constraints is bound to become too difficult for it to be effective. Conversely, Figure 2b suggests that the new notation visualizes Negation constraints by swapping solid cursors with empty ones and solid lines with dashed connectors. Clearly, this visualization approach facilitates the act of contrasting positive and negative constraints in a model at first glance, thus respecting the principle of perceptual discriminability to a higher degree.

Furthermore, the constraints *Precedence*(order, accepted) and *Precedence*(order, declined) in Figure 2b accurately reflect how the circular shaping of activities complies with the *principle of visual expressiveness*. The full range and capacities of a circular rationale is utilized to reduce the connecting lines' bending points and make the model appear more clearly arranged and visually appealing.

Figure 3 illustrates a car rental process, inspired by the work in [14], which is, again, visualized using both the old and new notation. The Existence constraints *Init*(register client data), *AtMostOne*(schedule check) and *Participation*(charge) demonstrate the fact that both notations equally respect the *principle of dual coding* as they com-

plement graphical symbols for Existence constraints with text identifiers. Yet, when looking at how these text identifiers are incorporated into the visual constructs, it appears that Figure 3b manages to include them *within* the activity element, resulting in a much cleaner and smoother image. Contrarily, the old notation requires an additional element to be put on top of the activity. Resultantly, the new notation accomplishes to keep the number of graphical symbols lower than the old notation and therefore better complies with the *principle of graphic economy*.

Finally, the representation of the constraint *AlternatePrecedence*(request, service) suggests that the new notation better complies with the *principle of semantic transparency*. In contrast to *Precedence*(a, b), the alternation of *AlternatePrecedence*(a, b) resides in the fact b cannot reoccur until a is executed again. As shown in Figure 3b, the new notation indicates this alternation by adding the counter "I" to the activation-part service. Just as one would assume, "I" indeed suggests its meaning – the activity complemented by it may only be performed *once* until the other one reoccurs. Hence, it respects the *principle of semantic transparency*, whereas the old notation's representation of the constraint is unable to provide any support for this principle.

4 Conclusion

In this paper, we presented a novel conceptual framework for representing declarative process models. By utilizing exemplary process models inspired by sample processes in existing literature [13, 14], we compared and contrasted both the old notation and new notation by means of Moody's notational design principles [12] and showed how the proposed framework is superior to the prevailing visualization approach. As this work is only concerned with the conceptual design and comparison of notations, future research investigating and empirically evaluating the framework is needed. In the context of process mining, the possibility of scaling the size of activity circles could be used to emphasize reoccurring activities and constraints in a model, as first addressed in [10]. Studies on the guidelines of declarative process modeling could be established, as already existing for imperative languages [11].

References

[1] De Giacomo, G., De Masellis, R., Montali, M.: Reasoning on LTL on finite traces: Insensitivity to infiniteness. In: Proceedings of the 28th AAAI Conference on AI. pp. 1027–1033 (2014)

[2] De Giacomo, G., Vardi, M.Y.: Linear temporal logic and linear dynamic logic on finite traces. In: Proceedings of the 23rd international joint conference on AI. pp. 854–860. AAAI Press (2013)

[3] van Der Aalst, W.M., Pesic, M., Schonenberg, H.: Declarative workflows: Balancing between flexibility and support. Computer Science-Research and Development 23(2), 99–113 (2009)

[4] Di Ciccio, C., Mecella, M., Catarci, T.: Representing and visualizing mined artful processes in MailOfMine. Springer (2011)

[5] Di Ciccio, C., Mecella, M., Scannapieco, M., Zardetto, D., Catarci, T.: MailOfMine – Analyzing mail messages for mining artful collaborative processes. In: Data-Driven Process Discovery and Analysis, pp. 55–81. Springer (2011)

[6] Fahland, D., Lübke, D., Mendling, J., Reijers, H., Weber, B., Weidlich, M., Zugal, S.: Declarative versus imperative process modeling languages: The issue of understandability. In: Enterprise, BP and IS Modeling, pp. 353–366. Springer (2009)

[7] Fahland, D., Mendling, J., Reijers, H.A., Weber, B., Weidlich, M., Zugal, S.: Declarative versus imperative process modeling languages: The issue of maintainability. In: BPM Workshops. vol. 43, pp. 477–488. Springer (2009)

[8] Hanser, M., Di Ciccio, C., Mendling, J.: A novel framework for visualizing declarative process models. In: ZEUS. CEUR Workshop Proceedings, vol. 1562, pp. 5–12. CEUR-WS.org (2016)

[9] Larkin, J.H., Simon, H.A.: Why a diagram is (sometimes) worth ten thousand words. Cognitive science 11(1), 65–100 (1987)

[10] Maggi, F.M., Bose, R.J.C., van der Aalst, W.M.: Efficient discovery of understandable declarative process models from event logs. In: Advanced IS Engineering. pp. 270–285. Springer (2012)

[11] Mendling, J., Reijers, H.A., van der Aalst, W.M.: Seven process modeling guidelines (7PMG). Information and Software Technology 52(2), 127–136 (2010)

[12] Moody, D.L.: The "physics" of notations: toward a scientific basis for constructing visual notations in software engineering. Software Engineering, IEEE Transactions on 35(6), 756–779 (2009)

[13] Pesic, M., Van der Aalst, W.M.: A declarative approach for flexible business processes management. In: BPM Workshops. pp. 169–180. Springer (2006)

[14] Pesic, M., Schonenberg, M., Sidorova, N., van der Aalst, W.M.: Constraint-based workflow models: Change made easy. In: On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS, pp. 77–94. Springer (2007)

[15] Pnueli, A.: The temporal logic of programs. In: Foundations of Computer Science, 18th Annual Symposium on. pp. 46–57. IEEE (1977)

This document is a pre-print copy of the manuscript
(Hanser, Di Ciccio, and Mendling 2016)
published in **journal**.

References

Hanser, Michael, Claudio Di Ciccio, and Jan Mendling (2016). "A New Notational Framework for Declarative Process Modeling". In: *Softwaretechnik-Trends* 36.2, pp. 53–56. URL: http://pi.informatik.uni-siegen.de/stt/36_2/03_Technische_Beitraege/ZEUS2016/beitrag_1.pdf.

BibTeX

```
@Article{         Hanser.etal/STT2016:NewNotationalFrameworkDeclare,
  author        = {Hanser, Michael and Di Ciccio, Claudio and Mendling, Jan},
  title         = {A New Notational Framework for Declarative Process
                   Modeling},
  journal       = {Softwaretechnik-Trends},
  year          = {2016},
  volume        = {36},
  number        = {2},
  pages         = {53--56},
  url           = {http://pi.informatik.uni-siegen.de/stt/36_2/03_Technische_Beitraege/ZEUS2016/beitrag_1.pdf}
}
```