A Novel Framework for Visualizing Declarative Process Models

Michael Hanser, Claudio Di Ciccio and Jan Mendling

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

Abstract The declarative approach to business process modeling has been introduced to deal with the issue of managing flexible processes. Instead of explicitly representing all the allowed enactments of a process, the approach describes the constraints that limit its behavior. However, current graphical notations for declarative processes are prone to be difficult to understand, thus hampering a widespread usage of the approach. To overcome this issue, we present a novel notation framework for visualizing declarative processes, which is devised in compliance with well-known notation design principles.

1 Introduction

Caused by an ever increasing demand for business processes to remain flexible, a declarative process modeling approach seeks to address the issue of current modeling languages lacking support for highly flexible scenarios [13]. Given the fact that a declarative approach is considered less intuitive and tougher to understand [6], a declarative modeling language and notation capable of conveying concepts in a quick and straight-forward manner is necessary. Current state of the art solutions struggle with effectively communicating explicit principles of how to interpret a declarative process model. Owing to the results in existing literature [6,7], a new notation, facilitating understandability and maintainability, is needed. The novel notation outlined in this paper is designed to ease the process of understanding declarative process models. Being developed in compliance with respected notation design principles [12], it offers a set of consistent and explicit mechanisms to effectively communicate semantic constructs. Our framework contributes to existing literature as it builds upon, refines and extends the notation approaches presented in [13,3,4,5].

This paper is structured as follows. Section 2 summarizes Declare and notational design. Section 3 describes the proposed notation and Section 4 discusses its notational quality. Section 5 concludes the paper.

2 Background

In contrast to the widely-used *imperative* paradigm of process modeling, a *declarative* modeling approach does not impose a strict order on activities, but

limits their behavior by using constraints. In fact, a declarative model allows any order, repetition or absence of activities, as long as it does not violate the constraints. As each constraint can either evaluate to *true* or *false* during the run time, the state of a process instance is *accepting*, and consequently considered *complete*, if and only if all constraints in the model evaluate to *true*. Declare 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) [14,1,2].

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. *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. Existence constraints are also used to mark the first and last activities in a process instance. *Init*(a) states that task a must be the first activity to be executed in a process instance. Likewise, the constraint *End*(a) indicates a as the very last activity to be performed. *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. For instance, *AlternatePrecedence*(a,b) forces activity b to be preceded by a, whilst allowing no further executions of b until a is performed again. Similarly, *Chain* constraints represent an even stricter limitation as they prohibit the execution of *any* other activity in between. E.g., *ChainSuccession*(a,b) forces activity a to be *directly* preceded by activity b and vice versa. Furthermore, certain Relation constraints signify the correlated execution of activities, with no restriction on their temporal order. *RespondedExistence*(a,b), for example, specifies that the execution of activity a also requires activity b to happen at some point in the process. Yet it does not matter whether this is before or after a occurs. Building upon the latter, *CoExistence*(a,b) also includes the converse behavior, thereby implying that the occurrence of a or b always implies the occurrence of one another. Ultimately, Negation constraints are based on existing Mutual Relation constraints, depicting their respective negated form. 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 – hence stating the opposite of *Succession*(a,b). Likewise, *NotChainSuccession*(a,b) states that a and b *cannot* occur one after the other, as

opposed to *ChainSuccession*(a,b). *NotCoExistence*(a,b) imposes that a and b are *not* allowed to occur in the same trace.

Visual notations such as the one of Declare can be evaluated using Moody's principles of *cognitive effectiveness*, which relate to the speed, ease and accuracy by which the human mind can process a visual notation [9]. Cognitive effectiveness is established as the primary design goal or *dependent variable* for comparing and evaluating visual notations and is thus suitable for making judgements on the goodness of notations. In order to facilitate designing cognitively effective notations, a set of principles is defined relating to the way the *visual vocabulary*, *grammar* and *semantics* should be combined to achieve a good visual notation [12]. In fact, Moody's principles have been demonstrated to positively influence a notation's perceived usefulness [8]. 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.

Various declarative notations have been defined up until now. Van der Aalst et al. propose to visualize declarative models by means of static diagrams that represent the entire process scheme at once [13,3]. Their notation, based on representing Declare constraint templates, has become the de-facto standard for visualizing declarative process models. Even though the notation's visual syntax facilitates a compact illustration of a declarative process model, its semantics tend to be difficult to understand at first sight. Especially when process models increase in size and complexity, as is common in the process mining field, the Declare notation discloses lack of providing a clean and comprehensible overview. Consequently, this increases the mental effort necessary for a user to process and interpret such a model. Given the shortcomings of the original Declare notation [6] in terms of understandability, alternative notations are needed.

3 Notation

Di Ciccio et al. [4] propose a visualization of declarative process models on the basis of Declare constraint templates [13,3] by means of three complementary views: *(a)* the *global view*, depicting a static bird-eye sketch of a process scheme; *(b)* the *local view*, focussing on one activity at a time; and *(c)* the *dynamic view*, visualizing the current state of a running instance. With the notation

primarily being devised for representing mined processes of e-mail collections, it is designed to handle larger and more complex process models. However, since the visual elements between these views do not remain consistent, understanding the connection between them can be a tough task.

Building upon the work in [4,5], the new notation employs two corresponding views on a process in a similar manner: *(i)* a static, multi-level *global view*, illustrating the entire process at once and *(ii)* a *local view*, focussing 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, i.e. we abstract away from various types constraints and merely indicate *positive* or *negative* relations between activities, thus increasing readability at first sight. For the sake of conciseness, this paper focusses on the more detailed "standard" granularity level of the global view. It bases its rationale on a network topology-like alignment of activities, which are accordingly depicted by means of circular elements and complemented by full text identifiers. Relation constraints are embodied by utilizing solid lines and cursors between activities, whereas Existence constraints are delineated by text annotations within the activity element. The notation illustrates constraints prescribing the cardinality of a task, e.g. *Participation*(a) or *AtMostOne*(a), by adding text to the upper half of the circular element. If the constraint specifies the first or last activity of a process, *Init*(a) or *End*(b) respectively, it is indicated by an annotation in the lower left or right part of the element.

A visualized constraint involving a dashed line always implies its belonging to the group of Negation constraints. The notation illustrates Relation constraints between activities by using *solid* cursors for positive constraints and *empty* cursors for Negation constraints, each of them connecting two activities per constraint. Relation constraints are perceived as "if-then" statements: The "if-part" or activation part is complemented by a cursor, being 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 rationale 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. Moreover, as it specifies that the respective target activity b must eventually be performed afterwards, the cursor is placed *outwards* on the activity border. Contrarily, in order to illustrate the *Precedence*(a,b) constraint, the cursor is now located at activity b, pointing *inwards*. The combination of both constraints, i.e. *Succession*(a,b), is depicted by joining the distinctive elements of their respective graphical representations. Figure 1 illustrates the graphical notation of Declare constraints.

In case a constraint allows executions of further tasks in between, these optional activities are visualized by means of smaller circles and complemented by a Kleene star ($*$), referring to "any other activity". If the constraint does

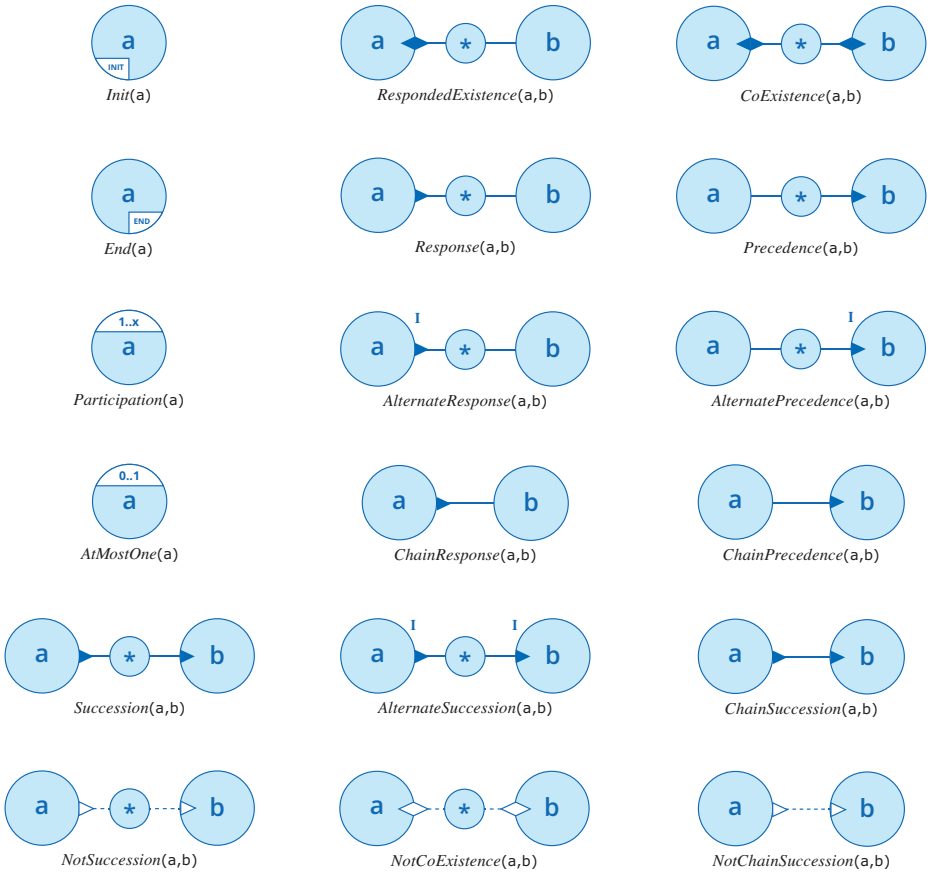

Figure 1. Constraint templates in Declare and their corresponding visual notation.

not prescribe a particular sequence, as in the case of *RespondedExistence*(a,b) or *CoExistence*(a,b), the notation employs two connected cursors, thus forming a diamond, which is placed at the activation part of the constraint. In order to indicate an *Alternate* limitation, the Roman symbol for 1 ("I") is added to the activation part of the constraint. This acts as a counter, stating that this very activity is allowed to only happen once until the other one is performed. Finally, *Chain* variations are depicted by leaving out *optional* activity circles, thereby specifying that no further activity must be performed in between. The process model in Figure 2 depicts an example of the global view.

In contrast to the panoramic global view, the *local view* only focuses on one activity and its *directly* related constraints at a time. As shown in Figure 3, it aims at providing a clear picture of what can, must or must not happen before and after the execution of the examined activity. As the local view's objective is to suggest a possible order of activities, two parameters are taken into account:

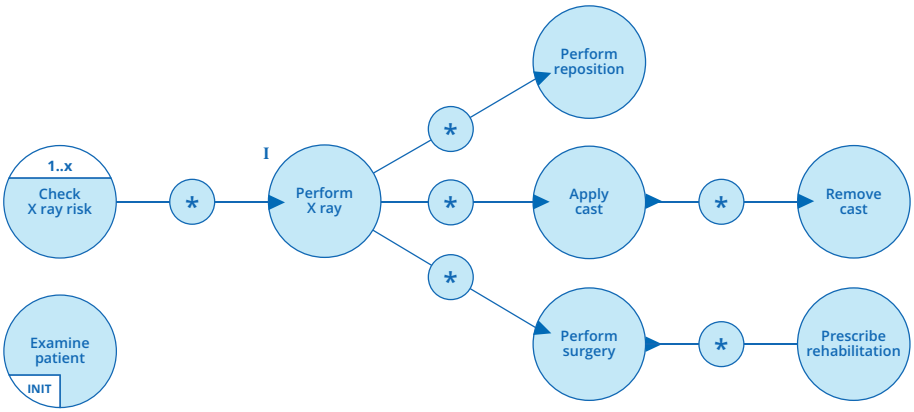

Figure 2. The enhanced global view level of a fracture treatment process.

time and *implication*. Based on the approach in [4,5], its rationale is inspired by the two-dimensional Cartesian coordinate system. With *time* being put on the x-axis, a timeline intuitively leads from left (past) to right (future), while the activity to be analyzed is put at the origin. Pointing from top to bottom, the upper part of the y-axis contains all activities that *imply* the activity located at the center. Conversely, the lower part of the y-axis encompasses all activities that *are implied by* the activity located at the origin.

4 Discussion

This section briefly discusses the implications of our findings with respect to Moody's nine principles [12] of designing cognitively effective visual notations.

The principle of *graphic economy* [12] is applied, i.e., the number of different symbols is being kept as low as possible in order to stay cognitively manageable. This principle explains, e.g., why optional unspecified activities (labeled with $*$) are being illustrated by means of the same geometrical shape as regular activities. The principle of *cognitive integration* [12] motivates the usage of the same set of graphical elements both in the global and the local view. This mechanism supports the integration and enhancement of information from the former to the latter. By employing a rationale with corresponding arrowheads for visualizing "if-then" statements, the notation builds upon using an explicit mechanism for dealing with complexity, as described in the principle of *complexity management* [12]. Moreover, employing this rationale applies to the principle of *semiotic clarity* [12], since a user can easily trace back how each graphic representation is constructed on the basis of its respective semantic construct. Finally, the same principle is considered in delineating *Alternate* constructs as they are indicated by adding a counter of 1, thereby specifying that an activity can be involved once in every alternation. Note that, by contrast, the standard notation of Declare [13] does not exploit explicit principles to increase comprehensiveness.

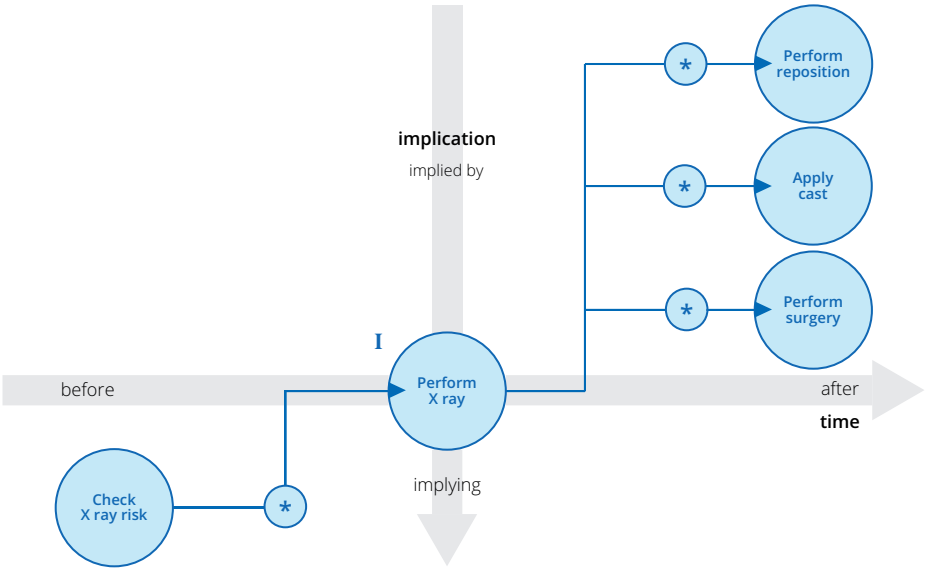

Figure 3. Design rationale of the local view examining the activity perform X ray.

Utilizing circular elements for depicting activities supports the alignment of activities in a more space saving and tidy way, hence enhancing readability and understandability. As cursors can easily be moved alongside the circular border, their connecting lines' bending points can be reduced to a minimum.

5 Conclusion

In this paper, we presented a novel conceptual framework for representing declarative process models on the basis of Declare constraint templates [3]. As this work is only concerned with the design of the notation, future research investigating and 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. In: USAB. pp. 83–94. Springer (2011)
5. Di Ciccio, C., Mecella, M., Scannapieco, M., Zardetto, D., Catarci, T.: Mailofmine – analyzing mail messages for mining artful collaborative processes. In: SIMPDA. 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. Figl, K., Derntl, M.: The impact of perceived cognitive effectiveness on perceived usefulness of visual conceptual modeling languages. In: Conceptual Modeling–ER 2011, pp. 78–91. Springer (2011)
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: Business Process Management Workshops. pp. 169–180. Springer (2006)
14. Pnueli, A.: The temporal logic of programs. In: Foundations of Computer Science, 18^{th} Annual Symposium on. pp. 46–57. IEEE (1977)