Towards a decision-aware declarative process modeling language for knowledge-intensive processes

Steven Mertens, Frederik Gailly, Geert Poels

Department of Business Informatics and Operations Management
Faculty of Economics and Business Administration
Ghent University, Tweekerkenstraat 2, 9000 Ghent, Belgium
{steven.mertens,frederik.gailly,geert.poels}@ugent.be

Abstract. Modeling loosely framed and knowledge-intensive business processes with the currently available process modeling languages is very challenging. Some lack the flexibility to model this type of processes, while others are missing one or more perspectives needed to add the necessary level of detail to the models. In this paper we have composed a list of requirements that a modeling language should fulfil in order to adequately support the modeling of this type of processes. Based on these requirements, a metamodel for a new modeling language was developed that satisfies them all. The new language, called DeciClare, incorporates parts of several existing modeling languages, integrating them with new solutions to requirements that had not yet been met. Deciclare is a declarative modeling language at its core, and therefore, can inherently deal with the flexibility required to model loosely framed processes. The complementary resource and data perspectives add the capability to reason about, respectively, resources and data values. The latter makes it possible to encapsulate the knowledge that governs the process flow by offering support for decision modeling. The abstract syntax of DeciClare has been implemented in the form of an Ecore model. Based on this implementation, the language-domain appropriateness of the language was validated by domain experts using the arm fracture case as application scenario.

Keywords: Business process management, Loosely framed processes, Knowledge-intensive processes, Declarative process modeling, Decision modeling, Resource reasoning

1 Introduction

A business process is a “collection of inter-related events, activities and decisions points that involve a number of actors and objects, and that collectively lead to an outcome that is of value to at least one customer” (Dumas, La Rosa, Mendling, & Reijers, 2013). Business process modeling (BPM) (Dumas et al., 2013; Weske, 2012) is the practice of creating models that represent business processes. These models can subsequently be used as a way of documenting a process, as a way of representing the as-is and to-be situations, as a means to check conformance and/or as a precise schedule for the execution of the process (Dumas et al., 2013). Business process modeling is being applied in a variety of industries (e.g., manufacturing, insurance, utilities, retail, telecommunications, finance, etc.). As a result of the diversity in uses and target domains, a multitude of process modeling languages exist (Goedertier, Vanthienen, & Caron, 2015; Krogstie, 2013; Lu & Sadiq, 2007; van der Aalst, 2013). Gantt and flow charts were part of the first wave and have since given way to modern-day techniques like BPMN (White, 2004), UML activity diagrams (Object Management Group, 2015) and Declare (Pesic, 2008; van der Aalst, Pesic, & Schonenberg, 2009).

A possible way of classifying business processes is according to their *predictability* (Di Ciccio, Marrella, & Russo, 2015; van der Aalst, 2013). First, a *tightly framed* process is always executed in just one of a few predefined, consistent and unambiguous ways (e.g., the mass production of electronics). When the process is less structured it is called *loosely framed*, which entails that a process is executed in a large, but finite and predefined, number of ways (e.g., the treatment of patients in a hospital). Next, a process is *ad hoc framed* if it is typically executed starting from a predefined process model, but ad hoc changes occur frequently during the execution. This leads to the models being used only once or perhaps a few times before being discarded or changed (e.g., project planning). Lastly, *unframed* processes are processes where each execution is unique and generally impossible to predefine (e.g., groupware systems). The scale ranging from tightly framed processes to unframed processes corresponds to processes becoming more and more knowledge-intensive (van der Aalst, 2013). Although there exist strongly framed processes that can be considered knowledge-intensive (e.g., the loan approval process in a bank), these are more the exception rather than the rule. (Di Ciccio, Marrella, et al., 2015) define *knowledge-intensive business processes (KiPs)*, in general, as follows:

"Processes whose conduct and execution are heavily dependent on knowledge workers performing various interconnected knowledge-intensive decision making tasks. KiPs are genuinely knowledge-, information- and data-intensive and require substantial flexibility at design- and run-time."

The former part of this definition references the complex nature of these processes and the unmistakable human factor in all of this. The execution of the process is completely dependent on *knowledge workers* (e.g., doctors), who each use their specific knowledge to contribute to the process. This knowledge is a form of processed information (i.e., understood patterns) and information (i.e., detected patterns) can be defined as data values (i.e., raw data) with a certain meaning or purpose (Rowley, 2007). So in essence, modeling knowledge requires the modeling language to support at least a *data* perspective (i.e., the available data records, attributes, values and their relationships). The latter part of the definition references the loosely to unframed nature of most KiPs. A knowledge worker (i.e., part of the *resource* perspective with the available resources and their relationships) requires a significant amount of freedom to operate, so that he or she can apply their knowledge to specific contexts. This freedom can be incorporated at design-time, at run-time or both (Reichert & Weber, 2012). For example, a model can explicitly specify a decision point and a specific process variation for each decision outcome at design-time or even leave these underspecified to be determined at run-time. Basically, this part of the definition indicates that a *functional* (i.e., the available activities/events and their instances) and *control-flow* perspective (i.e., restrictions on when activities/events occur) are integral parts of a modeling language for KiPs to be able to cater to this need for flexibility.

The classification can now be translated to a classification based on the relative importance of the functional, control-flow, resource and data perspectives to the process. For tightly framed processes the focus is mainly on precise definition of the functional and control-flow perspectives, and in some cases also on the resource perspectives (Di Ciccio, Marrella, et al., 2015). However, for loosely framed processes the data perspective is at least of equal importance as the other perspectives, because it is an essential requirement to model the knowledge that governs the other perspectives (Di Ciccio, Marrella, et al., 2015). The other perspectives are also of importance as demonstrated by the need for flexibility in the functional and control-flow perspectives in the definition of KiPs, while knowledge workers themselves are part of the resource perspective and knowledge about resources can heavily influence their decisions. The languages that currently target loosely framed process (e.g., Declare and DCR graphs) do not yet sufficiently address all of these perspectives together. The relative importance of the data and resource perspectives increases even more for ad hoc framed processes and ultimately dwarfs the other perspectives in the context of unframed processes.

The knowledge of process workers in KiPs guides the decision-making regarding which process variation is appropriate in a certain situation (e.g., how doctors diagnose and treat patients). The *tacit* nature of this knowledge is a typical symptom of the process being in a state of flux as the domain knowledge itself changes with time (Krogstie, 2013; Lenz & Reichert, 2007). The process workers cannot keep up with the complexity of the constant stream of new knowledge, which results in a situation where everyone has a different subset of the total, possibly even out-of-date, knowledge. Additionally, nobody knows exactly what the other process workers know. This tacit knowledge phenomenon is an important obstacle when attempting to accurately model these processes (Ferreira & Ferreira, 2006). Stimulating and managing the knowledge conversion processes in order to externalize the tacit knowledge (i.e., linking sets of data values to certain outcomes) is an important part of improving these processes. Firstly, the explicit knowledge can be used for explaining decisions that were made (Object Management Group, 2016). Secondly, making process knowledge explicit increases the transparency of the process to all stakeholders, and hence facilitates communication of process changes or new insights as the process evolves (Object Management Group, 2016). Thirdly, making the process knowledge explicit is beneficial for training process workers. Fourthly, explicit process knowledge enables offering a more uniform service, as the knowledge workers can compare and discuss the decision logic with each other in detail, which limits the amount of divergence (Shostack, 1987). Lastly, processes can be optimized more efficiently and effectively because they capture the reality more closely (Wastell, White, & Kawalek, 1994).

In this paper the *Design Science* methodology (Hevner, March, Park, & Ram, 2004; Peffers, Tuunanen, Rothenberger, & Chatterjee, 2007) is followed in order to design a mixed-perspective process modeling language, called *DeciClare*, that can capture information related to all four perspectives (i.e., functional, control-flow, data and resource) when modeling loosely framed KiPs, for example, in healthcare and incident management. The language was built on a declarative foundation, because of its innate support for modeling flexible process flows. DeciClare integrates several existing extensions for declarative process modeling languages as well as new solutions for yet unaddressed shortcomings in order to support the strongly linked perspectives of such processes. Because the different perspectives had to be integrated into one approach, DeciClare can be categorized as a mixed-perspective modeling language (a definition of our generalized interpretation of this term is presented in section 6).

Verwijderd: hybrid

Verwijderd: hybrid

Design Science research is an iterative process (Hevner et al., 2004) and this paper presents our second iteration of the subject. In this paper, most of the work presented in (Mertens, Gailly, & Poels, 2015) was redone, while the lessons learned from the previous iteration were incorporated. Firstly, this paper abandons the ambition to propose both an abstract and a concrete syntax for the language in one paper. As stated by (Moody, 2009), both require a different approach and evaluation (i.e., semantic versus cognitive). Consequently, it is better to split them in two separate research steps so that each can be performed in greater depth and with a separate evaluation. This paper presents the first part of the research project: the development of an abstract syntax for a modeling language that targets loosely framed KiPs. An implementation of this abstract syntax in the form of an Ecore model is the resulting design science artifact. Secondly, this paper starts from a better theoretical foundation in the form of specific requirements for the language based on the needs of the targeted processes. This allowed for a more thorough analysis of the existing solutions (i.e., mostly extensions of declarative process modeling languages) as well as to identify remaining shortcomings. The resulting requirements thus served as a blueprint for designing the envisioned language. Lastly, an evaluation of the effectiveness of the proposed language by way of interviews with domain experts was also added.

The paper is structured as follows. In section 2, the problem stated in the section 1 is translated into a set of requirements that must be met in order to adequately model loosely framed KiPs. Section 3 then examines each perspective in isolation. Existing modeling languages and their extensions are shown to fulfill some of these requirements, however, additional solutions are also proposed for the requirements that are not yet met. These separate solutions are then integrated into one metamodel in section 4 for the new mixed-perspective language fulfilling all the requirements from section 2. Next, in section 5, the new language is evaluated by way of expert validation based on a realistic KiP. In section 6, the related work is discussed and mixed-perspective languages are compared to hybrid and multi-perspective languages. Finally, the paper is concluded in section 7 and future work opportunities are described in section 8.

Verwijderd: hybrid

Verwijderd: a generalized definition is given of the term 'hybrid modeling language' as used in this paper.

2 Language requirements

The *main goal* of this paper is the development of a process modeling language that allows for the functional, control-flow, resource and data perspectives of loosely framed KiPs to be modeled in an integrated manner. These perspectives should not be treated in isolation, because the different perspectives are naturally intertwined. For instance, the required execution of an activity (i.e., functional/control-flow perspectives) could tie up certain resources thereby impacting the general resource availabilities (i.e., resource perspective), while the activity itself could use and produce certain data elements (i.e., data perspective). Alternatively, the unavailability of a resource (i.e., resource perspective) can result in the direct unavailability of an activity that requires the resource for its execution (i.e., functional/control-flow perspectives) and trigger decisions on how to respond to this unavailability (i.e., data perspective). Lastly, changing a data value (i.e., data perspective) could result in different activities needing to be executed (i.e., functional/control-flow perspectives) and adjusted resources availabilities (i.e., resource perspective). These domino effects between the perspectives are a common feature in many processes, but the relative prominence of all four of the perspectives in loosely framed KiPs (see section 1) increases the need to capture these effects in the process models. Therefore, the developed process modeling language will have to be of a mixed-perspective nature (see section 6) in order to integrate the different perspectives into one combined modeling technique.

Verwijderd: hybrid

The main goal can be translated into *five subgoals*: four subgoals expressing the need to support each of the perspectives and one additional subgoal for the integration of the different perspectives. These general subgoals can be operationalized to specific requirements for the modeling language by taking into account the properties and needs of loosely framed KiPs. There is one less requirement than there are subgoals, because the first requirement applies for both the functional and control-flow perspectives of the process. In the remainder of this section, these four requirements are presented and explained.

Req 1: The language must support 'flexibility by design'

A first property relates to the *functional* and *control-flow* perspectives. Loosely framed processes allow many different arrangements of their activities, leading to a large variety of process variations being valid (Di Ciccio, Marrella, et al., 2015; Rovani, Maggi, de Leoni, & van der Aalst, 2015). For example, take the diagnosis of a patient in an oncology department. The doctor can do any number of tests selected from a battery of possible tests to check whether or not the patient has cancer, and if so, determine the type, the size, the aggressiveness and the stage of the cancer. There are many different sequences in which these tests can be done, but some sequences can or cannot occur depending on certain prerequisite conditions specific to each process instance. To model such highly-dynamic, human-centric, knowledge-intensive and non-standardized processes, a language is needed that supports the flexibility to model these variations and defers the choice for a certain variation to the run-time execution (Goedertier et al., 2015). This type of flexibility is called *flexibility by design* (Schonenberg, Mans, Russell, Mulyar, & van der Aalst, 2008). Flexibility by design thus implies that many alternative process variations are designed into the process model implicitly, allowing the choice of a suitable variation to be made at run-time for each individual process instance.

Gewijzigde veldcode

Req 2: The language has to offer support for modeling complex decisions

The 'knowledge' in the KiPs alludes to the implicit or explicit knowledge needed by the process workers to execute the process. Let us retake the example from the oncology department from the previous paragraph and suppose the doctor has determined that the patient has a brain stem tumor. There are several ways for the process to proceed and, obviously, the process variation to follow is not chosen at random. To learn the type of cancer cells one would normally perform a biopsy, but, because of the location of the tumor, patient safety regulations state that this goes out of the window. The doctor would have to resort to using an MRI, CT and/or some other imaging test to proceed. This is an example of a rather simple rule using process relevant data, in this case data generated during one or more previous activities, followed by a decision on how to proceed. KiPs are governed by many rules and/or decisions processes (Borrego & Barba, 2014; Di Ciccio, Marrella, et al., 2015) and insight into these decisions is at least as important as what activities are available for execution, so support for decision modeling should be included in modeling language proposed in this paper. Because at its core decision modeling is nothing more than linking data values or combinations of data values to outcomes, this comes down to incorporating a *data* perspective into the language. The knowledge of a process is represented by the combination of specific data values and outcomes used in its decision-making. Typically, decisions that govern the process execution of KiPs have a high perceived complexity (Di Ciccio, Marrella, et al., 2015). Some contributing factors to the decision complexity are the number of data values and data value aggregations used, the number of data sources and the number of decision conditions as well as their interrelations. The data perspective of the language thus needs to offer support for modeling complex decisions that guide process execution.

Req 3: The language needs to allow reasoning about resources

Resources play a vital role in loosely framed KiPs. More often than not, the execution of certain activities from the functional and control-flow perspectives depends on the availability of certain resources and those resources have certain properties expressed in data values (e.g., availability, speed, cost, quality, etc.) that influence the decisions in the data perspective of the process model. So the modeling language needs to incorporate some form of *resource* perspective to help paint a complete picture of process and its resources. For example, resources need to be taken into account to make the models executable, to allow a thorough analysis of the process, to offer meaningful guidance to process workers and to facilitate simulation and optimization techniques.

Req 4: The language must define a clear link between the different perspectives while carefully considering the domain appropriateness

Finally, the last requirement relates the previous three requirements. As stated at the start of this section, the four perspectives are closely linked. These relationships are complementary in most cases, however, overlaps exist. The functional/control-flow and data perspectives have an innate overlap in expressive power. For example, stating that activity B has to be executed after activity A can, in theory, be represented as an part of either perspective. In the control-flow perspective it would result in a temporal relation between the two activities, while in the data perspective it would be a decision with the execution of activity A (= a fact, hence data) as condition and the execution of activity B as an outcome. This is an example of construct redundancy (Wand, Storey, & Weber, 1999), signaling that there exist multiple ways to represent the same ontological concept in a language. According to (Guizzardi, 2013) the truthfulness of a modeling language with respect to the domain depends on the degree of isomorphism between the metamodel of the language and the domain conceptualization, also referred to as the *domain appropriateness*. Construct redundancy, excess and overload are all properties that serve as telltale signs of a misalignment of the domain appropriateness of a language. As a consequence, the way in which the four perspectives are linked and complement each other should be carefully considered.

3 Designing the perspectives

In this section a standalone metamodel for each of the perspectives, required by *requirements 1, 2 and 3* from the previous section, will be developed. In section 4, we integrate these metamodels to also fulfil requirement 4. The concepts used in this section are defined on a rather abstract level to allow them to apply to a broad range of cases, which can make the link with real-life needs unclear. Therefore, a case description of realistic KiP will first be introduced to serve as a source of examples to demonstrate the practical relevance of the concepts used. The same case will also be used in section 5 to evaluate the proposed language.

3.1 The arm fracture case

As a practical example of a KiP, the realistic healthcare process of diagnosing and treating patients with suspected arm fractures [reference to Arm_fractures_case_-_Case_description.pdf] will be used. This is not an isolated process in real-life, but rather a part of the general process spanning the emergency and orthopedic departments of a hospital. For this reason, we will use the term ‘realistic’ instead of ‘real-life’ to indicate the difference between the artificially reduced scope of the case compared to the even more complex real-life case. The scope of arm fracture case is limited to patients with one or more suspected fractures in his or her fingers, hands, wrists, forearms, upper arms, shoulders, and/or collarbones. The activities related to the admission, diagnosis and treatment phases are included (see Table 1).

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt, Vet, Engels (VS)

Verwijderd: Table 1

Admission	Register patient, Unregister patient
Diagnosis	Take medical history of patient, Clinical examination of patient, Doctor consultation, Take X-ray, Take CT scan, Apply intra-compartmental pressure monitor
Treatment	Prescribe NSAID painkillers, Prescribe SAID painkillers, Prescribe anticoagulants, Prescribe stomach protecting drug, Prescribe rest (=no treatment), Apply ice, Apply cast, Remove cast, Apply splint, Apply sling, Apply fixation, Remove fixation, Apply bandage, Apply figure of eight bandage, Perform surgery, Let patient rest, Stay in patient room

Table 1. The activities used in the arm fracture case

In the remainder of this section, examples from this case will be used where possible. Just a few types of modeling concepts do not occur in this case. So for these concepts examples from other healthcare processes will be used.

3.2 Declare as a foundation

Support for the flexibility by design principle (*requirement 1*) in the functional and control-flow perspectives can be achieved in multiple ways. In BPM there are two major modeling approaches: *imperative* and *declarative* process modeling (Fahland et al., 2009; Goedertier et al., 2015). Process modeling languages like BPMN (Object Management Group, 2013) and Petri nets (Murata, 1989; Petri, 1962) are examples of imperative modeling languages (also known as procedural languages). These techniques can be described, in essence, as inside-out approaches to modeling because they describe exactly what is the allowed behavior during process execution (i.e., they provide an enumeration of all valid process variations) (van der Aalst et al., 2009). Consequently, imperative process modeling languages are well suited to model tightly framed processes as for such processes only a limited number of process variations need to be modeled. Declarative process modeling languages like Declare (Pesic, 2008) and DCR graphs (Mukkamala, 2012) take the opposite viewpoint with an outside-in approach. Declarative languages describe the rules and constraints by which the process has to abide (Goedertier et al., 2015). The unwanted process variations are prohibited based on these rules and constraints, while all other variations are allowed without explicitly having to enumerate all valid variations (van der Aalst et al., 2009).

Although imperative process modeling languages can also support the flexibility by design principle, declarative process modeling languages are better suited for modeling loosely framed processes as they do not require enumerating the potentially large number of valid process variations (Rovani et al., 2015). Generally, given the large number of allowed process variations in loosely framed processes, less effort will be required to describe the properties that make variations valid or invalid compared to enumerating all the valid process variations explicitly like imperative languages (Goedertier et al., 2015; van der Aalst, 2013).

The Declare modeling language will be used as the foundation of our language, since it currently is one of the most popular and comprehensive declarative process modeling languages and it has many available extensions. (Mulyar, Pesic, van der Aalst, & Peleg, 2007) has also shown that this language enables the representation of typical clinical guidelines, which are part of many loosely framed KiPs in healthcare. Declare is based on Linear Temporal Logic (LTL) (Pesic, 2008; van der Aalst et al., 2009) and allows for process models to be defined in terms of activities and constraints spanning these activities. The process environment is specified in terms of what is necessary and what is not allowed (i.e., rules expressing the modal verb ‘must’ and ‘must not’), restricting the possible process executions. Contrary to other declarative languages like DCR Graphs and BPCN (Lu, Sadiq, & Governatori, 2009), Declare also supports guidelines (referred to as optional constraints). Such constraints offer guidance (i.e., rules expressing the modal verbs 'should' and 'ought to'), while their soft character ensures flexibility is maintained (i.e., it is not necessary to enforce them) (Goedertier et al., 2015).

There are three main groups of Declare constraint templates (i.e., templates that can be used to create actual constraints in Declare models) defined in the description of the Declare language (Pesic, 2008):

- Existence constraints: unary constraints predicating the allowed or mandatory amount of executions of an activity (e.g., existence(3, A) stating that activity A must be executed at least 3 times).
- Choice constraints: n-ary constraints expressing a choice between activities from a given set (e.g., choice(2, [A, B, C]) stating that at least 2 distinct activities among A, B and C must be executed).
- Relation constraints: binary constraints enforcing a relation between two activities (e.g., response(A, B) stating that if A is executed, then B has to be executed at any time after A). Also negative versions of these relation constraints exist (e.g., responded_absence(A, B) stating that if A is executed, then B can never be executed).

In the arm fracture case, for example, the general flow of the process states that a doctor can choose from several activities to reach a diagnosis and a wide range of possible treatments activities. However, an X-ray should be taken in almost every case (we will ignore exception of life-threatening cases for now), immediately followed by a consultation with the patient to discuss the results. Imperative modeling language would require the explicit enumeration of every valid sequence of diagnosis and treatment steps. For example, a variation needs to be defined for when the doctor decides on a treatment consisting of an X-ray, a consultation, a prescription for painkillers and the application of a sling. But I should also be possible to give the patient the painkillers and/or sling before taking the X-ray. So even for this simple case, which does not even take into account the any prior steps, will result in six variation depending on the order of these steps. Declare allows this to be modeled with just two constraints: *existence(1, Take X-ray)* and *chain_response(Take X-ray, Doctor consultation)*. All the variations that do not violate these constraints are implicitly allowed.

We created a metamodel of the Declare language (~~Fig. 1~~), because a complete metamodel of Declare is not available in literature. This metamodel is based on a partial metamodel of the Declare language (Bernardi, Cimitile, Di Lucca, & Maggi, 2012) and the implicit metamodel of the concrete syntax proposed for the language (Pesic, 2008) and the corresponding modeling tool[1]. The constraint templates themselves have been omitted from the metamodel to maintain a manageable overview and will be presented later on. Having an explicit metamodel for a modeling language is important as it represents the core concepts of the language and their relationships. This prevents possible ambiguity, which can make a language more difficult to use. It is also a necessary step to be able to correctly integrate new elements into a language and for any kind of tool support to be developed.

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt, Engels (VS)

Verwijderd: Fig. 1

[1] http://www.win.tue.nl/declare/

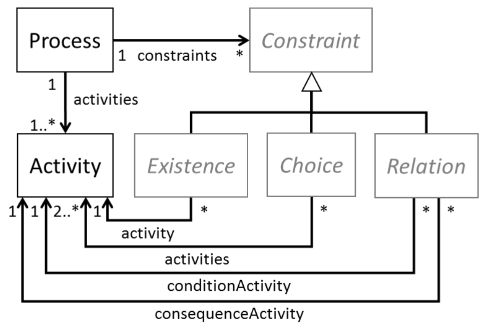

Fig. 1. The metamodel of the Declare language

The metamodel in Fig. 1 captures the Declare language as it has been used in most research, with the existence and choice templates largely corresponding to the functional perspective and the relation templates to the control-flow perspective. However, there is one change that could already be made. This entails using events in addition to activities, similar to how this was proposed in (Montali, 2010). (Pesic, 2008) defines the Declare constraint templates using activities as parameters. However, in the partial metamodel proposed in (Bernardi et al., 2012) constraints are actually already linked with events as opposed to activities. This ambiguity is probably due to the use of events in the underlying LTL definitions for each of the templates in (Pesic, 2008) and could have been avoided by providing an explicit metamodel of the language. The effect of allowing the templates to use event parameters is that it allows for much more fine-grained control when it comes to the expressive power of the constraints. Activities have a life cycle consisting of a start, end and cancellation event, mirroring the LTL definitions in (Pesic, 2008). For example, stating that a physical examination cannot be performed before the doctor or nurse has already started asking the patient questions about his/her medical history, can now be expressed with a precedence constraint: *precedence(start(Take medical history of patient), start(Clinical examination of patient))*. This still leaves the option to perform these activities simultaneously or sequentially. In comparison, the original precedence template is always assumed to relate the end event of the preceding activity to the start event of the subsequent activity (i.e., *precedence(end(Take medical history of patient), start(Clinical examination of patient))*). Other declarative languages like DCR graphs also work on the level of events for similar reasons (Slaats, Mukkamala, Hildebrandt, & Marquard, 2013).

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt, Engels (VS)

Verwijderd: **Fig. 1**

During the application of Declare in specific process contexts, several shortcomings of the original scope of the language have been noted (e.g., resources, time, etc.). In response to these shortcomings, many standalone extensions have been proposed for Declare in the past few years. These extensions will be used as inspiration or starting point for the proposed solutions for the other requirements.

3.3 Extending the functional and control-flow perspectives

With Declare as the foundation to build on, the match between its expressive power concerning the functional and control-flow perspectives and the needs of loosely framed KiPs can be analyzed. We specified a set of seven rule patterns that typically occur in this type of process based on the overview of rule patterns in (Caron, Vanthienen, & Baesens, 2013) (i.e., A, B and C) and additional insights obtained from studying several loosely-framed processes in healthcare organizations (i.e., D, E, F and G). Each pattern can entail multiple constraints that differ in effect, but are all related to a same underlying concept. These rule patterns, accompanied by an example of such a constraint in healthcare diagnosis and treatment processes, are presented in Table 2.

Verwijderd: Table 2

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt

	Rule pattern	Example
A	*Cardinalities*	A patient has to be registered at least once, but also at most once.
B	*Coexistence*	At least two of the following treatments will be administered: surgery, a cast, a fixation, bandages and painkillers.
C	*Temporal relationships*	If a patient has had surgery, then he will receive anticoagulants afterwards.
D	*Negative relationships*	NSAID and SAID painkillers should not be administered simultaneously.
E	*n-ary relationships*	After a physical examination, the doctor will decide on a treatment consisting of at least one of the following activities: prescribe rest, apply ice, apply cast, apply splint, apply sling, apply fixation, apply bandage, apply figure of eight bandage and/or perform surgery.
F	*Exception handling*	If a patient has an adverse reaction to the anesthesia causing the doctors to abandon the rest of the surgery, this is followed by a consultation to determine what needs to happen next.
G	*Guidelines*	Directly after surgery, the responsible doctor should visit the patient to explain what was done and what is next (i.e., a consultation).

Table 2. Seven typical rule patterns for flexible and KiPs

The available Declare templates can immediately fulfil several rule patterns. The existence templates of Declare can enforce the cardinalities of rule pattern *A* (for the example from Table 2: *existence(1, Register patient) and absence(1, Register patient)*). Rule pattern *B* can be fulfilled with the choice and responded existence templates *A* (for the example from Table 2: *choice(1, Perform surgery, Apply cast, Apply fixation, Apply bandage, Prescribe painkillers)*). Constraints belonging to rule patterns *C* and *D* can be expressed by using the relation templates (for the example from Table 2: *response(Perform surgery, Prescribe anticoagulants)* and *not_chain_succession(Prescribe NSAID, Prescribe SAID)*). And lastly, Declare also supports optional constraints, which can be used to model the guidelines of rule pattern *G* (for the example from Table 2: *optional response(Perform surgery, Doctor consultation)*). However, support for rule patterns *E* and *F* is lacking, and thus, needs to be addressed.

Verwijderd: Table 2

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt

Verwijderd: Table 2

Verwijderd: Table 2

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt

Verwijderd: Table 2

Before addressing the unsupported rule patterns, we will make a small change to the choice templates that were used to support rule pattern *B*. The exclusive choice template is replaced by the *AtMostChoice* template (the normal choice template will also be renamed to *AtLeastChoice*, see Table 7). The exclusive choice template states that exactly the given number of activities from the given set have to be executed. Keeping in mind that the normal choice template expresses the same thing but with the 'exactly' replaced by 'at least', it seems strange to ignore that the 'exactly' can also be replaced by a template expressing 'at most'. Similar to the other templates of the existence type, the 'exactly' is merely a combination of an 'at least' and an 'at most' constraint with the same bound and activities. Therefore, adding the AtMostChoice template results in an increase of the expressive power and symmetry of the constraint templates.

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt

Verwijderd: Table 7

Rule pattern *E* is the first unsupported rule pattern that will be discussed, as this entails a fundamental change to the Declare definitions. It conveys the need to express the constraints with more than the predefined number of connected elements. In the original definitions of the Declare constraint templates, constraints are connected to one (i.e., existence, with the exception of the choice templates) or two activities (i.e., relation) each. By *branching* the constraint templates, each connected element is replaced by a non-empty set. (Pesic, 2008) already described this as a possible extension of Declare, while later work elaborated further on the subject of branching. (Montali, 2010) provided a definition for branching relation templates, while the TBDeclare extension (Di Ciccio, Maggi, & Mendling, 2016) focusses on branching of the consequence side (sometimes referred to as the target side) of relation templates. The broadest definition of branching will be used, and thus, allow branching for all constraint templates. Although the definitions in the above mentioned extensions always use a disjunctive interpretation of the branched parameters, DeciClare will sometimes also allow conjunctions and represent them as logical expressions. The introduction of branching impacts the templates in different ways:

- The original definitions of the choice templates are, in a sense, already branched according to the interpretation of a disjunction. This will be expanded with conjunctions. For example, the choice template (later referred to as the AtLeastChoice template) can be used to specify that at least one out of the three following options have to be executed: w or x or (y and z). This constraint would allow event traces like [w], [x], [y, z], [w, x, y]…, but not [y], [z], [y, y]...
- The absence template (later referred to as the AtMost template) now supports both the inclusive disjunction and the logical conjunction for its logical expressions. For example, consider an event trace of [x, x, x, z, y, y]. It is correct to state that x *or* y occur at most 5 times (i.e., AtMost([x or y], 5)), corresponding to the sum of the occurrences of x and the occurrences of y being at most 5. On the other hand, it is correct to state that x *and* y occur at most 2 times (i.e., AtMost([x and y], 2)), corresponding to the minimum number of occurrences of x and y separately being at most 2. This is because at most 2 (x, y)-pairs can be found in the trace.
- For the existence template (AtLeast) similar logic applies for the disjunction. However, the conjunction is not supported because, for example, requiring at least a certain number of occurrences of (x, y)-pairs is equivalent to requiring x and y to separately occur at least that many times (i.e., two separate AtLeast-constraints). To represent this in one statement could be useful for a concrete syntax, but in the metamodel of the language this will be avoided (see syntactic sugar in section 4.3).
- The relation templates as presented in (Montali, 2010), support the inclusive disjunction. But this can also be further extended to include the expression of logical conjunctions. For example, consider a response constraint Response([x or [y and z]], [w]). This expresses that when event x has been executed or both events y and z have been executed, event w eventually needs to be executed. The example used in **Table 2** can now be modeled as *Response([Clinical examination of patient], [Prescribe rest, Apply ice, Apply cast, Apply splint, Apply sling, Apply fixation, Apply bandage, Apply figure of eight bandage or Perform surgery])*.

Verwijderd: Table 2

Rule pattern *F* corresponds to exception handling constraints. It can be supported by introducing a *failure event*. Similar to the cancellation event, it marks an alternative ending of the execution of the corresponding activity, although this time not in a user-driven fashion (e.g., due to a change of heart by the patient). The constraint templates can now be used to specify what has to happen when failure occurs. However, some combinations make little or no sense (e.g., the end event of activity A has to be responded by the failure of activity B), while others are just hard to enforce (e.g., the failure of activity A can occur no more than once). Our suggestion would be to use the event only in combination with the responded existence, response and precedence templates (including their variations) with the additional limitation that it cannot be included it in the condition parameter connected to the precedence templates, nor in the consequence parameter connected to the other templates. For instance, the failure event can be used in a ChainResponse constraint to express that if an MRI scan fails because the patient had unexpected metal items in his/her body (e.g., hip replacement), then immediate surgery needs be performed to repair the damage done by the MRI magnet. The example used in Table 2 can now be modeled as *ChainResponse([failure(Perform surgery)], [Doctor consultation])*.

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt

Verwijderd: Table 2

3.4 Adding a notion of time-awareness to the constraint templates

(Dadam, Reichert, & Kuhn, 2000) stated that the minimal and maximal time before, after and between activities is an important aspect in many loosely framed processes. The rule patterns presented in Table 2 do not deal with time as an explicit concept, as they largely coincide with expressive power of the original Declare language. (Caron et al., 2013) also proposed two more rule patterns that deal with time as a concept (*T1* and *T2* in Table 3). Additionally, (Lanz, Weber, & Reichert, 2014) have suggested a set of time patterns for supporting *time-awareness* in process-aware information systems based on empirical evidence from case studies (*T3*-*T7* in Table 3). Some of the time patterns suggested in (Lanz et al., 2014) have been omitted, because they do not apply in the context of loosely-framed KiPs or are redundant as a result of other patterns presented throughout this paper.

Verwijderd: Table 2

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt

Verwijderd: Table 3

Verwijderd: Table 3

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt

	Time pattern	Example
T1	*Time-oriented existence*	The triage has to be started within 5 minutes.
T2	*Time-oriented relationships*	After surgery for a displaced or unstable fracture of the leg, a cast is applied. This cast cannot be removed within the first 10 weeks.
T3	*Lag between two activities*	The time between on the one side removing a cast and having done an X-ray and on the other side starting a doctor consultation, should be at most 15 minutes.
T4	*Durations*	The triage should not exceed 10 minutes.
T5	*Lag between arbitrary events*	The time between triage and the examination by a doctor should not exceed 3 minutes for patients with a high risk level.
T6	*Schedule restricted elements*	Lab tests can be requested on Mondays between 8 am and 5 pm.
T7	*Cyclic elements*	Patients with medium risk level should be checked every hour.

Table 3. Seven time patterns to complement the control-flow perspectives

Declare supports an implicit idea of time (e.g., sequencing), but it lacks the support for *metric time* as is needed to satisfy the rule patterns in Table 3. However, the idea of supporting metric time in Declare is not new. (Barba, Lanz, Weber, Reichert, & Del Valle, 2012; Montali, 2010; Westergaard & Maggi, 2012) have all identified a need for constraint templates using time units. To solve this, a reformulation of Declare, called Timed Declare, was proposed in *Metric Temporal Logic* (MTL) that adds the option to express time units (Westergaard & Maggi, 2012). As a result, new versions of all Declare templates were defined. For example, the timed response template

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt

Verwijderd: Table 3

can express that a doctor consultation needs to take place at some point in time, t_1, after the taking an X-ray at t_0, with $t_1 \in [t_0 + a, t_0 + b]$. The inclusion or exclusion of the boundaries of the interval can be specified in the template using the parameters a and b which are expressed in time units (b is allowed to be ∞). When a and b respectively equal 0 and ∞, the formula collapses to the original definition of the response constraints of Declare. These MTL-definitions of the templates will be used instead of the original LTL-definitions as a starting point for our DeciClare language, because this reformulation increases the expressive power of Declare with a notion of time-awareness.

The time parameters of the Timed Declare constraint templates add a notion of time to the constraint templates that is sufficient to satisfy time patterns *T1* (for the example from **Table 3**: *AtLeast([Triage], 1, 0, 5minutes)*) and *T2* (for the example from **Table 3**: *NotResponse([Perform surgery], [Remove cast], 0, 10weeks)*). However, the other time patterns are not supported yet. *T3*, *T4* and *T5* can be treated jointly as the language already works on the granularity level of events. The time between two activities (*T3*) is defined as the time between the end event of the first activity and the start event of the second activity. *T4* generalizes this to the time between any two events of the process. This means that it also covers the duration of an activity (*T5*), because the duration is equal to the time between the start and end event of the activity. Therefore, two new time constraints will be defined to add support for these three patterns: *AtLeastLag* and *AtMostLag*. These constraint templates have two logical expressions using events (inclusive disjunction and conjunction) as input parameters and express that the time between the satisfaction of the first expression and the second expression can be no less or no more, respectively, than a given limit expressed in time units (for the example from **Table 3**: *AtMostLag([end(Remove cast) and end(Take X-ray)], [start(Doctor consultation))], 15minutes)*). The sixth time pattern (*T6*) makes it possible to express that certain activities are only available for execution according to a certain schedule. This can be generalized to also include expressing that certain activities are unavailable for execution according to a schedule (e.g., maintenance). For this, another two constraints are added: *EventAvailabilitySchedule* and *EventUnavailabilitySchedule*. Their parameters are a logical expression using events (inclusive disjunction) and a schedule. A schedule is a set of statements that take on the following form '(mandatory) every *[periodic point in time]*, (optional) between *[two smaller periodic points in time]*'. The example from **Table 3** would now be modeled as *EventAvailabilitySchedule([Lab tests], [every Monday between 8 am and 5 pm])*. The last time pattern (*T7*) enables the use of cyclic elements in the process. Where the Timed Declare definitions can be used to specify that an activity or event has to be executed a certain number of times within a certain time interval, *T7* adds to option to specify the time lag between the different executions. This can also be modeled with the first two new time constraint templates (i.e., *AtLeastLag* and *AtMostLag*), and so, requires no additional changes.

Taking all of the changes from this subsection and the previous one into account produces the extended Timed and Branched Declare metamodel in **Fig. 2**. This represents the functional and control-flow perspectives in response to *requirement 1*. Note that the left side of the metamodel corresponds with the functional perspective and the right side with the control-flow perspective of DeciClare.

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt

Verwijderd: Table 3

Verwijderd: Table 3

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt

Verwijderd: Table 3

Verwijderd: Table 3

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt, Engels (VS)

Verwijderd: Fig. 2

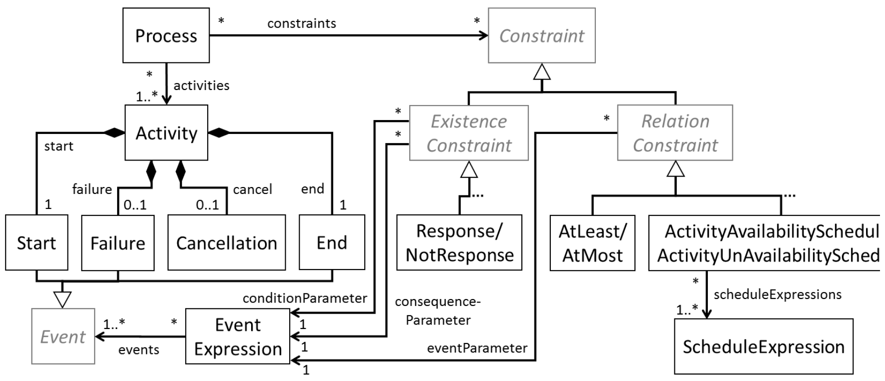

Fig. 2. The metamodel of the functional (left) and control-flow (right) perspectives

3.5 Supporting the data perspective

Until recently, no attempt had been made to support the data perspective in declarative languages (Debois, Hildebrandt, & Slaats, 2015). The languages lacked the expressive power to adequately model the (tacit) knowledge that governs the execution of loosely framed and KiPs (Lenz, Peleg, & Reichert, 2012; Mertens et al., 2015). Recent publications (Maggi, Dumas, García-Bañuelos, & Montali, 2013; Montali, Chesani, Mello, & Maggi, 2013; Slaats et al., 2013) have proposed extensions that add some data-awareness to the main declarative languages, including Declare. Even though this is a good start, none of the proposed extensions contain all of the elements needed to offer a complete solution for *requirement 2*. The extensions allow for data to be linked to constraints of the control-flow perspective (cf. section 4.1), but they do not sufficiently specify the data perspective itself (e.g., Which data attributes are there? When are they assigned/changed? etc.).

The decision modeling standard from the OMG group (Object Management Group, 2016), *Decision Model and Notation (DMN)*, served as inspiration during the formulation of a metamodel for the Declare data perspective. The primary goal of DMN is to provide a common notation for decision logic that can be understood by business users, business analysts and technical developers. This new standard is a response to the rising awareness that decisions can be just as important as, if not more important in some cases than, the process control-flow. By separating decisions from the process control-flow, the decision rules can be explicitly specified (Taylor, Fish, & Vanthienen, 2013; Van Der Aa et al., 2015). This means that the decision logic can be communicated, reused, adjusted to evolve within an ever-changing environment and used as justification for the choices being made.

DMN provides the constructs to model a complex decision in terms of its information requirements and the decision rules specifying the actual decision logic. For this purpose, a DMN decision model consists of two parts: a decision requirements graph and the decision logic. The former describes where the required information is coming from and can be depicted in one or more decision requirements diagrams. The latter describes the logic behind the decision, which is depicted in decision tables (Codasyl, 1982). Decision tables were chosen as the only representation of decision logic currently supported by DMN, because they are very popular and easy to understand. The current minimal scope of DMN will be expanded to include other representations (e.g., decision trees) and allow for references to other types of models (e.g., SBVR) in the future as the standard evolves (Vanthienen, 2014). The standard also describes its link to the popular imperative process modeling language BPMN explicitly through the Decision Task, but it positions itself as a separate perspective usable in other contexts as well (Object Management Group, 2016). However, it is unclear how it can be used in combination with other languages, including declarative process modeling languages.

For the purpose of this paper, the focus will only be on the underlying concepts used by DMN to model decisions. The same concepts will be integrated in our metamodel, as to keep the option open to also use the same visual notations in future research. The upper half of a decision table specifies the possible combinations of conditions, while the bottom half contains the actions to be taken (i.e., outcomes). This allows for complex decision logic to be modeled in one decision table or be spread over multiple tables. Fig. 3 shows the metamodel for decision tables. A metamodel of the decision requirements graphs has been omitted, as these graphs are non-essential for representing the actual decision logic and do not require integration into the other perspectives. Optionally, they can be used to specify where the data values in the decision tables come from as well as the relations between decision tables.

Verwijderd: Fig. 3

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt, Engels (VS)

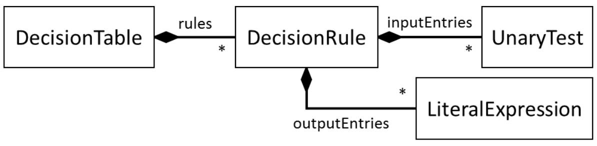

Fig. 3. The metamodel of the DMN language for specifying decision logic (based on Figure 51 from (Object Management Group, 2016))

The decision rules in Fig. 3 use unary tests as input entries. In turn, these tests use the values of data attributes to return a Boolean answer (e.g., patient gender=‘female’). Adding the concepts of a simple data model to Fig. 3 results in Fig. 4. A data record (e.g., patient file), possibly containing multiple sub records (e.g., medical history), consists of a set of data attributes (e.g., patient allergies), which each have one or more values connected to them (e.g., allergies={‘penicillin’, ‘peanuts’}). The unary tests are now Boolean statements using these values (e.g., ‘the patient has an allergy to penicillin?’).

Verwijderd: Fig. 3

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt, Engels (VS)

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt, Engels (VS)

Verwijderd: Fig. 3

Verwijderd: Fig. 4

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt, Engels (VS)

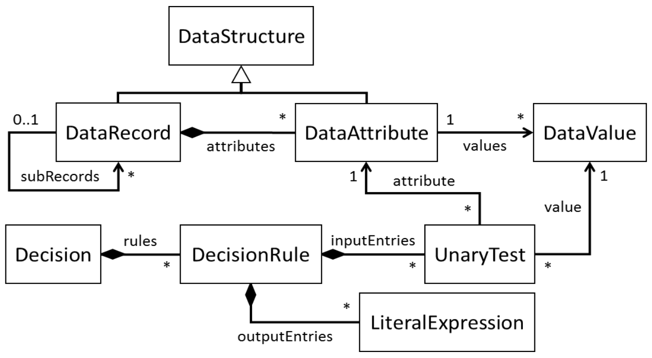

Fig. 4. The metamodel of the proposed data extension

Finally, a last remark on how the decisions themselves are to be modeled in DeciClare. The principles of imperative and declarative modeling can, in a sense, also be applied to decision logic. Imperative decision logic could correspond to decisions that have explicit outcomes for all possible combinations of the conditions (i.e., satisfy completeness property). Contrarily, a declarative description of decision logic could correspond to decisions that do not fulfill this property. These decisions only describe the outcomes for a subset of all the possible combinations of conditions, leaving the rest unspecified. For example, a decision based on one Boolean data attribute is specified completely (i.e., imperatively) if separate actions are defined for when the data value equals 'True' and when the data value equals 'False'. A declarative decision based on one Boolean data attribute specifies an action for one of the data values and leaves the other one unspecified. This follows the over- and underspecification tendencies of, respectively, the imperative and declarative way of thinking. As DeciClare takes on a declarative viewpoint, it follows that the specification of decision logic will also use this viewpoint.

3.6 Supporting the resource perspective

The resource patterns defined in (Russell, ter Hofstede, Edmond, & van der Aalst, 2004; Russell, van der Aalst, ter Hofstede, & Edmond, 2005; van der Aalst, 2011) can be used as inspiration for adding an resource perspective to the language (*requirement 3*). The patterns show typical use of resources during the execution of workflow processes. These patterns are also used to evaluate the resource perspective of workflow languages and systems based on their expressive power (Mulyar, 2005; Wohed, Russell, ter Hofstede, Andersson, & van der Aalst, 2009). Workflow processes are classified as tightly framed processes, so they do not fully correspond to our target processes. However, the patterns can also be used to identify general aspects related to resource use and availability that are applicable to loosely framed KiPs. **Table 4** presents those resource patterns that are not linked to other perspectives, in section 4.2 the other resource patterns will be presented. Based on the existing patterns, two additional patterns have been identified (denoted by an asterisk in **Table 4**), which are important for our context. The new patterns are related to resource availability. In order for the allocation of resources to happen correctly, information is needed about the availability of single resources and the amount of resources that can fulfill a certain role.

	Resource pattern	**Example**
R1	*Direct availability**	The hospital has two operating rooms (no.1 and no.2) and operating room no.1 is only available on Wednesdays between 8 am and 11 am.
R2	*Role-based availability**	The hospital has 20 available receptionists.
R3	*Simultaneous execution*	A recovery room can simultaneously handle up to 10 patients.

Table 4. Three resource patterns relevant to flexible and KiPs

To improve the practical usability of the original Declare language, a resource extension was already proposed, called ConDec-R (Jiménez-Ramírez & Barba, 2013). It enables reasoning about resources in Declare:

- Defining resources and the number of available instances of each resource type
- For each activity, linking the resource(s) required for its execution
- For each activity, adding estimates of its duration, cost or any other property

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt

Verwijderd: Table 4

Verwijderd: Table 4

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt

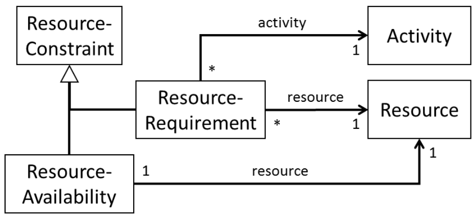

Fig. 5. The metamodel of the resource extension of the ConDec-R language

As there is again no metamodel publicly available, a metamodel was created based on the proposed semantics of (Jiménez-Ramírez & Barba, 2013) (Fig. 5). The ConDec-R extension does not clearly define what types of resources are included. Based on the example usage of the language in (Jiménez-Ramírez & Barba, 2013), resource reasoning capabilities were added with non-human resources in mind. However, the definition of the resources element in the metamodel can be expanded to also include human resources. Note that the metamodel already links Resource to Activity, and thus provides a link between the resource and functional/control-flow perspectives. This part of the metamodel will be discussed in section 4.2.

Verwijderd: Fig. 5

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt, Engels (VS)

In terms of the resources patterns of Table 4, this metamodel directly satisfies pattern *R2*. The example from Table 4 can be modeled as *ResourceAvailability(Receptionist, 20)*. However, that metamodel cannot express, for example, that the sum of available doctors and nurses is five (independent of the actual distribution). By introducing branched resource constraint templates, similar to the control-flow constraint templates, this can be modeled as *ResourceAvailability([Doctor and Nurse], 5)*. Additionally, some extra flexibility can be added by specifying a lower and upper bound similar to the existence templates, instead of specifying the exact number of available instances of a resources. Branched resource constraint templates *AtLeastAvailable* and *AtMostAvailable* have been added, which have a logical expression using resources (inclusive disjunction), a lower or upper bound and two time bounds as parameters. For example, *AtLeastAvailable([Doctor and Nurse], 5, 0, ∞)* expresses that the sum of available doctors and nurses at any time during the process has to be at least five. Although it is not directly required by the resources patterns, resource versions of the *ResourceAvailabilitySchedule* and *ResourceUnavailabilitySchedule* templates (*T6*) have also been added. The reasons for some activities to be (un-)available according to a certain schedule typically directly originates from the availability of a certain required resource according to the same or similar schedule. A typical example of this would be lab tests. Activities like performing blood and DNA testing rely on the availability of the lab for their execution. By adding these templates the models can represent the reason for this (un-)availability more precisely (i.e., Perform DNA test requires a lab and *ResourceAvailabilitySchedule([Lab], [every Tuesday between 11 am and 4 pm and every Friday between 9 am and 6 pm])*).

Verwijderd: Table 4

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt

Verwijderd: Table 4

Support for the direct availability of resources (*R1*), for example stating that operating room no.1 is only available on Tuesdays between 8 and 11 a.m., can be added by allowing a resource to be either a *direct resource* (e.g., operating room no.1) or a *resource role* (e.g., operating room). A direct resource then of course can be part of a resource role (e.g., operating rooms no.1 and no.2 can satisfy the role of operating room: *Operating room(operating room no.1, operating room no.2)*). By allowing the resources roles to be further specialized, even more elaborate reasoning about resources is made possible (e.g., a doctor can be have a specialization of neurosurgeon).

The last unsupported resource pattern deals with resources that can handle multiple resource allocations simultaneously (*R3*). This can be implemented by introducing one last constraint template: *SimultaneousCapacity*. It has a resource and an integer as input, combined with the standard two time parameters. The example from Table 4 can now be expressed as follows: *SimultaneousCapacity([Recovery room], 10, 0, ∞)*.

Integrating the solutions for the metamodel of the resource perspective, results in the metamodel presented in Fig. 6.

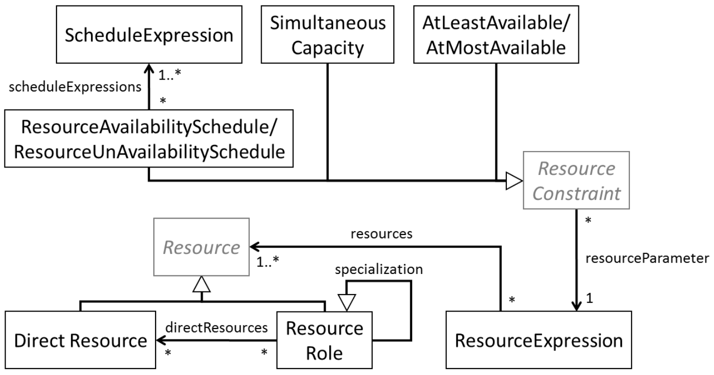

Fig. 6. The metamodel of the proposed resource extension

4 Integrating the perspectives

The previous section proposed extensions for Declare resulting in three standalone metamodels, enabling the functional/control-flow, data and resource perspectives of a process to be represented. These metamodels are merely partial solutions to *requirements 1, 2 and 3*, because these perspectives are heavily linked to one another. So the full expressive power of each perspective can only be achieved by considering its relations to the other perspectives. In this section, we will start from the metamodel of the functional and control-flow perspectives (Fig. 2) and step-by-step define the links with the data (Fig. 4) and resource (Fig. 6) perspectives. These links will open the door to expand the expressive power of the latter two perspectives even further. At the end, everything is integrated into one consistent metamodel that defines the new declarative process modeling language for loosely framed KiPs.

4.1 Integrating the data perspective into DeciClare

The rule patterns proposed by (Caron et al., 2013), which were already used in sections 3.3 and 3.4, also include two rule patterns (*D1* and *D2* in Table 5) that deal directly with the link between the data perspective (Fig. 4) on the one hand and the functional/control-flow perspectives (Fig. 2) on the other hand. Yet, similar data patterns can be defined by taking inspiration from database applications, which are typically used to store and access the data

Verwijderd: Table 4
Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt
Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt, Engels (VS)
Verwijderd: Fig. 6
Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt, Engels (VS)
Verwijderd: Fig. 2
Verwijderd: Fig. 4
Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt, Engels (VS)
Verwijderd: Fig. 6
Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt, Engels (VS)
Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt
Verwijderd: Table 5
Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt, Engels (VS)
Verwijderd: Fig. 4
Verwijderd: Fig. 2
Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt, Engels (VS)

in practical applications. Database query languages, like the popular Structured Query Language (SQL) (ISO/IEC, 2011), have defined a set of operations that can be performed on data: INSERT, SELECT, UPDATE and DELETE. This corresponds to how data records and their data attributes can be created, read, modified and removed during the execution of activities of a process. Based on these data operations from SQL, four additional data pattern have been formulated (*D3*, *D4*, *D5* and *D6* in **Table 5**).

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt

Verwijderd: Table 5

	Data pattern	Example
D1	*Data-driven existence*	If the patient has an open, displaced or unstable fracture, surgery will be performed at least once.
D2	*Data-driven relationships*	If the patient is a minor, then consent is required from the legal guardian before surgery can be performed.
D3	*Insert data*	If the patient does not have one yet for this hospital, a new patient file needs to be created during the registration of a patient.
D4	*Read data*	The doctor will need to examine the patient file during a consultation.
D5	*Update data*	If the patient reports information that does not correspond to his/her patient file (e.g., a different allergy) when taking his/her medical history, the patient file will be updated.
D6	*Delete data*	If a previously place internal fixation has been removed from the patient's arm, the data attribute in the patient's file stating that he/she has an internal fixation will be deleted.

Table 5. Six data patterns for KiPs

Data patterns *D1* and *D2* essentially require the option for all constraint templates supported by the functional and control-flow perspectives to link them with data. As a result, constraints would not need to apply for all process executions, as is customary in most declarative languages, but can apply for a mere subset of executions. For example, the two examples from **Table 5** can be modeled as *if('Open fracture = true' or 'Displaced fracture = true' or 'Unstable fracture = true') then AtLeast([Perform surgery], 1, 0, ∞)* and *if('Patient is a minor = true') then Precedence([Get consent from legal guardian], [Perform surgery], 0, ∞)*. To express that a constraint does apply to all process executions, it just needs to be linked with a trivial activation decision (i.e., always true). This will be represented in DeciClare by a constraint without the 'if'-part. For the constraint templates with a condition side (sometimes referred to as the source side), the activation decision on its own is not enough to activate the constraint (e.g., RespondedExistence, Response, Precedence…). Only the fulfilment of both the activation decision and condition of the constraint are enough for an activation. For example, consider a response constraint stating that if the patient has a displaced fracture, then surgery needs to be directly followed by applying a cast to the leg (i.e., *if('Displaced fracture = true') then Response([Perform surgery], [Apply cast], 0, ∞)*). For an activation of this constraint, the patient needs to have a displaced fracture *and* the doctors must have performed surgery. If one of these two conditions is not met, the constraint cannot be activated. A inactive constraint holds trivially (i.e., cannot be violated) and is commonly referred to as a vacuous satisfaction (Burattin, Maggi, van der Aalst, & Sperduti, 2012).

Verwijderd: Table 5

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt

Linking the functional, control-flow and data perspectives this way means that a constraint is always connected to an activation decision and the outcome of the decision is the activation of the constraint itself, while in the process making the LiteralExpression concept obsolete. This approach combines the power of the decision conditions to specify complex decisions based on data with the expressive power of the supported functional and control-flow perspectives as outcomes. This solution is similar to the decision-dependent constraints from (Mertens et al., 2015) and the first-order logic expressions from (Maggi, Dumas, et al., 2013), albeit here on a more general level. The decision-dependent constraints and first-order logic expressions can be seen as instantiations of the UnaryTest concept from Fig. 4. Linking the perspectives this way, has some consequences that need to be discussed in more detail and that sometimes also influence the design decisions made in previous section.

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt, Engels (VS)

Verwijderd: Fig. 4

First, an important question that arises: does a constraint that is activated after some activities have been performed cover only the activities after its activation, or also the activities that preceded its activation? The latter interpretation, that constraints apply retroactively, might be useful when the models are used for conformance checking of past execution, but can make enforcing them practically impossible when using the model to execute the process. For example, a constraint stating that a certain activity can only be executed once, can be activated after it has already been executed multiple times. Requiring the executing process instances to satisfy constraints up to and after the constraint activation seriously limits the expressive power and does not correspond to how decisions are taken in real life processes. So the applicability of an activated constraint will be defined as only the activities after the constraint activation.

A change to the definitions of the timed existence constraint templates is now required to maintain the consistency, because these templates specify that the time parameters (e.g., a and b) are relative to the process instance start, and thus, could have a retroactive effect. In order to prevent this, these parameters will be defined as relative to point in time when the constraint is activated. So for example, a timed 'AtLeast'-constraint could specify that a clinical examination has to take place within the first hour after the patient was marked as 'yellow' during the triage (i.e., *if('Triage color = yellow') then AtLeast([Clinical examination of patient], 1, 0, 1hour)*), independent of the fact that the patient might have arrived 20 minutes before he/she was triaged. The parameters can still express time relative to the process instance start by having a trivial activation decision or an activation decision that evaluates as true at the start of the targeted process instances.

Second, having a decision to activate a constraint is one thing, but in some cases it can be necessary to deactivate an activated constraint later on. Consider for example a constraint expressing that patients with a certain type of cancer need to receive at least six chemotherapy treatments. There are situations in which this constraint needs to be deactivated: if the patient reconsiders his consent after less than six treatments, if the patient experiences one or more complications after a couple of treatments and even the worst case scenario when the patient deceases before at least six treatments have been administered. As these reasons could not be foreseen when formulating the activation decision for the constraint, a way to deactivate constraints is needed. So every constraint will be given both an activation and deactivation decision. Of course, it is again allowed to be a trivial decision (i.e., always false) to represent a constraint that can never be deactivated. A constraint is not allowed to be violated as long it is activated, but does not always need to be fulfilled completely like as illustrated in the following example. Suppose the patient wants to stop the treatment after just two of the six treatments, then the constraint is deactivated even though the minimum of six treatments prescribed by the constraint was never satisfied. Of course, if an activated constraint would put a maximum on the number of treatments, this maximum is not allowed to be broken in the timeframe between the activation and deactivation of the constraint, although it is allowed after the deactivation. A consequence of having both an activation and deactivation decision is that is possible that a constraint is activated and deactivated multiple times during one process instance. For example, take a constraint expressing that if a patient has fluid in his/her longs, then this needs to be drained within 3 minutes unless the fluid has already been drained or dissolved by itself (i.e., *if('Fluid in longs = true') then AtLeast([Drain fluid], 1, 0, 3minutes) unless if('Fluid in longs = false')*). The first time fluid is found, the constraint is activated and the fluid will be drained. Afterwards, the constraint will be deactivated as the fluid has been drained. However, suppose the fluid immediately comes back. This activates the constraint again, leading to another deactivation after a second drainage.

Finally, a last unaddressed issue concerns the overlap in expressive power between the functional/control-flow and data perspectives (see requirement 4 in section 2). The main reason for this overlap is that the decisions can specify the same conditions as the conditions of the relation templates. For example, consider the response constraint stating that taking an X-ray has to be followed by a consultation (i.e., *Response([Take X-ray], [Doctor consultation], 0, ∞)*). This can also be expressed with an existence one constraint combined with an activation decision depending on the execution of the 'Take X-ray' activity (i.e., *if('Take X-ray has been executed = true') then AtLeast([Doctor consultation], 1), 0, ∞)*). This issue is resolved by not allowing decisions to use statements about the execution of events and activities. This eliminates the overlap, but at the same time reduces the expressive power of the resulting language. For instance, modeling a constraint like 'if surgery has been executed, then taking an X-ray will have to be executed at least 2 times' is no longer possible. To circumvent this side effect of the proposed solution, we will modify the relation templates to make such constraints possible once again. It comes down to allowing the relationship ends to be expressed in terms of existence templates. More specifically, the changes to the relation templates entail using branched existence (AtLeast), absence (AtMost), choice (AtLeastChoice) and AtMostChoice (see section 4.3) templates to express both the condition and consequence sides. Similar to the branching principle discussed earlier, a logic expression connecting these templates with the logical conjunction and inclusive disjunction operations will be used as condition or consequence side. The previous example can now be modeled as follows: *Response(AtLeast([Perform surgery], 1), AtLeast([Take X-ray], 2, 0, ∞)*). As a side effect, the negative versions of the relation templates are no longer needed explicitly, since they can be expressed by using the normal templates with an absence 0 (AtMost 0) constraint as its consequence side. The end result of this set of changes to the relation templates can express everything we could before the changes, but has eliminated the overlap and adds even more expressive power by making it possible to express relations between multiple executions of events.

Now that support for the first two data patterns has been added, the last four patterns (*D3*, *D4*, *D5* and *D6*) need to be addressed. For each of these data patterns two constraint templates can be defined to express whether this operation has to occur or cannot occur. This results in the following constraint templates: RequiredInsertion/ProhibitedInsertion, RequiredRead/ProhibitedRead, RequiredUpdate/ProhibitedUpdate and RequiredDeletion/ProhibitedDeletion. The templates relate data structures to activities of the process. The parameters of these templates are branched in a similar manner as for the control-flow constraint templates: a logical expression using activities (inclusive disjunction) and a logical expression of data structures (inclusive disjunction). Additionally, each of these templates has time parameters and an activation and deactivation decision. The last four examples of ~~**Table 5**~~ can now be modeled as follows:

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt

Verwijderd: **Table 5**

- D3: *if('Patient has been to this hospital before=false') then RequiredInsertion([Register patient], [Patient file], 0, ∞)*
- D4: *RequiredRead([Doctor consultation], [Patient file], 0, ∞)*
- D5: *optional RequiredUpdate([Take medical history of patient], [Patient file], 0, ∞)*
- D6: *RequiredDeletion([Remove fixation], [Patient file/fixations], 0, ∞)*

4.2 Integrating the resource perspective into DeciClare

The resource patterns defined by (Russell et al., 2004, 2005; van der Aalst, 2011) were used in section 3.6 to define a resource perspective for DeciClare. However, several patterns could not yet be discussed as they require links to the other perspectives. **Table 6** presents the remaining resource patterns relevant to loosely framed and KiPs. Again, an additional resource pattern was added (denoted by an asterisk). This pattern is a variation on the existing authorization pattern. The original pattern covers the authorization of a role to execute a certain task, while the added authorization pattern covers the authorization of roles to take certain decisions. As KiPs typically involve multiple process workers, establishing these authorization rules can be crucial in order to successfully execute the process.

	Resource pattern	Example
R4	*Direct allocation*	Brain surgery is performed in operation room no.1.
R5	*Role-based allocation*	Applying a cast requires at least one doctor or one nurse.
R6	*Activity-based authorization*	A surgeon is authorized to perform surgery.
R7	*Decision-based authorization**	A doctor is authorized to decide whether or not surgery to be performed.
R8	*Automatic execution (no human resources)*	Monitoring the heartrate of the patient is executed by the heartrate monitor machine.
R9	*Separation of duties*	For surgery, one person cannot take on both the surgeon and anesthesiologist roles.

Table 6. Six additional resource patterns linking the resource perspective to the other perspectives

The metamodel of the ConDec-R language (**Fig. 5**) directly satisfies resource pattern *R5*, so a similar solution will now be added to the metamodel of the resource perspective of DeciClare (**Fig. 6**). Analogous to how support was added to express resource availabilities, new constraint templates are added that use branching and upper/lower bounds to express resource allocations: *AtLeastUsage* and *AtMostUsage*. They have five parameters: a logical expression using activities (inclusive disjunction), a logical expression using resources (inclusive disjunction), a lower or upper bound and two time parameters. They can express that a certain activities needs at least or at most a certain number of resources (possibly, but not necessarily, of the same type). For example, it can state that applying a cast requires at least one doctor or one nurse: *AtLeastUsage([Apply cast], [Doctor or Nurse], 1, 0, ∞)*.

No further changes are required to support the direct allocation of resources (*R4*), as this is done analogous to the direct availability of resources (*R1*). For example, stating that all brain surgeries have to be performed in operating room no.1 can be expressed as *AtLeastUsage([Perform brain surgery], ["operating room no. 1"], 1, 0, ∞)*.

Verwijderd: Table 6

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt

Verwijderd: Fig. 5

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt, Engels (VS)

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt, Engels (VS)

Verwijderd: Fig. 6

The authorization patterns (e.g., *R6*, *R7* and *R8*) are also not yet supported by the metamodel. To implement these patterns two additional pairs of constraint templates are added: *(Not)DecisionAuthorization and (Not)ActivityAuthorization*. These templates have as input on the one hand a logical expression using resources (inclusive disjunction) and on the other hand, respectively, a constraint or an activity. They express that the connected resources have authorization to take the decision connected to a certain constraint or execute a certain activity. The examples for *R6*, *R7* and *R8* from **Table 6** can now be modeled as follows: *ActivityAuthorization([Orthopedic surgeon], [Perform surgery])*, *DecisionAuthorization([Doctor], [if('Displaced fracture? = true' or 'Open fracture? = true') then AtLeast([Perform surgery], 1, 0, ∞)])* and *ActivityAuthorization([Heart rate monitor], [Monitor heart rate])*. The last template can also be used to, for example, express that an expert system has authorization to diagnose a patient autonomously (i.e., *ActivityAuthorization([Expert system], [Diagnose patient])*).

Verwijderd: Table 6

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt

The final resource pattern is the one that expresses the principle of separation of duties (*R9*). To support this pattern, two more resource constraint templates are introduced. The resource *ResourceEquality* and *ResourceInequality* constraint templates are connected to two or more pairs of logical expressions, each consisting of one resource expression (inclusive disjunction) and one activity expression (inclusive disjunction). This allows for expressing that resources from certain activities have to be the same or have to be different, respectively, as the resources in some other activities. Consider the example from **Table 6** for *R9*, this can now be modeled as *ResourceInequality([Orthopedic surgeon or Anesthesiologist], [Perform surgery])*. An example of a resource equality on the other hand, would be to express that the surgeon that performed the surgery needs to be the same doctor as the one that does the follow-up examination of the patient (i.e., *ResourceEquality([Orthopedic surgeon], [Perform surgery or Doctor consultation])*).

Verwijderd: Table 6

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt

4.3 DeciClare

The metamodel of the DeciClare modeling language, which fully integrates the functional/control-flow, data and resource perspectives presented earlier, is presented in **Fig. 7**. Some constraint templates without additional relations have been omitted to improve readability. An Ecore implementation of the complete metamodel has been made available at [reference to DeciClare_ecore.zip].

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt, Engels (VS)

Verwijderd: Fig. 7

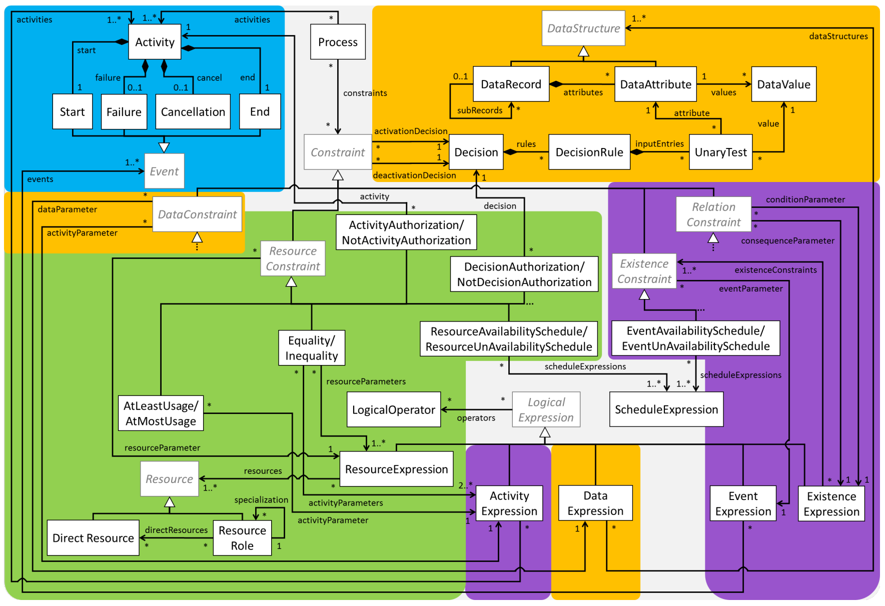

Fig. 7. The metamodel of the DeciClare language (blue = functional, purple=control-flow, orange = data and green = resource)

As summary of the solutions for each of the perspectives, the resulting constraint templates are presented in Table 7, Table 8 and Table 9. For Table 7, the second column contains the constraint templates from the most expressive Declare extension Declare^{++}. In Table 8 and Table 9 this column was omitted, as both perspectives are not supported by Declare^{++}. The last column of each table relates the constraint template to the corresponding patterns from Table 2, Table 3, Table 4, Table 5 and Table 6. The parameters X, I, C and B define the logical operators that can be used in the parameter expressions of the template as a result of the branching concept. The used letter refers to the option to use the exclusive disjunction (X), inclusive disjunction (I), conjunction (C) or both the inclusive disjunction and conjunction (B). The numerical bound enforced by some templates is expressed by the parameter n. The time interval linked to the Timed Declare definitions is represented by parameters a and b. The set of schedule expressions is denoted by S (*T6*). To save space, the activation and deactivation decisions for each constraint template and their optional versions have been omitted. The optional versions of each of the templates (rule pattern *G*) are of course still supported in the same way as the respective mandatory versions. In addition to presenting the solutions for the functional and control-flow perspectives of DeciClare, Table 7 also shows the results of a systematic review based on the principle of domain appropriateness (i.e., *requirement 4*) of the existing constraint templates. Some of the templates from Declare have been renamed in order to clarify their meaning and to emphasize the symmetry of the templates, while others have been omitted because they are either redundant (i.e., exclusive allowance) or syntactic sugar. The latter means that they can be easily expressed in terms of the other templates. Syntactic sugar can be categorized as construct redundancy, and thus, needs to be avoided in order to meet *requirement 4*. It is something that could be reintroduced when defining a concrete syntax to improve usability, but it does not belong in the metamodel of the language.

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt

Verwijderd: Table 7

Verwijderd: Table 8

Verwijderd: Table 9

Verwijderd: Table 7

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt

Verwijderd: Table 8

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt

Verwijderd: Table 9

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt

Verwijderd: Table 2

Verwijderd: Table 3

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt

Verwijderd: Table 4

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt

Verwijderd: Table 5

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt

Verwijderd: Table 6

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt

Verwijderd: Table 7

Type	Declare^{++}	DeciClare	Patterns
Existence templates	existence(n, Activity, a, b)	AtLeast(I, n, a, b)	*A, E, T1*
	absence(n, Activity, a, b)	AtMost(B, n, a, b)	*A, E, T1*
	exactly(n, Activity, a, b)	-	-
	init(Activity)	-	-
	last(Activity)	-	-
	choice(n, I, a, b)	AtLeastChoice(B , n, a, b)	*B, T1*
	exclusive choice(n, I, a, b)	-	-
	-	AtMostChoice(B, n, a, b)	*B, T1*
	-	EventAvailabilitySchedule(I, S)	*E, T6*
	-	EventUnavailabilitySchedule(I, S)	*E, T6*
Relation templates	responded existence(I_1, I_2)	RespondedPresence(B_1, B_2)	*B, D, E*
	responded absence(I_1, I_2)	-	-
	coexistence(I_1, I_2, a, b)	-	-
	not coexistence(I_1, I_2, a, b)	-	-
	response(I_1, I_2, a, b)	Response(B_1, B_2, a, b)	*C, D, E, T2*
	not response(I_1, I_2, a, b)	-	-
	chain response(I_1, I_2, a, b)	ChainResponse(B_1, B_2, a, b)	*C, D, E, T2*
	not chain response(I_1, I_2, a, b)	-	-
	alternate response(I_1, I_2, a, b)	AlternateResponse(B_1, B_2, a, b)	*C, E, T2*
	precedence(I_1, I_2, a, b)	Precedence(B_1, B_2, a, b)	*C, D, E, T2*
	not precedence(I_1, I_2, a, b)	-	-
	chain precedence(I_1, I_2, a, b)	ChainPrecedence(B_1, B_2, a, b)	*C, D, E, T2*
	not chain precedence(I_1, I_2, a, b)	-	-
	alternate precedence(I_1, I_2, a, b)	AlternatePrecedence(B_1, B_2, a, b)	*C, E, T2*
	succession(I_1, I_2, a, b)	-	-
	not succession(I_1, I_2, a, b)	-	-
	chain succession(I_1, I_2, a, b)	-	-
	not chain succession(I_1, I_2, a, b)	-	-
	alternate succession(I_1, I_2, a, b)	-	-
	-	AtLeastLag(B_1, B_2, n)	*E, T2/3/4/5/7*
	-	AtMostLag(B_1, B_2, n)	*E, T2/3/4/5/7*

Table 7. The control-flow constraint templates of Declare^{++} and DeciClare

Type	DeciClare	Patterns
Data templates	RequiredInsertion(I_{act}, I_{data}, a, b)	*D3*
	ProhibitedInsertion(I_{act}, I_{data}, a, b)	*D3*
	RequiredRead(I_{act}, I_{data}, a, b)	*D4*
	ProhibitedRead(I_{act}, I_{data}, a, b)	*D4*
	RequiredUpdate(I_{act}, I_{data}, a, b)	*D5*
	ProhibitedUpdate(I_{act}, I_{data}, a, b)	*D5*
	RequiredDeletion(I_{act}, I_{data}, a, b)	*D6*
	ProhibitedDeletion(I_{act}, I_{data}, a, b)	*D6*

Table 8. The data perspective constraint templates of DeciClare

Type	DeciClare	Patterns
Resource templates	AtLeastAvailable(I_{res}, n, a, b)	*R1, R2*
	AtMostAvailable(B, n, a, b)	*R1, R2*
	ResourceAvailabilitySchedule(I_{res}, S)	*T6*
	ResourceUnavailabilitySchedule(I_{res}, S)	*T6*
	SimultaneousCapacity(res, n)	*R3*
	AtLeastUsage(I_{act}, I_{res}, n, a, b)	*R4, R5*
	AtMostUsage(I_{act}, I_{res}, n, a, b)	*R4, R5*
	ActivityAuthorization(I_{res}, act)	*R6, R8*
	NotActivityAuthorization(I_{res}, act)	*R6, R8*
	DecisionAuthorization(I_{res}, constraint)	*R7, R8*
	NotDecisionAuthorization(I_{res}, constraint)	*R7, R8*
	ResourceEquality([I_{res}, I_{act}]$_{2..*}$)	*R9*
	ResourceInequality([I_{res}, I_{act}]$_{2..*}$)	*R9*

Table 9. The resource perspective constraint templates of DeciClare

5 Evaluation

(Hevner et al., 2004; Peffers, Rothenberger, Tuunanen, & Vaezi, 2012) present several types of evaluation that can be used to evaluate design science artifacts. A combination of expert evaluation and an illustrative scenario (Peffers et al., 2012) has been used to evaluate the artifact of this paper, which is the abstract syntax of the DeciClare modeling language. More specifically, domain experts have evaluated the abstract syntax with the help of the realistic arm fracture case, presented in section 3.1, as application scenario. The goal of this evaluation was to validate the effectiveness of the abstract syntax of the DeciClare language. The reasons for including certain modeling concepts in the abstract syntax have already been discussed in sections 2-4 and were each based on either literature or practical examples (i.e., informed arguments (Hevner et al., 2004)). In the first subsection, the set-up of the evaluation will be discussed. In the subsequent subsections the results of the evaluation concerning different quality types will be presented.

5.1 The evaluation set-up

The effectiveness of the design artifact, the metamodel of the DeciClare modeling language, can be assessed by applying the Conceptual Modeling Quality Framework (CMQF) (Nelson, Poels, Genero, & Piattini, 2012). This framework proposes several quality types based on what are called the quality cornerstones of conceptual modeling. The cornerstone that serves as the subject of our evaluation is the DeciClare metamodel, which in terms of the CMQF corresponds to a physical language that attempts to allow users to create a faithful representation of relevant concepts and phenomena in the physical domain, in this case loosely framed and KiPs.

In order to evaluate the DeciClare metamodel both directly and indirectly a realistic healthcare process of diagnosing and treating patients with suspected arm fractures [reference to Arm_fractures_case_-_Case_description.pdf] was introduced. Because a concrete syntax of DeciClare was not yet available, the textual representations of the proposed metamodel, as used throughout this paper, were used to serve as a temporary implementation of a concrete syntax for the DeciClare language. To make it as easy as possible for the participants, the metamodel instantiations were subsequently translated to more understandable natural language. For example, the constraint stating that surgery has to be performed within 1 hour if the symptoms for acute compartment syndrome have been fulfilled was originally modeled as "*AtLeast(Perform surgery, 1, 0, 1hour) activateIf[[Has acute compartment syndrome? = true]] deactivateIf[False]*" and now becomes "*If('Has acute compartment syndrome? = true') then Perform surgery has to be performed at least 1 time within at most 1 hour*". The original model of the case was based on the available literature and contained 15 activities, 11 resources, 15 data attributes and 79 constraints (35 functional/control-flow, 37 resource, 7 data and no guideline constraints) using 10 different constraint templates of the 22 available across all perspectives (not counting the mandatory/optional, positive/negative and at most/at least versions separately).

Using both the metamodel and the instantiation of the model, the CMQF framework can be used to evaluate the effectiveness of the language with three different quality types: perceived semantic quality, pragmatic quality and language-domain appropriateness. The perceived semantic quality of the case model represents the extent to which the model accurately and completely captures the domain knowledge of the experts, within the scope of the modeling task at hand. During the interviews issues with the semantic quality of the case model were treated as opportunities. They allow for discussions on the exact origin of the encountered issues and possible solutions. For instance, problems with the completeness of the language could be revealed, if it is not possible to formulate a satisfactory solution (i.e., missing concepts or relations in the DeciClare metamodel). The pragmatic quality on the other hand captures the extent to which the experts completely and accurately understand the parts of the case model that are relevant to their roles in the actual process. Basically, this gives an idea of the usefulness and value of the case model to the stakeholders of the process, and per extension of modeling with DeciClare in general.

Both perceived semantic quality and pragmatic quality indirectly evaluate the DeciClare metamodel by means of the developed model. The language-domain appropriateness quality type of the CMQF has as object of interest the metamodel of the language. It is the quality type related to the relationship between the physical language and the physical domain. However, an objective measurement of this relationship is impossible, because the universe of discourse related to the domain of loosely framed and KiPs is impossible to capture in its entirety. To circumvent this issue, a more subjective quality type can be used as approximation: the perceived language-domain appropriateness (Nelson et al., 2012). This quality type states that the metamodel, as understood by the users/modelers, must be appropriate to the understanding of the real-world universe of discourse by these users/modelers and for the ultimate use of the models created with the modeling language. This can be assessed by domain experts, who possess the in-depth knowledge of loosely framed KiPs to assess the appropriateness of the DeciClare metamodel to allow creating faithful representations of such processes.

By way of several semi-structured interviews with domain experts, all three quality types were evaluated for the DeciClare modeling language. In the first part of the interview, the expert was introduced to the research project and the DeciClare modeling language. In the second part, a brief introduction to the arm fracture case and its scope was given, followed by some time to thoroughly read the case description. When the experts were fully aware of the scope of the case and sufficiently understood the modeling language, they were presented with the natural language model of the case. All the rules of the model were discussed one by one as part of the semantic quality evaluation. If a rule did not conform to reality, corrections were discussed. In the last part of the interview the pragmatic quality of the model was discussed followed by a general discussion about the perceived language-domain appropriateness of the modeling language. The arm fracture case takes place in the emergency and orthopedic departments of hospitals. Therefore, four medical professionals that fulfill different roles along these departments in three of the biggest hospitals in Ghent were interviewed as domain experts: a surgeon from the AZ Sint-Lucas hospital, an emergency nurse/process quality promotor from the Ghent University Hospital and both an orthopedic surgeon and an orthopedic surgeon in training (intern) from the AZ Maria Middelares hospital in Ghent. The first three were very experienced and offered detailed insight into the process in practice, while the orthopedic surgeon in training was able to cast a different light on the process based on his training and his recent emergence into the complexity of real-life cases.

5.2 Evaluating the perceived semantic quality

In order to verify the sematic quality of the model the participants were instructed to validate the constraints of the model one by one based on their correctness and completeness. This revealed some interesting feedback on the model of the case: new activities and constraints were added, while others were modified or some even removed. The reasons ranged from a lack of detail in the original model concerning certain treatments and decisions to a sense that certain deadlines were difficult to enforce or merely hospital policies that differ from hospital to hospital. All of the suggested changes could be incorporated into the model with the available modeling constructs supplied by the DeciClare metamodel as to the eventual satisfaction of the experts. So this did not expose any issues concerning the completeness of the language. After each interview the model was updated with additional insights, resulting in a final model containing 25 activities, 17 resources, 21 data attributes and 282 constraints (126 functional/control-flow, 115 resource, 7 data and 34 guideline constraints) using 16 different constraint templates [ref to Arm_fractures_case_-_DeciClare_model_v5.pdf]. Most of the DeciClare examples used throughout this were taken from this updated model of the arms fracture case.

5.3 Evaluating the pragmatic quality

The pragmatic quality of the case model assesses the extent to which the experts understood the model and were able to use it. To this purpose, the experts were asked to identify examples of conforming and non-conforming process instances based on the model. All of them were able to perform this task satisfactory. For the non-conforming examples, they were also able to pinpoint the violated constraint or set of constraints.

Furthermore, the experts were asked if they could use the model to support their daily job. The responses confirmed the usefulness of the model. They all agree with the statement made earlier in the paper, that it can be a valuable tool when training new personnel. The orthopedic surgeon in training also added that he had not encountered anything similar during his education and training. They were taught using all sorts of specific clinical pathways (Rotter et al., 2010), but never got a higher level view of the process. He agreed that a model like this could have the potential to significantly shorten the internship duration, as they would no longer be limited to gaining experience from shadowing cases of more experienced doctors. Also, the experts see it as a great way to communicate changes to the process, be it due to policy changes, new diagnostic tests or new treatments. However, it was noted that the direct value of the model for a certain process actor somewhat diminishes as the experience level of the process actor rises. This was to be expected, as a rising experience level means that the knowledge made explicit by the model becomes more and more known. So less experienced process actors will be able to learn more from the model, than those with more experience. Yet, as noted by the emergency nurse, more experienced process actors can still get more insight into how other process actors, fulfilling different roles, make decisions. Her role in the process is to prepare the patient before seeing the doctor and provide support throughout. The model could help her identify and acquire the specific bits of information that the doctors will need in subsequent activities faster.

The experts were also asked how they currently use the information described in the model in their daily job. While they acknowledged the tacit nature of the knowledge by referring to its application as a sort of autopilot mode, they did say that they think more in terms of clinical pathways due to the nature of their training. The constraints expressed by the model refer to one or a few activities directly or to a relation between them, however, the experts would group them more according to certain clinical pathways (e.g., a Colles' fracture). An interesting feature to consider would be to allow filters on the model, showing only the constraints that apply for specified data values. This would allow the constraints for each clinical pathway to be presented separately, while still being able to connect them to the complete process. This is somewhat similar to how (Maggi, Mooij, & Van Der Aalst, 2011) filtered their Declare models based on previously executed activities to make them more understandable. Additionally, they indicated that in reality many decisions also come down to patient preferences. Typically, the patient is directly or indirectly responsible for much of the variability present in a healthcare process. For example, the constraints in the model do not account for patients that refuse treatment. Of course the patient preferences can be represented by data values, but this would make the decision logic of the affected constraints more complex. They agreed that for communication purposes this would be unfavorable, but for analysis and training purposes it could be very useful to know exactly which decisions need to account for patient preference and which do not.

5.4 Evaluating the language-domain appropriateness

In the last part of the interview, the DeciClare metamodel textual representations were discussed independently of the presented case. The experts agreed that the language is sufficiently expressive to also model the real underlying process without the limitations put on the scope (i.e., the complete emergency services process). They feel that the model would become a lot more complex, but that the added complexity would be in proportion to the added complexity of the real-world process. When considering applying the technique to other healthcare processes that fit the mold of flexible and KiPs, they were unable to think of examples (i.e., processes or types of constraints) that could not be expressed using the DeciClare language. The lack of identified construct deficits (Guizzardi, 2013) at this point or during the discussion of the case model, allows us to conclude that there are no apparent issues with the completeness of the language. Also, no construct excesses (Guizzardi, 2013) were uncovered, since real-life examples of each of the constraint templates could be given. Of course, this is because special attention was given to these properties during the design of the language.

One expert did, however, mention that after discussing some of the modeling elements from Declare that were left out due to being syntactic sugar, she would prefer to have these elements included as to simplify the models. The example that we discussed was to include the 'Init' template, from the Declare language, to express that the patient registration must happen as the first activity of a process instance (*Init([Register patient])*). Instead, this was expressed by 24 precedence relations between the patient registration activity and every other activity (i.e., *Precedence([AtLeast([Register patient], 1)], [AtLeast([Take medical history of patient], 1)], 0, ∞)* and the same for the 23 other activities). As stated in section 4.3, we are certainly open to adding these types of modeling elements as part of the eventual concrete syntax of the language. Introducing the Init/Last templates and the 'exactly' versions of each of the AtLeast/AtMost templates would have reduced the final case model from 282 to 218 constraints (78 functional/control-flow, 99 resource and 7 data and 34 guideline constraints). However, these templates were not included in the temporary concrete syntax used for the purpose of this evaluation, because it was a direct instantiation of the metamodel of DeciClare.

When considering the DeciClare language as a whole, the experts were most intrigued by how the different perspectives complemented each other. A good example of this from the case model were the rules that modeled what should happen when a patient has a confirmed displaced wrist or forearm fracture. The normal treatment involves surgery to reposition the broken bone (i.e., *if('Displaced fracture = true' or 'Open fracture = true') then AtLeast([Perform surgery], 1, 0, ∞)*), which requires at least a free operating room (i.e., *AtLeastUsage([Perform surgery], [operating room], 1)*), a surgeon (i.e., *AtLeastUsage([Perform surgery], [orthopedic surgeon], 1)*), an anesthesiologist (i.e., *AtLeastUsage([Perform surgery], [anesthesiologist], 1)*) and a surgical assistant or one or more OR or scrub nurses (i.e., *AtLeastUsage([Perform surgery], [surgical assistant or scrub nurse], 1)* and *AtLeastUsage([Perform surgery], [OR nurse], 1)*). But we also modeled the frequently occurring situation where the operating time (i.e., the time for which all the necessary resources need to be available) is not yet available (i.e., *if(['Displaced fracture? = true' and 'Operating time available? = false' and 'Confirmed Fracture? = Wrist'] or ['Displaced fracture = true' and 'Operating time available = false' and 'Confirmed Fracture = Forearm']) then Precedence([[AtLeast([Apply cast], 1) or AtLeast([Apply splint], 1)] and AtLeast([Stay in patient room], 1)], [AtLeast([Perform surgery], 1)], 0, ∞) unless if('Operating time available = true')*). Instead, the responsible doctor will order a temporary cast or splint to be applied (by himself or a nurse) to bridge the time until the operating time becomes available again. This situation could be modeled completely in DeciClare, because it supports and combines the necessary decisions, functional/control-flow restrictions and resource reasoning to express it. Therefore, the general conclusion of the experts was that the perceived model-domain appropriateness of DeciClare is satisfactory and that it is sufficiently expressive to model loosely framed and KiPs with the needed level of detail.

6 Related work

As stated in section 1, the languages that currently target loosely framed process (e.g., Declare and DCR graphs) do not sufficiently address all of its needs. Recently, some steps have been taken to add the support for different perspectives to Declare (Pesic, 2008). Many of these extensions were integrated in our solution (DeciClare) as explained in section 3. However, most of the extensions have been presented as separate languages. The Declare^{++} language is the only extension to combine multiple aspects of the functional/control-flow perspectives with a data perspective, but it does not fully support complex decisions (i.e., no disjunctions) and lacks a resource perspective. Compared to DeciClare, Declare^{++} is also less expressive when it comes to time concepts. Other declarative languages have recently also put more emphasis on the different perspectives of a process. (Slaats et al., 2013) added a data extension to DCR graphs and Guard-Stage-Milestone language (Hull et al., 2010) was introduced, which incorporates a data-centric workflow model with declarative elements for modeling life cycles of business artifacts. These languages and language extensions are, however, not enough to model some of the more complex decisions found in KiPs because they only allow simple decision logic (i.e., no disjunctions and decisions based on only one or at most just a few data values). Their focus is still very much on the process flow itself, and less on the decision processes that govern them. As stated by (Di Ciccio et al., 2016), the integration of the existing extensions is considered an avenue for future work, and to our knowledge had not yet been addressed in a satisfactory manner. In this paper, the integration of the different perspectives has been performed, and additionally, new elements have been added as to increase the expressive power of the resulting language.

A summary of the expressive power of all languages and extensions used in this paper in relation to the requirements of section 2 and their operationalization by means of patterns in sections 3 and 4 is presented in . When a pattern is covered by the language this is denoted by '+', when unaddressed by '-' and when not applicable by '/'. In some cases a certain pattern is somewhat addressed, but not to the full extend, which is denoted by '±'.

Verwijderd: Support for time semantics (Montali, 2010; Westergaard & Maggi, 2012) (Declare^{++} and Timed Declare) and branching (Di Ciccio et al., 2016; Montali, 2010) (Declare^{++} and TBDeclare) were added to the functional and control-flow perspectives. Furthermore, (Jiménez-Ramírez & Barba, 2013) proposed a basic resource perspective for Declare called ConDec-R. And finally, for the data perspective multiple extensions have been proposed. (Borrego & Barba, 2014; Montali et al., 2013) both propose extensions (Declare^{++} and unnamed) for Declare to add simple decision logic to Declare constraints (e.g., one data value). (Maggi, Dumas, et al., 2013) proposed a more powerful extension for Declare based on the First-Order Linear Temporal Logic (LTL-FO). They also connected data to constraints by way of activation conditions, but without the notion of deactivation conditions.

Omlaag verplaatst [2]: As stated by (Di Ciccio et al., 2016), the integration of the existing extensions is considered an avenue for future work, and to our knowledge had not yet been addressed in a satisfactory manner. In this paper, the integration of the different perspectives has been performed, and additionally, new elements have been added as to increase the expressive power of the resulting language.

Verwijderd:

Verplaatst (invoeging) [2]

Verplaatst (invoeging) [1]

Verwijderd: given

Gewijzigde veldcode

		Declare (Pesic 2008)	Timed Declare (Westergaard et al. 2012)	TBDeclare (Di Ciccio et al. 2016)	ConDec-R (Jiménez-Ramírez et al. 2013)	Data-aware Declare (Borrego et al. 2014)	Declare++ (Montali 2010; Montali et al. 2013)	Declare based on LTL-FO (Maggi et al. 2013)	DMN (Object Management Group 2016)	Declare-R-DMN (Mertens et al. 2015)	DeciClare
1	A	+	+	+	+	+	+	+	/	+	+
	B	+	+	+	+	+	+	+	/	+	+
	C	+	+	+	+	+	+	+	/	+	+
	D	+	+	+	+	+	+	+	/	+	+
	E	-	-	±	-	-	+	-	/	-	+
	F	-	-	-	-	-	-	-	/	-	+
	G	+	+	+	+	+	+	+	/	+	+
	T1	-	+	-	-	-	-	-	/	-	+
	T2	-	+	-	-	-	+	-	/	-	+
	T3	-	-	-	-	-	-	-	/	-	+
	T4	-	-	-	-	-	-	-	/	-	+
	T5	-	-	-	-	-	-	-	/	-	+
	T6	-	-	-	-	-	-	-	/	-	+
	T7	-	-	-	-	-	-	-	/	-	+
2		-	-	-	-	±	±	+	+	+	+
3	R1	-	-	-	-	-	-	-	/	-	+
	R2	-	-	-	+	-	-	-	/	+	+
	R3	-	-	-	-	-	-	-	/	-	+
4	R4	-	-	-	-	-	-	-	/	-	+
	R5	-	-	-	+	-	-	-	/	+	+
	R6	-	-	-	-	-	-	-	/	-	+
	R7	-	-	-	-	-	-	-	/	+	+
	R8	-	-	-	-	-	-	-	/	-	+
	R9	-	-	-	-	-	-	-	/	-	+
	D1	-	-	-	-	+	+	+	/	±	+
	D2	-	-	-	-	+	+	+	/	+	+
	D3	-	-	-	-	-	-	-	/	-	+
	D4	-	-	-	-	-	-	-	/	-	+
	D5	-	-	-	-	-	-	-	/	-	+
	D6	-	-	-	-	-	-	-	/	-	+

Table 10. A comparison of the languages used in this paper based on the requirements from section 2 and their operationalization as patterns from section 3 and 4

The DeciClare language as proposed in this paper is categorized as a mixed-perspective modeling language and not as hybrid languages. Within business process modelling research, a hybrid language is used to signify the combination of imperative and declarative modeling languages into one approach (De Smedt, De Weerdt, Vanthienen, & Poels, 2016; Goedertier et al., 2015; Madhavji & Schafer, 1991; Maggi, Slaats, & Reijers, 2014; Sadiq, Orlowska, & Sadiq, 2005). So the question remains how this work compares to others that position themselves as hybrid modeling techniques. For hybrid modelling languages, the idea is that some of the information can be better expressed using an imperative language, while for other parts this is true when using a declarative language. DeciClare does not focus on combining representation within one specific perspective but instead focuses on combining different perspectives.

Met opmaak: Lettertype:(Standaard) Times New Roman

Verwijderd:

Verwijderd: hybrid

Verwijderd: , but this term is not clearly defined in literature.

Verwijderd: In a lot of literature it

Gewijzigde veldcode

Met opmaak: Nederlands

Met opmaak: Nederlands

Verwijderd: ,

Verwijderd: however, some other papers have used it in a broader sense (Frank, 2014; Westergaard & Slaats, 2013).

Met opmaak: Engels (VS)

Met opmaak: Engels (VS)

Verwijderd: Therefore, this paper will be positioned in this domain according to the biggest differentiating properties of hybrid modeling techniques. Hybrid modeling languages have in common that they all attempt to combine multiple perspectives into one modeling approach. This definition does not specify which perspectives are included or even how these are combined. However, these two aspects can be used to classify existing process modeling languages that are positioned as hybrid modeling techniques.

Verwijderd: Most hybrid modeling languages focus on combining multiple perspectives within one of these two classes. All of the papers that combine imperative and declarative process modeling languages into one modeling approach, as referenced in the previous paragraph, combine two perspectives related to representation. Both types of languages can express the same information, but they differ in the way that they represent the information to the users.

Verwijderd: T

The DeciClare language proposed in this paper is classified as a mixed-perspective process modeling language and not as a multi-perspective process modelling language. The latter combines two or more perspectives by using them side by side, keeping the metamodels of these languages separated. Mixed-perspective process modeling language combine the perspectives by integrating the metamodels into one resulting metamodel. Coloured Petri nets (Jensen, 1997) is an example of mixed-perspective approach, as it uses one integrated metamodel to combine the control-flow perspective of regular Petri nets with a data perspective. The Declare extensions Declare^{++}, ConDec-R, data-aware Declare and Declare based on LTL-FO can also be classified as mixed-perspective modeling languages. The difference is that they cover at most three perspectives, where this paper covers the main four.

7 Conclusion

In this paper the metamodel of the mixed-perspective declarative modeling language DeciClare has been presented. This metamodel is the final step in a journey that started with the formulation of requirements of a modeling language for loosely framed and KiPs. These requirements were translated into sets of smaller requirements (i.e., patterns) based on relevant literature, which represent the operationalization of the original requirements. Subsequently, the patterns were analyzed one by one and addressed. Although several existing languages already targeted this type of processes, none could offer the means to model all of its intricacies as specified by the requirements. The foundation for the DeciClare language was the popular declarative process modeling language Declare. Based on the requirements, an evaluation was performed to identify the aspects supported and unsupported by Declare. Because the Declare language received many standalone extensions since its conception, the relevant extensions were also evaluated in the same way. This helped reveal exactly those parts that could be reused and those that necessitated new solutions. After the development of separate solutions for the different perspectives in response to the requirements, an integration step was performed. The integration of the perspectives is an important step for the academic and practical applicability of declarative languages. The myriad of standalone extensions might each add support for some specific aspects, but an approach combining of all of these extensions is a vital step to capturing these processes completely in a process model.

An essential aspect of modeling KiPs is, of course, modeling the knowledge itself. This knowledge governs the process and all of its numerous variations. As it is typically of a tacit nature, it is paramount that a language to model these processes offers a way to explicitly represent the decision logic that expresses it. DeciClare links each declarative constraint with decision logic that can make the constraint active and that can deactivate an active constraint. Process models can therefore represent the process very precisely, as well as the knowledge governing it. This facilitates the sharing of knowledge with other knowledge workers and with other stakeholders of the process (e.g., the patient). Additionally, a more uniform and transparent service can be offered when the knowledge workers are more informed and they have a way to communicate changes to the process at their disposal.

Verplaatst (invoeging) [3]

Verwijderd: can be

Verwijderd: mixing perspectives that relate to content. The metamodels of the different content perspectives were integrated into one metamodel, because they cannot be viewed separately from one another due to the use of overlapping concepts and relations. The DeciClare language does not combine perspectives that relate to representation, as the language is fully declarative. Adding multiple visual perspectives has not yet been considered as a visual syntax for the language is not in the scope of this paper.

Verwijderd: mixing perspectives that relate to content.

Verwijderd: Alternatively, Coloured Petri nets (Jensen, 1997) combines two perspectives related to content: the control-flow perspective of regular Petri nets and a data perspective. These two perspectives express different and complementary information, each related to another functional concept.

Omhoog verplaatst [3]: The DeciClare language proposed in this paper can be classified as a mixed-perspective process modeling language mixing perspectives that relate to content. The metamodels of the different content perspectives were integrated into one metamodel, because they cannot be viewed separately from one another due to the use of overlapping concepts and relations. The DeciClare language does not combine perspectives that relate to representation, as the language is fully declarative. Adding multiple visual perspectives has not yet been considered as a visual syntax for the language is not in the scope of this paper. The Declare extensions Declare^{++}, ConDec-R, data-aware Declare and Declare based on LTL-FO can also be classified as mixed-perspective modeling languages mixing perspectives that relate to content. The difference is that they cover at most three perspectives, where this paper covers the main four.

Verwijderd:

Verwijderd:

Verwijderd:

Verwijderd:

Verwijderd:

Met opmaak: Lettertype:(Standaard) Times New Roman

Verwijderd:

Met opmaak: Standaard,Uitgevuld, Tabs: 2,47 cm, Links

Verwijderd:

Verwijderd:

Verwijderd:

Verwijderd: hybrid

Along the way, this paper make also several contributions to the research domain in the form of other metamodels. The first metamodel that was presented is the one of the Declare language (Fig. 1). Although a partial metamodel for Declare had been proposed before (Bernardi et al., 2012), it was not fully compatible with the original definitions of the language and its corresponding modeling tool. This metamodel was subsequently enriched with additional concepts and relations in response to the requirements. The resulting model (Fig. 2) presents a direct extension of the Declare language, since it originally only covered the functional and control-flow perspectives. When considering the addition of a data perspective, the concepts from DMN and a simple data representation were used to create a metamodel for the data side (Fig. 4). And finally, a metamodel of the resource perspective was presented in Fig. 6. This model was based on the proposed metamodel of ConDec-R extension (Fig. 5) for Declare, for which a metamodel was also lacking in literature.

The effectiveness of the abstract syntax of the DeciClare language was evaluated by way of semi-structured interviews with domain experts. An example of a loosely framed and KiP was used (i.e., a simplification of the emergency services process only considering patients with possible fractures in the arms) to make the discussions and questions more relatable for the experts. Based on a general explanation of DeciClare, the case description and a textual DeciClare model of the case, the language-domain appropriateness of the language was assessed. The semantic and pragmatic quality of the case model were discussed, and afterwards a general discussion took place about the DeciClare language. This lead to two main takeaways.

The first conclusion that can be drawn from the evaluation is that the DeciClare language appears to be sufficiently expressive to model loosely framed and KiPs. The original case model was enriched during each of the interviews, but nothing was encountered that could not be expressed in DeciClare. The domain experts also saw no potential problems for the model of the case to be expanded to represent the whole process of diagnosis and treatment in the emergency service department, apart from the proportional increase in complexity. Additionally, they were unable to come up with constraints that could not be captured by a DeciClare model, but might occur in different processes of this type.

The second conclusion is that the experts deemed modeling the processes with DeciClare useful for training and communication purposes. It can be a tool to make these processes, which are generally perceived as black boxes by management, more transparent to all stakeholders. It also allows for process actors to anticipate the needs of other process actors more correctly, because the models can offer insight into their role in the process and their corresponding needs. However, experience plays a vital role here. Inexperienced process actors will benefit more from the added insight, as more experienced actors already know much of the information relayed by the model.

In general, DeciClare offers modelers a new way to model loosely framed KiPs. Although it is a new language, it still has a familiar feel to it because existing modeling concepts were integrated whenever possible. Compared to alternative languages, the main advantage of DeciClare stems from its integration of the different perspectives of the process. Most of the languages targeting the same types of processes offer only one perspective to the process, resulting in important information getting lost during the modeling phase (as illustrated by). DeciClare allows each of the essential perspectives of the processes to be represented, as well as the links between these perspectives. This is somewhat similar to what the BPMN/DMN language extension attempts to do for tightly framed processes. BPMN/DMN also combines the functional, control-flow, resource and data perspectives, but using the imperative modeling paradigm. On the flipside, approaches like DeciClare and BPMN/DMN introduce a lot of added

Verwijderd: Fig. 1

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt, Engels (VS)

Verwijderd: Fig. 2

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt, Engels (VS)

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt, Engels (VS)

Verwijderd: Fig. 4

Verwijderd: Fig. 6

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt, Engels (VS)

Verwijderd: Fig. 5

Met opmaak: Lettertype:(Standaard) Times New Roman, 10 pt, Engels (VS)

Gewijzigde veldcode

complexity. It is a typical tradeoff between simplicity and expressive power. In case of DeciClare, we feel that the latter is justified because the targeted processes are inherently complex. The resulting models are still an abstraction of reality that offers a view of the process with reduced complexity, but they are also comprehensive enough to represent the most important intricacies of the process. Additionally, we envision most practical applications of these models (e.g., process mining, simulation, conformance checking and run-time support) to be supported by way of (semi-)automated techniques.

Finally, the DeciClare language was positioned in the business process modeling domain using several possible classifications. The first classification is according to the types of processes that the language targets, in this cases loosely framed KiPs. Of course, this was a design decision. The second classification positions the language within the mixed-perspective modeling domain. DeciClare can be classified as a mixed-perspective modeling technique because multiple content perspectives (i.e., functional, control-flow, data and resource) are integrated into one metamodel. The last classification is according to the usage scenarios supported by the language. At design-time it provides the means to convert tacit knowledge generally associated with loosely framed processes to explicit knowledge, allowing this knowledge encapsulated in the models to be communicated with others. The language makes it possible to explain and justify why things are done a certain way, by making the decision logic transparent for all stakeholders. In an execution context it facilitates conformance checking of past and present process executions.

Verwijderd: hybrid

Verwijderd: Two important dimensions of hybrid modeling were identified: representation and content.

Verwijderd: hybrid

Verwijderd: that takes on a mixed-perspective approach and mixes perspectives related to content. This means that the language uses just one representational perspective (i.e., declarative), but

8 Limitations and future work

Firstly, the scope of this paper was limited to the abstract syntax of the modeling language. A visual syntax for a modeling language requires a different research approach (i.e., cognitive instead of semantic) and a different evaluation, and so we chose to split the two into separate research steps that each can be performed in greater depth. Therefore, the next step will be to develop a visual syntax for the DeciClare language. A second limitation is intrinsic to declarative modeling, namely that the models tend to be complex and difficult to understand for human users. Of course, the choice for a good visual syntax will also play a role in this, but an extended metamodel will also see the complexity of possible models rise. There is some research being done on how to reduce the size of declarative models (Di Ciccio, Maggi, Montali, & Mendling, 2015; Di Ciccio & Mecella, 2015; Maggi, Bose, & van der Aalst, 2013), which could help mitigate the problem. However, it is not yet clear if these techniques are still applicable, and if so, how they would be applied taking into account the introduction of activation decisions and deactivation decisions for constraints. Another possible way to reduce this issue came up during the evaluation, specifically the option to add data-filters to the visualization of a model. This makes it possible to view certain cases in isolation (e.g., clinical pathways), removing all unrelated or inactive constraints from the model. A third limitation was already discussed in section 3.1, namely that the case used in this paper had some limitations on the scope. Instead of modeling the complete real-world process of the emergency and orthopedic departments of a hospital, we limited the scope to certain types of arm fractures. This was necessary to reduce the complexity process and its resulting DeciClare model to reasonable levels. As was noted by the domain experts during the evaluation, the added complexity would not have changed anything to the conclusions of this paper. In the next step of our research project, we are going to use DeciClare to model the whole real-life process in the emergency department of a big Belgian hospital. A final limitation relates to a design decision made in section 4.1. Although most of the language very much supports conformance checking, the choice to not have activated constraints apply retroactively can be seen as a small restriction on its conformance checking potential. However, these types of constraints would be very hard to enforce, as process actors would only know in hindsight whether some constraint should have been enforced. So this concept is not very useful for any sort of active use of process models.

The future research plans start with the first three limitations discussed above. Additionally, we will study how DeciClare can support simulation techniques and provide a foundation for offering guidance and recommendations to the process workers at run-time.

References

Barba, I., Lanz, A., Weber, B., Reichert, M., & Del Valle, C. (2012). Optimized time management for declarative workflows. *Lecture Notes in Business Information Processing*, *113 LNBIP*, 195–210. http://doi.org/10.1007/978-3-642-31072-0_14

Bernardi, M. L., Cimitile, M., Di Lucca, G., & Maggi, F. M. (2012). Using declarative workflow languages to develop process-centric Web Applications. In *Proceedings of IEEE International EDOC 2012 Workshops* (pp. 56–65). Beijing, China. http://doi.org/10.1109/EDOCW.2012.17

Borrego, D., & Barba, I. (2014). Conformance checking and diagnosis for declarative business process models in data-aware scenarios. *Expert Systems with Applications*, *41*(11), 5340–5352. http://doi.org/10.1016/j.eswa.2014.03.010

Burattin, A., Maggi, F. M., van der Aalst, W. M. P., & Sperduti, A. (2012). Techniques for a posteriori analysis of declarative processes. In *Proceedings of IEEE International EDOC 2012* (pp. 41–50). Beijing, China. http://doi.org/10.1109/EDOC.2012.15

Caron, F., Vanthienen, J., & Baesens, B. (2013). Comprehensive rule-based compliance checking and risk management with process mining. *Decision Support Systems*, *54*(3), 1357–1369. http://doi.org/10.1016/j.dss.2012.12.012

Codasyl. (1982). A Modern Appraisal Of Decision Tables. In *Report of the Decision Table Task Group, ACM* (pp. 230–232). New York, USA. http://doi.org/https://feb.kuleuven.be/prologa/CODASYL82/CODASYL82.pdf

Dadam, P., Reichert, M., & Kuhn, K. (2000). Clinical Workflows - The Killer Application for Process-oriented Information Systems? In *Proc. of BIS 2000* (pp. 36–59). Poznan, Poland: Springer-Verlag. http://doi.org/10.1007/978-1-4471-0761-3

De Smedt, J., De Weerdt, J., Vanthienen, J., & Poels, G. (2016). Mixed-Paradigm Process Modeling with Intertwined State Spaces. *Business & Information Systems Engineering*, *58*(1), 19–29. http://doi.org/10.1007/s12599-015-0416-y

Debois, S., Hildebrandt, T., & Slaats, T. (2015). Concurrency and Asynchrony in Declarative Workflows. In *BPM'15. LNCS, vol. 9253* (pp. 72–89). Innsbruck, Austria: Springer.

Di Ciccio, C., Maggi, F. M., & Mendling, J. (2016). Efficient discovery of Target-Branched Declare constraints. *Information Systems*, *56*, 258–283. http://doi.org/10.1016/j.is.2015.06.009

Di Ciccio, C., Maggi, F. M., Montali, M., & Mendling, J. (2015). Ensuring model consistency in declarative process discovery. In *BPM'15. LNCS, vol. 9253* (pp. 144–159). Innsbruck, Austria: Springer.

http://doi.org/10.1007/978-3-319-23063-4_9

Di Ciccio, C., Marrella, A., & Russo, A. (2015). Knowledge-Intensive Processes: Characteristics, Requirements and Analysis of Contemporary Approaches. *Journal on Data Semantics*, *4*(1), 29–57. http://doi.org/10.1007/s13740-014-0038-4

Di Ciccio, C., & Mecella, M. (2015). On the Discovery of Declarative Control Flows for Artful Processes. *ACM Trans. Manage. Inf. Syst.*, *5*(4), 24:1-24:37. http://doi.org/10.1145/2629447

Dumas, M., La Rosa, M., Mendling, J., & Reijers, H. A. (2013). *Fundamentals of business process management*. Springer-Verlag Berlin Heidelberg. http://doi.org/10.1007/978-3-642-33143-5

Fahland, D., Lübke, D., Mendling, J., Reijers, H. a., Weber, B., Weidlich, M., & Zugal, S. (2009). Declarative versus imperative process modeling languages: The issue of understandability. In T. A. Halpin, J. Krogstie, S. Nurcan, E. Proper, R. Schmidt, P. Soffer, & R. Ukor (Eds.), *BPMDS/EMMSAD'09 Workshop on Enterprise, Business-Process and Information Systems Modeling, held at CAiSE'09. LNBIP, vol. 29* (pp. 353–366). Amsterdam, The Netherlands: Springer. Retrieved from http://link.springer.com/chapter/10.1007/978-3-642-01862-6_29

Ferreira, H. M., & Ferreira, D. R. (2006). An Integrated Life Cycle for Workflow Management Based on Learning and Planning. *International Journal of Cooperative Information Systems*, *15*(4), 485–505. http://doi.org/10.1142/S0218843006001463

Frank, U. (2014). Multi-perspective enterprise modeling: Foundational concepts, prospects and future research challenges. *Software and Systems Modeling*, *13*(3), 941–962. http://doi.org/10.1007/s10270-012-0273-9

Goedertier, S., Vanthienen, J., & Caron, F. (2015). Declarative business process modelling: principles and modelling languages. *Enterprise Information Systems*, *9*(2), 161–185. http://doi.org/10.1080/17517575.2013.830340

Guizzardi, G. (2013). Ontology-Based Evaluation and Design of Visual Conceptual Modeling Languages. In I. Reinhartz-Berger, A. Sturm, T. Clark, S. Cohen, & J. Bettin (Eds.), *Domain Engineering. Product Lines, Languages and Conceptual Models* (pp. 317–347). Springer Berlin Heidelberg. http://doi.org/10.1007/978-3-642-36654-3_13

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design science in Information Systems research. *MIS Quarterly*, *28*(1), 75–105.

Hull, R., Damaggio, E., Fournier, F., Gupta, M., Heath, F., Hobson, S., … Vaculín, R. (2010). Introducing the guard-stage-milestone approach for specifying business entity lifecycles. In *Web Services and Formal Methods. LNCS, vol. 6551* (pp. 1–24). Springer. http://doi.org/10.1007/978-3-642-19589-1_1

ISO/IEC. (2011). *ISO/IEC 9075-2:2011(en): Information technology - Database languages - SQL - Part 2: Foundation (SQL/Foundation)*. Retrieved from https://www.iso.org/obp/ui/#iso:std:iso-iec:9075:-2:ed-4:v1:en

Jensen, K. (1997). *Coloured Petri Nets. Basic concepts, analysis methods and practical use*. Berlin: Springer-Verlag.

Jiménez-Ramírez, A., & Barba, I. (2013). Generating multi-objective optimized business process enactment plans. In *Advanced Information Systems Engineering. LNCS, vol. 7908* (pp. 99–115). Springer. Retrieved from http://link.springer.com/chapter/10.1007/978-3-642-38709-8_7

Krogstie, J. (2013). Perspectives to process modeling. In *BPM. SCI, vol. 444* (pp. 1–39). Springer.

Lanz, A., Weber, B., & Reichert, M. (2014). Time patterns for process-aware information systems. *Requirements Engineering*, *19*(2), 113–141. http://doi.org/10.1007/s00766-012-0162-3

Lenz, R., Peleg, M., & Reichert, M. (2012). Healthcare Process Support: Achievements, Challenges, Current Research. *International Journal of Knowledge-Based Organizations (IJKBO)*, *4*(2).

Lenz, R., & Reichert, M. (2007). IT support for healthcare processes - premises, challenges, perspectives. *Data and Knowledge Engineering*, *61*, 39–58. http://doi.org/10.1016/j.datak.2006.04.007

Lu, R., & Sadiq, S. W. (2007). A survey of comparative business process modeling approaches. In *BIS'07. LNCS, vol. 4439* (pp. 82–94). Poznan, Poland: Springer. Retrieved from http://link.springer.com/chapter/10.1007/978-3-540-72035-5_7

Lu, R., Sadiq, S. W., & Governatori, G. (2009). On managing business processes variants. *Data & Knowledge Engineering*, *68*(7), 642–664. http://doi.org/10.1016/j.datak.2009.02.009

Madhavji, N. H., & Schafer, W. (1991). Prism - Methodology and Process-Oriented Environment. *IEEE Transactions on Software Engineering*, *17*(12), 1270–1283. http://doi.org/10.1109/32.106987

Maggi, F. M., Bose, R. P. J. C., & van der Aalst, W. M. P. (2013). A knowledge-based integrated approach for discovering and repairing declare maps. In *CAiSE'13. LNCS, vol. 7908* (pp. 433–448). Valencia, Spain. http://doi.org/10.1007/978-3-642-38709-8_28

Maggi, F. M., Dumas, M., García-Bañuelos, L., & Montali, M. (2013). Discovering data-aware declarative process models from event logs. In F. Daniel, J. Wang, & B. Weber (Eds.), *BPM'13. LNCS, vol. 8094* (pp. 81–96). Beijing, China: Springer. http://doi.org/10.1007/978-3-642-40176-3_8

Maggi, F. M., Mooij, A. J., & Van Der Aalst, W. M. P. (2011). User-guided discovery of declarative process models. *CIDM 2011: IEEE Symposium on Computational Intelligence and Data Mining*, 192–199. http://doi.org/10.1109/CIDM.2011.5949297

Maggi, F. M., Slaats, T., & Reijers, H. A. (2014). The automated discovery of hybrid processes. In *BPM'14. LNCS, vol. 8659* (pp. 392–399). Haifa, Israel: Springer. Retrieved from http://link.springer.com/chapter/10.1007/978-3-319-10172-9_27

Mertens, S., Gailly, F., & Poels, G. (2015). Enhancing declarative process models with DMN decision logic. In *Enterprise, Business-Process, and Information Systems Modelling at BPMDS/EMMSAD'15, held at CAiSE'15. LNBIP, vol. 214* (Vol. 214, pp. 151–165). Stockholm, Sweden: Springer. http://doi.org/10.1007/978-3-319-19237-6_10

Montali, M. (2010). *Specification and verification of declarative open interaction models. A Logic-Based Approach. LNBIP, vol. 56*. Springer. Retrieved from http://link.springer.com/content/pdf/10.1007/978-3-642-14538-4.pdf

Montali, M., Chesani, F., Mello, P., & Maggi, F. M. (2013). Towards Data-Aware Constraints in Declare. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing. SAC* (pp. 1391–1396). New York: ACM Press. http://doi.org/10.1145/2480362.2480624

Moody, D. L. (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. Retrieved from http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5353439

Mukkamala, R. R. (2012). *A Formal Model For Declarative Workflows - Dynamic Condition Response Graphs*. IT-Universitetet i København.

Mulyar, N. (2005). Pattern-based Evaluation of Oracle-BPEL (v.10.1.2). In *Technical report BPM-05-24, BPMcenter.org*.

Mulyar, N., Pesic, M., van der Aalst, W. M. P., & Peleg, M. (2007). Declarative and procedural approaches for modelling clinical guidelines: addressing flexibility issues. In *BPM'07 Workshops. LNCS, vol. 4928* (Vol. 4928, pp. 335–346). Brisbane, Australia: Springer. Retrieved from http://link.springer.com/chapter/10.1007/978-3-540-78238-4_35

Murata, T. (1989). Petri nets: properties, analysis and applications. *Proceedings of the IEEE*. http://doi.org/10.1109/5.24143

Nelson, H. J., Poels, G., Genero, M., & Piattini, M. (2012). A conceptual modeling quality framework. *Software Quality Journal*, *20*(1), 201–228. http://doi.org/10.1007/s11219-011-9136-9

Object Management Group. (2013). *Business Process Model and Notation (BPMN2.02)*. Retrieved from http://www.omg.org/spec/BPMN/

Object Management Group. (2015). *Unified Modeling Language (UML)*. Retrieved from http://www.omg.org/spec/UML/

Object Management Group. (2016). *Decision Model and Notation (DMN1.1)* (Vol. 1.0). Retrieved from http://www.omg.org/spec/DMN/

Peffers, K., Rothenberger, M. A., Tuunanen, T., & Vaezi, R. (2012). Design Science Research Evaluation. *Design Science Research in Information Systems. Advances in Theory and Practice*, 398–410. http://doi.org/10.1007/978-3-642-29863-9_29

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, *24*(3), 45–77.

Pesic, M. (2008). *Constraint-based workflow management systems: shifting control to users*. Technische Universiteit Eindhoven.

Petri, C. A. (1962). *Kommunikation mit automaten*. Universität Bonn.

Reichert, M., & Weber, B. (2012). *Enabling flexibility in process-aware information systems*. Springer.

Rotter, T., Kinsman, L., James, E. L., Machotta, A., Gothe, H., Willis, J., … Kugler, J. (2010). Clinical pathways: effects on professional practice, patient outcomes, length of stay and hospital costs. *Cochrane Database of Systematic Reviews*, (3), 1–170. http://doi.org/10.1002/14651858.CD006632.pub2

Rovani, M., Maggi, F. M., de Leoni, M., & van der Aalst, W. M. P. (2015). Declarative process mining in healthcare. *Expert Systems with Applications*, *42*(23), 9236–9251. http://doi.org/10.1016/j.eswa.2015.07.040

Rowley, J. (2007). The wisdom hierarchy: representations of the DIKW hierarchy. *Journal of Information Science*, *33*(2), 163–180. http://doi.org/10.1177/0165551506070706

Russell, N., ter Hofstede, A. H. M., Edmond, D., & van der Aalst, W. M. P. (2004). *Workflow resource patterns*. *BETA Working Paper Series, WP 127*.

Russell, N., van der Aalst, W. M. P., ter Hofstede, A. H. M., & Edmond, D. (2005). Workflow Resource Patterns: Identification, Representation and Tool Support. In *Advanced Information Systems Engineering. LNCS, vol. 3520* (pp. 216–232). Springer. http://doi.org/10.1007/11431855_16

Sadiq, S. W., Orlowska, M. E., & Sadiq, W. (2005). Specification and validation of process constraints for flexible workflows. *Information Systems*, *30*(5), 349–378. http://doi.org/10.1016/j.is.2004.05.002

Schonenberg, H., Mans, R. S., Russell, N., Mulyar, N., & van der Aalst, W. M. P. (2008). Process flexibility: A survey of contemporary approaches. In *Advances in Enterprise Engineering I. LNBIP, vol. 10* (pp. 16–30). Springer. Retrieved from http://link.springer.com/chapter/10.1007/978-3-540-68644-6_2

Shostack, G. L. (1987). Positioning Through Structural Change. *Journal of Marketing*, *51*(1), 34–43.

Slaats, T., Mukkamala, R. R., Hildebrandt, T., & Marquard, M. (2013). Exformatics declarative case management workflows as DCR graphs. In *BPM'13. LNCS, vol. 8094* (Vol. 8094 LNCS, pp. 339–354). Beijing, China: Springer. http://doi.org/10.1007/978-3-642-40176-3_28

Taylor, J., Fish, F. A., & Vanthienen, J. (2013). Emerging Standards in Decision Modeling—an Introduction to Decision Model & Notation. In *iBPMS: Intelligent BPM Systems: Intelligent BPM Systems: Impact and Opportunity* (pp. 133–146). Retrieved from https://lirias.kuleuven.be/bitstream/123456789/424966/1/14+Emerging+Standards+in+Decision+Modeling-Taylor+et+al+final.pdf

Van Der Aa, H., Leopold, H., Batoulis, K., Aa, H. Van Der, Leopold, H., & Batoulis, K. (2015). Integrated Process and Decision Modeling for Data-Driven Processes. In *BPM'15 Workshops, DeMiMoP'15. LNBIP, vol. 256* (pp. 405–417). Innsbruck, Austria: Springer.

van der Aalst, W. M. P. (2011). Workflow Resource Patterns. Retrieved October 6, 2015, from http://www.workflowpatterns.com/patterns/resource/

van der Aalst, W. M. P. (2013). Business Process Management: A Comprehensive Survey. *ISRN Software Engineering*, *2013*, 1–37. http://doi.org/10.1155/2013/507984

van der Aalst, W. M. P., Pesic, M., & Schonenberg, H. (2009). Declarative workflows: Balancing between flexibility and support. *Computer Science - Research and Development*, *23*(2), 99–113. http://doi.org/10.1007/s00450-009-0057-9

Vanthienen, J. (2014). *Introducing a proposal to standardize a Decision Model and Notation*. Retrieved from http://www.omg.org/news/meetings/tc/agendas/va/DMN_pdf/Vincent_SainteMarie_vanThienen.pdf

Wand, Y., Storey, V. C., & Weber, R. (1999). An ontological analysis of the relationship construct in conceptual modeling. *ACM Trans. Database Syst.*, *24*(4), 494–528. http://doi.org/10.1145/331983.331989

Wastell, D. G., White, P., & Kawalek, P. (1994). A methodology for business process redesign: experiences and issues. *The Journal of Strategic Information Systems*, *3*(1), 23–40. http://doi.org/10.1016/0963-8687(94)90004-3

Weske, M. (2012). *Business Process Management: Concepts, Languages, Architectures*. Springer. http://doi.org/10.1007/978-3-642-28616-2

Westergaard, M., & Maggi, F. M. (2012). Looking Into the Future about Timed Declarative Process Models. In *OTM 2012* (Vol. 7565, pp. 250–267). http://doi.org/10.1007/978-3-642-33606-5_16

Westergaard, M., & Slaats, T. (2013). Mixing Paradigms for More Comprehensible Models. *BPM'13 Workshops. LNCS, Vol. 8094*, 283–290. http://doi.org/10.1007/978-3-642-40176-3_24

White, S. A. (2004). Introduction to BPMN. *IBM Cooperation*, 1–11. Retrieved from http://yoann.nogues.free.fr/IMG/pdf/07-04_WP_Intro_to_BPMN_-_White-2.pdf

Wohed, P., Russell, N., ter Hofstede, A. H. M., Andersson, B., & van der Aalst, W. M. P. (2009). Patterns-based evaluation of open source BPM systems: The cases of jBPM, OpenWFE, and Enhydra Shark. *Information* *and Software Technology*, *51*(8), 1187–1216. http://doi.org/10.1016/j.infsof.2009.02.002