Artificial Intelligence In Medicine 137 (2023) 102495

Contents lists available at ScienceDirect

Artificial Intelligence In Medicine

journal homepage: www.elsevier.com/locate/artmed

Research Paper

A model-driven transformation approach for the modelling of processes in clinical practice guidelines

Begoña Martínez-Salvador [a,*,1], Mar Marcos [a,1], Patricia Palau [b,c,2], Eloy Domínguez Mafé [b,d,2]

[a] Department of Computer Engineering and Science, Univ. Jaume I, Spain
[b] Cardiology Department, Hospital Clínico Universitario de Valencia, Univ. de València, Spain
[c] INCLIVA Instituto de Investigación Sanitaria, Univ. de València, Spain
[d] Predepartmental Unit of Medicine, Univ. Jaume I, Spain

ARTICLE INFO

Keywords:
Clinical practice guidelines
Computer-interpretable guidelines
Modelling of processes in clinical practice guidelines
Model-driven development
BPMN
PROforma
ATLAS transformation language

ABSTRACT

Clinical Practice Guidelines (CPGs) include recommendations aimed at optimising patient care, informed by a review of the available clinical evidence. To achieve their potential benefits, CPG should be readily available at the point of care. This can be done by translating CPG recommendations into one of the languages for Computer-Interpretable Guidelines (CIGs). This is a difficult task for which the collaboration of clinical and technical staff is crucial. However, in general CIG languages are not accessible to non-technical staff. We propose to support the modelling of CPG processes (and hence the authoring of CIGs) based on a transformation, from a preliminary specification in a more accessible language into an implementation in a CIG language. In this paper, we approach this transformation following the Model-Driven Development (MDD) paradigm, in which models and transformations are key elements for software development. To demonstrate the approach, we implemented and tested an algorithm for the transformation from the BPMN language for business processes to the PROforma CIG language. This implementation uses transformations defined in the ATLAS Transformation Language. Additionally, we conducted a small experiment to assess the hypothesis that a language such as BPMN can facilitate the modelling of CPG processes by clinical and technical staff.

1. Introduction

Clinical Practice Guidelines (CPGs) are "statements that include recommendations intended to optimise patient care that are informed by a systematic review of evidence and an assessment of the benefits and harms of alternative care options" [1]. Research has demonstrated that CPGs have the potential of translating clinical research results to practice, and to improve the quality and outcomes of healthcare. To achieve these benefits, CPGs should be available at the point where the encounter between the patient and the clinician occurs [2]. An effective way to accomplish this is by translating the recommendations within CPGs into a computer-interpretable format [3]. Thus, Computer-Interpretable Guidelines (CIGs) can be defined as formalised versions of CPG contents intended to be executed as part of clinical decision support systems.

Over the last 20 years, several modelling languages for CIGs were proposed in the Medical Informatics and AI in Medicine areas. Among the most important ones, in chronological order, are [2,4,5]: Arden Syntax, PROforma, Asbru, EON, Prodigy, GLIF, and GUIDE. An important part of these languages refers to the procedures to perform, covering both concrete clinical actions (e.g., diagnostic tests) and more or less complex combinations of actions (e.g., choices, sequences) [6]. Additionally, these CIG languages usually provide editing tools to facilitate the authoring of models. However, encoding clinical recommendations into a CIG language is a difficult and demanding task. On one hand, a proper understanding of CPG recommendations necessitates from a clinical background. On the other hand, CIG languages are in general poorly accessible to clinicians without technical skills. For these reasons, it has been acknowledged for some time that the collaboration of clinical and technical staff is crucial for the authoring of CIGs [7–9]. Some authors even have demonstrated that the models obtained by mixed (clinical and technical) teams were superior to those produced by either clinical or technical staff alone [10].

The research problem we address is the difficulty of encoding the recommendations of CPGs into a CIG language. In this context, we advocate promoting the involvement of clinicians in the encoding of CPG recommendations. Besides, clinicians should be able to recognise

* Corresponding author.
E-mail address: begona.martinez@uji.es (B. Martínez-Salvador).
[1] PhD
[2] MD, PhD

https://doi.org/10.1016/j.artmed.2023.102495
Received 25 April 2022; Received in revised form 13 December 2022; Accepted 18 January 2023
Available online 23 January 2023

the recommendations they have authored in the resulting CIGs. Rather than directly encoding the guideline in one of the above-mentioned languages, we propose to support the authoring of CIGs based on transformations, from a preliminary description in a more accessible language, into a detailed implementation in a CIG language. For this purpose, and specifically focusing on the modelling of clinical processes in the CPG, we use the Business Process Modelling and Notation (BPMN) [11] as a bridge language between the textual CPG and the CIG language. BPMN, whose latest specification is BPMN 2.0, is an OMG proposal which is widely accepted for process modelling in many areas [12].

In the field of Medicine, BPMN has been in use for over a decade. In this sense, some works have reported a quick and intuitive familiarisation with the language, and a better understanding of the processes for non-technical staff, facilitating collaboration and communication [13,14]. In recent years, there has been an increase in the publishing activity in this regard, as evidenced by recent review papers [15,16]. The literature analysis by De Ramón et al. [15] confirms the utility of the use of BPMN in the design, optimisation and automation of clinical processes, as well as a greater involvement of clinical staff derived from this use. The review work by Tomaskova and Kopecky [16] identifies the description of clinical processes and decision-making among the main purposes of analysed publications. The scope of the latter review is widespread, including several approaches related to knowledge representation of clinical processes [6,17–19]. Zerbato et al. [17] worked on the use of BPMN for modelling clinical protocols, with special focus on modelling temporal aspects. Kaiser and Marcos [6] compared the expressiveness of CIG and business process languages (not only BPMN) in terms of the so-called workflow control patterns. Combi et al. [18] proposed a methodology to model decision-intensive care pathways, combining BPMN and DMN (Decision Modelling Notation, also an OMG proposal). Lastly, Martínez-Salvador and Marcos [19] proposed a transformation-based refinement approach that uses BPMN to support the encoding of clinical processes into CIG languages, jointly by clinical and IT staff. Following a similar idea, the work by Gonzalez-Lopez et al. [20] (in this special issue) presents a process-oriented methodology, based on BPMN, for modelling surgical processes involving healthcare domain experts.

Following a transformation-based approach, in this work we tackle the transformation to support the authoring of CIGs applying the Model-Driven Development (MDD) paradigm from Software Engineering. In MDD, models are first-class artefacts in the development of software and transformations are the main operation. In this context, we define MDD transformations to obtain the target model in a CIG language from a source model in BPMN, dealing with the metamodels (or schemes) of these languages and implementing the solution in terms of the elements of those metamodels. The MDD transformation itself is defined in the ATLAS Transformation Language (ATL) [21].

We demonstrate this novel approach by choosing the above-mentioned BPMN notation and the PROforma CIG language as source and target languages, respectively. PROforma is an executable CIG language tailored to capture medical knowledge which has been successfully used for deploying clinical decision support systems [22]. Compared to BPMN, PROforma is in general less intuitive for clinicians not only because it includes much more process detail (it is an executable language) but also because many of these details are not displayed graphically. Note that a comparison of the BPMN and PROforma languages is outside the scope of this work.

Additionally, we describe the results of a small experiment we have conducted to evaluate our hypothesis that a more accessible language facilitates the early stages of the CPG modelling task by mixed teams with clinical and technical staff. The observations gathered during our experiment show that, after a short introduction to the BPMN language, clinicians can not only be engaged but also take a leading role in the modelling task. The results of our experiment also support the greater accessibility to clinicians of BPMN notation.

2. Materials and methods

2.1. Representation of clinical processes

CPGs contain evidence-based recommendations for the best management of patients with a specific clinical condition. Regarding their appearance and structure, they are usually text documents of varying length, describing in detail the recommended actions for the diagnosis and/or treatment of a particular disease. Sometimes CPGs are augmented with more structured information like tables summarising key recommendations and flowcharts structuring and specifying some recommendation steps for ease of understanding.

Clinical processes in guidelines typically include requests for data, and cover both concrete clinical actions (e.g., diagnostic tests) and more or less complex combinations of actions such as choices and sequences. Most times an order is established in the mentioned steps, for example, a sequence. However, in other situations, some actions could be done in parallel or simply the order is not relevant. In most cases, clinical processes within CPGs are rather complex. Therefore, the use of elements for encapsulating some parts in the representation may increase readability. For example, a set of interventions to perform a complex task could be encapsulated for the sake of clarity. Encapsulation can be also useful when there are different treatments according to certain patient features, for example for postmenopausal women. As a result, CPGs could be considered as a hierarchy of tasks or clinical processes.

It is also worth mentioning that guidelines could contain some recommendations that affect only a subgroup of patients. For example, an additional test may be recommended for a group of patients while not for the rest. Moreover, usually guidelines contain iterative steps, specially related to some therapeutic actions, like in the case of chemotherapy. Finally, notice that CPGs are formulated in natural language, therefore the processes they contain are inherently structured [6], which means that for every process branching it is possible to define a corresponding joining, and thus that all branch-join points can be properly paired. All these features must be considered when modelling guidelines in both BPMN and PROforma.

2.1.1. Clinical guidelines in BPMN

BPMN is a complex graphical language with about 50 constructs. However, some studies conclude that a common core set and an extended core set of elements, representing about 24% of the total elements, constitute the most frequently used elements to design processes [23]. In this line, our experience modelling in BPMN the clinical processes of different CPGs confirms that the BPMN constructs included in the core and extended sets more than cover the representation needs of CPGs. Note that in this section, we restrict our discussion to the minimal set of BPMN elements required to represent clinical processes according to our experience. These include: events, tasks, gateways, sequence flows, and sub-processes. See Fig. 1 for an example of clinical process modelled in BPMN.

Events, tasks and gateways are types of BPMN flow objects. *Events* are something that happens during the course of a process. Graphically, events are represented as circles. *Tasks* are atomic activities and represent the actions to be performed. They are depicted as rounded rectangles. *Gateways* control branching and merging of flows in a process. There are different types of gateways that correspond to different control flow structures: alternative paths, that can be deterministic (XOR-gateway) or not deterministic (OR-gateway), and parallel paths (AND-gateway). Gateways are represented as diamonds with a mark inside that indicates the type of gateway. Moreover, if gateways branch the control flow, they are known as diverging gateways. Conversely, gateways that merge flows are known as converging gateways.

The elements in a BPMN process are connected using *sequence flows*, graphically depicted as arrows (see Fig. 1). Sequence flows define the order of execution of the BPMN flow objects. Each sequence flow has exactly one source and one target. Sequence flows starting in a

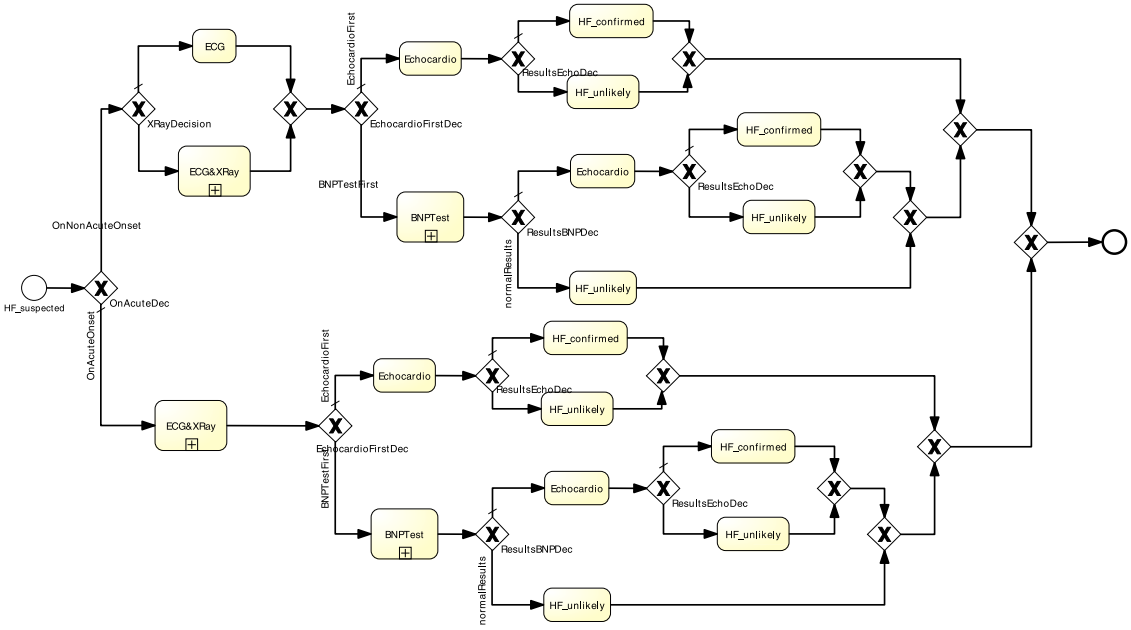

Fig. 1. BPMN diagram for the diagnosis of heart failure in the cases of acute and non-acute onset. Circles are the start and end events. Rounded squares represent tasks or sub-processes (marked with a plus sign). Diamonds are diverging and converging gateways. In this example all the gateways are XOR-gateways. Sequence flows (arrows) connect all the elements.

diverging (X)OR-gateway[3] are conditional sequence flows since they have conditions that must be met in order to follow the corresponding flow. These diverging gateways might also be the origin of a default sequence flow that would be chosen when none of the conditions of the alternative sequence flows are satisfied.

Sub-processes are compound activities. A sub-process is depicted as a single node but contains its own process diagram. Graphically, it is represented as a rounded rectangle marked with a plus sign (+), thus indicating that its own diagram can be displayed. They are used to hide the complexity of a process by encapsulating a set of tasks that are needed to reach a goal, or to define a specific kind of execution for the activities within the sub-process, such as any sequential ordering. As explained in the previous section, clinical processes are rather complex and therefore sub-process encapsulation is used to increase readability. In BPMN, it is also possible to indicate that a task or sub-process can be repeatedly executed. This feature can be used for iterative procedures included in CPGs.

A clinical process is represented in BPMN as a directed acyclic graph [19]. Each graph has a start event and an end event as starting and finishing points, respectively. In the graph, events, tasks, gateways and sub-processes are connected by means of sequence flows. Diverging gateways split the flow into two or more paths. Conversely, converging gateways merge flows. As an illustration, Fig. 1 shows the BPMN graph for the diagnosis of chronic heart failure in the cases of acute and non-acute onset. The graph includes several sub-processes which contain their own graph, resulting in a hierarchy of acyclic graphs. It can be observed that it is a structured graph, i.e. every diverging gateway has its corresponding converging gateway (of the same type), and all pairs of diverging–converging gateways are properly nested. As mentioned before, we deal with structured graphs because clinical processes in CPGs are inherently structured. Structuredness is a desirable property of business process graphs to avoid structural conflicts such as deadlocks [24].

[3] From now on, we will use the notation (X)OR-gateway to refer to both XOR-gateways and OR-gateways.

2.1.2. Clinical guidelines in PROforma

As mentioned before, PROforma is a CIG language which has been successfully used for implementing numerous clinical decision support systems [22]. In PROforma, a guideline is modelled as a plan made up of one or more tasks [25]. *Tasks* are the building blocks of PROforma. There are four types of tasks: enquiries, decisions, actions and plans. *Enquiries* are tasks that request data, i.e. the value of one or more data items or sources, from the external environment (for example a user or a database). *Actions* represent those activities to be performed by an external agent (for example, clinical procedures). *Decisions* are tasks that represent a choice among different options or candidates. Finally, *plans* group together a set of other tasks. Since plans may contain other plans in turn, PROforma allows the definition of hierarchical task networks. The tasks in a plan are usually ordered using *scheduling constraints* and/or different types of task conditions. If none of them are given, then a parallel execution is assumed.

Decisions are a key element of the PROforma language and require additional attributes. As explained above, they involve a choice among different options, named *candidates*. Each candidate has one or more arguments which are truth-valued expressions that determine the choice of that candidate. These expressions usually describe the arguments for (in favour) or against the candidate. Additionally, each candidate has a recommendation rule, which is an expression that calculates the support for the candidate considering all its arguments. Finally the choice mode of the decision, single or multiple, determines how many candidates can be recommended by the decision.

In the PROforma graphical notation each plan is depicted as a directed acyclic graph (see example in Fig. 2) in which nodes represent tasks and arcs represent scheduling constraints. The different shapes of the nodes correspond to the different types of tasks. Concretely, squares correspond to actions, circles correspond to decisions, diamonds represent enquires, and round-edged rectangles correspond to plans. Scheduling constrains are directed arcs indicating that the task at the head of the arc cannot be considered for execution until the task at the tail (antecedent) has finished.

Regarding the execution of a PROforma plan, a task will be considered for activation when all its scheduling constraints have been met, that is, when all its antecedent tasks have been either completed or discarded [25]. In that case, the task will be activated if at least

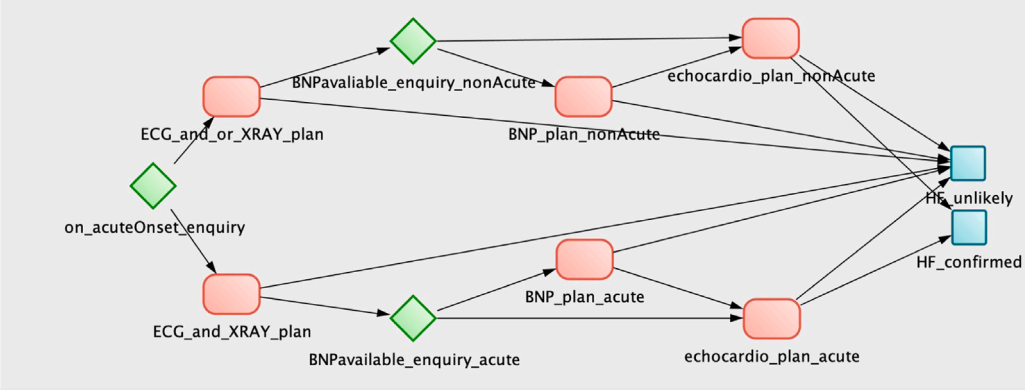

Fig. 2. PROforma top-level plan for the diagnosis of heart failure on acute and non-acute onset, modelled from scratch from the textual CPG.

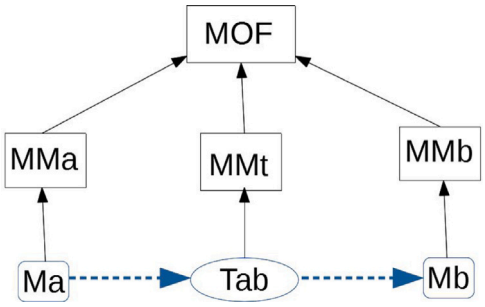

Fig. 3. Schema of a model-driven transformation in the context of OMG. *Source:* Adapted from Figure 2 in [21].

one of its antecedent tasks has been completed. Otherwise, it will be discarded. Additionally, tasks may also have different types of conditions that have to be met before activation. These include preconditions and wait conditions. Both are truth-valued expressions that are checked once the scheduling constraints of the tasks are met. Then, in the case of a precondition, the task will be activated if the precondition holds and discarded otherwise. In the case of a wait condition, the task will stay dormant until the condition holds. For more details on PROforma, refer to OpenClinical.net resources [26].

Note that many of the above elements (e.g. candidates, arguments, preconditions, wait conditions) are described as code and are not displayed in the PROforma graphical notation. As a consequence, although at first glance the process in Fig. 2 may seem simpler than the one in Fig. 1, is not so for the clinician because she/he must be able to understand the associated PROforma code for a correct interpretation of the process. For the sake of simplicity, only the graphical description is shown in the figure.

2.2. Model-driven development to support modelling of CPG processes

Starting from the hypothesis that a more accessible language such as BPMN can facilitate the CPG modelling task, in this work we support the authoring of PROforma CIGs using transformations between these languages. MDD is a known approach used to improve software development processes by means of models and transformations, which are used to map information from a source model to obtain the target model [27]. MDD was developed to solve some of the most important problems in software development, namely: to increase productivity, to document the transformation, to increase portability, and to improve interoperability. It has been argued that MDD has several advantages over traditional software development, such as increasing productivity and reliability [28,29].

A recent study [30] comparing MDD with respect to traditional software development concludes that MDD is the most useful method in the long run, with a greater learning curve but compensated with a lower development effort. MDD relies on transformations between source and target models. Transformations following the MDD paradigm are described in a more abstract and readable way, compared to transformations written in a general-purpose programming language. Thus, the MDD approach is also useful to trace and document the transformation [31].

In the literature, model transformations can be classified according to different criteria regarding their source and target artefacts or models [32]. According to these criteria, we classify our transformation as follows. Regarding the number of input and output models, it is a one-to-one transformation, since there is a single input model and a single output model. Since the input and output modelling languages are different, it is an exogenous transformation. Because the input language, BPMN, is not as specific (and detailed) to capture the rich content of clinical processes as the PROforma language, we can say that the models lie in different abstraction levels. Thus, we consider the transformation as a vertical transformation or refinement. Finally, according to the criterion technical space, our transformation belongs to the same technical space since all the metamodels conform to the Meta-Object Facility by OMG (see below). To sum up, we deal with a one-to-one exogenous vertical transformation (i.e. a translation) [32].

Hereafter, the main elements of our MDD approach, namely the metamodels of the BPMN and PROforma languages and the ATL language used for the implementation of transformations, are described.

2.2.1. Metamodels

Metamodels are a prerequisite for MDD. A metamodel is a precise description of the structure, elements and well-formed rules of the language of the model. A model-driven transformation[4] consists in the automatic generation of a target model from a source model by means of a transformation definition, which is a set of rules that describe how a model (source) is transformed into another model (target) [32] in terms of the metamodel elements. Model transformation can support a wide range of tasks, including refinement, synthesis, abstraction, translation, refactoring, or merging of models. A model transformation follows a common pattern known as model transformation pattern [21]. This pattern has three elements (see Fig. 3): *Ma*, *Mb* and *Tab*. *Ma* is the input model that conforms to the source metamodel *MMa*. *Mb* is the output model conforming to the target metamodel *MMb*. In other words, Ma is an instance of the metamodel MMa, and similarly with Mb and MMb. *Tab* is a transformation definition in a transformation language that automatically generates *Mb* from *Ma*. Ultimately, *Tab*

[4] Henceforth, the term model transformation is used instead of model-driven transformation.

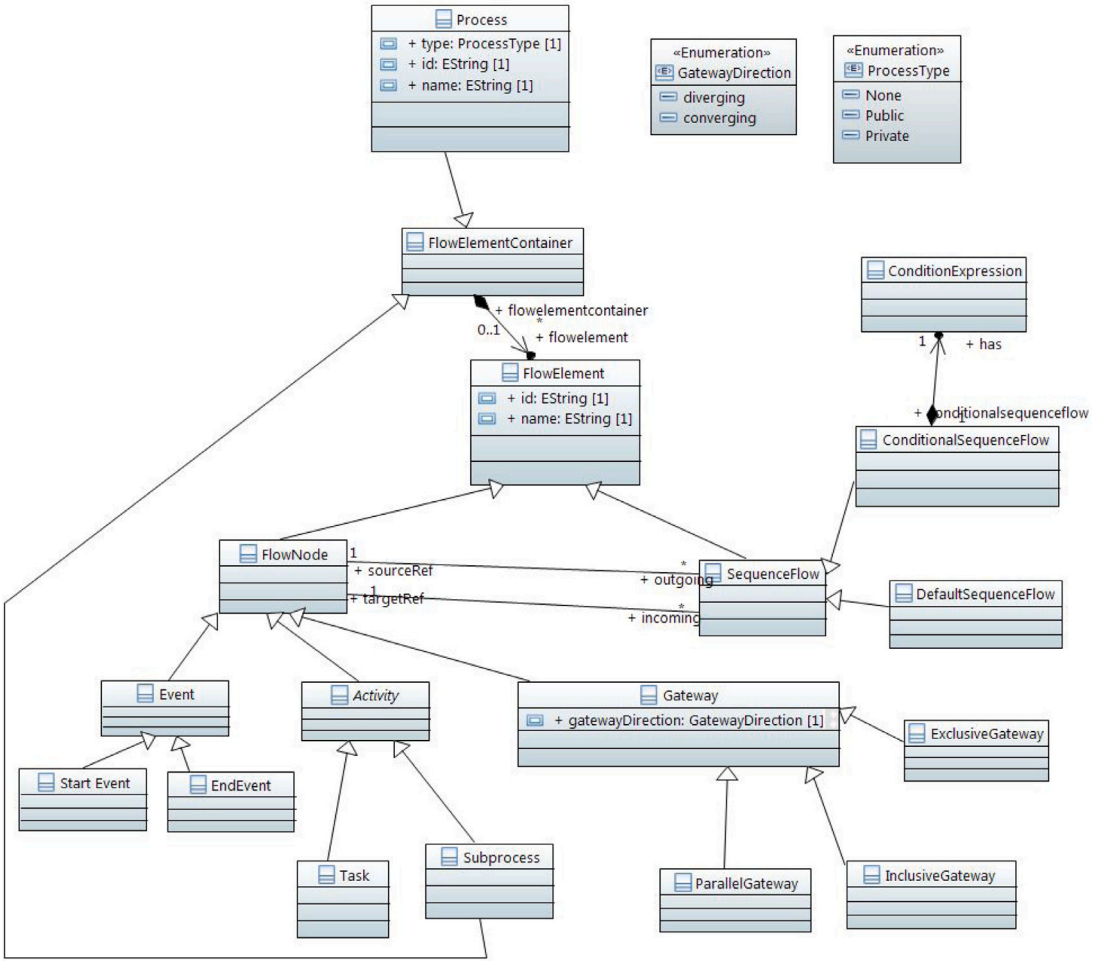

Fig. 4. Simplified UML class diagram of the BPMN metamodel.

conforms to *MMt*, which corresponds to the abstract syntax of the transformation language. Then, the transformation definition *Tab* describes how *Ma* can be transformed into *Mb*. Note that all the metamodels conform to a meta-metamodel, which in our case corresponds to the Meta-Object Facility (MOF[5]) promoted by OMG.

BPMN 2.0 metamodel. The source metamodel is the official BPMN metamodel provided by OMG, which is MOF-compliant and is based on the BPMN 2.0 specification [11]. This metamodel has a considerable size, consistent with the great number of language elements. We describe next a selected subset of these elements, shown in a simplified manner in Fig. 4. The main element of a BPMN model is a `Process`, which is a `Flow Element Container` composed of 0 or more `Flow Elements`. `Sequence Flow` and `Flow Node` are both subtypes of the entity `Flow Element`. Likewise, the entities `Event`, `Activity` and `Gateway` are subtypes of `Flow Node`, and `Task` and `Sub Process` are subtypes of `Activity`. A `Sub Process` is also a type of `Flow Element Container`, which means that it may contain some `Flow Elements`. With respect to events, `Start Event` and `End Event` are subtypes of `Event`. Finally, the entity `Gateway` has the attribute `gateway Direction`, amongst others, and the subtypes `Exclusive Gateway`, `Inclusive Gateway` and `Parallel Gateway`.

`Flow Nodes` are referenced as a source and/or as a target in `Sequence Flows`. Focusing on `Events`, a `Start Event` can only be a source for `Sequence Flows`, while an `End Event` can only be a target for them. Both `Activities` and `Gateways` may be a source and/or a target for `Sequence Flows`. Note that if a node is a source for one or more sequence flows, this implies that it has one or more outgoing sequence flows; similarly, being the target for one or more sequence flows means that there are one or more incoming sequence flows (see the `outgoing` and `incoming` labels in Fig. 4). Lastly, the entity `Sequence Flow` has as subtypes `Conditional Sequence Flow` and `Default Sequence Flow`. These subtypes are only allowed for sequence flows going out from diverging gateways. On the other hand, a `Conditional Sequence Flow` must have a `Conditional Expression`.

The BPMN 2.0 editor we have used in our work is the BPMN Modeller plugin available for the Eclipse IDE, which is based on the BPMN metamodel by OMG, and therefore is MOF-compliant.

Proforma metamodel. The developers of the target language, PROforma, do not provide a metamodel defining its syntax. For this reason, we have defined a UML class diagram for the elements of the language (see Section 2.1.2) and, based on it, we have generated a metamodel suitable for our purposes (see below). Note that this metamodel includes all the details of the PROforma language and therefore can be used to derive the PROforma format, which is plain text.

The metamodel of PROforma (Fig. 5) contains 14 entities and 24 relations among those entities. The entity `Process` must contain a single `Plan`, through the relation `topLevelPlan`, and may have some `Data` and also some `Triggers`. Every `Plan` has a `SubProcess`, which contains one or more `Tasks` and may contain several `SchedulingConstraints`. `Task` is a generalisation of the entities

[5] MOF is an OMG standard for MDD.

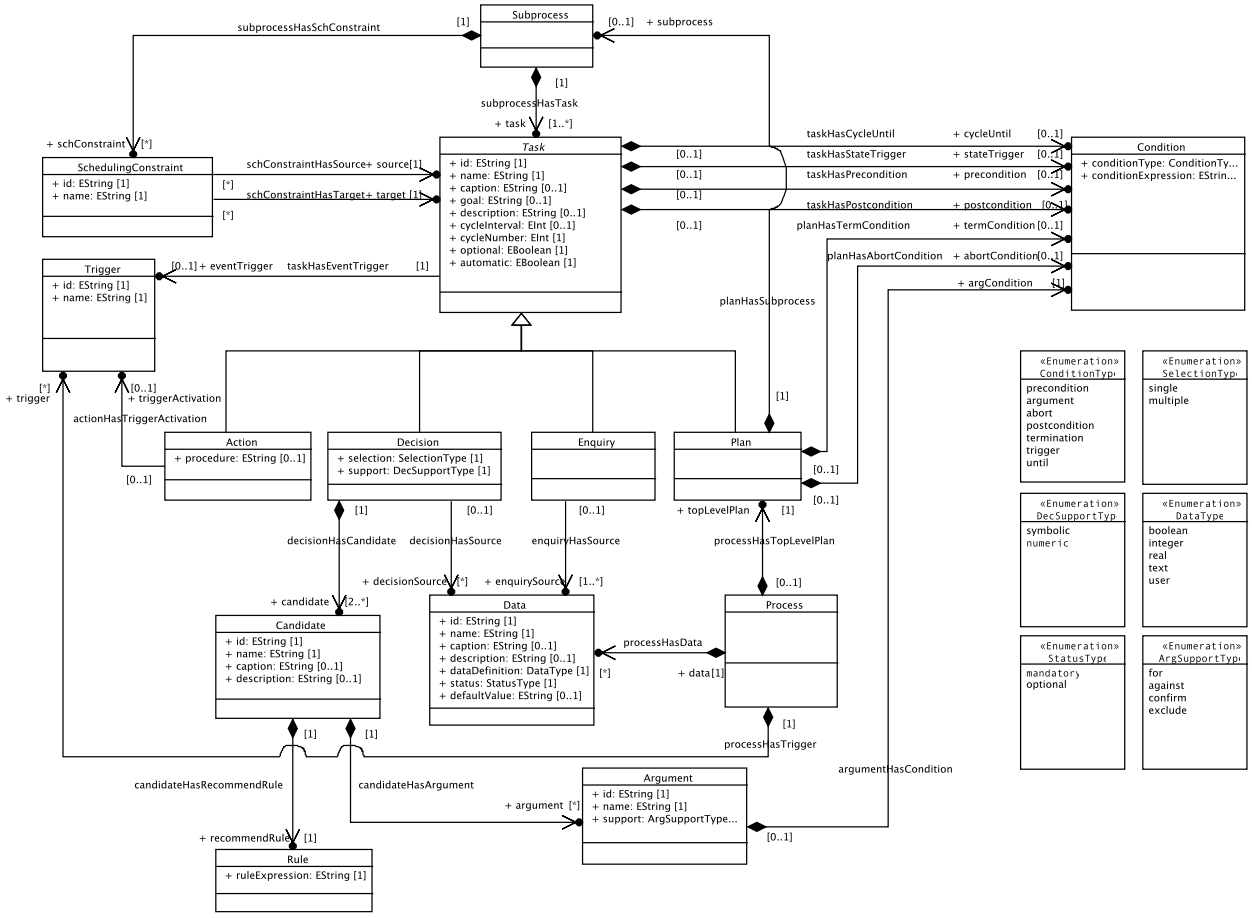

Fig. 5. UML class diagram of PROforma metamodel.

`Action`, `Enquiry`, `Decision` and `Plan`, corresponding to the four types of tasks in PROforma. The entity `SchedulingConstraint` is associated with the entity `Task` via two relations, since each scheduling constraint must have one source task and one target task. The entity `Enquiry` is associated with `Data` that represents the data to be entered by the user. The entity `Decision` can also have some `Data`, and additionally it has two or more `Candidates`. In turn, a `Candidate` must have a `Rule` and may have some `Arguments`. Finally, each `Argument` is related to one `Condition`. This entity `Condition` has several other associations with the entities `Task` and `Plan`, in order to model the different types of conditions.

Based on this UML class diagram, we have obtained the PROforma metamodel (`PROforma.ecore` file) using the Eclipse Modelling Framework (EMF) Generator Model tool.[6] Additionally, with other EMF tools we have derived an editor for PROforma models compliant with our metamodel. This editor has been used to validate the metamodel, by modelling a significant CPG sample (including the CPGs from the OpenClinical.net repository[7] used in a previous work [33]). The guideline used in this work has also been modelled conforming to the PROforma metamodel with the same tool.

2.2.2. *The ATLAS transformation language*

In MDD, the characteristics of the transformation language or tool are equally important to the ones of the model being transformed [32]. Among the desirable characteristics are: (i) the transformation language should allow creating/editing/deleting transformations; (ii) it should allow grouping and composing transformations; (iii) the defined transformations should be generic; (iii) the language should support traceability; (iv) it should be compliant to standards; and (v) it should be accepted by the user community. In this work, we use the ATLAS Transformation Language (ATL) [21], that meets most of these properties and also conforms to MOF.

ATL is a hybrid model transformation language, i.e. it contains declarative and imperative constructs, developed as part of the ATLAS Model Management Architecture (AMMA). ATL is supported by a set of development tools built on the Eclipse environment: an editor, a compiler, a debugger, and an engine [34]. ATL is applied in the context of the above-mentioned model transformation pattern (see Fig. 3): a source model *Ma* is transformed into a target model *Mb* according to a transformation definition *Tab*, that corresponds to a programme *mma2mmb.atl* written in the ATL language.

As mentioned above, ATL provides both declarative and imperative constructs. The use of the declarative constructs is encouraged since the transformation has more advantages in this way [34]: (i) the transformation tends to be closer to the way developers intuitively perceive it; (ii) it hides the details, thus complex transformations can be written with a simple syntax; and (iii) it allows traceability between the source and target models. In the rest of the section we focus on the declarative aspects of ATL, which are the ones we have used in this work.

A transformation programme is a set of rules that describe how to obtain the output model from the input model. The transformation rule is the basic construct of the ATL language. A rule is a description of how one or more constructs in the input model should be transformed into one or more constructs in the output model. Transformation rules in ATL are organised in *modules*. A module has a mandatory header

[6] https://www.eclipse.org/modeling/emf/ (last access: April 21st, 2022).
[7] https://www.openclinical.net/library/ (last access April 21st, 2022).

section and a set of helpers and transformation rules. *Helpers* are methods applied to an element of the source metamodel and can be called from different points of the ATL transformation programme. *Matched rules* are the declarative ATL rules and are composed of a source pattern and a target pattern. They are executed over matches of the source pattern in the source code. For a given match, the elements of the target pattern are created and their attributes are initialised according to the bindings, which are the elements of the source metamodel that appear in the target pattern. If the value of the binding expression is a source element, then it is first resolved into an element of the target model.

There are different types of matched rules differing on the way they are triggered. *Standard matched rules* are applied once for every match found in the source model. *Lazy rules* are triggered by other rules and they can be applied on a single match as many times as they are referred to by the rules, generating a new set of target elements each time. Lastly, *unique lazy rules* are lazy rules but they are applied only once for a given match. A more comprehensive description of the ATL language can be found in [21].

3. Implementation of the approach

As explained before, model transformations work at metamodel level. In the context of the transformation from BPMN to the PROforma CIG language, it is not possible to establish a 1:1 correspondence between all the elements of both metamodels. For example, the transformation of a choice among alternative options has to deal with different elements of the metamodels. In general, transformations are not trivial and require to define complex queries on the source model and produce complex results, since the solution does not depend on a single metamodel element but on a set of elements and involve related features. For this reason, we do not approach the transformation of the entire model as a whole, but we consider smaller parts whose elements are related. In order to accomplish this, we leverage the graph-oriented quality of the source model. Taking advantage of the fact that BPMN clinical guidelines are structured graphs (see Section 2.1.1), we divide the model in smaller parts by implementing a structure-identification strategy [35] (see Section 3.2), and then, we define model transformations for those smaller parts or structures.

The structure-identification strategy consists in identifying in the source language a series of structures of interest (or components) considering the target language. In our target model (PROforma) there are sets of tasks executed in sequential order, in parallel or in non-deterministic order, and we also have decisions. Accordingly, we need to identify in the BPMN model sequences, parallel branches and alternative branches. For example, since our source model (BPMN) is a structured graph, a parallel structure will be defined by a diverging AND-gateway along with the sequence flows starting from that gateway, and will be delimited by the corresponding converging AND-gateway. The rest of structures will be identified in a similar way.

In the rest of the section we describe the main transformation algorithm, i.e. the structure-identification strategy, the mapping of identified BPMN components, and the ATL rules.

3.1. Transformation algorithm

The input to the algorithm (source model) is a BPMN representation of the clinical processes of the CPG, that is stored as a graph data structure. More accurately, as a hierarchy of graphs since the model uses sub-processes which are themselves graphs. In such representation, we can find different types of nodes connected by sequence flows that define sequences, split branches and join branches.

The transformation method we have implemented has two parts that we have called the *reducing* phase and the *expanding* phase, as shown in Algorithm 1. In each iteration of the reducing phase, the innermost structure of interest (or one of them), is identified in the input graph (line 5 of Algorithm 1). Then, a new node is created that stores the identified structure as a stand-alone BPMN process, which requires that a start and an end event are added to the sub-graph so that the resulting structure conforms to the BPMN metamodel. At this point, the corresponding ATL transformation for the identified structure of interest is called and the PROforma fragment resulting from the transformation is also stored in the new node (lines 7 and 8 of Algorithm 1). Finally, the whole sub-graph (of the identified structure) is replaced by the new node and the number of nodes of the input graph is updated (lines 9 and 10 of Algorithm 1). This reducing phase finishes when the whole graph has been reduced to a single node.

```
1  Algorithm: Bpmn2PROformaATLTransf
   input : A BPMN model of the CPG/CPG fragment conforming
           to the BPMN metamodel
   output: A PROforma model of the CPG/CPG fragment
           conforming to the PROforma metamodel
2  Store the BPMN model in a graph (a hierarchy of graphs
     if it contains sub-processes);
   // Reducing phase
3  nNodes ← number of nodes of the current graph;
4  while nNodes > 1 do
5  |   Identify the sub-graph that constitutes the
   |    innermost structure of interest;
6  |   Store a BPMN process with this sub-graph in node;
7  |   Call the proper ATL module;
8  |   Store the resulting PROforma code in node;
9  |   Replace sub-graph by node;
10 |   Update nNodes;
   // Expanding phase
11 topNode ← single node of the graph after the reducing
     phase;
12 JoinPROforma (topNode);
13 return PROforma model;
```

Algorithm 1: Bpmn2PROformaATLTransf algorithm to obtain a PROforma model from a BPMN one using ATL transformations.

The *expanding* phase starts with this last node (lines 11 and 12 of Algorithm 1). The algorithm recursively extracts the computed PROforma fragment and replaces the node by its content (see Algorithm 2). Then, it properly embeds the PROforma code of each node.

```
1  Algorithm: JoinPROforma
   input : currentNode, with a BPMN model of an identified
           structure and a partial PROforma model
           (without the details of innermost identified
           structures)
   output: A complete PROforma model for the identified
           structure
   // Recursive algorithm
2  PROformaModel ← PROforma code stored in currentNode;
3  foreach node in the BPMN model of currentNode do
     JoinPROforma (node);
4  return PROforma model
```

Algorithm 2: JoinPROforma algorithm to compose the PROforma model of an identified structure of interest.

The final PROforma code is embedded from top to bottom. Each time a node is replaced by its PROforma content, it is necessary to connect the first element/s with its/their predecessors through PROforma scheduling constraints, and respectively the last element/s with its/their successors. Notice that it is impossible to know in advance all the details (predecessors and successors) until the JoinPROforma algorithm finishes. To illustrate this, consider the fragment in Fig. 6. In this example, Fig. 6(a) shows a possible first expansion of the last node after finishing the reduction phase. In Fig. 6(b), the expansion of node1 is schematically represented. Shaded nodes represent final PROforma elements and dotted arrows represent temporary PROforma scheduling constraints. Fig. 6(c) shows possible expansions of

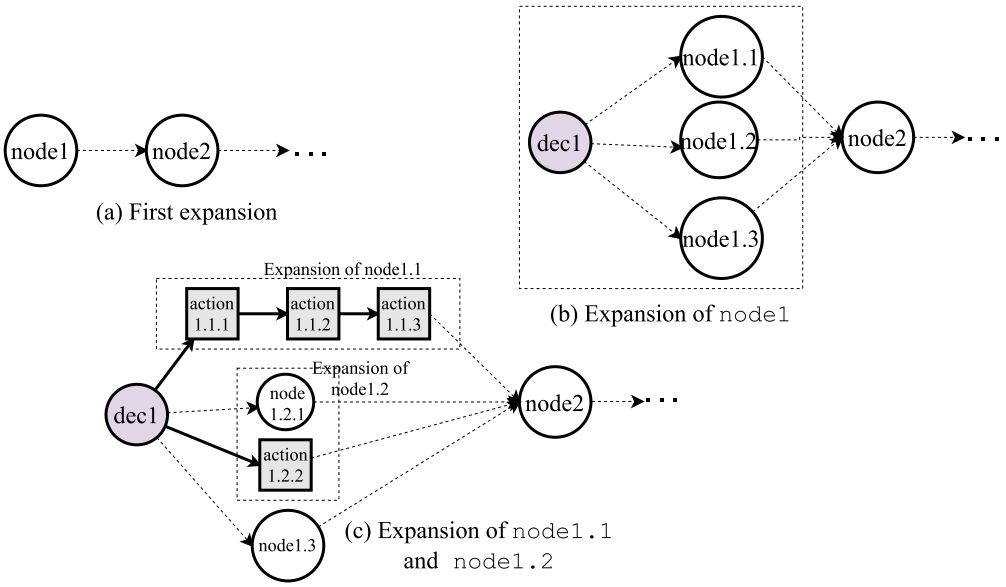

Fig. 6. Schema to illustrate the expanding phase.

node1.1 and node1.2. Notice that the scheduling constraints may change as the expanding phase (Algorithm 2) progresses. As we describe below (see Section 3.3), we map parallel components to plans, since they facilitate the introduction of these scheduling constraints (see dotted areas in Fig. 6).

The implementation of the algorithm has been carried out in the Eclipse IDE. It combines imperative methods in Java and model transformations in ATL. Both the identification of the structures of interest, in the reducing phase, and the expanding phase have been implemented in Java. For this part (identification of structures of interest and graph reduction), we have used the open-source Java JDOM API[8] to manipulate the XML information of the source and target models.

The ATL rules are organised into modules that are programmatically called from Java methods in the reducing phase. To do that, we have used the ATL's EMF Transformation Virtual Machine,[9] (EMFTVM), and we have cloned and adapted the ATLauncher[10] project to our purposes. Each module is self-contained and includes all the rules and helpers needed to complete the transformation of a structure of interest (see Section 3.4).

3.2. Identification of structures of interest

The structure-identification strategy we use was proposed to implement generic strategies for transforming from a graph-oriented language (BPMN) to a block-oriented one, such as BPEL [35,36]. As sketched before, the strategy consists in identifying in the source language a series of structures of interest considering the target language.

Henceforth, we refer to such structures of interest as *components* [19]. A component is a connected sub-graph with at least two nodes, a single entry node and a single exit node, and without any start and end events. We distinguish between parallel components and sequential components or sequences. In a sequence, every node has a single predecessor and a single successor, except for the entry and exit points. A parallel component has as entry point a diverging gateway node and as exit node a converging gateway of the same type. According to the type of the gateway, we classify parallel components into XOR-parallel components, OR-parallel components or AND-parallel components. Note that these components should not be confused with the homonymous BPMN constructs. XOR-parallel components represent a single choice among the branches starting from it, depending on which condition of the sequence flow is met. In the case of OR-parallel components, several branches may be executed if more than one condition is met. Finally, AND-parallel components represent parallel execution of the subsequent branches. To identify the proper components in the input model, we have adapted the token analysis algorithm [37] to the singularities of clinical guidelines [19].

In most cases, a guideline in BPMN consists of a combination of sequence and parallel components. These components can be combined in different ways, for instance any fragment of a sequence can be a parallel component, and in a parallel component it is possible to have sequences and other parallel components in turn. Our structure-identification implementation covers also such cases. Moreover, the implementation deals with BPMN sub-processes by means of recursive calls that address the graphs they contain.

3.3. Mapping of BPMN components to PROforma

We address the model transformation of the whole BPMN guideline by identifying in the input simpler (and easier to transform) components. Thus, each time a component is identified, the adequate ATL module is called. In order to define the ATL rules to be included in each module (see Section 3.4), the necessary mappings must be established between the elements of the BPMN metamodel and of the PROforma metamodel. Table 1 shows these mappings.

In Table 1 we distinguish the mappings related to the BPMN components, at the top, from the mappings of single BPMN elements, at the bottom. Regarding the mappings of BPMN components, any sequence in BPMN will be translated into a sequence of tasks in PROforma, where tasks can be actions or plans. Next, AND-parallel components will be modelled in PROforma as a plan where the successor elements of the diverging AND-gateway are part of the content of the plan. Finally, any (X)OR-parallel component will be mapped to a plan with an inner decision (in fact, the diverging gateway will be transformed into the decision). The mapping to PROforma plans facilitates the construction of the final PROforma model, as explained in Section 3.1.

With respect to the mappings for single BPMN elements, note that some elements either do not have an explicit representation in PROforma and thus are ignored in the transformation, or are considered

[8] http://www.jdom.org/docs/apidocs/org/jdom2/input/SAXBuilder.html (accessed April 21st, 2022)

[9] https://wiki.eclipse.org/ATL/EMFTVM (last access April 21st, 2022).

[10] https://github.com/guana/ATLauncher by Victor Guana (last access April 21st, 2022).

Table 1
Mappings between BPMN and PROforma.

BPMN	PROforma
Sequential component	Sequence of elements
AND-parallel component	Plan
XOR-parallel component	Plan with decision
OR-parallel component	Plan with decision
Start event	–
End event	–
Sequence Flow	Scheduling constraint
Task	Action
Sub-process	Plan
Diverging (X)OR-gateway	Decision
Condition in Conditional Sequence Flow	Argument in decision
Target reference of a sequence flow outgoing from a diverging (X)OR-Gateway	Candidate in decision Rule for a candidate in decision
Diverging AND-gateway	–
Any converging gateway	–

in combination with other PROforma elements. For example, start and end events are implicitly represented in PROforma by means of a plan containing the sub-graph in between them. As an illustration of the second case, a conditional expression in a conditional sequence flow coming out from a diverging XOR-gateway is used to define an argument in a PROforma decision.

3.4. ATL rules for BPMN to PROforma transformation

We have defined ATL rules according to the correspondences shown in Table 1. Each component has a dedicated ATL module, that contains several matched rules and helpers to perform the model transformation from BPMN to PROforma. As an illustration, in this section we explain the ATL module for transforming an (X)OR-parallel component into a PROforma plan with an inner decision, since it is one of the most complex ones and uses several ATL features. Note that both XOR and OR-parallel components are translated to a plan with an inner decision derived from the same BPMN elements.

Standard matched rules. As previously said, an ATL matched rule is composed of a source pattern and a target pattern. The source pattern starts with the keyword `from` and the target pattern starts with `to`. For a given match of the source pattern in the source model, the target elements and their attributes are created and they are initialised using the binding expressions. For example, consider Listing 1: every BPMN process is transformed to a PROforma top-level plan `p`, with the set of attributes listed in parentheses, such as `id`, `name`, `cycleNumber`, etc. Those attributes are initialised with the binding expression on the right. If the binding expression contains a source element, it should first be resolved into a target element. For example, this is the case of the binding expressions in lines 17 and 19.

Listing 1: ATL rule for transforming a BPMN (X)OR-parallel component into a PROforma plan with a decision. The rule implicitly triggers other standard ATL rules for each BPMN sequence flow going out from the diverging exclusive gateway, for each task and for the exclusive gateway itself.

```
1  rule Process2Process {
2  from
3    bpmnPro: bpmn20!Process
4  to
5    t: proforma!Process(
6       topLevelPlan <- p
7    ),
8    p: proforma!Plan(
9      id <- 'Plan_' + bpmnPro.id,
10     name <- 'Plan ' + bpmnPro.name,
11     cycleNumber <- 1,
12     optional <- 'false',
13     automatic <- 'false',
14     subprocess <- s
15   ),
16   s: proforma!Subprocess(
17   schConstraint <- (bpmnPro.flowElements-> select(e |
18     e.sequenceFlowFromDivExcGateway())),
19   task <- (bpmnPro.flowElements-> select(e | e.isTask() or
20     e.divergingExcGateway()))
21   )
22 }
```

In line 17, a scheduling constraint is created and initialised from each BPMN sequence flow outgoing from a diverging exclusive gateway. Since this is a source element, it first has to be resolved to a target element. This causes the implicit triggering of rule `SqFlow2SchConstraint`. In order to select the adequate source sequence flows, the helper `SqFlowFromDivExcGateway` is used. Notice that a scheduling constraint is created for every match in the source model. Similarly, in line 19 a PROforma task is created for every BPMN task or diverging exclusive gateway. Again, the proper ATL rule is implicitly executed to transform those elements into target elements. Two rules are possible here, depending on the case: the rule that transforms a BPMN task into a PROforma action (`Task2Action`), and the one that transforms a diverging exclusive gateway into a PROforma decision (`ExcGateway2Decision`). The latter is shown in Listing 2.

Lazy rules. Lazy rules are triggered by other rules and can be executed over a single match as many times as they are referred to by the other rule. For example, rule `ExcGateway2Decision` triggers the lazy rule `SqFlow2Candidate` in the binding expression of the attribute `candidate` (see line 12 in Listing 2), in order to create the candidates and their decision rules corresponding to the PROforma decision. Listing 3 shows the code of this lazy rule. The `to` section of the rule creates a candidate along with the PROforma rule for that candidate. Notice that the binding of the attribute `argument` (line 9 in Listing 3) is either a BPMN conditional expression (line 10) or the result of triggering the lazy rule `SqFlow2DefaultArgument` (line 12), depending on whether the sequence flow contains a condition or not. In the first case, the source element is resolved by implicitly triggering a standard rule.

Listing 2: ATL rule for transforming a BPMN diverging exclusive gateway into a PROforma decision. The candidates of the decision and their attributes are obtained through the explicit triggering of the lazy rule `SqFlow2Candidate`

```
1  rule ExcGateway2Decision {
2   from
3     bpmnExcGtw: bpmn20!ExclusiveGateway(
4       bpmnExcGtw.divergingExcGateway() )
5   to
6     t: proforma!Decision(
7         id <- bpmnExcGtw.id,
8         name <- bpmnExcGtw.name,
9         cycleNumber <- 1,
10        optional <- 'false',
```

```
        automatic <- 'false',
        candidate <-  (bpmnExcGtw.outgoing -> collect (e |
             thisModule.SqFlow2Candidate(e)))
    )
}
```

Listing 3: ATL rule for building a candidate of the PROforma decision, its argument and its recommendation rule. If the candidate is the `targetRef` of a conditional sequence flow, the stardard rule `FormalExp2Argument` is implicitly triggered. However, if the argument corresponds to a default sequence flow, the lazy rule `SqFlow2DefaultArgument` is called.

```
lazy rule SqFlow2Candidate {
  from
    bpmnSF: bpmn20!SequenceFlow( bpmnSF.
         sequenceFlowFromDivExcGateway() )
  to
    t: proforma!Candidate(
        id <- 'C_' + bpmnSF.targetRef.id,
        name <- 'C ' + bpmnSF.targetRef.name,
        recommendationRule <- r,
        argument <- (if not bpmnSF.conditionExpression.
             oclIsUndefined() then
                    Sequence{bpmnSF.conditionExpression}
                    else
                    Sequence {thisModule.SqFlow2DefaultArgument(
                         bpmnSF)}
                  endif)
    ),
    r: proforma!Rule(
        ruleExpression <- 'net_support(' + bpmnSF.sourceRef.id + ',
             ' + 'C_' + bpmnSF.targetRef.id + ') >= 1'
    )
}
```

Helpers. Finally, the module for an (X)OR-parallel component contains several helpers. Helpers can be used either in the source or in the target patterns of the rule. Most of them are attribute helpers, which return a Boolean value that is used to confirm whether the element has certain features, for instance a diverging exclusive gateway (see rule `ExcGateway2Decision` in Listing 2). In the case of the rule `SqFlow2Candidate`, the proper matches are the sequence flows outgoing from diverging exclusive gateways (see `from` section in Listing 3). Additionally, we have implemented an operation helper for the default condition of the component, built as the negation of all the other conditions. The definition of this helper uses the imperative features of the ATL (see Listing 4).

Listing 4: Operation helper returning a string for the default sequence flow as the negation of all the other conditions

```
helper def : getConditions(excGtw: bpmn20!ExclusiveGateway) : String
      =
   excGtw.outgoing -> collect(e| e.conditionExpression) -> asSet()
        ->
     iterate( cond; conj: String='' | conj +
        if (not cond.oclIsUndefined()) then
           if conj='' then
              'not (' + cond.body +')'
           else
              ' and not ('+ cond.body + ')'
           endif
        else
           ''
        endif)
;
```

Fig. 7 shows all the rules and helpers involved in the module to transform a BPMN (X)OR-parallel component into a PROforma plan with a decision task. Rectangles represent the rules and the font indicates the type of rule: regular font for standard rules and italics for lazy rules. The arrows indicate the dependencies among the rules, i.e. implicit triggering or explicit triggering in the case of lazy rules. Helpers are indicated as ovals connected to the rules which use them.

As a result of the use of the structure-identification and graph reduction strategies, the design of the ATL rules has been approached in a manageable way. For example, all the XOR-gateways in Fig. 1 are ultimately reduced to a schema similar to the one shown in Fig. 8, which makes model transformation affordable. As the transformation algorithm proceeds, these elements will expand in new PROforma processes until the final transformation is obtained, as explained before.

4. Results

4.1. Application of the transformation algorithm

We have applied our approach to an extract of the CPG for the diagnosis and treatment of acute and chronic heart failure (HF henceforth), developed by the European Society of Cardiology [38]. As most CPGs, it is a comprehensive document (85 page long) that includes several tables and flow-charts that visually support the recommendations. Concretely, the selected extract deals with the diagnosis of HF in the cases of acute and non-acute onset.

In order to apply our approach to a real use case, the first step is modelling the recommendations for the selected extract in BPMN. For this purpose we have used the Eclipse BPMN Modeller, which is a graphical editor for BPMN models. Fig. 1 shows the resulting process diagram in BPMN, which is a graph that includes 10 diverging XOR-gateways, representing choices, and their corresponding converging gateways. The graph also has four sub-processes, which contain their own process diagram.

The input to the transformation algorithm is the XML file of the above BPMN graph, as generated by the BPMN Modeller tool. The reduction phase of this graph includes the identification of the XOR-parallel components and of the sequential components that subsequently appear when replacing those parallel components by a node. Notice that the algorithm always identifies the innermost component. As explained before, each one of these components is transformed to PROforma by executing the corresponding ATL module. The final PROforma representation is obtained by embedding the resulting PROforma, starting from the last node (top-level node) after the graph reduction completes.

The output of the transformation algorithm is shown in Fig. 9. Since the BPMN process starts with a diverging XOR-gateway, the top-level plan of the PROforma resulting from the transformation (transformed PROforma henceforth) has a single plan. This plan contains one decision, four scheduling constrains, and four plans (with additional contents). Fig. 10 shows the representation of the contents of this plan in the PROforma graphical notation. Besides, Fig. 11 shows the original BPMN model highlighting the BPMN elements and components in Fig. 10. Both figures (also the XML code in Fig. 9) have circled numbers to indicate a mapping between elements. Number ① corresponds to the decision, which is further detailed in Fig. 12, and numbers ④, ⑥, ⑧, and ⑨ are the PROforma plans.

Notice that the use of the structure-identification strategy forces the introduction of new plans. For these additional elements, and also for the PROforma elements that do not have a BPMN counterpart, the transformation algorithm generates names using a predefined naming convention, such as `Decision_ExclusiveGateway_7`.

The decision task ① has two candidates, each of them with one argument and one recommendation rule. The argument condition for the first candidate (line 16 in Fig. 12) is obtained from the conditional expression of the corresponding BPMN conditional sequence flow. Similarly, the argument condition for the second candidate (line 22) is derived from a default sequence flow using the ATL helper shown in Listing 4.

Some discussion follows on the results of the application of our transformation algorithm. We have checked that it is able to generate

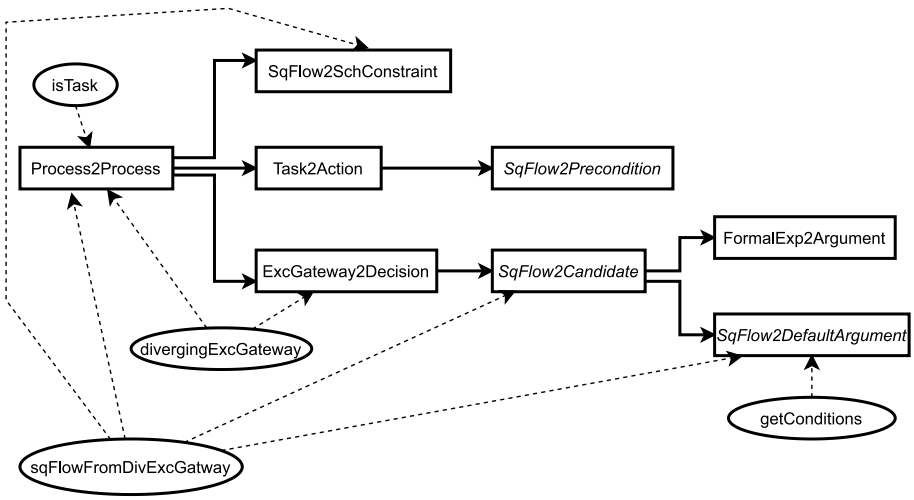

Fig. 7. ATL rules and helpers in the module for the transformation of BPMN (X)OR-parallel components.

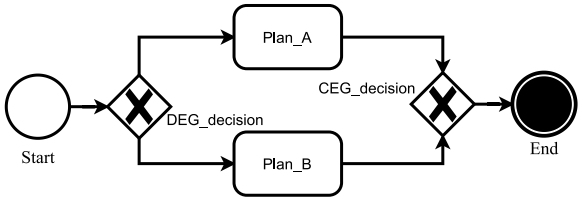

Fig. 8. Schema of the BPMN XOR-parallel component obtained after the reduction phase of the graph in Fig. 8.

all the needed PROforma elements, including the candidates of decision tasks along with all their details, from a structured BPMN model given as input. The only missing parts are the source data elements required by the conditions in the candidate arguments. For this reason, some final editing is required to fine-tune the transformed PROforma model. This is due to the plain text format used in the conditional expressions of the original BPMN model. Note that these elements could be properly generated if a more structured format for BPMN conditions were used.

For the time being, the verification of the PROforma model obtained in the transformation (after the required editing) is done manually. This includes not only verifying that all elements have been correctly translated into the model, but also that its execution produces the results intended by the modeller. An important part is verifying that all conditions have been properly transcribed. For instance, in the case of (X)OR-parallel components, this implies checking that the expressions in BPMN conditional sequence flows have been copied to the argument/condition of the adequate candidate in the corresponding PROforma decision. Note that all PROforma conditions will be treated as hard constraints, although the clinician will usually be able to confirm which action to undertake (or which candidate to choose).

The transformed PROforma model is structured since it reflects the structure of the input BPMN model. When comparing the transformed PROforma model (in Fig. 10) with a manually developed PROforma model for the same process (in Fig. 2), it can be seen that the former contains more levels and elements. The reason for that is the structure-identification strategy, which divides the BPMN model in more manageable structures and thus forces the introduction of new plans. Related to this, the necessary ATL rules are designed for (and applied to) more simple and clearly defined model components. Otherwise, the features of the BPMN language would make the application of an MDD approach very challenging.

4.2. Modelling of clinical processes in BPMN by a clinical–technical team

To assess our hypothesis that a more accessible language such as BPMN can foster the engagement of clinicians in the CPG modelling task from its initial stages, we have conducted a small experiment. The experiment involved one knowledge engineer (BMS coauthor) together with two specialists in Cardiology (PP and EDM coauthors). The other knowledge engineer (MM) participated as an observer during the process. The initial objective of the experiment was to model the same process fragment both in BPMN and PROforma, to obtain feedback (qualitative and comparative) from the cardiologists on the suitability of these two languages for modelling clinical processes, but also to observe the collaborative modelling process. The process fragment that the cardiologists chose to model was the flowchart for the diagnosis of HF included in the 2021 version of the guideline. As modelling tools, the knowledge engineers selected intuitive and easy-to-use graphical editors, when possible.

The structure designed for the modelling exercises consisted of the following steps: (1) basic training in the language (BPMN or PROforma) by the knowledge engineer; (2) initial modelling of the clinical process individually by each cardiologist, to better get acquainted with the language; (3) discussion on the models and resolution of doubts and queries, jointly by the cardiologists and the knowledge engineer; (4) refinement of the initial models and agreement on a single shared model, by the cardiologists alone; and (5) discussion on the shared refined model and agreement on the final model, again by the cardiologists and the knowledge engineer. Note that the discussions on the models usually include both technical details of the language and clinical aspects of the process.

The BPMN modelling exercise was carried out in all its phases without major problems. However, the PROforma modelling exercise could not be completed because the cardiologists, after the initial training in the language, found it too complex to use it themselves. This confirms that not all languages are equally accessible to clinicians. As a demonstration of the language after not having completed the exercise, a PROforma model of the same process carried out by the knowledge engineer was presented to the cardiologists and its behaviour was discussed with them.

At the end of the experiment, the cardiologists were asked to fill out a short survey which consisted of 15 Likert statements (i.e. with 5-point scale answers) plus 4 open questions. According to the answers (mostly agree or strongly agree, for all statements), they agree on the ease of use of the BPMN language after a basic initial training, and on the understandability of the resulting BPMN models. They view favourably that BPMN models facilitate the discussion of clinical issues,

```
<?xml version="1.0" encoding="UTF-8"?>
<proforma:Process
    xmlns:proforma="http://www.keg.uji.es/proforma"
    xmlns:xmi="http://www.omg.org/XMI"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <topLevelPlan id="HF_Diagnosis.proforma" name="HF_Diagnosis.proforma" caption
        ="HF_suspected" cycleNumber="1" optional="false" automatic="false">
        <subprocess>
            <task xsi:type="proforma:Plan" id="Decision_ExclusiveGateway_1" name
                ="Decision_ExclusiveGateway_1" cycleNumber="1" optional="false"
                automatic="false">
                <subprocess>
                    <schConstraint id="ExclusiveGateway_1_Decision_ExclusiveGateway_3"
                        name="ExclusiveGateway_1_Decision_ExclusiveGateway_3" source
                        ="ExclusiveGateway_1" target="Decision_ExclusiveGateway_3"/>
                    <schConstraint id="ExclusiveGateway_1_AdHocSubProcess_2" name
                        ="ExclusiveGateway_1_AdHocSubProcess_2" source
                        ="ExclusiveGateway_1" target="AdHocSubProcess_2"/>
                    <task xsi:type="proforma:Decision" id="ExclusiveGateway_1" name
                        ="AcuteDecision" cycleNumber="1" optional="false" automatic
                        ="false">...</task>                                          } 1
                    <schConstraint id
                        ="Decision_ExclusiveGateway_3_Decision_ExclusiveGateway_5" name
                        ="Decision_ExclusiveGateway_3_Decision_ExclusiveGateway_5"
                        source="Decision_ExclusiveGateway_3" target
                        ="Decision_ExclusiveGateway_5"/>
                    <task xsi:type="proforma:Plan" id="Decision_ExclusiveGateway_3"
                        name="Decision_ExclusiveGateway_3" cycleNumber="1" optional
                        ="false" automatic="false">
                        <precondition conditionExpression="result_of
                            (ExclusiveGateway_3) = CDecision_ExclusiveGateway_3"/>
                        <subprocess>...</subprocess>                                 } 4
                    </task>
                    <task xsi:type="proforma:Plan" id="Decision_ExclusiveGateway_5"
                        name="Decision_ExclusiveGateway_5" cycleNumber="1" optional
                        ="false" automatic="false">
                        <subprocess>...</subprocess>                                 } 6
                    </task>
                    <schConstraint id="AdHocSubProcess_2_Decision_ExclusiveGateway_7"
                        name="AdHocSubProcess_2_Decision_ExclusiveGateway_7" source
                        ="AdHocSubProcess_2" target="Decision_ExclusiveGateway_7"/>
                    <task xsi:type="proforma:Plan" id="AdHocSubProcess_2" name
                        ="ECGXray" optional="false" automatic="false" cycleNumber="1">
                        <precondition conditionExpression="result_of
                            (ExclusiveGateway_1) = CAdHocSubProcess_2"/>
                        <subprocess>...</subprocess>                                 } 8
                    </task>
                    <task xsi:type="proforma:Plan" id="Decision_ExclusiveGateway_7"
                        name="Decision_ExclusiveGateway_7" cycleNumber="1" optional
                        ="false" automatic="false">
                        <subprocess>...</subprocess>                                 } 9
                    </task>
                </subprocess>
            </task>
        </subprocess>
    </topLevelPlan>
```

Fig. 9. Top-level plan of the transformed PROforma model obtained from the BPMN model shown in Fig. 11.

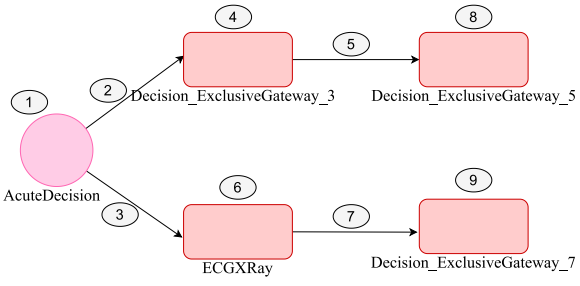

Fig. 10. Top-level plan of the transformed PROforma model in graphical notation.

both with clinicians of the same speciality and with other clinical staff. They also agree that BPMN models help in a deeper understanding of clinical processes and that, consequently, could improve the practical application of CPG recommendations. In a different area, they note that BPMN could help in the training of Cardiology residents. As far as PROforma is concerned, the cardiologists feel that, since the building blocks of the language are more complex, it would take more time to get acquainted with the language and that the modelling task would be more time-consuming for them. On the positive side, they point out that PROforma models would be a good tool to keep track of the dynamics of the clinical process, since these models can be executed.

For her part, the knowledge engineer highly values the involvement of the experts in the process modelling task from the very beginning, and all that this implies (the possibility of discussing clinical and technical aspects with them, or the refinement of the BPMN model in collaboration with them). Consequently, the final BPMN model is more comprehensive and better conforms to the knowledge of the intended clinical user, compared to a BPMN model developed by knowledge engineers alone.

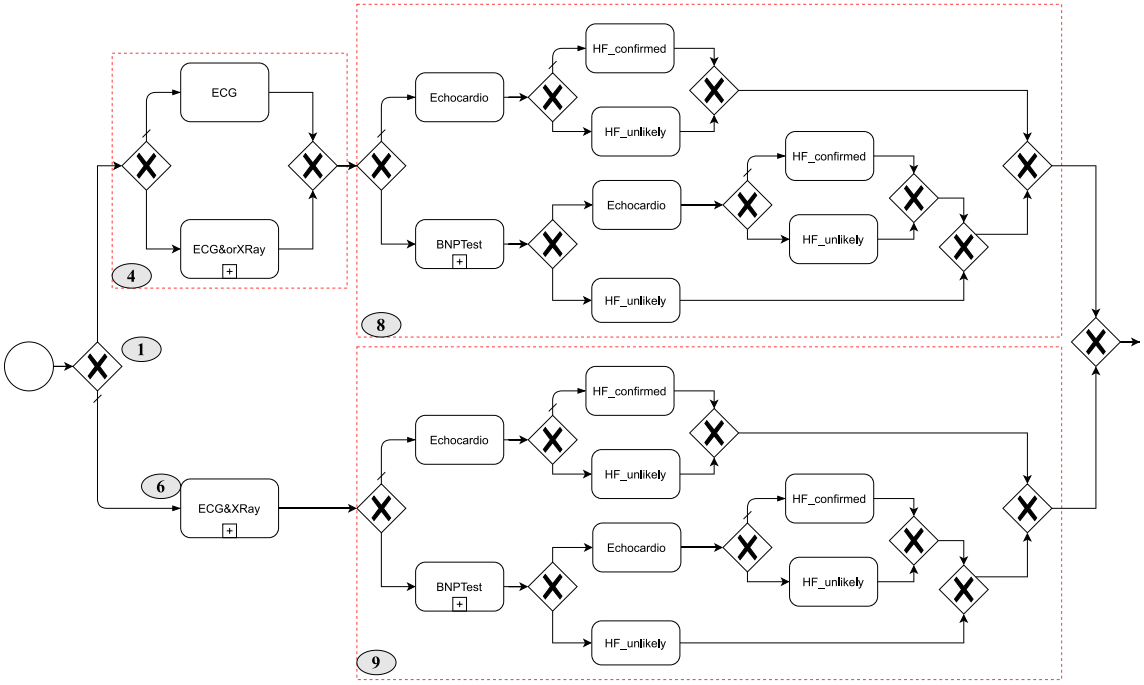

Fig. 11. Input BPMN model showing the correspondences with the PROforma elements in Figs. 9 and 10.

```
12    <task xsi:type="proforma:Decision" id="ExclusiveGateway_1" name
          ="AcuteDecision" cycleNumber="1" optional="false" automatic
          ="false">
13        <candidate id="CDecision_ExclusiveGateway_3" name
              ="CDecision_ExclusiveGateway_3">
14            <recommendRule ruleExpression="netsupport
                  (ExclusiveGateway_1, CDecision_ExclusiveGateway_3) &gt
                  ;= 1"/>
15            <argument id="arg_01" name="arg_01" support="confirm">
16                <argCondition conditionType="argument"
                      conditionExpression="acuteOnset = false"/>
17            </argument>
18        </candidate>
19        <candidate id="CAdHocSubProcess_2" name="CAdHocSubProcess_2">
20            <recommendRule ruleExpression="netsupport
                  (ExclusiveGateway_1, CAdHocSubProcess_2) = &gt;= 1"/>
21            <argument id="Default_Argument" name="Default Argument"
                  support="confirm">
22                <argCondition conditionType="argument"
                      conditionExpression="not (acuteOnset = false)"/>
23            </argument>
24        </candidate>
25    </task>
```

Fig. 12. Details of decision task ① in Fig. 9.

To sum up, in view of the results of our small experiment we can conclude that clinicians can be more easily engaged in the modelling of clinical processes when using a language such as BPMN, and that they can even take a leading role in this task. Moreover, this easy-to-use notation allows for the collaboration of mixed clinical–technical teams, resulting in models of higher quality.

5. Conclusions

With the aim of facilitating the development of CIGs and to involve clinicians in this task, we propose to support the modelling of clinical processes in CPGs based on transformations, from a preliminary description in the BPMN notation for business processes to a detailed implementation in a CIG language, in this case PROforma. We tackle these transformations following the MDD paradigm, which is a relatively unexplored approach in the area of CIGs.

We have implemented an algorithm that exploits the graph-oriented nature of the source BPMN model, identifying structures of interest that are then transformed to the target PROforma model by applying transformation rules in the ATL language. In addition to the algorithm itself, we have developed the necessary PROforma metamodel and ATL transformation rules. We have tested our implementation (algorithm plus ATL rules) on an extract of a CPG for the diagnosis and treatment of HF. Besides, we have conducted a small experiment to assess whether BPMN is accessible to clinicians, and that by using it they can be more easily involved in the modelling of clinical processes.

Our proposal to support the modelling of clinical processes using BPMN as a bridge language and transformations is based on the ideas of Kaiser et al. [6]. In previous works we have addressed this problem with the same rationale of using BPMN and transformations but without following the model-driven paradigm [19,39]. This work is also inspired by Murzek et al. [40] work, which focuses on the transformation

of structural patterns but suggests a segmentation of the input model to overcome the difficulties of a full model-driven transformation between two business process modelling languages.

In the area of CPGs, the work by Pérez et al. [41] is pioneering in the application of an MDD approach. This work presents a model-to-text tool that transforms a CPG model in terms of UML statecharts into a model checking representation thereof, for verification purposes. Our approach is similar but differs in two important aspects, apart from the final goal. On one hand, our method implements a model-to-model transformation based on a declarative transformation language, and on the other hand, the source and target languages we use are very different. In particular, we claim that the source BPMN language we use is better tailored to the purpose of involving clinicians in the modelling of CPG processes. The work by Farkash et al. [42] focuses on a solution for the representation of CPGs so that clinicians can easily read and verify them against patient data in the Electronic Health Record. They propose modelling CPGs as NRL rules and then translating them into OCL constraints by using MDD transformations. Although the simplicity of rules may be ideal in certain scenarios, they cannot easily express the variety of clinical processes that can be found in complex CPGs. Moreover, it can be argued that in general rules are a less intuitive and engaging formalism for clinicians compared to BPMN. Less directly related to our work, the goal of the work by Nyameino et al. [43] is the gamification of CPG content for clinicians in training. For this purpose, they exploit the entity model of a clinical encounter and the workflow model of clinical processes involved. In this case the approach is model-driven but no transformations are used.

The MDD paradigm used in this work allows us to describe complex transformations in a more compact way, and with a higher level of abstraction, compared to our previous solutions. As a consequence, our approach benefits from improved explainability, traceability, and comprehension of the transformations. Besides, the transformations are standalone entities implemented as ATL modules, which facilitates their maintainability and reusability. An additional advantage of our approach is that it can be generalised to transform BPMN models to models in other target CIG languages. By defining rules for other languages, it would be possible to capitalise on the effort of modelling a guideline in BPMN. A parallel could be drawn with the model-driven architecture proposed by OMG [44], which distinguishes platform-independent models (PIMs) from platform-specific models (PSMs), and where a PIM could be transformed into various PSMs. In our case, BPMN is the notation for PIMs whereas PROforma and other CIG languages serve for PSMs. We believe that it would be easier to convince a medical institution to adopt BPMN (which is an OMG standard) than to adopt a particular CIG language. In this context, our approach would be highly beneficial.

As mentioned above, a limitation of our approach is that the resulting PROforma models have a more complex structure, with more elements, compared to manually developed PROforma models. Notice that this is related to the structure-identification strategy used, rather than to the use of MDD tools. Another problem is related to our choice of BPMN notation as a preliminary modelling language. This choice is motivated by our focus on the procedural part in CPGs. This implies that, in the case of CPGs with higher declarative content and few procedural parts, the BPMN notation would not be appropriate and therefore our approach would be less beneficial.

To sum up, we have not only tested our implementation on a realistic CPG fragment, but we have also shown in a preliminary experiment that clinicians can play an active and collaborative role in CPG modelling when using a language such as BPMN. Taken together, these results suggest that our BPMN and transformation-based approach to support the modelling of CPG processes (and hence CIGs) is feasible. As future work, it would be necessary to carry out a further and more comprehensive evaluation to determine if the approach and developed tools adequately serve this purpose.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This research has been supported by the Spanish Ministry of Economy and Competitiveness through project TIN2014-53749-C2-1-R.

References

[1] Graham R, Mancher M, Wolman DM, Greenfiled S, Steinberg E, editors. Clinical practice guidelines we can trust. The National Academies Press; 2011.
[2] Peleg M. Computer-interpretable clinical guidelines: A methodological review. J Biomed Inform 2013;46(4):744–63.
[3] Sonnenberg FA, Hagerty CG. Computer-interpretable clinical practice guidelines. Where are we and where are we going? Yearb Med Inform 2006;145(58):145–58.
[4] Peleg M, Tu S, Bury J, Ciccarese P, Fox J, et al. Comparing computer-interpretable guideline models: A case-study approach. J Am Med Inform Assoc 2003;10(1):52–68.
[5] De Clercq PA, Blom JA, Korsten HHM, Hasman A. Approaches for creating computer-interpretable guidelines that facilitate decision support. Artif Intell Med 2004;31(1):1–27.
[6] Kaiser K, Marcos M. Leveraging workflow control patterns in the domain of clinical practice guidelines. BMC Med Inform Decis Making 2016;16:1–23.
[7] Patel VL, Arocha JF, Diermeier M, Greenes RA, Shortliffe EH. Methods of cognitive analysis to support the design and evaluation of biomedical systems: The case of clinical practice guidelines. J Biomed Inform 2001;34(1):52–66.
[8] Seyfang A, Miksch S, Marcos M, Wittenberg J, Polo-Conde C, Rosenbrand K. Bridging the gap between informal and formal guideline representations. In: Frontiers in Artificial Intelligence and Applications, vol. 141, 2006, p. 447–51.
[9] Seyfang A, Martínez-Salvador B, Serban R, Wittenberg J, Miksch S, Marcos M, et al. Maintaining formal models of living guidelines efficiently. In: Bellazzi R, Abu-Hanna A, Hunter J, editors. Artificial intelligence in medicine. Berlin, Heidelberg: Springer Berlin Heidelberg; 2007, p. 441–5.
[10] Patel VL, Allen VG, Arocha JF, Shortliffe EH. Representing clinical guidelines in GLIF: Individual and collaborative expertise. J Am Med Inform Assoc 1998;5(5):467–83.
[11] Object Management Group (OMG). Business process model and notation (BPMN) version 2.0. 2011.
[12] Reichert M. What BPM technology can do for healthcare process support. In: Artificial intelligence in medicine. 2011, p. 2–13.
[13] Scheuerlein H, Rauchfuss F, Dittmar Y, Molle R, Lehmann T, Pienkos N, Settmacher U. New methods for clinical pathways -Business process modeling notation (BPMN) and tangible business process modeling (t.BPM). Langenbeck's Arch Surg 2012;755–61.
[14] Kirchner K, Malessa C, Scheuerlein H, Settmacher U. Experience from collaborative modeling of clinical pathways. In: M. Hess HS, editor. Modellierung Im gesundheitswesen: Tagungsband des workshops im Rahmen der modellierung. 2014, p. 13–24.
[15] De Ramón Fernández A, Ruiz Fernández D, Sabuco García Y. Business process management for optimizing clinical processes: A systematic literature review. Health Inform J 2020;26:1305–20.
[16] Tomaskova H, Kopecky M. Specialization of business process model and notation applications in medicine - A review. Data 2020;5:1–42.
[17] Zerbato F, Oliboni B, Combi C, Campos M, Juarez JM. BPMN-based representation and comparison of clinical pathways for catheter-related bloodstream infections. In: Proceedings - 2015 IEEE international conference on healthcare informatics. Institute of Electrical and Electronics Engineers Inc.; 2015, p. 346–55.
[18] Combi C, Oliboni B, Zardini A, Zerbato F. Seamless design of decision-intensive care pathways. In: Proceedings - 2016 IEEE international conference on healthcare informatics. Institute of Electrical and Electronics Engineers Inc.; 2016, p. 35–45.
[19] Martínez-Salvador B, Marcos M. Supporting the refinement of clinical process models to computer-interpretable guideline models. Bus Inf Syst Eng 2016;58(5):355–66.
[20] Gonzalez-Lopez F, Martin N, de la Fuente R, Galvez-Yanjari V, Guzmán J, Kattan E, et al. ProDeM: A process-oriented delphi method for systematic asynchronous and consensual surgical process modelling. Artif Intell Med 2022;135:1–14.
[21] Jouault F, Allilaire F, Bézivin J, Kurtev I. ATL: A model transformation tool. Sci Comput Program 2008;31–9.
[22] Sutton DR, Fox J. The syntax and semantics of the proforma guideline modeling language. J Am Med Inform Assoc 2003;433–43.

[23] Zur Muehlen M, Recker J. How much language is enough? Theoretical and practical use of the business process modeling notation. In: Advanced information systems engineering. 2008, p. 465–79.
[24] Mendling J, Reijers HA, van der Aalst WM. Seven process modeling guidelines (7PMG). Inf Softw Technol 2010;52:127–36.
[25] Marcos M, Campos C, Martínez-Salvador B. A practical exercise on re-engineering clinical guideline models using different representation languages. In: AI in medicine: Knowledge representation and transparent and explainable systems. 2019, p. 3–16.
[26] OpenClinical CIC: OpenClinical.net. 2020, http://www.openclinical.net. [Accessed 22 May 2020].
[27] Bézivin J. In search of a basic principle for model driven engineering. Spec Novatica Issue - UML Model Eng 2004;21–4.
[28] Mellor SJ, Clark AN, Futagami T. Model-driven development. IEEE Software 2003;20:14–8.
[29] Selic B. The pragmatics of model-driven development. IEEE Software 2003;20:19–25.
[30] Martínez Y, Cachero C, Meliá S. MDD vs. traditional software development: A practitioner's subjective perspective. Inf Softw Technol 2013;55:189–200.
[31] Sendall S, Kozaczynski W. Model transformation: The heart and soul of model-driven software development. IEEE Software 2003;42–5.
[32] Mens T, Van Gorp P. A taxonomy of model transformation. Electron Notes Theor Comput Sci 2006;125–42.
[33] Torres-Sospedra J, Martínez-Salvador B, Sancho CC, Marcos M. Process model metrics for quality assessment of computer-interpretable guidelines in proforma. Appl Sci 2021;11–(7).
[34] Jouault F, Kurtev I. Transforming models with ATL. In: Models 2005 workshops, LNCS 3844. 2006, p. 128–38.
[35] Mendling J, Lassen KB, Zdun U. On the transformation of control flow between block-oriented and graph-oriented process modelling languages. Bus Process Manag J 2008;96–108.
[36] Ouyang C, Dumas M, Van Der Aalst W, Hofstede AT, Mendling J. From business process models to process-oriented software systems. ACM Trans Software Eng Methodol 2009;1–37.
[37] Götz M, Roser S, Lautenbacher F, Bauer B. Token analysis of graph-oriented process models. In: Proceedings - IEEE international enterprise distributed object computing workshop. 2009, p. 15–24.
[38] Ponikowski P, et al. ESC guidelines for the diagnosis and treatment of acute and chronic heart failure. Rev Española de Cardidol 2016;2129–200.
[39] Martínez-Salvador B, Marcos M, Riaño D. An algorithm for guideline transformation: From BPMN to SDA. In: Procedia Comput Sci. 63, 2015, p. 244–51.
[40] Murzek M, Kramler G, Michlmayr E. Structural patterns for the transformation of business process models. In: Proc. - 10th IEEE int. enterprise distributed object computing conf. workshops. 2006, p. 1–18.
[41] Pérez B, Porres I. Authoring and verification of clinical guidelines: A model driven approach. J Biomed Inform 2010;43(4):520–36.
[42] Farkash A, Timm JT, Waks Z. A model-driven approach to clinical practice guidelines representation and evaluation using standards. In: Studies in health technology and informatics, vol. 192, 2013, p. 200–4.
[43] Nyameino JN, Rabbi F, Ebbesvik BR, Were MC, Lamo Y. A model driven approach to the development of gamified interactive clinical practice guidelines. In: ENASE 2019 - Proceedings of the 14th international conference on evaluation of novel approaches to software engineering. 2019, p. 147–58.
[44] OMG Standards Development Organization. Model driven architecture. 2022, https://www.omg.org/mda/. [Accessed 18 August 2022].