École Polytechnique Fédérale de Lausanne

BPMN process generation and optimization for readability in the context of telemedecine

by Charline Montial

Master Thesis

Approved by the Examining Committee:

Prof. Dr. Viktor Kunčak
Thesis Advisor

Antoine Morand
External Expert

Dr. Barbara Jobstmann
Thesis Supervisor

EPFL IC
IC - LARA
CH-1015 Lausanne

April 23, 2022

Acknowledgments

I would like to warmly thank my supervisor Dr. Barbara Jobstmann. She was always available for weekly meetings to suggest ideas, discuss my results and guide me.

Secondly I would also like to express my special thanks to my intern supervisor Antoine Morand. He was always enthusiast about my project and would bring interesting ways of thinking.

Also, I am really grateful to Romain Boichat, the operational director of Soignez-moi.ch. Without him, this project could not have been possible. I would also like to mention his great interest and invested follow-up of my progress.

I also would like to express my gratitude to Prof. Viktor Kunčak for his extensive review of my report.

With many thanks to Robin who supported me and often provided me great advice.

I express my gratitude to my parents Jean-Pierre and Susanne who have always been present and interested in my studies.

I would like to thank my sister Delphine who reviewed my report for her constant moral support.

Finally, I thank my horse Charlotte for the good walks that refreshed my mind and my cat Yami for his calming presence.

Lausanne, April 23, 2022 Charline Montial

Abstract

In business, automated processes can be specified using the BPMN standard, which stands for Business Process Model and Notation. It is presented as flowcharts with standardized graphical elements to show the possible flows and activities of a process. These diagrams can be seen as a graph with logical expressions conditioning their edges. The telemedecine start-up Soignez-moi.ch uses BPMN to describe complex medical forms in a readable way. Soignez-moi.ch offers medical consultations via an online platform. The first step for the patient is to fill a questionnaire about his/her symptoms. In the second step, a doctor accesses to the answers and proceeds to call the patient for the tele-consultation. The complexity of these medical forms comes from the fact that the questions are adapted to the patient answers, which leads to different possible flows.

Complex forms from another telemedecine website, Diaana, had to be translated to BPMN to be compatible and used at Soignez-moi.ch. Diaana is a contraction of the words diagnosis and anamnesis. The way these forms are specified is very different from what has been modeled in Soignez-moi.ch: instead of having separated branches, there is a sequential list of questions. In this list, every question is evaluated on its own to see if it is asked or not. The question conditions are Boolean expressions.

If a direct translation to BPMN is performed, there is no branching to show the relationships between the questions. The aim of my work is to highlight similarities in the branch conditions to make the diagrams more readable and editable by humans. Doctors and collaborators read the medical BPMN processes and may modify them. It should be a pleasant experience for them to go through the diagrams. They should quickly understand in a visual way if questions are related, and how they are related. The BPMN process readability has been achieved by extracting patterns, showing the implications between the questions to have a readable diagram with meaningful branches. This translation has been applied to 18 questionnaires. Two questionnaires were not improved because they were only composed of one question. One questionnaire was also not improved because all its questions were unconditional, meaning that no relationship between the question conditions could be highlighted. Another questionnaire with conditional questions was not improved because its conditions were unrelated. 14/18 questionnaires have had their readability improved.

Résumé

En entreprise, certains processus métier peuvent être modélisés à l'aide du standard BPMN, dont l'abréviation signifie "Business Process Model and Notation". Ce standard se présente sous la forme d'un diagramme avec des flèches (les "flots" possibles) avec des éléments graphiques standardisés pour montrer les issues possibles et les activités d'un processus. Ces diagrammes peuvent être vus comme des graphes avec des expressions logiques sur leurs arêtes. La start-up de télémédecine Soignez-moi.ch utilise BPMN pour décrire des questionnaires médicaux complexes de façon à ce qu'ils soient facilement compréhensibles. Soignez-moi.ch offre un service de consultations médicales via une plateforme en ligne. La première étape pour le patient est de remplir un questionnaire à propos de ses symptômes. Ensuite, un médecin accède aux réponses et appelle le patient pour la télé-consultation. La complexité de ces questionnaires médicaux vient du fait que le questionnaire est adapté aux réponses du patient au fur et à mesure, conduisant à différents déroulements possibles.

Des questionnaires médicaux complexes d'un autre site de télémédecine nommé Diaana doivent être traduits en BPMN afin d'être compatibles et utilisés chez Soignez-moi.ch. Diaana est une contraction des mots diagnostic et anamnèse. La façon dont ces questionnaires médicaux sont spécifiés est très différente de ce qui a été modélisé chez Soignez-moi.ch : à la place d'avoir des branches séparées, ce questionnaire se présente sous la forme d'une liste de questions dont l'ordre est fixe. Dans cette liste, chaque question est affichée ou non selon si sa condition d'affichage est satisfaite. Les conditions des questions sont des expressions Booléennes.

Si une traduction directe en BPMN est opérée, il n'y a pas de branches qui montrent les relations entre les questions. Le but de mon travail est de montrer les liens et similarités dans les conditions des branches pour rendre les diagrammes BPMN plus lisibles et modifiables par des êtres humains. Les médecins et collaborateurs consultent régulièrement les processus BPMN et peuvent les modifier. Cela devrait être agréable et pratique pour eux de parcourir les diagrammes. Ils devraient être en mesure de comprendre rapidement et de manière visuelle si des questions ont un lien entre elles, et quel est leur lien. La lisibilité des processus BPMN a été réalisée en exposant les implications entre les questions, le but étant d'avoir un diagramme avec des branches qui donnent de l'information. Cette traduction a été appliquée à 18 questionnaires. Deux questionnaires n'ont pas pu être améliorés car ils étaient seulement composés d'une question. Un questionnaire n'a également pas été amélioré car toutes ses questions sont

inconditionnelles, signifiant qu'aucune relation entre les questions ne pouvait être extraite. Un autre questionnaire avec des questions conditionnelles n'a pas changé car les conditions de ses questions n'avaient pas de liens entre elles à exploiter. 14 questionnaires ont vu leur lisibilité améliorée.

Contents

Acknowledgments **2**

Abstract **3**

1 Introduction **8**

2 Background **14**
2.1 BPMN medical form representation 14
2.2 Diaana 15

3 Design **18**
3.1 General BPMN structure 18
3.2 Maximum logical expressions factorization 19
3.3 Highlight branching similarities 20
3.3.1 Multiple occurrences of one condition 21
3.3.2 Highlight common prefixes among questions 22
3.4 Mutually exclusive questions 24
3.5 Questions reordering 26
3.6 Question implications 27
3.6.1 Core trigger implications 28
3.6.2 Consequential triggers implication 31
3.7 Remove condition redundancies 31

4 Implementation **34**
4.1 Technical details 34
4.2 Reordering implementation 36
4.3 Condition factorization implementation 37
4.4 Form generation 38
4.5 Triggers updating 38
4.6 Runtime 42

5 Results **43**

6 Related Work **52**

7 Conclusion **54**

Bibliography **55**

Chapter 1

Introduction

Soignez-moi.ch is a telemedicine start-up that offers medical consultations via an online platform. The patient must first fill a questionnaire about how he/she feels, providing his/her symptoms. The doctor reviews the answers that have been provided and takes care of the patient over the phone. Figure 1.1 presents the first question the patient has to fill, which asks what is the consultation reason. This questionnaire system contains variables that condition the next questions, which are asked to the patient or not depending on the answers. For the doctor to get the most useful information about the patient, some information (called "flags") are displayed according to specific conditions. These flags can be a piece of information, a treatment proposal, or even a diagnosis. Soignez-moi.ch uses the BPMN (Business Process Model and Notation) standard to describe its complex medical forms. BPMN is a standard that is presented as a flowchart. It allows one to visually describe a sequence of (business) activities and expose the different possible situations and their treatment. Medical forms are not business processes in the strict sense, but this standard, combined with the engine used to run these processes, Flowable, made it a suitable solution. Figure 1.2 shows an example of a questionnaire represented in BPMN. Each user task (squares with texts) represents a question. The branches are the arrows, and the text next to the branches are their condition. We can also see that there is a branch called `SPORT`, a `YOGA` subbranch and a `SWIMMING` subbranch. We can see that there is a question that is general to all sports on the `SPORT` branch, a specific question to the `YOGA` subbranch and another one to the `SWIMMING` branch. At each XOR gate, there are always two possibilities: the flow with the condition if it is met, and a branch with no condition that allows one to skip a question or a whole branch when the condition is not met. The flows with no conditions that are needed to skip the branches are called "default flows" and can be recognized by the little line that crosses the flow right after the XOR gate. In this example, these branches indicate which questions are only asked if the patient practises sports, and the two subbranches if the sports is yoga or swimming in particular. The general question about sports asks if the training sessions have been suddenly much more frequent, and the specific questions ask about the training details of the particular sport.

Diaana, whose name is the contraction of diagnosis and anamnesis, is another website used for online medical consultations which uses a different way of describing its medical forms. Instead of using a known standard, it created its own customized way to model forms. This mean to describe its forms is not presented as a diagram, but rather as a sequential list of questions, each having a Boolean logical expression to check whether it will be displayed to the patient or not. The questions are organized as a list, and the order of the questions is fixed. For clarity, the Diaana questions are grouped in sections. The purpose of these sections is to group the questions in logical themes. Soignez-moi.ch needs to have these questionnaires translated to BPMN to be able to use them. These Diaana questionnaires translated to BPMN must be human readable. A BPMN process had a good readability if a doctor can easily understand the relationship between the questions by looking at the branches structure.

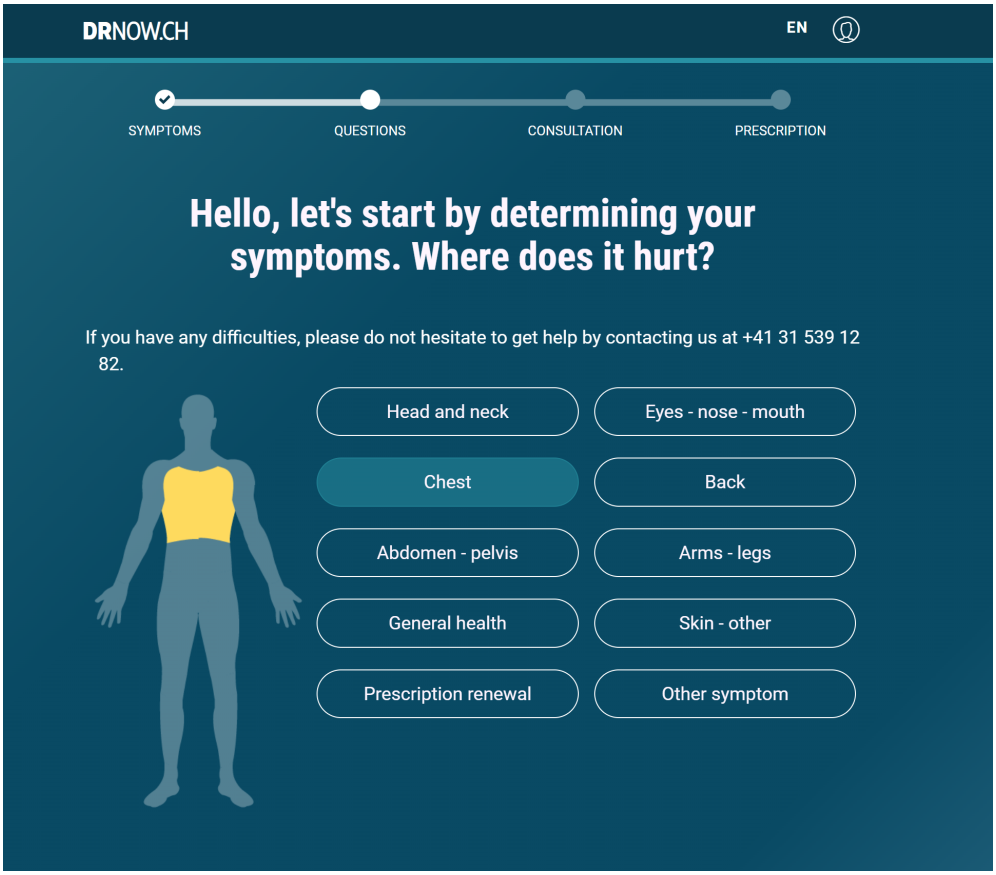

Figure 1.1: This is the first question the patient has to fill to begin its online medical consultation.

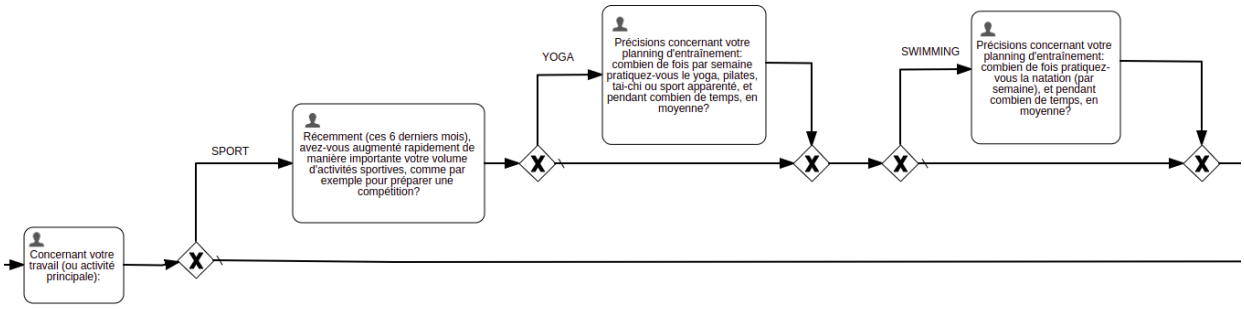

Figure 1.2: This BPMN diagram of a questionnaire shows that if the patient practises sport, and more precisely yoga or swimming in this case, he/she will have specific questions.

If a direct approach is used to translate the forms used in Diaana to BPMN for Soignez-moi.ch, the BPMN advantage of branching readability is not present. Diaana is modeled as a linear sequence of questions. The result of a direct translation is a BPMN diagram with a single chain of questions, each of them being evaluated if it will be displayed or not. There is no branch or subbranch. From a readability point of view, it not desirable. When a doctor looks at the questionnaire diagram, he should understand the relationship between the questions. For example, if he sees a branch named `WOMAN`, he knows that all the questions on this branch will only be asked to women. He may need to add some questions, and appending a question to a precise subbranch is clearer: by looking at the branch structure, the question relationships are clear. We can see in the upper part of Figure 1.3 that a direct translation outputs an individual branch for each conditional question. The first two conditions are: `(not REPETITIVETRAUMA) and TRAUMA` and: `(not REPETITIVETRAUMA) and TRAUMA`. Upon reading them, we can see that they are exactly the same and could be both appended on one branch as it is presented in the lower part of the figure. If we look at the upper part of the image again, we can see that the last two questions conditions are: `TRAUMA and WRIST` and: `TRAUMA and WRIST and (not HANDSTRETCHEDSHOCK)`. We can notice that there is a common part (`TRAUMA and WRIST`) in these two conditions. This common part can be used to create a branch that holds the question with this condition and a subbranch with the last question that has the additional `(not HANDSTRETCHEDSHOCK)` part. Finally, there is a common part in all for conditions, which is `TRAUMA`. It means we can have a common `TRAUMA` branch that holds all these questions and subbranches. Let us notice in both lower and upper parts of the figure that at each XOR gate, if the condition is not met, there is a flow with no condition that allows to skip a question and not get stuck. We can see the results with the question relationships exposed in the lower part of the figure: there is a common `TRAUMA` branch, a subbranch called `(not REPETITIVE TRAUMA) and TRAUMA`, another subbranch named `TRAUMA and WRIST` and a sub-subbranch on

top of this one called `TRAUMA and WRIST and (not HANDSTRETCHEDSHOCK)`.

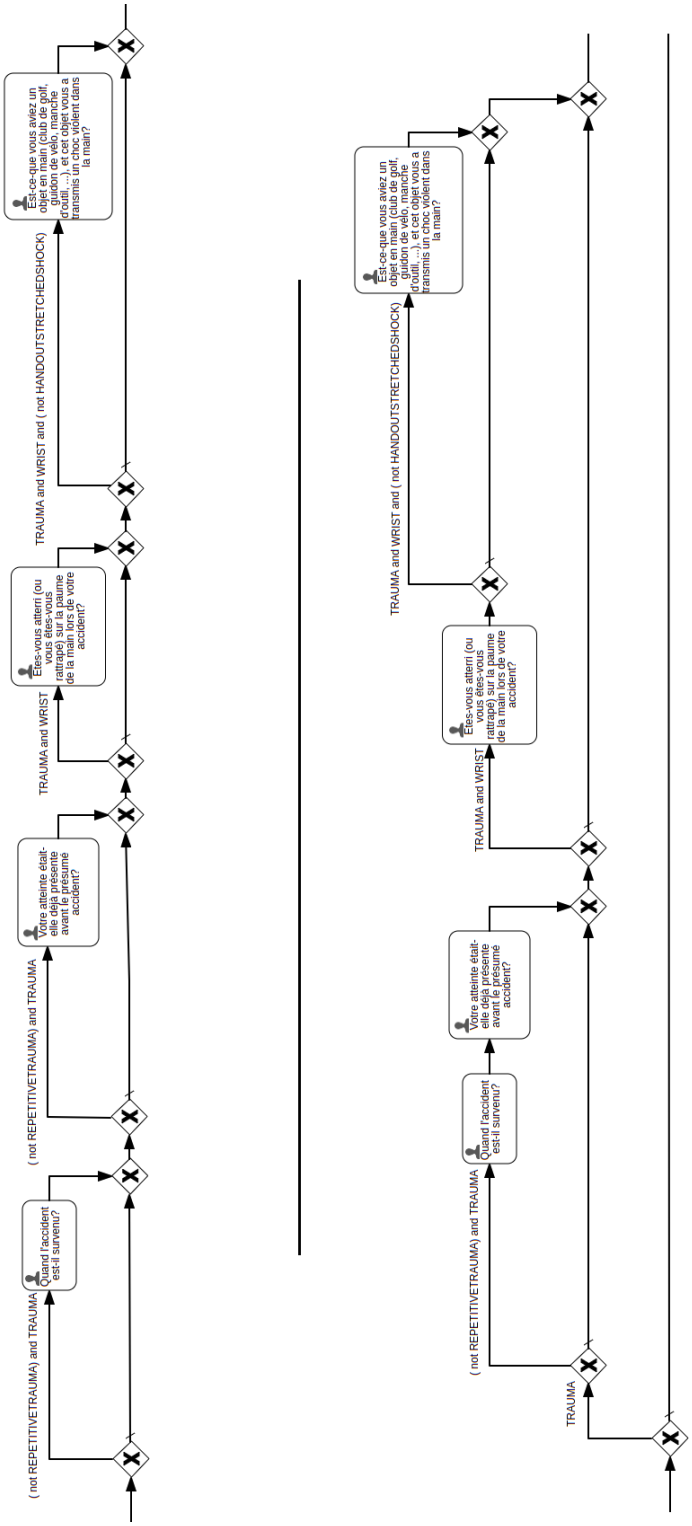

Figure 1.3: The upper part of the figure shows the result of a direct translation from a Diaana form to its BPMN version. Each question is evaluated individually. The potential relationship between the questions is not visible. The lower part shows the result of a translation that exposes the question relationships hidden in the Boolean conditions.

The challenge of the readability optimization is to perform a translation that can expose the links between the logical expressions under which the questions are displayed. The aim is to have a diagram that is human readable. An output that is considered to be "successful" could for example show a separate branch dedicated to the pregnant women only, or simply have a single branch for all consecutive questions that present the exact same condition. It should be clear from the structure which questions are related, without reading the conditions. This is what we mean by "optimization for readability".

My approach to solve this problem consists of a set of rules which, combined together, provide the desired result. The conditions of the questions that need to be translated are expressed in their disjunctive normal form. Some variable names are quite long, resulting in conditions that are not pleasant to read. The first rule to improve the readability is to factorize the expressions as much as possible. This feature does not improve the branching but it improves the readability and is helpful for the subsequent steps. The second key idea is to highlight similarities between the logical Boolean expressions, for example append a question to an already existing branch if both questions have the exact same conditions, or if a prefix is identical to both questions, create a branch with this prefix and have two subbranches appended to it. The third point is to exploit the mutual exclusivity of the questions. When two questions are mutually exclusive, we can have two parallels flows since only one path can be taken by execution. To further expose the common part of the conditions, the questions can be reordered to be grouped by conditions. To keep a logical order of the questions, they have been reordered within their section while respecting the constraints due to their mutual dependency. The last key idea is that some questions are not visually related, even though they actually are. Implications between different Boolean variables used can be noticed, and by taking them into account, additional branching can be modeled.

Chapter 2

Background

2.1 BPMN medical form representation

In Soignez-moi.ch, the questionnaire is represented as a BPMN process. The questions are modelled with BPMN user tasks and the conditional aspect as sequence flows with conditions and XOR gates. Each of these user tasks is linked to a form to fill. This form defines the answer variable names and their type. These variables can be used at anytime in the process. Questions with a common theme can be grouped under a section. A section is a subprocess within a main process. A section is defined in the main process with a BPMN call activity. At each question answered, the Flowable engine looks for the next path to take given the sequence flows. For this use case, namely model complex medical forms, only a very restricted subset of features of the BPMN standard is used. The aim was to keep these medical forms as simple and clear as possible. BPMN is normally used to describe business processes with many roles, parallel flows and other complex situations that were not relevant to our case, where there is one linear flow with a single user with one type of task, namely a form to fill.

One can notice in Figure 2.1 on the left the question as it appears in the BPMN diagram. A question is represented as a BPMN user task. The circle on the left is the start of the questionnaire and is called a start event in BPMN. The arrows show the different paths and are called sequence flows in BPMN. This question is unconditional because there is only one path with no condition. We can read that the question is : "Votre douleur est-elle présente constamment ou de manière fluctuante?". This user task is linked to a form, which is what appears on the right. We see that the question is the first line of the form, then we can see the possible options that the patient can answer: "Ma douleur est constante et d'intensité stable", "Ma douleur est constante, mais elle varie en intensité.", "Ma douleur apparaît et disparaît (elle est fluctuante)." and finally "Mon problème ne déclenche jamais de douleurs.". This view comes from the form editor where the question type is defined, along with the answer type and the choices. The grey expression in camel case named `votreDouleurEstEllePresenteConstammentOuDeManie` is the variable name

where the answer is going to be stored.

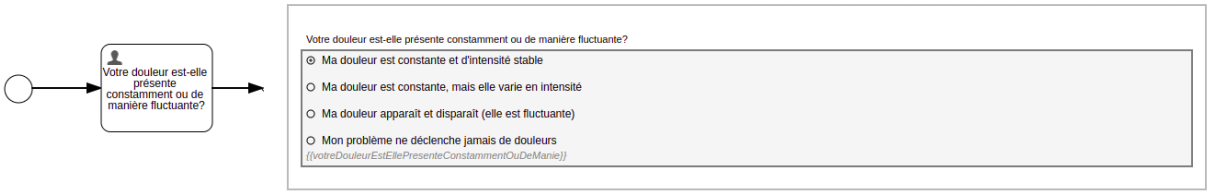

Figure 2.1: On the left, we can see the question as it appears in the BPMN process (BPMN User task). On the right, we can see the full question screen.

2.2 Diaana

Diaana (diagnosis and anamnesis) is a medical questionnaire that is designed to provide a detailed anamnesis of the patient. This anamnesis purpose is to help the doctor choose the best treatment for the patient by providing an extensive view of the patient problem, but also about its general health that is a necessary context. There is a linear list of 243 questions. Each question may be conditional. If it is conditional, its condition will be evaluated to know if the question will be displayed. Some answers can trigger what is called "triggers" which are Boolean variables set to "True". All these variables are initialized to "False" and can be activated (or "triggered") according to the answers. There are two types of these triggers variables: the triggers that are activated when some answers are chosen - a trigger can be activated by more than one answer. Symmetrically, an answer can activate more than one trigger. These variables are the core of the conditional system and for the anamnesis generation. For this reason, I am referring to them as "core triggers". For readability purposes, there is another kind of triggers that can be used in the exact same way as the triggers just described: they are not activated upon answering a specific option, but are described as a logical Boolean expression composed of triggers of any kind. I refer to them as "composed triggers". They are useful for the doctor because it is more concise to use a composed trigger instead of its definition which is a logical expression. For example, in the questionnaire, the patient can click on the finger(s) which hurt. In the next questions, it is needed to ask questions that are general to all finger wounds. Instead of expressing `FINGER1 || FINGER2 || FINGER3 || FINGER4 || FINGER5`, a trigger called `FINGER` equal to this condition helps the doctor understand more quickly that these questions concern every case where one or many fingers is/are hurt. If this same condition is needed more than once, it is also more practical and faster for the doctor to use the composed trigger instead of its definition. Finally, composed triggers are useful when they are used in logical expression. It helps keeping these logical expressions shorter and thus, clearer.

There are 732 triggers in the Diaana questionnaire which needs to be translated. 326 of them are core triggers and the 406 remaining are composed. There are a lot of them because Diaana's

purpose is to provide a precise and detailed anamnesis to the doctor. Both core and composed triggers are used by the doctor to define the diagnosis structure, including a list of the patient risk factors. Figure 2.2 shows how a question is created in Diaana, and more precisely, we can see that after each choice answer, the desired number of core triggers can be added to it. It means that when this option is picked, the associated core triggers will be set to true. The question in this case is "Votre douleur est-elle présente constamment ou de manière fluctuante ?", and the possible answers below are first "Ma douleur est constante et d'intensité stable". Its associated triggers are `24/24PAIN` and `PAIN`. The second option is "Ma douleur est constante, mais elle varie en intensité." and its associated triggers are `INTERMITTENTPAIN`, `24/24PAIN` and `PAIN`. The third option is "Ma douleur apparaît et disparaît (elle est fluctuante)." and its associated triggers are `INTERMITTENTPAIN` and `PAIN`. The last option is "Mon problème ne déclenche jamais de douleurs." and it does not have any associated triggers. Then, we can see in Figure 2.3 how a composed trigger is defined. We see than the composed trigger named `ARM` is equal to `ARMLATERAL or ARMPOSTERIOR or ARMMEDIAL`.

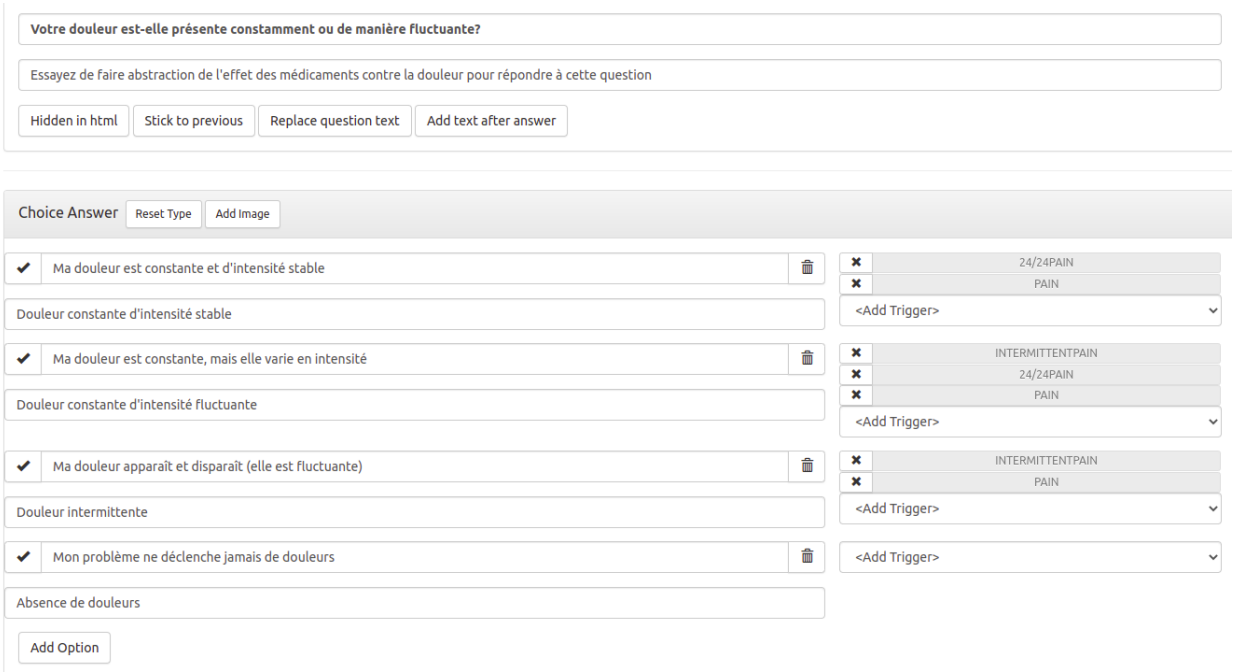

Figure 2.2: In Diaana, the creation of a question also implies defining which core triggers are activated upon answering certain options.

Figure 2.3: In Diaana, the composed triggers are defined with a logical Boolean expressions of other core and/or composed triggers.

Each time a question is answered, the state of the composed triggers is computed and they can be activated or deactivated each time a question is answered. This new state will determine which is the next question that is going to be asked. If a question is skipped because its condition was not met, it cannot be asked later because the questions in Diaana are in a sequential list of questions with a fixed order. The Diaana form has sections to group the questions by themes for readability. A section can also be conditional.

Chapter 3

Design

3.1 General BPMN structure

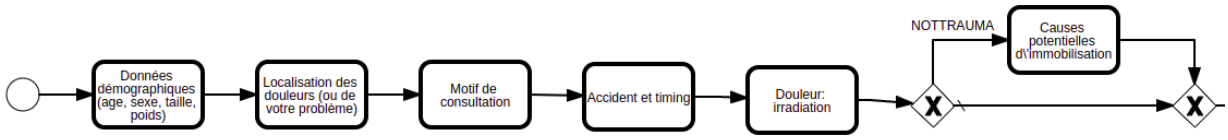

Figure 3.1: This is the main BPMN process. Each task is a subprocess which represents a section of questions from Diaana.

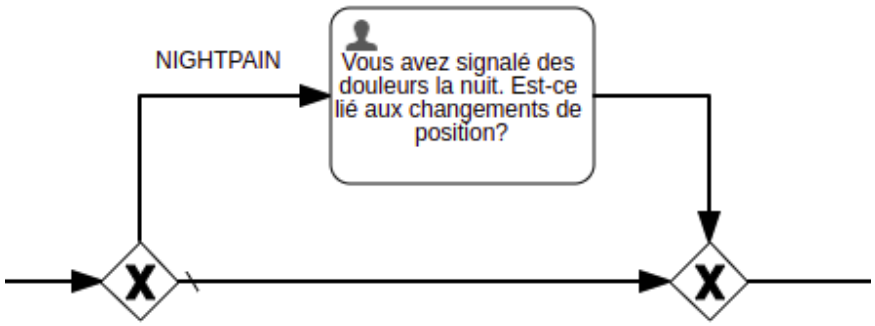

Figure 3.2: A default flow in BPMN

3.2 Maximum logical expressions factorization

In Diaana, all conditions are in their disjunctive normal forms. Some of these conditions can be quite long. One such example could be : `(SPRAIN_TRAUMA_REPORTED and ANKLE) or (SPRAIN_TRAUMA_REPORTED and MIDFOOT) or (SPRAIN_TRAUMA_REPORTED and LEG) or (TRAUMA_WITH_CRACKING and ANKLE) or (TRAUMA_WITH_CRACKING and MIDFOOT) or (TRAUMA_WITH_CRACKING and LEG)`. This causes two readability problems. The first problem is that a long condition is cumbersome to read, which is not practical for a doctor when looking at the questionnaire. The other problem is that it takes up a lot of space in the screen. It prevents the reader from seeing other questions at the same time on screen. This situation can be observed in Figure 3.3. We can see in the upper part of this figure that it is not immediately clear under which condition this question appears and that a long condition "stretches out" the diagram horizontally. We also cannot see the previous questions nor the next ones because the condition occupies a considerable area of the screen. To improve the readability of such conditions and also of the general aspect of the graph, one idea is to factorize them. Let us have a look at this condition (all variables in this logical expression being Boolean): `(SPRAIN_TRAUMA_REPORTED and ANKLE) or (SPRAIN_TRAUMA_REPORTED and MIDFOOT) or (SPRAIN_TRAUMA_REPORTED and LEG) or (TRAUMA_WITH_CRACKING and ANKLE) or (TRAUMA_WITH_CRACKING and MIDFOOT) or (TRAUMA_WITH_CRACKING and LEG)` which is a long condition that deteriorates the readability of the diagram, not only upon reading the condition to understand what it means from a medical point of view, but also because the diagram is forced to be spread such that this long condition can be displayed correctly. The first factorization that can be done is `((SPRAIN_TRAUMA_REPORTED) and (ANKLE or MIDFOOT or LEG)) or ((TRAUMA_WITH_CRACKING) and (ANKLE or MIDFOOT or LEG))`. This can even be further factorized as `(SPRAIN_TRAUMA_REPORTED or TRAUMA_WITH_CRACKING) and (ANKLE or`

`MIDFOOT or LEG)` which yields a shorter expression (one can actually notice that in this case, we went from the disjunctive normal form to the conjunctive normal form, which seems to be the most compact way of expressing this condition). In the lower part of Figure 3.3, we can see that it is clearer to read the condition and that some space is gained. This example sums up the essence of this feature. Some cases may not be as convenient as this one, and could be only factorized partially, or not at all. The implementation will be covered in chapter 4 which is about the implementation.

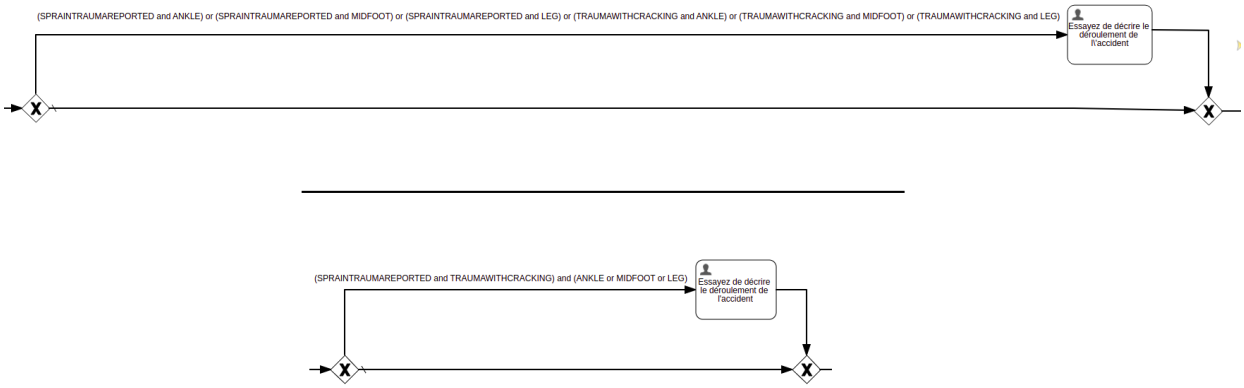

Figure 3.3: When a condition is not factorized (and in this particular case, a very long expression), not only is it harder for the user to understand it, but it also stretches out the diagram, deteriorating the readability. The upper part of the Figure shows when the condition is not factorized, and the lower part when the expression is factorized.

3.3 Highlight branching similarities

The purpose of the following optimizations is to improve the branching structure. More precisely, the ideas that will be presented use the Boolean conditions of the questions to compare them to extract similarities when possible. Then, the relationships between the questions (thus their condition) are integrated in the branching structure to show them visually and not only by reading the conditions.

3.3.1 Multiple occurrences of one condition

If two questions have the same conditions, they can be appended onto the same branch. This is only possible when there is no question between them that has a completely different condition. If we have a look at the upper part of Figure 3.4, we can see that the two questions have a separate branch. The condition of both of these branches is `PARESY`. The fact that these two conditions are identical is not taken into account in the branch structure. In the lower part of the figure, we can on the contrary see that both questions are on a single branch whose condition is `PARESY`. The fact that the condition of the questions is identical has been taken into account in the branch structure. We can visually see that these questions have the same condition.

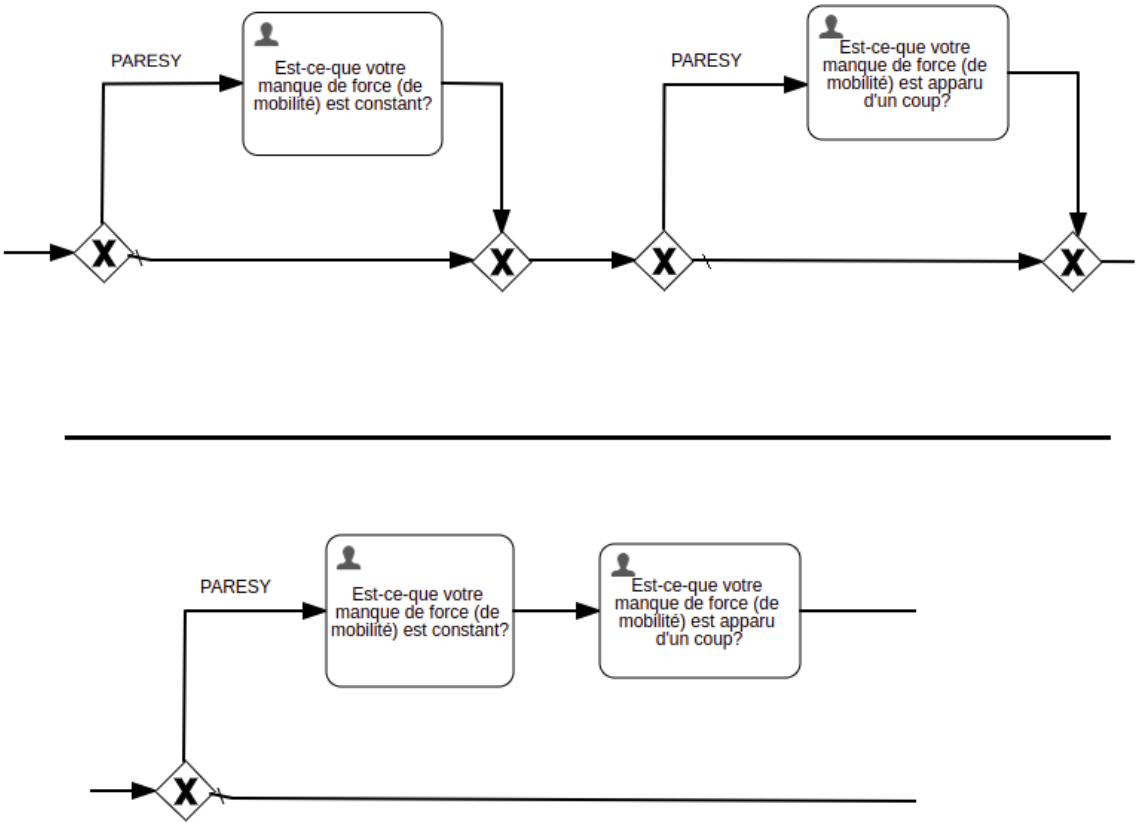

Figure 3.4: In the upper part we can see that with a direct translation, even if the two conditions are the same, we have two separated branches. In the lower part of the picture, the fact that these questions have the same condition has been taken into account. This results in having both questions on the same branch.

3.3.2 Highlight common prefixes among questions

This next rule takes into account common parts of question conditions. The first case can be observed in Figure 3.5. Let us look at the upper part of the image first. What can be noticed in this example is that the condition of the first question is `PAIN`, and the condition of the second question is `PAIN and (not TRAUMA<1WEEK)`. The intersection of these two conditions is `PAIN`. It is

actually the first question condition. Because of the non-empty intersection of their condition, a subbranch can be created for the second question to emphasize their common part. In this case, I say that the first question condition is a prefix of the second one's condition. The result by taking these observations into account can be seen in the lower part of the figure. The other case is the one that can be seen in Figure 3.6. This situation is similar. Let us look at the upper part: the intersection of the two conditions being: `TRAUMAYOUNGERTHAN6WEEKS and (NECK or HIGHBACK or LOWBACK)` and : `TRAUMAYOUNGERTHAN6WEEKS and FINGER` yields `TRAUMAYOUNGERTHAN6WEEKS`. Unlike in the previous case, the intersection is not the condition of any question, but it is still common to both of them. To highlight the link between the two questions, a branch with their intersection is created to support both subbranches as it can be seen in the lower part of this Figure.

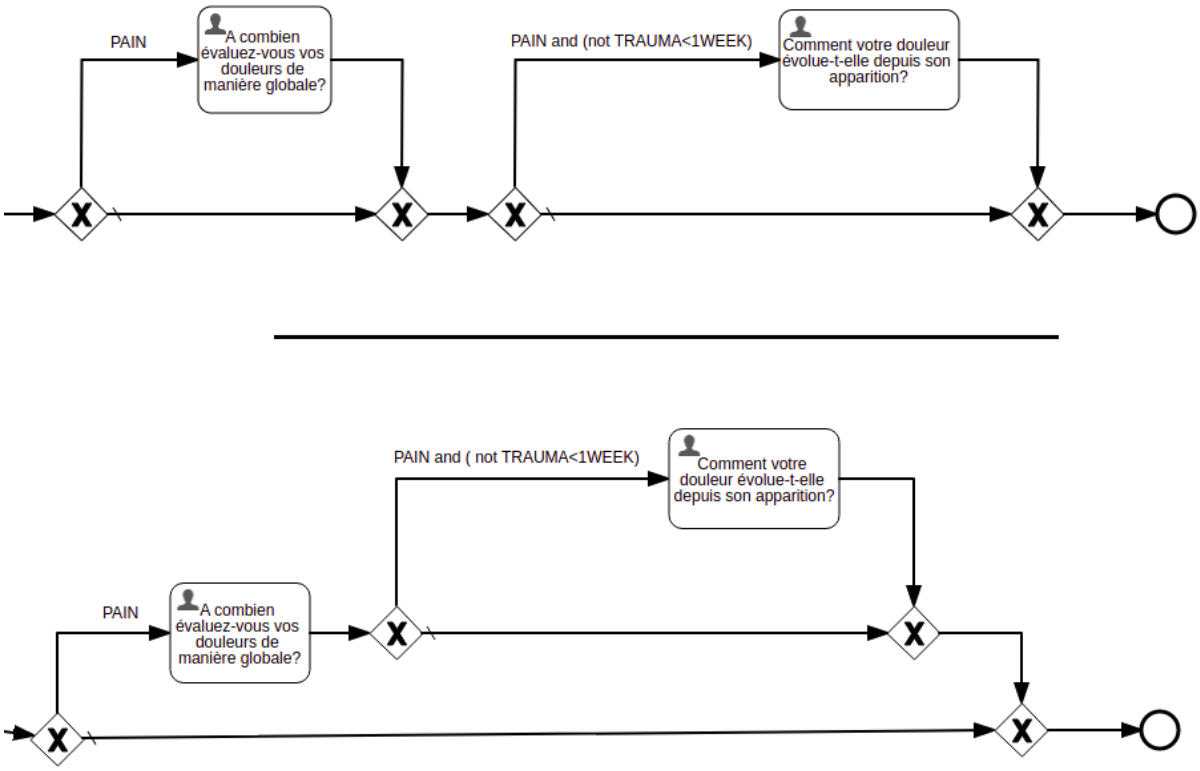

Figure 3.5: We can see an example where a subbranch is created due to the common part of both conditions of these two questions, in comparison to the upper part where the common part of the conditions is not taken into account.

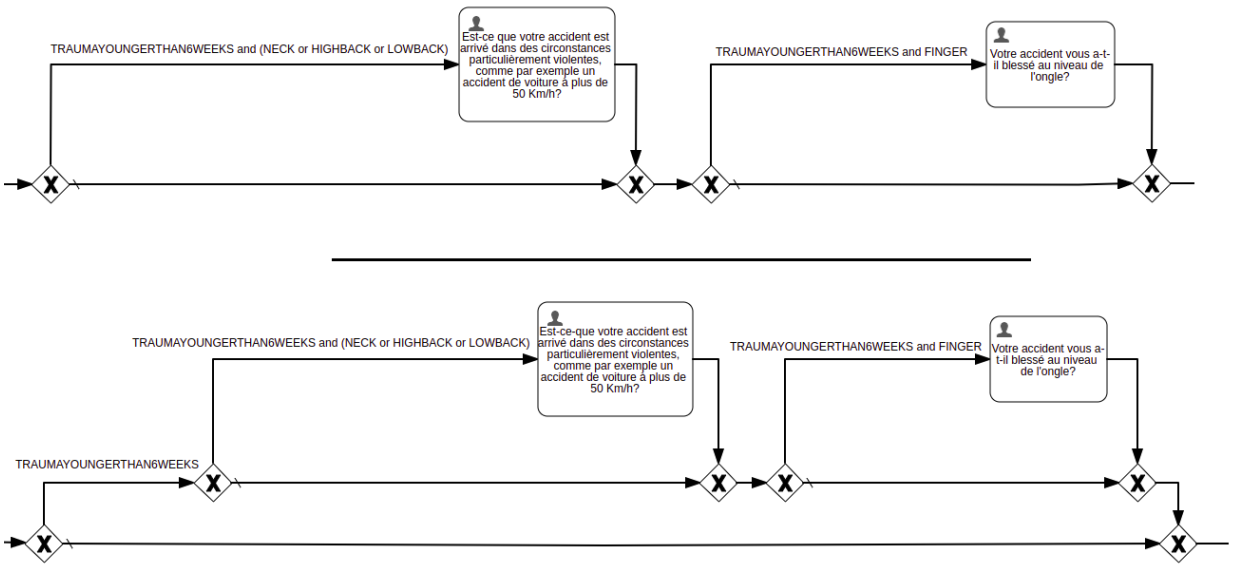

Figure 3.6: This situation illustrates when the intersection of two consecutive questions conditions is not empty, which this leads to a branch that highlights this common part and support both subbranches, in comparison to the upper part where the common part of the condition is not taken into account.

3.4 Mutually exclusive questions

Some consecutive questions are mutually exclusive. When such a pair of questions is found, we know that in one execution of the questionnaire, only one of them can be asked. This can be enforced and made visible in the BPMN process by displaying them in two parallel flows. We can see in Figure 3.7 that : `(not SHOULDER) and (not ARM) and (not ELBOW) and (not LOWBACK) and (not LATERALLOWBACK)` and : `SHOULDER and ARM and ELBOW and LOWBACK and LATERALLOWBACK` are mutually exclusive. The upper part of the figure does not take this mutual exclusivity in the branching structure, while the lower part exposes this property to have a more meaningful branching: the questions are put in parallel flows, emphasizing that they can never be asked both in one run of the process.

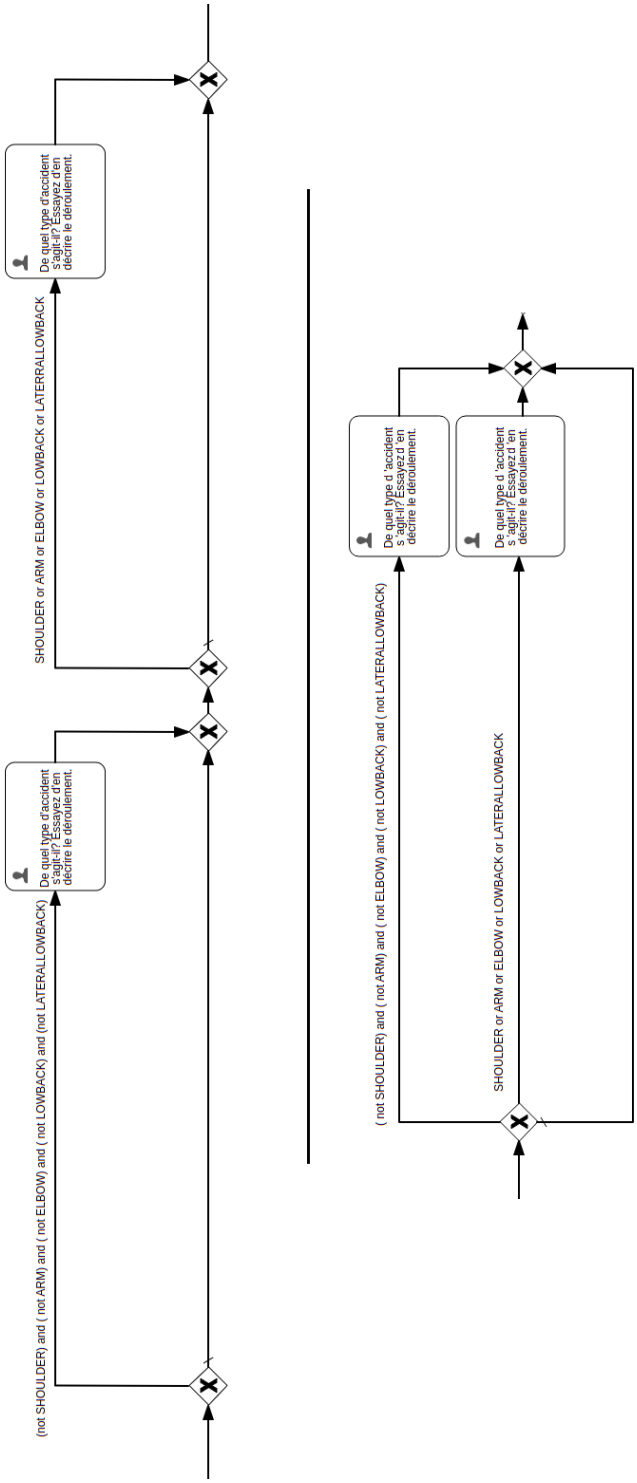

Figure 3.7: We can see that due to their condition, these questions can never be asked in the same execution. The upper part of the picture shows the branching when this is not taken into account, while the lower part exposes their mutual exclusivity.

3.5 Questions reordering

The readability can be further optimized with the following idea: I noticed that some questions that could theoretically be appended to the same branch, or that have a common part in their condition are not displayed in the most readable way because of other questions in between them with totally unrelated conditions that prevent a branching hierarchy to appear. Upon noticing this, the idea of reordering the questions occurred. If a reordering is performed, questions that have common parts could be grouped together. To keep a logical structure to the questionnaire, I decided questions could only be reordered within a section, since a section groups questions of the same topic. The reordering had to be performed carefully because one should not move a question that triggers a variable used in another question condition after the latter, because this question could never have a chance of being displayed. This reordering has to keep in mind the dependencies between the questions for the questionnaire to function properly. An example can be seen in Figure 3.8. In this picture, we can see that in the upper part of the picture, before the reordering, the question in the middle whose condition is `ATCDTRAUMA` prevents the two left and right questions to have a common branch, since they both have `ATCDSURGERY` as their condition. The result after the reordering can be seen in the lower part of the same figure.

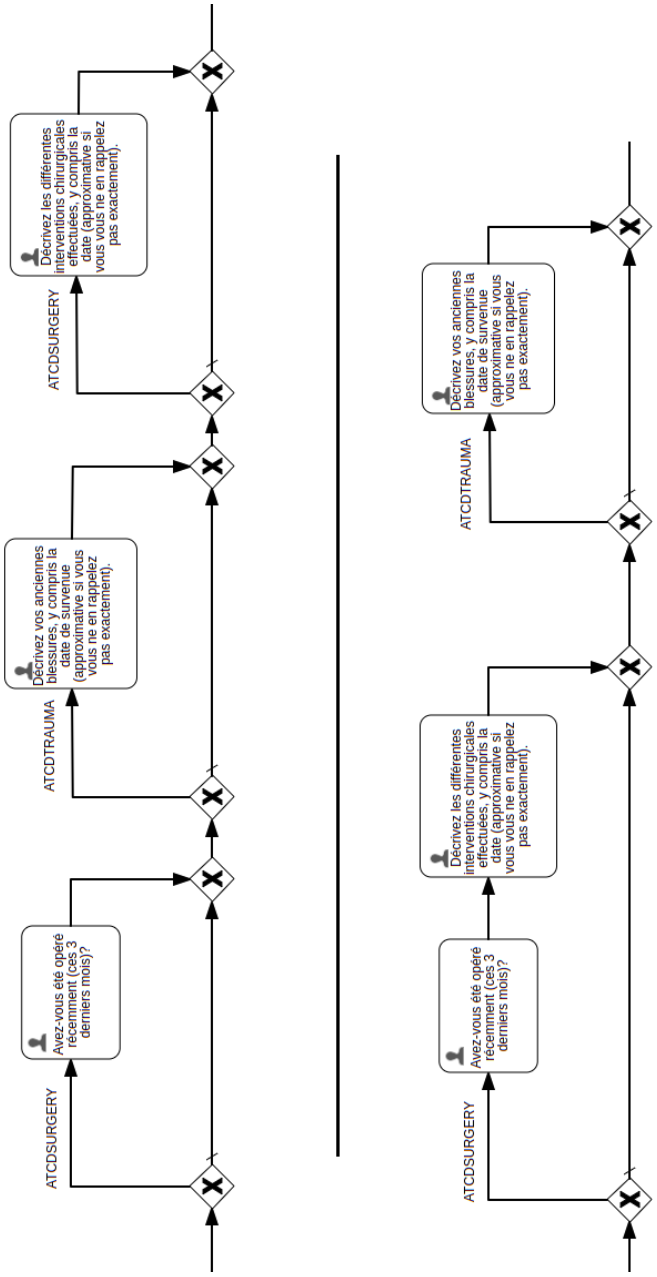

Figure 3.8: In the upper part of the picture, we can see that before the reordering, the question in the middle prevented a common branch for the left and right question.

3.6 Question implications

By construction, there are triggers that imply others. If these implications can be found, a more precise branching can be modeled. To give a first example, we can see in Figure 3.9 in the upper part the original result without exposing mutual implications. We can however intuitively think

that the trigger `NIGHTPAIN` implies `PAIN`, and identically for `MORNINGPAIN`. It can be verified in the triggers definition that it is indeed the case. The explanation of their implication will be developed further. We can safely assume that it is the case in this example. If this implication can become visible by adding `PAIN` to both conditions, the branching of the lower part of the picture will be generated. This branching can be considered as better than the upper one from the perspective of readability because it visually shows the relationship between the questions.

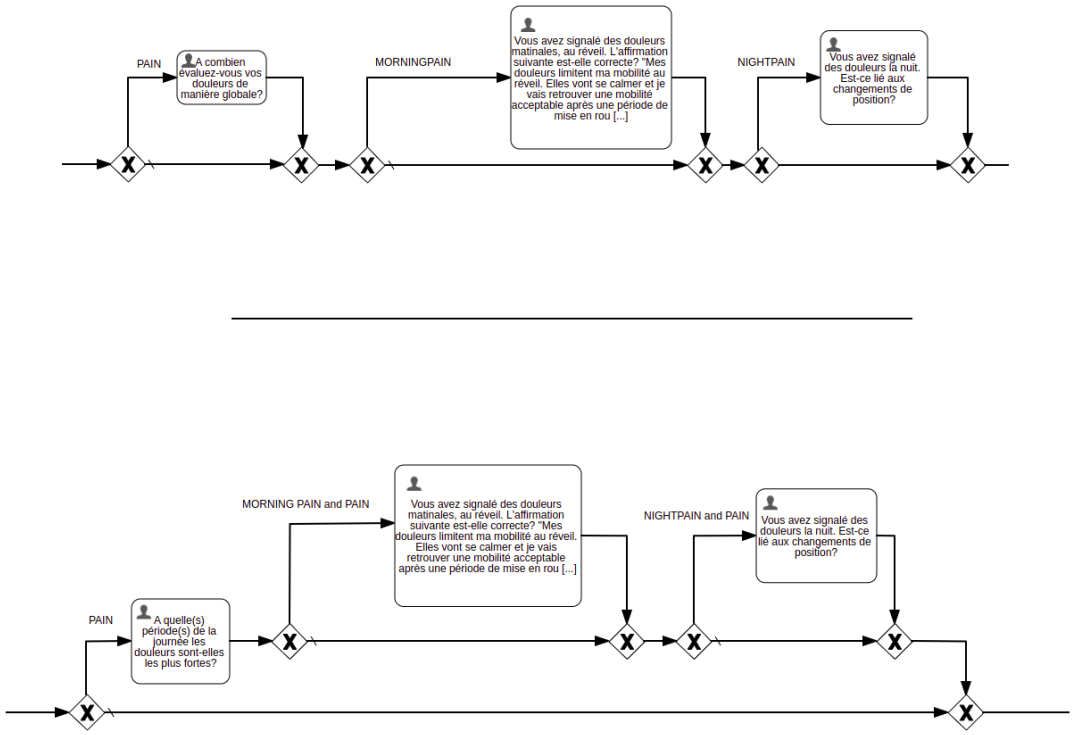

Figure 3.9: When the implications are used in the conditions, it leads to a better branching that shows the relationship between the questions. In the upper part the implications are not exploited in the branching structure, while in the lower part they are taken into account.

3.6.1 Core trigger implications

The first case of trigger implications relies on the core trigger definitions. For example, Figure 3.10 shows an extract of some answer options in the Diaana questionnaire editor. We can see the core triggers associated with each answer option. Each answer is associated with two core triggers. One can notice that every sport (skiing, swimming, fitness, yoga, ...) implies by design

the more general trigger named SPORT. Generally, some core trigger X implies a core trigger Y if every time X is triggered, Y is also triggered. To integrate this information in my branching, all X occurrences in the conditions were replaced by X and Y. By doing this, I make the relationship between X and Y visible and taken into account in the branching. We can see in Figure 3.11 that, because of the implication between the particular sports and the more general trigger that characterizes sports in general, we have a more informative branching that groups all the sports (SWIMMING, SKIING and RACKETSPORT) on a more general SPORT branch (the original conditions were SPORTS and SWIMMING, SPORTS and SKIING and SPORTS and RACKETSPORT but the SPORTS part has been removed for readability purposes).

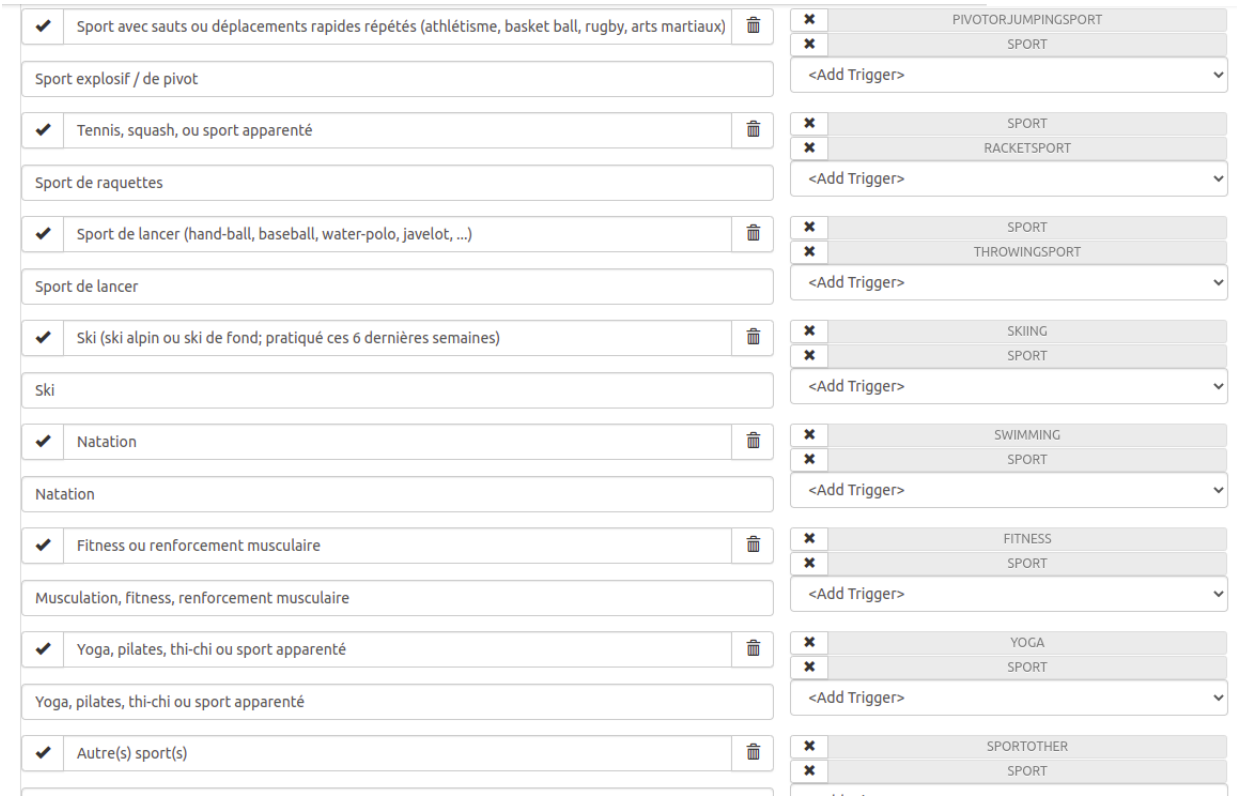

Figure 3.10: We can see that all particular sports imply the more general trigger called "SPORT".

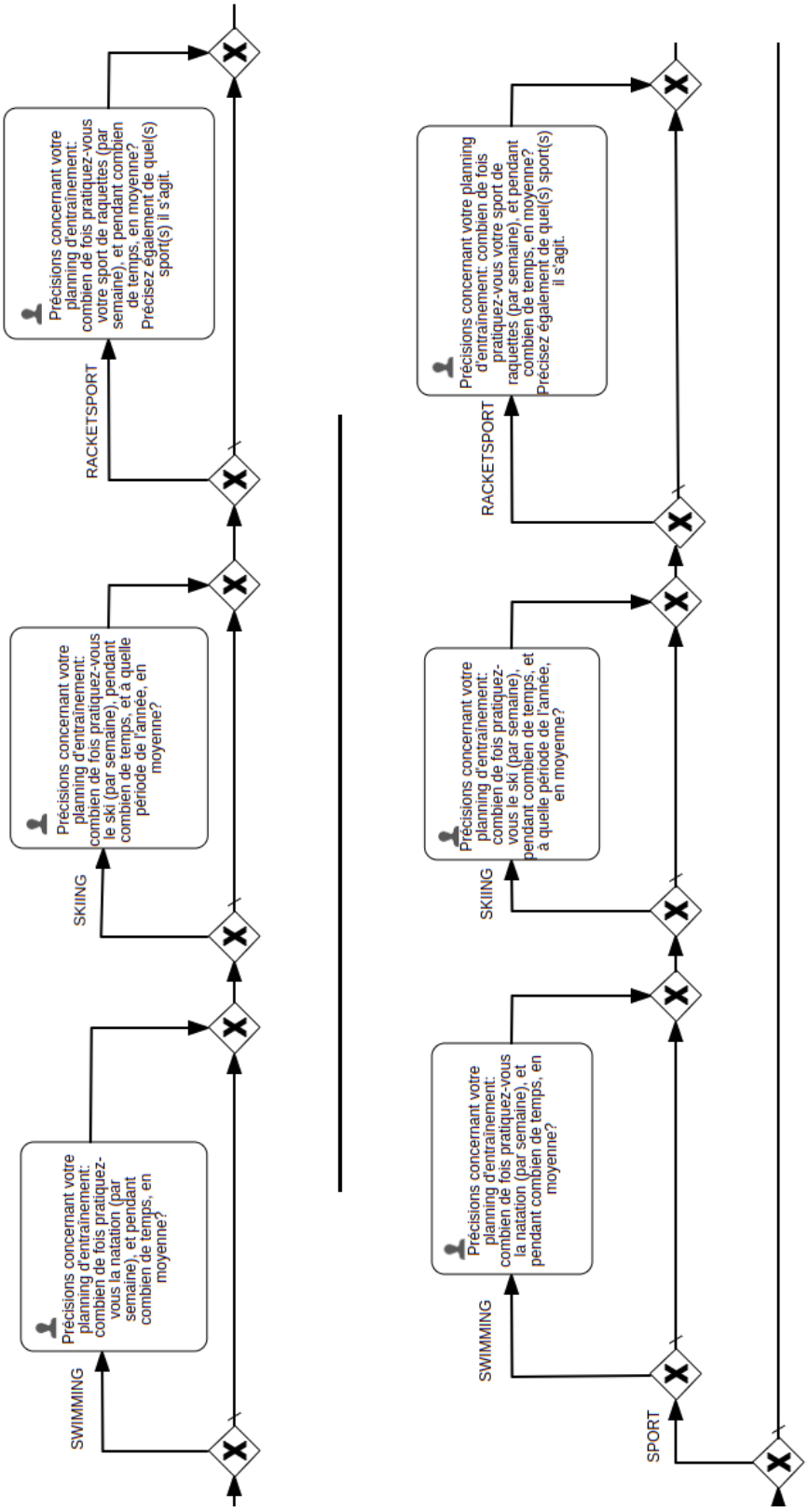

Figure 3.11: The implications between the particular sport and the more general sport triggers are used for a more informative branching (lower part of the figure).

3.6.2 Consequential triggers implication

I am going to illustrate the second case of implications with a concrete example. If a conditional question whose condition is `WOMAN` can potentially activate the trigger called `PREGNANCY`, and then the next question is conditional under `PREGNANCY`, it should be logical that the question whose condition is `PREGNANCY` is a subbranch of the `WOMAN` branch. By detecting this, The `PREGNANCY` condition can be changed to `PREGNANCY and WOMAN` to get the desired branching. We actually have such an example in Figure 3.12. It is indeed designed that way because the trigger `PAINTYPEOTHER` can only be activated upon answering the first question of this figure, which itself is conditioned under the `PAIN` trigger. The upper part of the figure does not take this into account while the lower part of the figure shows the result when this implication is taken into account by having a `PAINTYPEOTHER` subbranch from the `PAIN` branch. (Originally, the condition was `PAIN and PAINTYPEOTHER`, but `PAIN` has been removed from this condition because it was redundant.

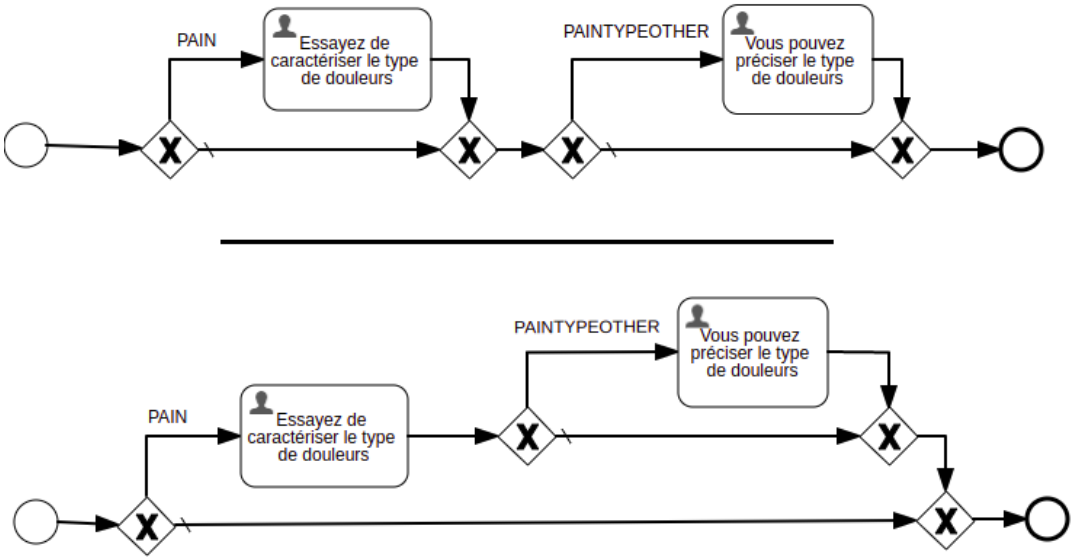

Figure 3.12: The fact that the question under the PAINTYPEOTHER condition is only displayed if the question under the PAIN condition is asked should be visible in the branch structure, which is not the case in the upper part of the picture. The lower part of the figure shows the expected result.

3.7 Remove condition redundancies

In the original Diaana format, there is no branching and each question see its condition evaluated individually to know if it is displayed. In BPMN, we take different paths (flows). When, for example, a subbranch is reached, we do not start from zero and already know

that if we arrived at this subbranch, the logical expression under which the parent branch is conditioned must be true. If conditions are not reworked, this leads to redundancies. They are necessary during the implementation of the features to see if what is implemented is correct, however in the final version, this should be removed for readability purposes. Not only does it add superfluous information, but also, similarly to non-factorized logical expressions, the diagram sees itself stretched out, and the information is less dense. We can see in Figure 3.13 in the upper part that the trigger `GLENOHUMERALLUXATIONLOCALIZATION` is repeated in the subbranches conditions which are: `GLENOHUMERALLUXATIONLOCALIZATION and (SHOULDERELEVATIONPAIN or PUSHINGPAIN)` and: `GLENOHUMERALLUXATIONLOCALIZATION and (not ISOLATEDACUTEANDRECENTRAUMA)`. It is not necessary in BPMN, since the fact that the subbranches are appended on the `GLENOHUMERALLUXATIONLOCALIZATION` branch already models the implication. In the lower part, we can see the same questions when the redundancies are removed. The diagram is compacter and is more pleasant to read.

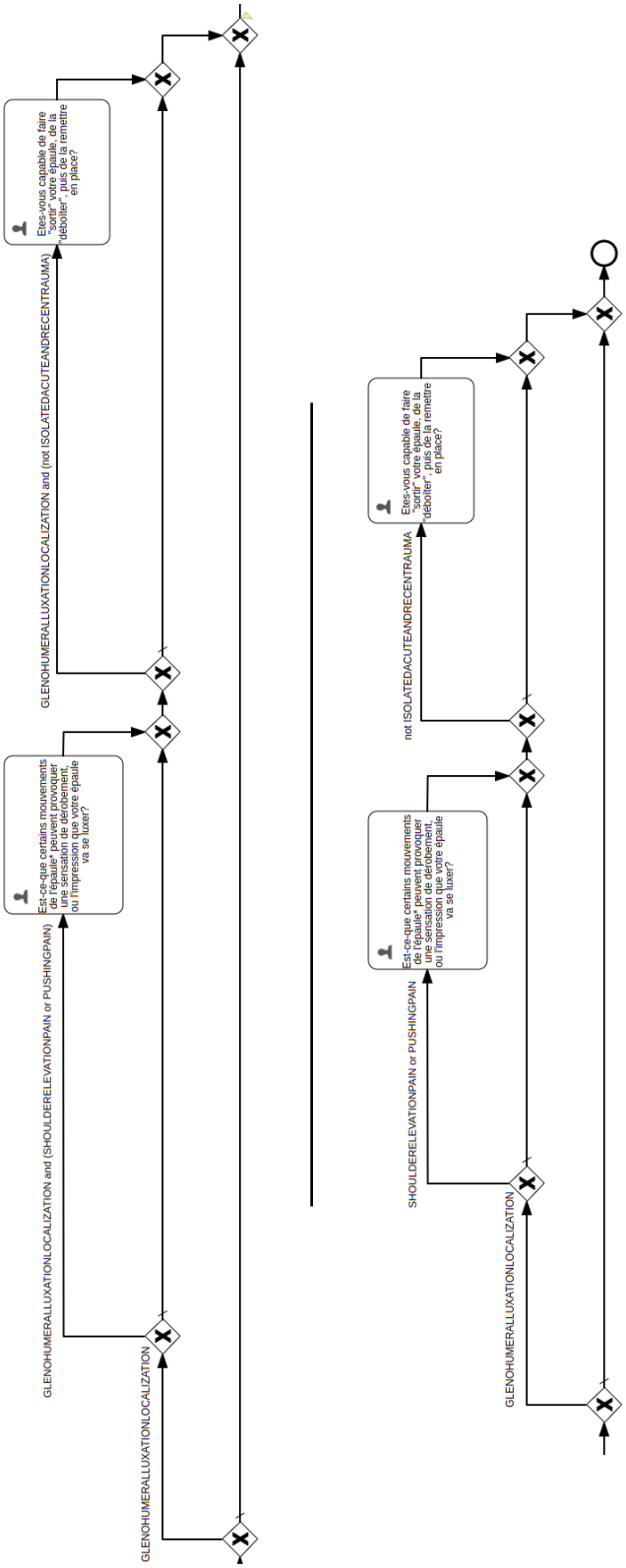

Figure 3.13: When the condition redundancies are removed as in the lower part of the figure, the diagram is compacter and more pleasant to read, especially when the removed trigger names are long, in comparison to the upper part of the figure.

Chapter 4

Implementation

4.1 Technical details

The implementation takes the form of a Python script which receives as an input the Diaana questionnaire in the form of a Json file. This Json file contains all the questions, the information about the sections, trigger definitions and everything else that is needed. Then, using this Json file, the script generates the necessary BPMN and form files. These files can then be imported into the Flowable Designer. There is one BPMN file by subprocess, which represents a section and a main BPMN file which links the sections together (see Figure 4.1 for the main file). The algorithm that generates the files works as follow: first, it stores all the questions from the current section in an array and makes a few modifications to this array. These modifications first concern the reordering of the questions. Then, to further improve the branching, empty questions with specific conditions are temporarily added to trigger the correct branching. Let us look at a concrete example: we can see in Figure 4.2 that the intersection of the condition `SMRIGHT and RIGHT` and the other one `IMRIGHT and RIGHT` is `RIGHT`. The intersection is not the condition of any of these two questions, so I decided to temporarily add an empty question called `TO_REMOVE` that has a condition which produce the desired branching, which in this case is `RIGHT`. We can see in this figure that this temporary question triggered the branch that supports both subbranches, highlighting their common part. (For explanation purposes, condition redundancies were not removed in this figure.)

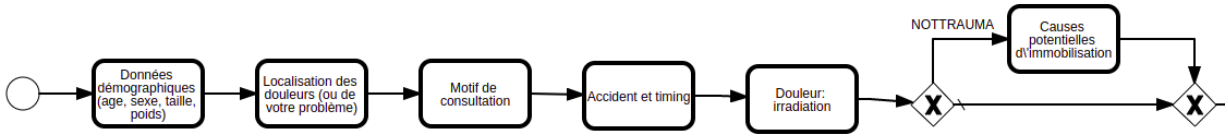

Figure 4.1: This is the main BPMN process. Each task is a subprocess which represents a section of questions from Diaana.

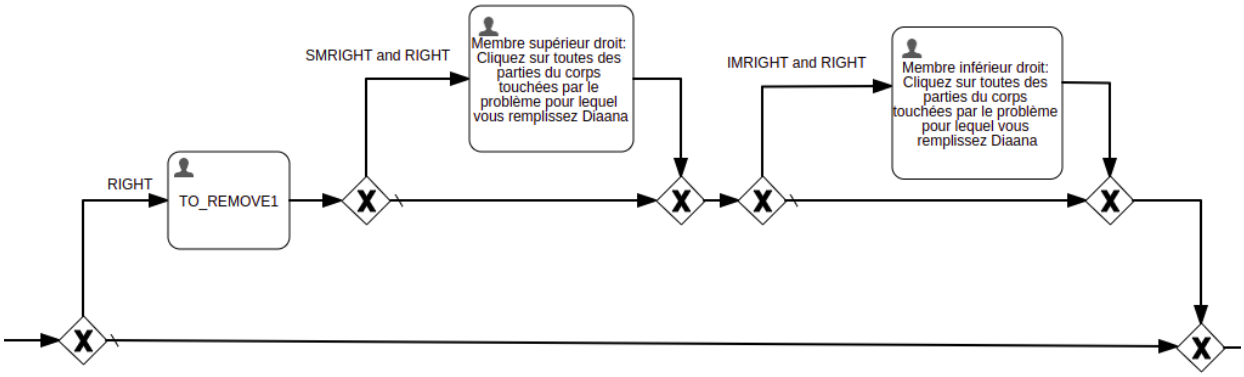

Figure 4.2: The temporary question TO_REMOVE1 has been added to have the script model the correct branching.

Then, every question is linearly appended (appended means that the BPMN element description has been written to the file, each question following each other). To append the question in the correct way, the most specific case to the most general is tested in order for the script to have the most appropriate behaviour. If no case has been found, the default one is simply to close all current open branches and open a new one for this question that cannot be related to any open

branches.

Flowable cannot process user tasks whose name is longer than 255 characters, so I had to ensure that such question labels were cropped (it is not a problem for the questionnaire as the question information is in the form file; The purpose of the user task name is to help the reader understand the diagram).

4.2 Reordering implementation

The reordering implementation is a greedy algorithm that consists of multiple passes. The first path gathers the questions with similar conditions together, and the next passes fine-tune the new ordering: they move the questions in order to find the neighbours with the largest condition intersection. To perform this, each question computes the score (i.e., the size of the intersection) of all the next questions. A question can stay if the best score is the question right behind, or move forward if the best choice is further. A question cannot go backwards in the array, i.e. lower in the index. I call a candidate all questions after the question we may move whose score is computed and the one right behind it. The scores are ordered in a decreasing way. The first (and higher) score is picked and we need to check if the move next to this question is possible. A candidate is only eligible if moving the question next to it does not lead to a violation. The following situation constitutes a violation: if between the question we desire to move and the candidate, there is a question that uses as a condition a trigger the question we want to move activates, then if it is moved past that problematic question, this particular question can never be displayed, because the trigger will always be tested before it can even be activated. If a candidate is not eligible, the problematic encountered question is pushed after the candidate if possible. If the problematic question cannot be moved due to its own constraints, the next candidate is tested to see if the score is high enough (the higher the pass number is, the higher the minimum accepted score is). All the questions of the array are traversed this way and the questions are moved in place. The final array contains the questions reordered. In some situations, the questions can keep moving around and the algorithm may not terminate. I added a condition that checks that if the same situation is encountered three times, the algorithm is stopped.

Let us look at a concrete example. This example will not contain the variable dependency constraint, meaning that each question can be moved at any place in the array. This example is only going to have one pass. The input array contains the conditions of the questions in the same order as their question. This array is $[A \wedge B, D, A, D \wedge E]$. It begins with condition $A \wedge B$. The scores (intersection size of their conditions) between A and all next elements are summarized in this array: [0, 1, 0]. It is the first condition so the previous neighbour score is not computed. The best non-zero score is 1, corresponding to the condition A. Thus, the condition $A \wedge B$ will be put after condition A (after because its size is longer, so a subbranch could be modeled). The new array is $[D, A, A \wedge B, D \wedge E]$. Then, condition D has the following potential scores: [0, 0, 1] (also no previous neighbour score since it is now the first element of the array). The best score is 1,

corresponding to the condition $D \wedge E$. Since its length is shorter than its future neighbour, it will be put before it. The resulting array is: $[A, A \wedge B, D, D \wedge E]$. A does not move because it is at its best position. $A \wedge B$ also does not move, then D and $D \wedge E$ don't move as well as they all found the best position. We have the reordered array of questions with similar conditions being grouped.

4.3 Condition factorization implementation

The condition factorization is implemented as a greedy recursive function. It takes as input a condition described in its disjunctive normal form. It will first try to see if there is a non-empty intersection from all terms in order to extract it. If not possible, it will try to find the largest term subset where there is a non-empty intersection. If such a case is found, the function will be run again recursively on the remaining terms and on the differences of the terms where an intersection has been found as a subset of them could lead to an inner factorization, and so on. Below is the pseudo code of the recursive factorization (the function being called `factorize`). The input is an array that itself contains arrays that represent the terms of the logical expression. Each of these arrays contains the variables of each term. This array is called `dnf_array`. The algorithm tries to find the largest non-empty intersection which explains why the `for` loop indexes are picked in a decreasing order.

```
factorize (dnf_array):
found = false;
for ( i = size(dnf_array) downto 2 )
    foreach ( combination of size i from dnf_array )
        if intersection of all terms of the combination ≠ ∅ then
            The intersection is factorized as a common factor;
            factorize(terms that were not in the combination);
            factorize(combination \ common factor)
            found = true;
            break;
        end
    end
    if found then
        break;
    end
end
```

The first `for` loop has an index of a minimum value of 2, because we need at least two terms to compute an intersection. The Boolean variable `false` is used to be able to stop both loops when an intersection is found. One additional thing to know is that all combinations are computed dynamically when accessed, meaning that only the needed combinations are computed. About

the worst case complexity: if our DNF input array has m terms, and that the maximum number of terms is n, the worst execution would be that we only find non-empty intersections of 2 terms. In this case, the first `for` loop would be fully executed each time. It means that we have $m* \binom{m}{i}$, that recursively calls $\binom{m-2}{i}$. We can thus approximate that the recursive calls take $m* \binom{m}{i}$, which leads to a total of $m * m* \binom{m}{i}$. To estimate i, I will take $\lfloor m/2 \rfloor$ as it yields the largest result. We have now $m * m* \binom{m}{m/2}$. By using Stirling's approximation, we get that $\binom{m}{m/2} \approx 2^m \frac{1}{\sqrt{m}}$. This yields in total a worst case complexity of O($m^2 \frac{2^m}{\sqrt{m}}$) = O($\sqrt{m} m 2^m$). Even though it is an exponential runtime, it has been considered as acceptable as the script does not need to be fast, and that m (the number of terms in the DNF expression) rarely exceeds 5 in practice as the conditions are designed by humans.

4.4 Form generation

The generation of the forms consists of generating one Json file for each question, which resulted in 243 Json form files. A Json form file describes the questions in the following way: the label (the question itself), the question type and the answer options when needed. The question types implemented are radio buttons (only one answer can be ticked among proposed answers), checkboxes (none, or all the options can be ticked), dates, numbers, and texts. We can see in Figure 2.1 the result of a generated form. This form is linked to a specific user task (question). We can see the question label of this form, the potential answer options and its type (radio button in this case).

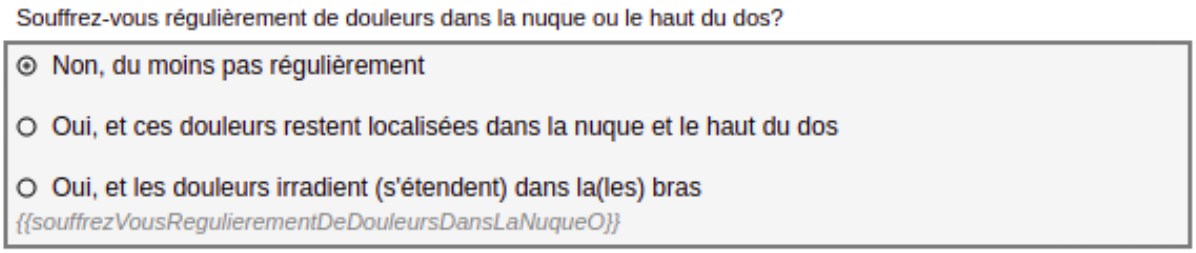

Figure 4.3: A form represents one question.

4.5 Triggers updating

In Diaana, the core and composed triggers are automatically updated after each question. In Flowable, the variable processing is not automatic. Moreover, setting the answer variables is not enough, as many triggers can be activated upon answering certain options. To update the

triggers, script tasks have been used. A script task allows updating as many triggers as needed. The downside of script tasks is that they appear in the same way as a user task (a question in our use case) and it hinders the readability. To make the update triggers tasks less inconvenient, there are only present when necessary. All core triggers have to be initialized to `false` at the beginning of the main process, then are set to `true` later if the answers trigger its activation. Then, updating triggers is also needed before a XOR gate. In this case, triggers have to be updated to ensure the correct path is taken. Secondly, at the end of a section, another script task is needed to update the core triggers that were not yet set because no XOR gate had been encountered since (in the case where the last questions of the section are unconditional). Finally, at the end of the main process, all composed triggers need to be updated, because some are not used in question conditions and thus were not updated previously. These composed triggers are useful information for the doctor and need to be retrieved.

Unlike in Diaana, for performance reasons, not all 732 triggers are updated each time a question is answered, but only the necessary ones needed at that particular part of the questionnaire. When a question is answered and its options trigger a core trigger, there is a script task after the user task to update the core triggers. Before a XOR gate, the composed triggers that are in the condition are updated. The minimal needed amount of triggers are updated when a question is answered for performance reasons.

We can see in Figure 4.4 how a script from a script task looks like. We can notice that the same variable called `lorsDesMouvementsOuEncoreDurantUneActivitePhysiqueLegere` is used to set two triggers named `MOBILIZATIONPAIN` and `PAININCREASEWITHSOFTEXERCISE`. We can see in Figure 4.5 how these script tasks are integrated among the questions. These script tasks are called "Update triggers". One can notice that for the two last questions on the right that are on the same subbranch (whose condition has been omitted for the readability of the figure), there is no script task in between them because no flow decision has to be made. For readability purposes it is better to put the least "Update triggers" script tasks not to obstruct the questions.

If a doctor or a collaborator wants to delete a core trigger, he should go to the "Initialize triggers" script tasks to remove it, and also in the "Update triggers" script tasks after the question where it is associated. If it is a composed trigger that should be deleted, the doctor should only delete the declaration lines before each time the composed trigger is used, and at the end of the main process where the last value of the composed triggers are fetched. One should be careful to modify or remove triggers that may be used in composed trigger definitions. If a core trigger is created, the option that is linked to this core trigger should be declared in the script task after the triggering question. A new composed trigger only needs to be updated before it is used in a condition, and at the end to be retrieved in its up to date value.

Script

```
var lorsDesMouvementsOuEncoreDurantUneActivitePhysiqueLegere = execution.getVariable("lorsDesMouvementsOuEncoreDurantUneActivitePhysiqueLegere");

if(lorsDesMouvementsOuEncoreDurantUneActivitePhysiqueLegere) {
 execution.setVariable("MOBILIZATIONPAIN", true);
 }



var lorsDesMouvementsOuEncoreDurantUneActivitePhysiqueLegere = execution.getVariable("lorsDesMouvementsOuEncoreDurantUneActivitePhysiqueLegere");

if(lorsDesMouvementsOuEncoreDurantUneActivitePhysiqueLegere) {
 execution.setVariable("PAININCREASEDWITHSOFTEXERCICE", true);
 }



var durantLesEffortsOuUneActivitePhysiquePlusSoutenue = execution.getVariable("durantLesEffortsOuUneActivitePhysiquePlusSoutenue");

if(durantLesEffortsOuUneActivitePhysiquePlusSoutenue) {
 execution.setVariable("MOBILIZATIONPAIN", true);
 }



var durantLesEffortsOuUneActivitePhysiquePlusSoutenue = execution.getVariable("durantLesEffortsOuUneActivitePhysiquePlusSoutenue");

if(durantLesEffortsOuUneActivitePhysiquePlusSoutenue) {
 execution.setVariable("PAININCREASEDWITHHARDEXERCICE", true);
```

Figure 4.4: A script task used to update triggers.

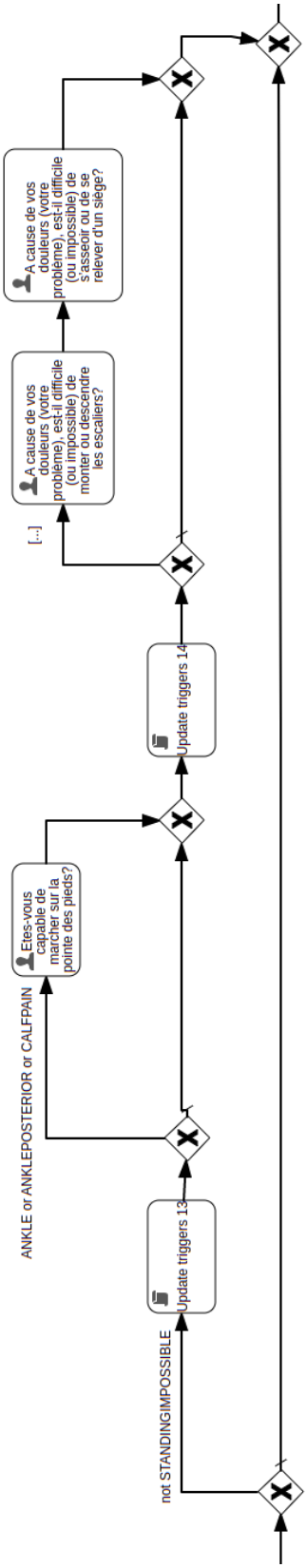

Figure 4.5: The script tasks called "Update triggers" can be seen before a XOR gate. They are necessary to update the needed triggers for the process to choose the correct path.

4.6 Runtime

The provided runtimes are an average of 20 measures, excluding outliers. The input questionnaire contains 243 questions and 17 sections. The program completes in approximately 3 minutes and 40 seconds. Without the reordering feature, it completes in 3 minutes and 28 seconds approximately. Without the reordering and the factorization of the composed triggers, it completes in less than a tenth of a second. The majority of the runtime is spent on the factorization of the composed triggers, and then on the reordering. More precisely, the factorization of one condition with 25 terms leads to this latency. This is due to the implementation of the factorization feature which tries to find non-empty intersections of a subset of the condition terms. The aim is to factorize parts of the condition when a global common factor extraction is not possible. The part of the algorithm which finds potential non-empty intersections within a term subset tries all possibilities and stops when one is found. This explains why the runtime can be lengthened when conditions with lots of combination are analyzed.

Because the program has to linearly go through all the questions multiple times (to reorder them, add the additional questions to trigger nice branching, and then to add them to the final file result), it is expected that in average, a Diaana questionnaire with more questions will have a longer runtime. However, as previously explained, some features spend more time computing than others (the factorization of the conditions being the most time-consuming one, with the reordering feature coming second). It means that an input questionnaire with 10 questions could have the same runtime, or even a longer one than another input questionnaire of 20 questions for example: the execution duration is highly variable according to the input questionnaire, and more precisely to the conditions under which these questions are displayed or not, since it is the factorization operation that consumes the majority of the runtime. The runtime is thus also bounded by $O(\sqrt{m}m2^m)$.

Chapter 5

Results

As seen throughout the Design section, the features work as intended and each of them contributes to the readability improvement. Soignez-moi needed to have one questionnaire translated. This questionnaire is constituted of 18 sections, which can be considered as sub-questionnaires. I am referring to them as questionnaires. These are 18 questionnaires that have been translated. 14 questionnaires have an improved readability. The four remaining questionnaires could not be optimized. Two of them only had a single question, meaning that no relationship between questions could be exposed since there was only one question. The questions of the third questionnaire were all unconditional. This means that the branching structure could not show any link between questions because all of them have no condition and should be asked in all cases. This can be observed in Figure 5.1 where the two questions have no conditions. There is in this case only one possible flow that should go through these questions. The last questionnaire that could not be improved has questions whose conditions are unrelated.

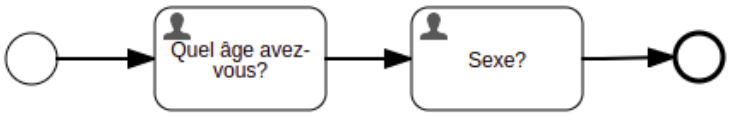

Figure 5.1: This questionnaire could not be improved because all its questions are unconditional.

Let us look at some of the generated BPMN processes. We can see in Figure 5.3 the result of some combined features, among them mutually exclusive questions putting in parallel the branches `NOTTRAUMA` and `not NOTTRAUMA`. On the `not NOTTRAUMA` branch, there is another mutual exclusivity case with two questions whose conditions are `THORAX` and `(not ARTHRITISORMUSCLEDOSEASE) and (not THORAX)`. The common parts of the conditions

(`NOTTRAUMA` for the upper branch and `not NOTTRAUMA` for the lower branch) have been exploited to create subbranches. The removal of the condition redundancies yield a compacter diagram. Figure 5.4 also presents a translated section with branches, subbranches and sub-subbranches, along with Figure 5.5. Figure 5.6 is an additional example where we can see a global view of a translated section. We can also notice branches, subbranches and sub-subbranches. As stated in the Design section, there is a slight visibility loss in the usable version. The usable version differs because it has some "technical tasks" which are script tasks to update the triggers. Figure 5.2 shows the comparison between the non-technical version of a diagram (upper part) and its technical counterpart (lower part). We can see the first task that is necessary to initialize the core triggers and the script tasks to update the needed triggers before each XOR gate. It is less compact than the purely visual version, but these scrips are needed to run the processes.

It is also worth mentioning that some sections have few questions whose conditions can be linked. If there are a certain number of unrelated questions, the produced BPMN diagram will have portions that only present separate branches for each question following each other. The produced BPMN results highly depend on the input forms, and more precisely on the question conditions.

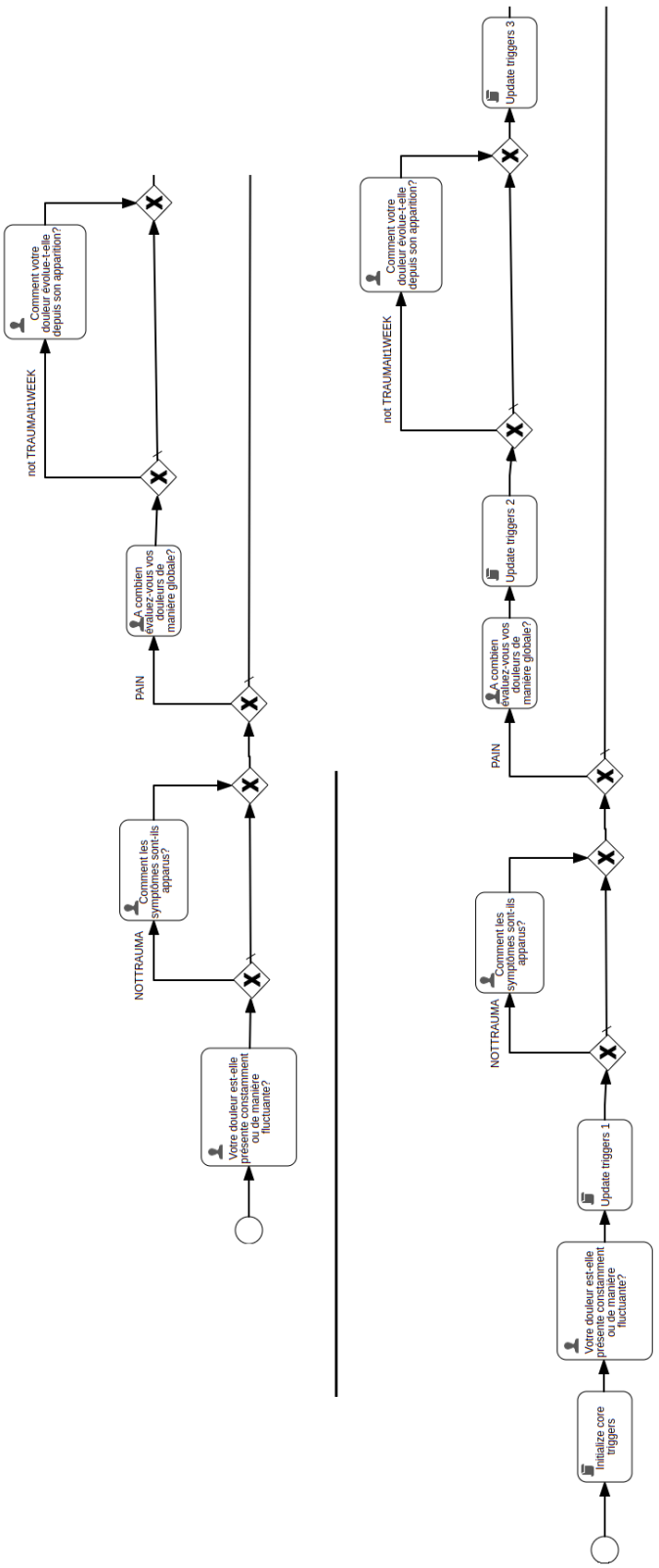

Figure 5.2: In the upper part of the figure we can see the diagram without the technical tasks, and in the lower part the graph with the additional technical tasks to update the trigger variables.

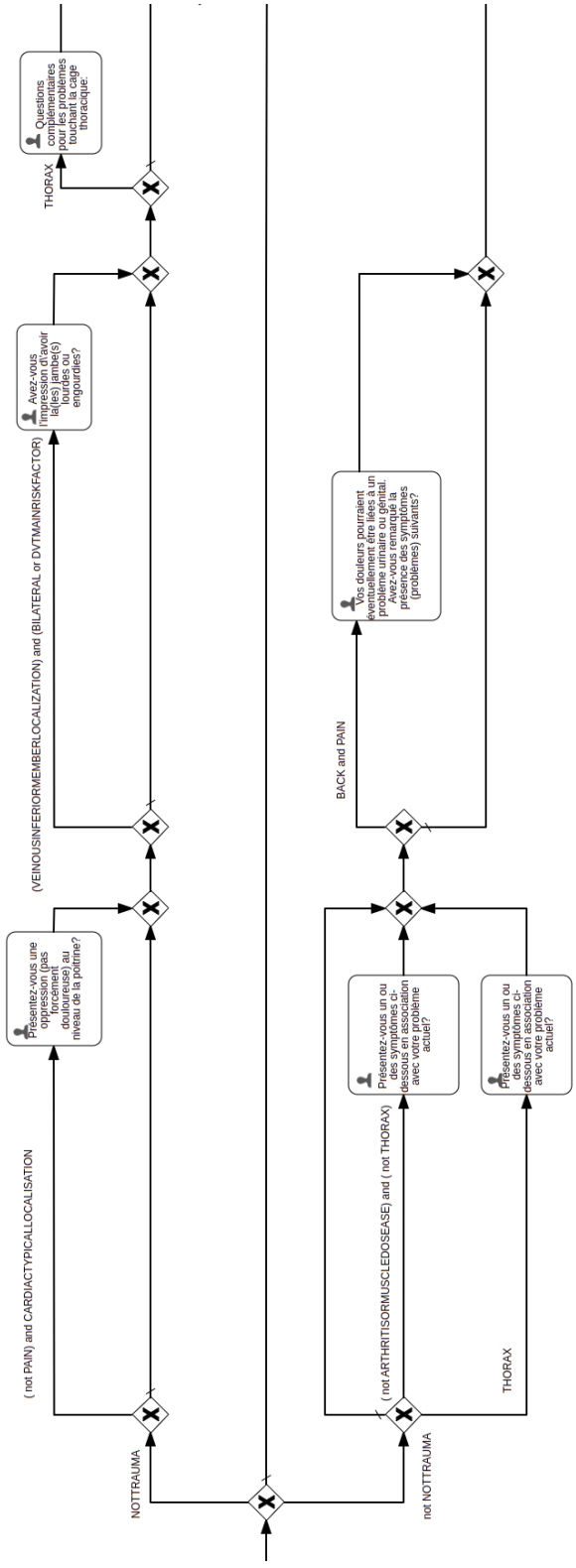

Figure 5.3: The combined features led to results that improved the readability of the section named "Symptômes (problèmes) associés".

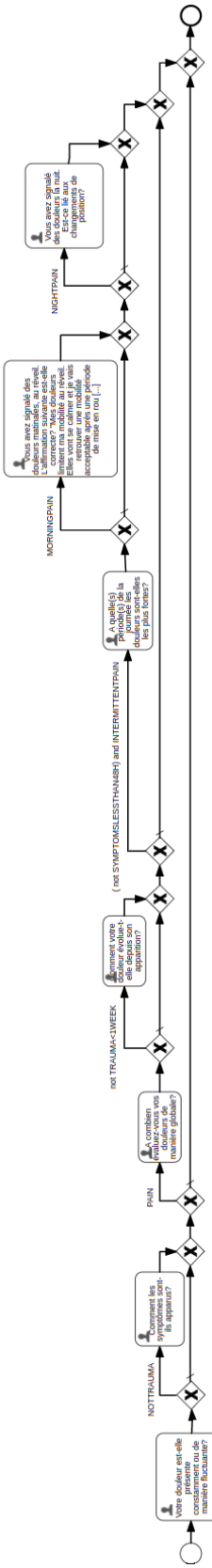

Figure 5.4: A global view of the section named "Douleur (problème): temporalité.

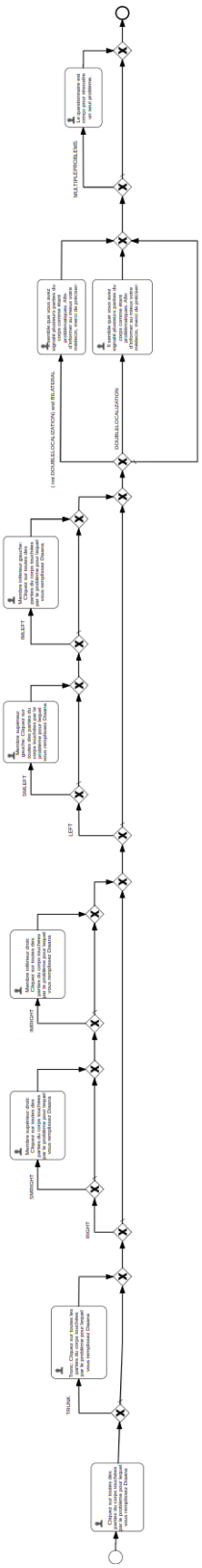

Figure 5.5: A global view of the section named "Localisation des douleurs (ou de votre problème)"

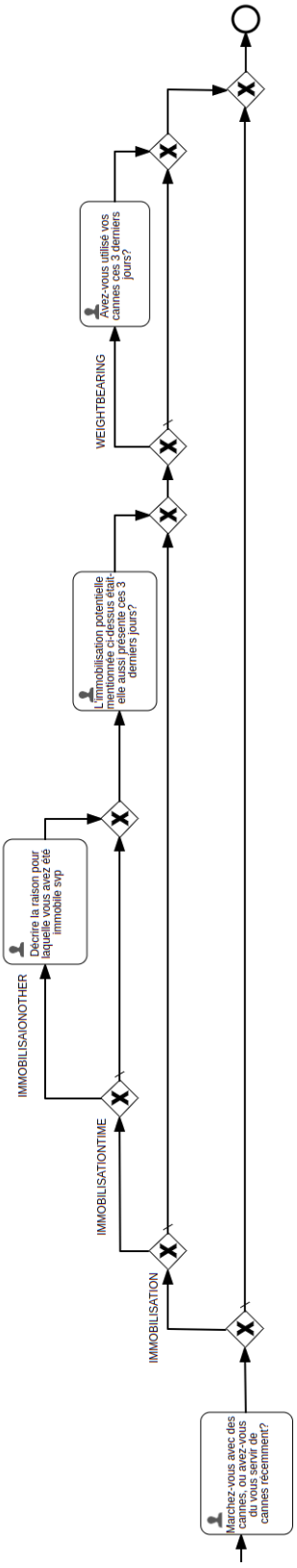

Figure 5.6: A global view of the section named "Causes potentielles d'immobilisation"

There is an aspect of the script that could have been improved for the user experience: when importing the files, since the script only describes the relationship between the BPMN elements, it is the task of the Flowable designer to actually place them correctly on the plane. This does not provide satisfying results in terms of visualization and some manual work is needed to have a proper diagram. The diagrams are sequential and follow a horizontal path, which means that desired element coordinates could have been computed and added to the files such that there is no manual work required to space out the BPMN elements after the importation. Also, the size of the squares of the user tasks, which hold the question texts, does not adapt itself to the size of the text. This leads to user tasks with texts that overflow the square in which it should be contained, which is not visually pleasant. This also has to be manually handled.

Another downside of the translation is that the core triggers cannot be defined next to the options that trigger them in the form editor, which is practical in Diaana. Let us look at Figure 5.7: we can see in this figure that in Diaana (uppert part), on the right of each option, there is the list of triggers associated to it, if any. In this case, when the patient answers "Oui" to the question "Avez-vous déjà eu un problème de genou par le passé?", the trigger `KNEEATCD` is activated. In BPMN (lower part), we have the BPMN diagram. The question is represented as a user task. If we take a look at the form linked to this task (arrow going from the question), we can see the detailed form. We can see the question, the possible answers, and the variable name which is the question text in its camel case version. There is not the possibility to associate triggers to the answers at this place. One should look at the next script task, in this case it is called `Update triggers 19`. The script of this task defines this trigger by retrieving the answer variable and testing its value. While it is not shown in this figure, the trigger `KNEEATCD` has to be initialized to `false` at the beginning of the process.

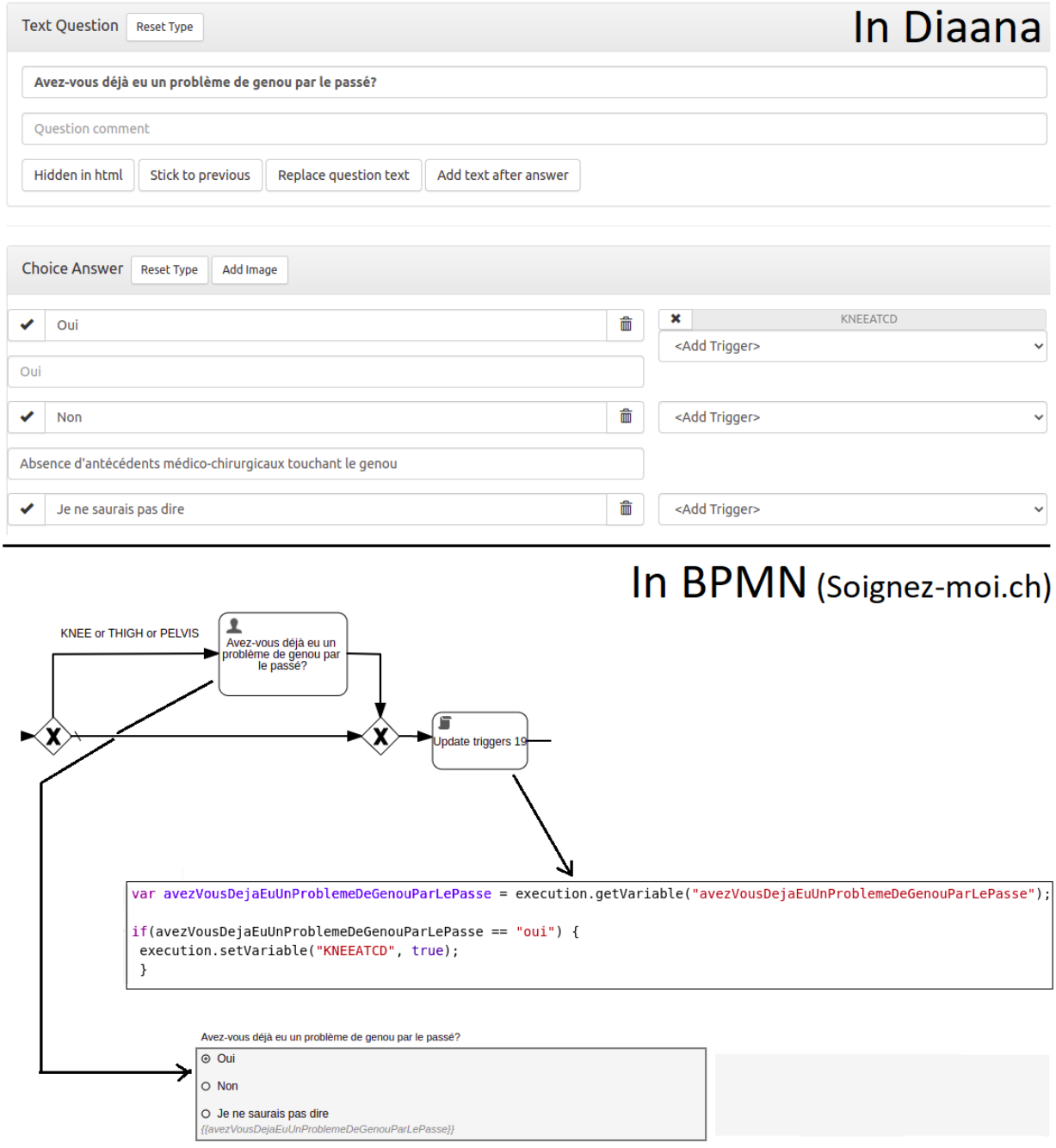

Figure 5.7: How core triggers are defind in Diaana in comparison to how they are defined in BPMN

Chapter 6

Related Work

In [2], Farah Ben-Naoum and Christophe Godin describe a method to perform height compression of unordered trees. The method has a roughly similar high level idea, namely to identify similar patterns and combine them to avoid repeating useless information. This result in unordered trees being translated into directed acyclic graphs that contain the same information, but in a more efficient and compact way. Even though the high level concepts are somewhat similar, it cannot be applied to my use case as the input are trees, and the output directed acyclic graphs and not BPMN processes. Similarly, in [3], Shenfeng Chen and John H. Reif present an approach to perform an efficient lossless compression of Trees and Graphs which generalizes the same kind of ideas. However, the aim of this paper is to compress the graphs for memory purposes, not for readability. In that sense, Fang Zhou ([8]) applies the compression to help understand the essential information the input graphs contain and improve visualization for data analysis for example. In the domain of height compression, in [1], a compression of trees via linear directed acyclic graphs is performed using BFS. The aim of this compression is mainly to meet the demand of storage and transfer of large size data, which is also not related to my use case. In [4], the authors study how to have semi-automated modelling of optimized BPMN processes. The goal is also to help users modelling the most optimized processes through various intermediate outputs expressed as simplified graphs to show the most important features of the process. This helper is not based on the logical expressions of the flows as I did. Finally, in [7], BPMN processes optimization by workflow analysis is studied. The aim and method is similar to the previous cited work. It is also directed towards non-experts in BPMN to assist them during the process modeling. In both of these papers, it is in a more general BPMN use to design correct processes from scratch and is not a translation from another format as it is in my case.

About Boolean expressions, Denis Ponomaryov presents a polynomial-time algorithm to decompose a DNF expression into a conjunction of DNFs in [6]. It is explained that this decomposition facilitates finding a more compact representation of the Boolean function. In that sense, it is similar to my problem. I factorized DNF expressions to get a more compact representation as well. However, this algorithm outputs a conjunction of DNFs, which does not apply in my case.

In the article [5], the authors present methods of decomposition of Boolean functions in their conjunctive normal form. One of them consists of representing the CNF expression as a hypergraph. It could have been interesting if applied to DNF, but I opted for a more straight-forward recursive algorithm.

Chapter 7

Conclusion

The combined ideas presented provided a result that showed improvements in comparison to the direct translation: the relationship between the questions have been highlighted to deliver a better result compared to the default single chain, where each question is evaluated individually, regardless of its implications with other questions. By first performing the factorization of the conditions, the removal of the condition redundancies, and then by taking into account their condition intersections, implications and their mutual exclusivity along with the reordering, the readability has been improved: the doctors and collaborators can visually see the relationships of the questions by looking at the branch structure of the diagram. Over 18 questionnaires, 14 of them have been optimized from a readability point of view. The four ones left remained the same because the first two only had one question, the third one only had unconditional questions and the last one had questions whose conditions were unrelated.

Bibliography

[1] Romain Azaïs, Jean-Baptiste Durand, and Christophe Godin. “Lossy compression of trees via linear directed acyclic graphs”. In: *HAL Open science* (Mar. 2016). URL: https://hal.archives-ouvertes.fr/hal-01294013v1/file/paper.pdf.

[2] Farah Ben-Naoum and Christophe Godin. “Algorithmic height compression of unordered trees”. In: *Journal of Theoretical Biology* 389 (2016), pp. 237–252. ISSN: 0022-5193. DOI: https://doi.org/10.1016/j.jtbi.2015.10.030. URL: https://www.sciencedirect.com/science/article/pii/S0022519315005299.

[3] Shenfeng Chen and John H. Reif. “Efficient Lossless Compression of Trees and Graphs”. In: *Proceedings of Data Compression Conference - DCC ’96* (1996), pp. 428–. DOI: 10.1109/DCC.1996.488356. URL: https://users.cs.duke.edu/~reif/paper/chen/graph/graph.pdf.

[4] Yliès Falcone, Gwen Salaün, and Ahang Zuo. “Semi-automated Modelling of Optimized BPMN Processes”. In: *SCC 2021 - IEEE International Conference on Services Computing* (Oct. 2021). URL: https://hal.inria.fr/hal-03330330/document.

[5] N I Gdanskiy, A A Denisov, and M L Rysin. “Methods of decomposition of Boolean functions”. In: *IOP Conference Series: Materials Science and Engineering* 862.4 (May 2020), p. 042026. DOI: 10.1088/1757-899x/862/4/042026. URL: https://doi.org/10.1088/1757-899x/862/4/042026.

[6] Denis K. Ponomaryov. “A Polynomial Time Delta-Decomposition Algorithm for Positive DNFs”. In: *CoRR* (2019). URL: https://arxiv.org/pdf/1805.08177.pdf.

[7] Ahmad Shraideh, Hervé Camus, and Pascal Yim. “Business process optimization by workflow analyzis”. In: *Proceedings of the ISCA 22nd International Conference on Computer Applications in Industry and Engineering* (Jan. 2009). URL: https://www.academia.edu/20683572/Business_Process_Optimization_by_Workflow_Analysis.

[8] Fang Zhou. “Graph compression”. In: *Encyclopedia of Big Data Technologies* (2019). URL: https://www.cs.helsinki.fi/u/htoivone/teaching/seminarS10/reports/zhou-graph-compression-v2.pdf.