A hybrid process modeling approach

Master’s thesis
Vivian C.T. Hermans
Vivian C.T. Hermans

s081059

April 2014

Graduation Committee:

prof. dr. ir. H.A. Reijers
Chairman
Professor in Business Process Technologies
Department of Computer Science & Mathematics
Section Architecture of Information Systems
Eindhoven University of Technology (TU\e)

dr. ir. I.T.P. Vanderfeesten
Member
Assistant Professor in Information Systems
Department of Industrial Engineering & Innovation Sciences
Section Information Systems
Eindhoven University of Technology (TU\e)

dr. M. Zimakova
Member
Software Engineer
Research & Development Group
Perceptive Software
Automating business processes has been a popular business strategy in the past few years. These processes were modeled using an imperative modeling language. A known issue of this way of modeling is that it is difficult to model flexible parts of processes, since every sequence needs to be modeled explicitly. Flexible parts or parts that need user input of a process need to be made structural in order to be able to automate them. On the other hand, with a declarative modeling language it is not convenient to model structured parts of a process. A modeling approach that is able to model both the structured and flexible parts does not exist.

This thesis presents a hybrid process modeling approach that addresses this issue. The hybrid modeling language consists of a combination of the imperative modeling language BPMN and declarative modeling language Declare. It allows the user to jointly model flexible and structured parts of a process. A method is developed to create a hybrid process from a process description and guidelines are created to help the user decide between modeling a group of activities using the imperative or declarative part of the hybrid modeling language. An empirical experiment is conducted to evaluate the set of guidelines. The analysis of the results indicates that these guidelines are found useful, but the ‘ease-of-use’ aspect needs improvement.
This thesis is the result of almost seven months of research performed as the conclusion of the Master’s program Business Information Systems at the Eindhoven University of Technology. The research was partly performed at the company Perceptive Software located in Apeldoorn. I am grateful to numerous people who have helped me during this time.

First off, I would like to thank my supervisor Hajo Reijers (TU/e) for guiding me throughout the project. Our (weekly) meetings in the car on our way to Apeldoorn, often made me to think of other directions to explore. It was nice to brainstorm about the project but also to talk about all kind of things. Your enthusiasm and knowledge about the research topic, inspired me to work hard again especially at times that I was stuck.

I would like to thank my supervisor Maria Zimakova (Perceptive Software) for supporting me during this time. I appreciate the time you’ve freed up to answer my questions and give me advice on different topics of the research.

I would like to thank Irene Vanderfeesten (TU/e) for letting me conduct my experiment in one of her lectures. Furthermore did she help me setting up the experiment and indicating the attention points.

Furthermore, I would like to thank everyone who invested time in giving me feedback. Whether that was on my thesis or performing the (pre-test of the) experiment. Thank you all for your dedication and time.

I would like to thank Perceptive Software to give me the opportunity of performing this research at their office in Apeldoorn. I would like to thank the employees of Perceptive Software for making me feel welcome and allowing me to be a part of them. You made it worthwhile for me to travel four hours a day to go from Eindhoven to Apeldoorn and back. I enjoyed the lovely lunches and wonderful Wednesday-walks.

Finally, I would like to thank my family, boyfriend and friends for supporting me during the years I studied and for making it a wonderful time that I will never forget.

Vivian Hermans
Eindhoven, March 2014
# Contents

1. **Introduction**  
1.1 Problem indication ........................................... 1  
1.2 Problem statement ......................................... 2  
1.3 Scope ..................................................................... 2  
1.4 Research method & thesis outline ......................... 3  

2. **Process modeling state of the art**  
2.1 Models and applications ....................................... 5  
2.2 Imperative modeling languages ............................... 6  
2.3 Declarative modeling languages ............................. 8  

3. **Requirements analysis**  
3.1 Requirements ................................................... 14  
3.2 Modeling language analysis .................................. 15  

4. **Hybrid modeling language**  
4.1 New modeling element: Block ............................... 17  
4.2 Declarative block in BPMN .................................... 18  
4.3 Imperative block in Declare ................................. 19  
4.4 Implementation .................................................. 20  

5. **Method and Guidelines**  
5.1 Case study ...................................................... 21  
5.2 Method to create hybrid process model .................. 22  
5.3 Define groups .................................................... 23  
5.4 Define blocks .................................................... 25  
5.5 Define relations between blocks ......................... 29  
5.6 Define relations between activities .................... 32  

6. **Empirical evaluation**  
6.1 Experimental definition and planning ................. 39  
6.2 Experimental operation ..................................... 42  
6.3 Data analysis ..................................................... 42
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>The regulative cycle</td>
<td>3</td>
</tr>
<tr>
<td>2.1</td>
<td>Overview of the basic BPMN elements</td>
<td>6</td>
</tr>
<tr>
<td>2.2</td>
<td>An example of a BPMN process model</td>
<td>7</td>
</tr>
<tr>
<td>2.3</td>
<td>An EPC model</td>
<td>8</td>
</tr>
<tr>
<td>2.4</td>
<td>A PetriNet model</td>
<td>8</td>
</tr>
<tr>
<td>2.5</td>
<td>A BPEL model</td>
<td>8</td>
</tr>
<tr>
<td>2.6</td>
<td>The Activity</td>
<td>9</td>
</tr>
<tr>
<td>2.7</td>
<td>The Init Activity</td>
<td>9</td>
</tr>
<tr>
<td>2.8</td>
<td>The Last Activity</td>
<td>9</td>
</tr>
<tr>
<td>2.9</td>
<td>The Exactly Constraint</td>
<td>9</td>
</tr>
<tr>
<td>2.10</td>
<td>The Existence Constraint</td>
<td>9</td>
</tr>
<tr>
<td>2.11</td>
<td>The Absence Constraint</td>
<td>9</td>
</tr>
<tr>
<td>2.12</td>
<td>The responded and co-existence constraints</td>
<td>10</td>
</tr>
<tr>
<td>2.13</td>
<td>Three precedence constraints</td>
<td>10</td>
</tr>
<tr>
<td>2.14</td>
<td>Three response constraints</td>
<td>10</td>
</tr>
<tr>
<td>2.15</td>
<td>Three succession constraints</td>
<td>10</td>
</tr>
<tr>
<td>2.16</td>
<td>Two models displaying the precedence and response constraint</td>
<td>11</td>
</tr>
<tr>
<td>2.17</td>
<td>Three models displaying the precedence, alternate precedence and chain precedence constraint</td>
<td>11</td>
</tr>
<tr>
<td>2.18</td>
<td>Three negation constraints</td>
<td>12</td>
</tr>
<tr>
<td>2.19</td>
<td>The two choice constraints</td>
<td>12</td>
</tr>
<tr>
<td>2.20</td>
<td>An example of a DCR Graphs model</td>
<td>13</td>
</tr>
<tr>
<td>4.1</td>
<td>Example of a block structured process</td>
<td>18</td>
</tr>
<tr>
<td>4.2</td>
<td>Collapsed Declarative Block</td>
<td>18</td>
</tr>
<tr>
<td>4.3</td>
<td>Expanded Declarative Block</td>
<td>18</td>
</tr>
<tr>
<td>4.4</td>
<td>A declarative block in a BPMN-process</td>
<td>19</td>
</tr>
<tr>
<td>4.5</td>
<td>Collapsed Imperative Block</td>
<td>19</td>
</tr>
<tr>
<td>4.6</td>
<td>Expanded Imperative Block</td>
<td>19</td>
</tr>
<tr>
<td>4.7</td>
<td>Procedural Sub-Process in Declare</td>
<td>20</td>
</tr>
<tr>
<td>5.1</td>
<td>Iterative process of creating a hybrid process model</td>
<td>22</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

5.2 Example of defining groups ......................................................... 23
5.3 The division of the activities into groups ........................................ 25
5.4 Imperative (a) and declarative (b) way of modeling the sequence A, maximum of 3 B’s, C ......................................................... 26
5.5 Imperative and declarative way of modeling ...................................... 27
5.6 Imperative and declarative way of modeling ...................................... 28
5.7 The first attempt to divide of the activities into blocks ......................... 29
5.8 An example of two blocks which need to be combined ......................... 31
5.9 The final distribution of the activities over the blocks ......................... 32
5.10 Two models including the init and last activity .................................. 33
5.11 Two possible models using precedence and exactly constraints ............. 34
5.12 Two possible models using the exactly and absence constraints .......... 35
5.13 Two possible models using ‘not (chain) succession’ constraints .......... 35
5.14 Two different models including the not co-existence constraint .......... 35
5.15 The three different alternate constraints ........................................ 36
5.16 The model of the imperative block ............................................. 37
5.17 The final hybrid model ............................................................ 38

6.1 Variables tested in the experiment .................................................. 39
6.2 Process description task ............................................................. 40
6.3 Structure of the experiment .......................................................... 42
6.4 CPN Tools implementation vs. BPMN implementation ....................... 50

A.1 Imperative model ................................................................. 55
A.2 Declarative model ................................................................. 56
A.3 Legend of the constraints of A.2 ................................................. 56
A.4 Imperative main process with a declarative sub-process ....................... 58
A.5 Declarative main process with an imperative sub-process ................... 59
A.6 Imperative main process with an imperative and declarative sub-process 59

E.1 Old en new version of the init and last activities ............................... 72
E.2 Old and new version of the existence constraints ............................ 73
E.3 Old and new version of the not-constraints .................................... 73
E.4 Old and new version of the chain-constraints .................................. 73
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Possibilities of modeling different blocks</td>
<td>30</td>
</tr>
<tr>
<td>6.1</td>
<td>Modeling experience</td>
<td>43</td>
</tr>
<tr>
<td>6.2</td>
<td>Guidelines vs. No Guidelines</td>
<td>44</td>
</tr>
<tr>
<td>6.3</td>
<td>Level of difficulty</td>
<td>45</td>
</tr>
<tr>
<td>6.4</td>
<td>Level of difficulty grouped by correct and incorrect answers no guidelines</td>
<td>45</td>
</tr>
<tr>
<td>6.5</td>
<td>Level of difficulty grouped by correct and incorrect answers guidelines</td>
<td>46</td>
</tr>
<tr>
<td>6.6</td>
<td>Usage of intended guideline</td>
<td>46</td>
</tr>
<tr>
<td>6.7</td>
<td>Correctness vs. Usage of intended guideline</td>
<td>46</td>
</tr>
<tr>
<td>6.8</td>
<td>Group ‘&lt; 8 # None’s’ vs. group ‘≥ 8 # None’s’ correctness</td>
<td>47</td>
</tr>
<tr>
<td>6.9</td>
<td>Group ‘&lt; 8 # None’s’ vs. group ‘≥ 8 # None’s’ difficulty</td>
<td>48</td>
</tr>
<tr>
<td>6.10</td>
<td>Usefulness guidelines</td>
<td>48</td>
</tr>
<tr>
<td>6.11</td>
<td>Clearness process descriptions</td>
<td>49</td>
</tr>
<tr>
<td>B.1</td>
<td>Most used constraints of Declare</td>
<td>67</td>
</tr>
<tr>
<td>D.1</td>
<td>P-values of the level of difficulty values compared to the neutral value 4</td>
<td>70</td>
</tr>
<tr>
<td>D.2</td>
<td>P-values of the level of difficulty values compared to the neutral value 4 sorted on correctness (no guidelines)</td>
<td>70</td>
</tr>
<tr>
<td>D.3</td>
<td>P-values of the level of difficulty values compared to the neutral value 4 sorted on correctness (guidelines)</td>
<td>71</td>
</tr>
</tbody>
</table>
1.1 Problem indication

Automating business processes has been a popular business strategy in the past few years. Business process models were developed to gain insights in the business processes before they were automated. In general, these process models were mainly developed using an imperative (or procedural) modeling language. This type of modeling language excels in creating (highly) structured process models. On the other hand, over-specification can arise when the processes are too flexible or need user input. Over-specification means that the process model becomes too restricted in comparison with the actual process. The flexible parts of the process are turned into structured parts. This can also cause less readable process models, since all possible sequences need to be modeled separately.

Another modeling language exists which allows flexibility, the declarative modeling language. This is a constraint-based modeling language. Constraints are put between activities to prohibit behavior. All behavior that is not prohibited by constraints, is allowed. A disadvantage of this modeling language is that it is complex to model (highly) structured processes. According to the modeling experts that participated in the workshop described in [24], the declarative modeling language is not a process modeling language that can be used to model entire business processes in practice. However, a mix between imperative and declarative modeling languages seemed as an attractive idea.

A mix between two (or more) modeling languages is called a hybrid modeling language. Such a modeling language is a component of a hybrid process modeling approach. A hybrid process modeling approach also includes a method of how to create a hybrid process model with the hybrid modeling language. The following papers describe already existing ways of combining imperative and declarative approaches. Wainer and Bezerra [31], present an idea and prototype of a workflow system defined through constraints. They claim that declarative constraints can be defined by defining pre- and post-conditions for activities. Lienhard and Künzi [13] proposed an approach where business logic is separated from process logic. This way they claim that constraints could be defined within the business logic about the process logic. While Kumar and Yao [12] extended this approach by developing an imperative template to model a process accompanied by declarative rules. Furthermore Caron and Vanthienen [6], consider hybrid, declarative and imperative modeling concepts in the context of the business process life-cycle. The different phases
of the business process life-cycle are discussed by means of these three concepts. Finally, Schöning, Zeising and Jablonski [26] present a framework to work with specified and unspecified processes. These specified processes can be of imperative or declarative nature and can be executed, while the unspecified processes should be discovered first. The processes may also be hybrid. All these papers describe ways of how to deal with or create hybrid (process) models but none of them developed a (graphical) hybrid modeling language.

Methods and guidelines are developed to support the user in the process of creating a process model. For the imperative modeling language, papers [16] and [2] describe guidelines. These guidelines mention attention points which can be taken into account when modeling. Paper [10] states that, in general, analysts tend to think sequentially when trying to understand a process model. This tells us that the declarative mindset does not come natural to modelers. To support modelers to model with this type of modeling language, guidelines are needed. Such guidelines do not yet exist. There are also no guidelines mentioned in literature which help users model with a hybrid modeling language.

1.2 Problem statement

Section 1.1 shows that there are problems with modeling processes that contain both structured and flexible parts. The imperative modeling languages cannot cope well with flexible (parts of) processes and the declarative modeling languages have difficulties with structured (parts of) processes. However, a combination between these two does not exist yet. Literature indicates that guidance is needed to support the use of declarative modeling languages. Guidance in the form of a method, which presents a structured way of creating a hybrid process model, and guidelines, which help the user deal with the differences between imperative and declarative modeling languages, also do not exist yet.

Therefore the main objective addressed in this thesis is:

“Develop a hybrid process modeling approach.”

This hybrid process modeling approach consist of three components. These are captured in the following sub-objectives:

1. Develop a hybrid modeling language.
2. Develop a method for creating a hybrid process model.
3. Develop a set of guidelines which helps the user with creating hybrid process models.

The following sub-objective is not a part of the hybrid process modeling approach, but it evaluates a component of it:

4. Evaluate the set of guidelines by means of an empirical experiment.

1.3 Scope

In this section, the scope of the research is presented. Per sub-objective it is described to which extent it will be investigated.

1. Develop a hybrid modeling language.

The design for the hybrid modeling language is theoretical. It is described how imperative and declarative modeling languages can be combined and what is needed to do so. It is not the case that a fully executable
modeling language will be developed. However, suggestions will be made of how to implement the new elements of this language.

2. Develop a method for creating a hybrid process model.
The method describes which steps have to be taken to create a hybrid process model, starting from a process description. No other starting points will be taken into account.

3. Develop a set of guidelines which helps the user with creating hybrid process models.
For each step of the method, guidance needs to be present. The main focus is on a set of guidelines which helps the user identify imperative and declarative activities from a process description.

4. Evaluate the set of guidelines by means of an empirical experiment.
Only the set of guidelines that focuses on identifying imperative and declarative activities from a process descriptions will be evaluated with an empirical experiment. The guidelines are tested on usefulness, understandability and effectiveness.

1.4 Research method & thesis outline

This section presents the research method used for this thesis, as well as the thesis outline. The regulative cycle, developed by P.J. van Strien [30], is the research method that is used in this thesis. Figure 1.1 shows the steps of this cycle and each step is discussed in this section. Additionally, the chapters are presented which discuss the information gathered/developed in this phase.

Figure 1.1: The regulative cycle

In chapter 2 background information is given on process models and their applications. Furthermore, imperative and declarative modeling languages are discussed in general and the modeling languages BPMN and Declare are discussed in detail.

Problem statement
In this phase, the problem statement is determined. This is already done in this chapter (section 1.2). In chapter 3 a brief part is dedicated to this topic.

Analysis
In this phase, an analysis is made of the problem. Requirements are established for all components of the hybrid process modeling approach. Also, an analysis is made of different imperative and declarative modeling languages. Eventually for both languages a particular choice has been made. These chosen modeling languages are used in the design of the hybrid modeling language. This phase is described in chapter 3.
Design
In the design phase, designs for the components of the hybrid process modeling approach are presented. In chapter 4 the design for the hybrid modeling language is provided. The modeling languages chosen in the analysis phase are combined using new modeling elements. The implementation details are also briefly discussed. In chapter 5 the method and guidelines supporting this method are introduced.

Implementation & Evaluation
These two phases are combined in 6. An experiment was conducted to evaluate a set of guidelines. The experimental design, planning and execution are explained. The chapter also includes the data analysis and the interpretation of the results.

Finally, chapter 7 shows the conclusions drawn from this research. All objectives are discussed and to which extent they were achieved. Additionally, topics for future work are presented.
CHAPTER 2

Process modeling state of the art

This chapter contains background information on process models, its applications, imperative modeling languages and declarative modeling languages. This information is needed to get a clear overview of the area of interest and for the upcoming chapters.

Section 2.1 elaborates on different applications of process models and their purposes. Section 2.2 describes imperative modeling languages in general. In particular, the imperative modeling language BPMN is described since it is used in the hybrid modeling language (chapter 4). Furthermore, a brief overview is given of other imperative modeling languages. In section 2.3 the general purposes of declarative modeling languages are described. The declarative modeling language that is highlighted is the Declare language. This particular language is also used in the hybrid modeling language. This section is concluded with a brief overview of other declarative languages.

2.1 Models and applications

Nowadays, process modeling is used for different purposes and to achieve different goals. This section illustrate a couple of the applications making use of process modeling. It also shows different perspectives for using process models.

Paper [8] mentions three important applications in which process modeling is used. These are:

**Business process re-engineering.** Process modeling is used in the redesign of an organization’s business processes to make them more efficient.

**Coordination technology.** Process modeling act as an aid to managing dependencies among the agents within a business process, and provides automated support for the most routinized component processes.

**Process-driven software development environments.** Process modeling can lead towards an automated system for integrating the work of all software-related management and staff; it provides embedded support for an orderly and defined software development process.
According to Curtis et al. [8] there are four most commonly represented perspectives to use process models: functional (what is performed), behavioral (when is it performed), organizational (where and by whom is it performed) and informational (what information is produced or manipulated). A combination of these four perspectives should result in a complete model of the analyzed process.

Besides the four different perspectives described above, there are two other types of models: the execution (or technical) model and the communication (or user) model. The execution model combines all four perspectives described and therefore could be seen as a complete model. This type of model is typically used by software engineers, e.g. to create an automated workflow system. All aspects of the process are modeled, implemented and syntactically correct. The communication model on the other hand is mostly used by consultants to communicate with the client about the processes of their organization. Therefore, the client has to recognize the processes of its organization in the model. For this purpose, the communication model should be easy to understand by non-modeling experts. It is also not always syntactically correct. However that is not an issue since a new execution model is made for the implementation of the process.

2.2 Imperative modeling languages

The most commonly used modeling approach is the imperative (or procedural) process modeling language. This way of modeling focuses on how a process works. It explicitly models every possible order of activities. To keep the process model at a manageable size, decisions about the order of activities has to be made as early as possible in the designing process. Due to these restricting decisions over-specification could arise. Too much of the flexibility is given up to limit the size of the model. Therefore this way of modeling is most suited for (highly) structured processes. More flexibility leads to bigger process models and decreases the readability drastically.

2.2.1 BPMN

Business Process Model and Notation (BPMN) is one of the many imperative modeling languages which can specify a business process models in a graphical way. BPMN is the only BPM standard, which makes it one of a kind and commonly used. A full specification of BPMN 2.0 can be found in [20].

![Overview of the basic BPMN elements](image)

Figure 2.1: Overview of the basic BPMN elements

Figure 2.1 shows an overview of the basic elements of BPMN 2.0. Each of these elements is explained briefly.

‘Activities’ can be seen as actions that have to be executed during the process. A ‘Task’ is one of such actions and a ‘Sub-process’ is a group of these actions which are related to each other.
An ‘Event’ can be seen as the state of a process. For example a ‘None start event’ is the state before any task of that (sub-)process is executed. The state after all tasks are executed is called the ‘None end event’. The ‘None intermediate event’ is a state somewhere in the middle of a process. In the core BPMN set of elements also triggered events are included. These events can throw or catch, for example, a message.

The ‘Sequence flow’ is the most used connecting object of BPMN, it determines the sequences of tasks. ‘Message flow’ is only used to communicate a message between participants and ‘Association’ associates text to modeling elements.

‘Gateways’ are used on a crossing of branches. With a ‘Gateway’ it can be decided how many branches are chosen. An ‘Exclusive Gateway’ ensures an exclusive choice between one or more branches. A ‘Parallel Gateway’ ensures that all branches are chosen and executed in parallel.

‘Artifacts’ exist to make the process model more clear, but they are not necessary for the execution of the process model. ‘Text annotations’ are used to add a textual explanation to modeling elements and ‘Groups’ are dotted rounded rectangles which can be placed around a number of tasks which belong together.

‘Swimlanes’ are another way of grouping modeling elements together but they also have a function for the execution of the process model. ‘Lanes’ are most of the time introduced for a specific role and all tasks that are included in a lane may be executed by a person of that role. ‘Pools’ can be seen as containers of the lanes, e.g. as the company or a blackbox. A ‘Participant’ is a company or department that is modeled in the process model. It can be depicted as a lane or pool.

Figure 2.2: An example of a BPMN process model

Figure 2.2 shows an example of a BPMN process model. The model describes the following restaurant process: A customer arrives at the restaurant, asks for the specials and gets a greeting from the server. The server provides the customer with a list of the specials and the customer orders something, whereafter the order is entered in the system by the server and both the server and the customer wait for the next activity.

The model contains two lanes named ‘Server’ and ‘Customer’. The tasks within these lanes can only be performed by the server or customer respectively. Both these lanes have a (none) start- and (none) end-event, these are the circles at the beginning and at the end of the sequences respectively. Inside the lanes, the sequence flow is used to model the sequence of the tasks. Between the two lanes, dashed arrows are drawn. These are message flows and exchange messages between the server and the customer. The greeting is for example one of the messages which is exchanged between the server and the customer.
2.2.2 Others

Beside BPMN there exist other commonly used imperative modeling languages. The most important examples are Event-driven Process Chains (EPCs), Petri Nets and BPEL (Business Process Execution Language). Figures 2.3 - 2.5 show examples of these three imperative languages.

![Figure 2.3: An EPC model](image1)

![Figure 2.4: A PetriNet model](image2)

![Figure 2.5: A BPEL model](image3)

All these languages are used to graphically model process models. Something they have in common is that they use activities and directed arrows to model the flow of the process. There are also a lot of differences between these languages and BPMN. A difference between BPMN and Petri Nets, for example, is that Petri Nets include places which can contain tokens. BPEL is a service interface based execution language and is more focused on the execution of the tasks and less focused on a intuitive layout in comparison to BPMN. EPC’s are flowcharts which also contain less functionality compared to BPMN.

2.3 Declarative modeling languages

The other way of creating a process model is by using a declarative modeling language. This is a constraint-based language, which is specifically useful for flexible processes. Flexible processes are processes in which activities exist, that are allowed to happen multiple times and/or in various orders. A declarative modeling language can specify the constraints under which the activities can be executed. This section elaborates on the declarative language Declare (subsection 2.3.1) developed at the Eindhoven University of Technology and takes a brief look at other declarative modeling languages (subsection 2.3.2).

2.3.1 Declare

Declare is a declarative modeling language that uses constraints to specify relations between activities. Declare contains three different activities and 19 different constraints. In this subsection first the activities are described, second the meaning of the different constraints are explained. This subsection also shows a new graphical notation for the activities and constraints of Declare. This graphical notation is developed by two members of the UX team (i.e. User Experience Team) from Perceptive Software. Appendix E.4 shows a comparison between the old and the new graphical notation accompanied with the reasons for changing.
2.3. DECLARATIVE MODELING LANGUAGES

Activities

There are three, differently distinguishable activities in Declare. The semantics and graphical representation are described below.

Activity: Initially, this type of activity is always enabled and can be executed any number of times. This may change due to the constraints attached or the presence of the init- or last-activity. It is depicted as a rectangle (see Figure 2.6).

Init activity: This type of activity must be the first activity to be executed within a (sub-)process. After its execution, it is possible to execute the same activity multiple times if it is allowed by the constraints attached to it (if any). It is depicted as a pentagon (see Figure 2.7).

Last activity: This type of activity must be the last activity which is executed in a (sub-)process. After its execution, it is still possible to execute this activity with or without other activities in between. This activity, however, must always be the last activity which is executed (of a (sub-)process). It is depicted as shown in Figure 2.8.

Constraints

Declare contains 19 different constraints. These constraints can be placed between activities. They limit the behavior of the activities in their own way. The constraints can be divided into four categories: existence-, relation-, negation- and choice-constraints. Per category the semantics and graphical notation of the constraints are described. In the continuation of this section the capital letters ‘A’ and ‘B’ are written instead of ‘Activity A’ and ‘Activity B’ respectively.

Existence constraints

Existence constraints indicate how often an activity is allowed to occur. The graphical notation of these three constraints is shown in Figures 2.9, 2.10, 2.11.

Exactly(A,n) A must occur exactly n times (see Figure 2.9).

Existence(A,n) A must occur at least n times (see Figure 2.10).

Absence(A,n) A must occur at most n times (see Figure 2.11).

Relation constraints

Relation constraints indicate what kind of relation exists between activities. For each constraint the graphical notation is shown along with a description and some example traces.

Responded existence(A,B) Only the first A should be preceded or followed by a B (not necessarily directly) (see for the notation Figure 2.12 (a)).
Co-existence($A,B$) Both $A$ and $B$ should occur (at least once, in a random order, not necessarily directly after each other) or none of them (see for the notation Figure 2.12 (b)).

Precedence($A,B$) Only the first $B$ should be preceded by at least one $A$ (not necessarily directly) (see for the notation Figure 2.13 (a)).

Alternate precedence($A,B$) Every $B$ should be preceded by at least one $A$ (not necessarily directly) (see for the notation Figure 2.13 (b)).

Chain precedence($A,B$) Every $B$ should directly be preceded by at least one $A$ (see for the notation Figure 2.13 (c)).

Response($A,B$) Every $A$ must eventually be followed by at least one $B$ (not necessarily directly). Other $A$’s may happen in between (see for the notation Figure 2.14 (a)).

Alternate response($A,B$) Every $A$ must eventually be followed by at least one $B$ (not necessarily directly). There are no other $A$’s allowed in between (see for the notation Figure 2.14 (b)).

Chain response($A,B$) Every $A$ must directly be followed by at least one $B$ (see for the notation Figure 2.14 (c)).

Succession($A,B$) Every $A$ is eventually followed by at least one $B$ (not necessarily directly). Every $B$ is preceded by at least one $A$ (not necessarily directly) (see for the notation Figure 2.15 (a)).

Alternate succession($A,B$) Every $A$ is eventually followed by exactly one $B$ (not necessarily directly). Every $B$ is preceded by exactly one $A$ (not necessarily directly) (see for the notation Figure 2.15 (b)).

Chain succession($A,B$) Every $A$ is directly followed by exactly one $B$. Every $B$ is directly preceded by exactly one $A$ (see for the notation Figure 2.15 (c)).

Figure 2.15: Three succession constraints
Example: precedence versus response.
Figure 2.16 shows two models displaying the precedence (1) and response (2) constraints. To clarify the differences between these two constraints the behavior of both models is explained.

Figure 2.16: Two models displaying the precedence and response constraint.

In the first model (precedence), the client should order room service at least once before receiving the bill. After the first time, the client can get room service and receive a bill any number of times and in any order.

In the second model (response), every time when a client receives room service, eventually he/she has to receive a bill. In this model it is also possible that the client gets a bill without receiving room service. Whenever a client receives (multiple) room services at least one bill has to be received as well.

Example: three types of precedence.
Figure 2.17 shows three constraints: precedence (1), alternate precedence (2) and chain precedence (3). To clarify the differences between the ‘normal’ constraint precedence, its alternate and chain version, the behavior of the three models is explained.

Figure 2.17: Three models displaying the precedence, alternate precedence and chain precedence constraint.

In the first model ((normal) precedence), only the first time you want to order food, you have to receive a menu. After that, ‘Receive menu’ and ‘Order food’ can happen in any order and any number of times.

In the second model (alternate precedence), every time you want to order food, you must have received a menu sometime before. It is not possible to order food twice in a row, without receiving a menu (again) sometime before. On the other hand it is possible to receive a menu twice in a row and sometime after that order food.

In the third model (chain precedence), every time you want to order food, you must have received a menu directly before. It is not possible to order food two times in a row, because a menu must be received in between and directly before ordering food.

Negation constraints
For every relation constraint, a negation constraint exists. As some of them are equivalent to another negation constraint, only three constraints are needed to cover all negation constraints. The not responded existence-constraint coincides with the not co-existence-constraint; the not response-constraint and the not precedence-constraint coincide with the not succession-constraint; and the not chain response-constraint and the not chain precedence-constraint coincide with the not chain succession-constraint. Also the negation of the alternate-constraints coincide with the negative chain constraints.

Not co-existence(A,B) A and B cannot happen together in any order (see for the notation Figure 2.18 (a)).

Not succession(A,B) A cannot be followed by B (not necessarily directly).
Traces which contain an A somewhere before a B are prohibited, whereas traces with a B somewhere before an A are allowed (see for the notation Figure 2.18 (b)).

**Not chain succession(A,B)** A cannot be directly followed by B. Traces which contain an A directly followed by a B are prohibited, whereas traces which contain an A somewhere before a B or contain a B somewhere (or directly) before an A are allowed (see for the notation Figure 2.18 (c)).

![Figure 2.18: Three negation constraints](image)

**Choice constraints**
Choice constraints indicate that a choice could/must be made between tasks. In [21] eight different choice constraints are described, but they can be captured in the two constraints explained below. Both these constraints are similar to the corresponding BPMN elements. The ‘exclusive m of n’ constraint is similar to the BPMN XOR-gateway, the ‘m of n’ constraint is similar to the BPMN OR-gateway.

**Exclusive m of n(A₁, ..., Aₙ)** Exactly m of the n activities must be executed one or more times, while the other n − m activities cannot be executed at all (see for the notation Figure 2.19 (a)). (m < n and n ≥ 2)

**m of n(A₁, ..., Aₙ)** At least m of the n activities has to be executed, but all activities can be executed an arbitrary number of times (see for the notation Figure 2.19 (b)). (m < n and n ≥ 2)

![Figure 2.19: The two choice constraints](image)

### 2.3.2 Others

In order to provide a complete information base, this subsection describes other declarative languages. These languages are shortly described and a graphical example is included for illustration purposes.

The other well-known declarative language is DCR Graphs as described in [11]. It is developed at the IT University of Copenhagen. Figure 2.20 shows an example of a DCR Graph model. Rectangles are used to express activities. On the top right of these rectangles roles can be indicated. A dashed rectangle means that the activity is initially excluded from the set of activities that can be executed. Relations between the activities can be expressed by constraints. DCR Graphs only has five different constraints: condition, response, milestone(⋄), inclusion(+) and exclusion(%). The condition and response constraints are similar to the precedence and response constraints of Declare respectively. The inclusion and exclusion constraints can include or exclude activities from the set of activities that can be executed. It also supports hierarchical models. Although DCR Graphs only have five defined constraints, the developers claim it has the same expressiveness (concerning finite models) as Declare.

Other declarative approaches discussed in [1] and [25] do not have a graphical representation, but also use a declarative approach.
Figure 2.20: An example of a DCR Graphs model
This chapter describes both the problem statement and analysis phase. As can be said concluded from the advantages of both modeling approaches discussed in chapter 2, the imperative approach is made for modeling (highly) structured processes whereas the declarative approach does a better job modeling (highly) flexible processes. Since most processes are not only structured or only flexible, a combination of the two languages seems ideal. I.e., whenever a part of the process is structured, the imperative part of the language can be used and whenever a part of a process is flexible, the declarative part of the language can be used.

This chapter addresses requirements for the hybrid modeling approach (section 3.1) as well as an analysis of both imperative and declarative modeling languages. One of both is chosen as a base for the hybrid modeling language (section 3.2).

3.1 Requirements

This section lists and explains the functional and non-functional requirements that the hybrid process modeling approach should satisfy. These requirements are indicated with an R followed by a number. Unless it is stated otherwise, the requirements are functional requirements.

3.1.1 Hybrid process modeling approach

First of all, all three components of the hybrid process modeling approach should be present: hybrid modeling language, method to create a hybrid process model, and guidelines for creating a hybrid process model.

R1: The hybrid process modeling approach should include a hybrid modeling language.

R2: The hybrid process modeling approach should include a method to create a hybrid process model.

R3: The hybrid process modeling approach should include a set of guidelines for creating a hybrid process model.
Functional and non-functional requirements are given for each component in the next subsections.

### 3.1.2 Hybrid modeling language

This process modeling language should support both imperative and declarative modeling elements. It also should support hierarchical modeling elements, to be able to model with both types of languages in one process. The hybrid modeling language can be used to close the gap between a communication model and the execution model (explained in section 2.1). Therefore, the hybrid approach should be graphically intuitive for the client, so that the client is still able to recognize the processes of its organization. On the other hand, it should be executable to be used as an execution model. Below, a list of requirements for the hybrid modeling language is shown.

- **R1.1:** The hybrid modeling language should support imperative modeling elements.
- **R1.2:** The hybrid modeling language should support declarative modeling elements.
- **R1.3:** The hybrid modeling language should support hierarchical modeling elements.
- **R1.4:** The hybrid modeling language should be executable.
- **R1.5:** The hybrid modeling language should be graphically intuitive. *(Non-functional)*

### 3.1.3 Method to create hybrid process model

The method for creating a hybrid process model should have well-defined steps to follow. It should also be iterative, since it must have room for adjustments. The following two requirements can be established:

- **R2.1:** The method should have well-defined steps. *(Non-functional)*
- **R2.2:** The method should be iterative. *(Non-functional)*

### 3.1.4 Guidelines for creating hybrid process model

The most important set of guidelines is the one which help the user to decide whether to use imperative or declarative modeling elements. The guidelines should also be easy to use and useful. The next three requirements capture these aspects:

- **R3.1:** A set of guidelines should be present to decide between imperative and declarative modeling elements.
- **R3.2:** The guidelines should be easy to apply. *(Non-functional)*
- **R3.3:** The guidelines should be useful. *(Non-functional)*

### 3.2 Modeling language analysis

As stated in the previous section, a hybrid process modeling approach should support both imperative and declarative modeling elements. Considering the time aspect of this project, the choice was made not to develop a whole new language, but combine two already existing languages. Another reason is that there are already a lot of procedural languages which are successfully used in practice. This section describes which languages are used and the reasoning behind it.
3.2.1 Imperative language

There are a lot of imperative languages to choose from e.g. EPC, Petri Net, BPMN. BPMN is the only BPM standard and recently presented a new version BPMN 2.0 [20]. Additionally, BPMN is by far the most popular imperative modeling language and, many companies use it to model their processes. There are over a dozen editors which support BPMN and it is open source. These reasons support the choice to use BPMN 2.0 as the imperative language of the hybrid approach.

3.2.2 Declarative language

At the moment the two best-known declarative languages are Declare and DCR Graphs. To decide which language is best for developing the hybrid language, a comparison between the two languages is made. This is done on the following areas: expressiveness, understandability and the option to use hierarchy.

Expressiveness The major difference between Declare and DCR Graphs is that the latter one only has five different constraints whereas Declare has around 30 different constraints. [11] states that a follow-up paper proves that DCR Graphs can express all behavior that can be expressed by Declare and even more. The extra expressiveness is due to their inclusion- and exclusion-constraints. It is stated that in infinite models, DCR Graphs can express more behavior. Since the scope of this project is only in process models from practice, which are not infinite, this extra expressiveness is not an advantage.

Understandability For practical purposes it would be more convenient to use the language which is the most understandable. To investigate this area, paper [24] elaborates about the understandability of declarative models. It states that, there was no notable difference in difficulty between the two languages. At the moment there is no other research available comparing the understandability of these two languages.

Hierarchy Since the use of hierarchy is a major part in the hybrid language, the possibilities to achieve this in a declarative language must be considered. DCR Graphs has already implemented this aspect using nested graphs whereas for Declare this is not the case. However, a conceptual idea and the semantics of sub-processes in Declare is already posted [34].

Based on this investigation, the choice is made to use Declare as the declarative language. As mentioned above, there are no significant differences on the expressiveness considering practical process models; there was no preference regarding understandability within the experts; there are no major problems anticipated for using hierarchy within Declare. Besides these reasons, the Declare language is developed at the Eindhoven University of Technology (TU/e) whereas the concept of DCR Graphs is developed in Denmark. The developers of Declare also made a tool which combines the imperative language ‘Petri Nets’ with Declare. This tool is, in the course of the project, used to make example models of the hybrid language and possible new features can easily be requested to the developer, since this developer is also present at the TU/e.
In this chapter the first part of the design phase is discussed. The design of the hybrid (process) modeling language is given. This hybrid modeling language consists of an imperative modeling language (BPMN 2.0 [20]) and a declarative modeling language (Declare [22]). These are combined with a newly introduced element, which is called a block. The basic idea of combining the two languages is described in section 4.1. In sections 4.2 and 4.3 the details of the declarative and imperative blocks are discussed respectively.

4.1 New modeling element: Block

To avoid the problem of having both imperative and declarative elements in one layer of a process, a new modeling element is created for both languages. This new element is called a block. A block can be seen as a sub-process. Inside this sub-process, a new process can be modeled. A block may contain only imperative or declarative elements. These types of blocks are called imperative block and declarative block respectively. The process that contains a block, is called the parent-process of that block.

To clarify the block structure idea, Figure 4.1 shows a fictive example of two possible block combinations. The top layer of the process (declarative block 1), is modeled with a declarative modeling language. It contains two blocks: one imperative block and one declarative block. There are no constraints modeled, so both blocks are allowed to be executed in any sequence and any number of times. The processes modeled inside these blocks are located on the second layer of the process.

It is allowed to have an imperative block inside an imperative block. The semantics of this combination are not different from the ones of an ordinary sub-process of the BPMN language. It is also allowed to have a declarative block inside a declarative block. The semantics of this combination are not different from a sub-process construction within the Declare language. The semantics of the other block combinations are discussed in the following sections.
4.2 Declarative block in BPMN

In declarative process models, multiple activities can be enabled at the same time. In this case user input is needed to decide, for example, which activity must be executed next. Usually this is not a problem. However, when there are multiple activities enabled at the beginning of a process or at the end of a process, there is no explicit start or end of that process. This is not ideal to have inside a block. It is desirable that a block has one start- and one endpoint. The process one layer higher needs to know when the block has started and when it has finished, so the process in that layer can continue. Therefore the following assumption is made:

Assumption 1: A declarative block has one explicit start and one explicit end. (4.1)

In a BPMN-process, the declarative block should be considered as another BPMN-activity. It can be instantiated via a sequence flow coming from another activity or gateway. When this happens, the activities within the block can be executed. When there are no violated constraints, the user can decide to terminate the block. The parent-process can now continue. It is also possible to model activities in parallel with a declarative block. It is not allowed to connect an activity outside the block with an activity inside the block \[34\].

The graphical representation of a declarative block is shown in Figures 4.2 and 4.3. The left symbol represents a collapsed declarative block and the right symbol represents the expanded declarative block. To meet assumption 4.1 an explicit start and end are added to the declarative block. This is established by adding an init- and last-activity, which must be executed precisely once. Therefore a declarative block consists of two layers. The first layer contains the init-, last-activity and another activity which consists of the actual process to be modeled inside the block.

Figure 4.2: Collapsed Declarative Block

Figure 4.3: Expanded Declarative Block

Figure 4.4 shows a small example is of a BPMN-process containing a (n expanded) declarative block. The depicted process is a small restaurant process. First the restaurant is entered by the visitor and the visitor sits down at a table. After that, the declarative block can start. First, the init-activity ‘Start’ needs to be executed, after that either the second layer of the block can be executed or the last-activity ‘Finish’. When the second layer of the block is executed, first the menu should be received. After that, drinks and food can be ordered and eventually received. Recall that the indicated constraints between ‘Receive menu’ and ‘Order drinks’ means that the customer should first receive a menu after which the
customer can order drinks multiple times. The constraint between ‘Order drinks’ and ‘Receive drinks’ means that the customer can order multiple drinks and will eventually receive these drinks. When the visitor has had enough, the last-activity ‘Finish’ is executed and the execution of the declarative block has ended. Now the BPMN-process can continue, which means the bill is received by the visitor and after it is paid the visitor leaves the restaurant.

4.3 Imperative block in Declare

In a declarative parent-process the imperative block should be seen as another declarative activity. Therefore, constraints can be attached to it and it can happen multiple times. The block must be instantiated manually. When it is instantiated, all activities connected with a sequence flow to the start event are enabled. Whenever the end event of the block is reached, the parent-process can continue and the imperative block has ended. It is not allowed to put a constraint or message flow between an activity outside the block to an activity inside the block or the other way around.

A BPMN-process can have multiple start and end events. There are also a lot of different types of start and end events. In this thesis, the main focus is on connecting the imperative and declarative languages. Therefore the following assumption is made:

**Assumption 2:** An imperative block has only one start and one end event. (4.2)

The graphical notation of the imperative block is shown in Figures 4.5 and 4.6. The left symbol represents a collapsed imperative block and the right symbol represents an expanded imperative block.

![Figure 4.5: Collapsed Imperative Block](4.5)

![Figure 4.6: Expanded Imperative Block](4.6)

Figure 4.7 shows a small example of an (expanded) imperative block within a declarative process. The process describes a way to spend time on your computer. First, the computer should be started. Then it is possible to go on Facebook or Twitter in any order and for any amount of time. When an email is received a new process can be started (manually) to process the email (the imperative block). The email is read and a decision is made whether to respond to, throw away or archive the email. When one of these options is chosen, the user can go back to Facebook and/or to Twitter. When the user is done, the computer can be shut down and the process ends.
4.4 Implementation

The actual implementation of the hybrid modeling language is not part of the scope of this research. Nevertheless, this section gives a proposal of how to implement a declarative block in BPMN 2.0 and to implement an imperative block into Declare.

4.4.1 Implementation of declarative block

Within BPMN 2.0 there are a lot of different types of sub-processes. There is one kind of sub-process which allows for declarative behavior: the ad-hoc sub-process. As [20] explains: “An ad-hoc sub-process is a specialized type of sub-process that is a group of activities that have no required sequence relationships. A set of activities can be defined for the [sub-process], but the sequence and number of performances for the activities is determined by the performers of the activities. Activities within the [ad-hoc sub-process] are generally disconnected from each other. During execution of the process, any one or more of the activities may be active and they may be performed multiple times. The performers determine when activities will start, what the next activity will be, and so on.”.

According to the class diagram, the ad-hoc sub-process can only contain BPMN tasks. However, the BPMN 2.0 standard supports the addition of customized elements. Therefore a new type of sub-process should be introduced which can only contain declarative activities. These declarative activities could be implemented with a new type of task. The two special types of declarative activities, init- and last-activity, should also be defined. This can be done, for example, by adding to boolean attributes to the task which indicate whether the task should be seen as an init-, last-activity or a normal declarative activity. The declarative constraints cannot be inherited from the BPMN ‘flow element’ because a constraint does not direct a flow from one task to another. A suggestion would be to introduce a new type of connecting object. This object should be able to specify the different graphical notations and functionalities of the Declare constraints.

4.4.2 Implementation of imperative block

Only the declarative language DCR Graphs has an implemented version of the sub-process (or block) concept in a declarative language. For the language Declare, paper [34] introduces a sub-process element. However, this is only for conceptual purposes and there are no implementations yet where such a sub-process is used in combination with the Declare language.

There already is an implementation which combines the modeling elements of PetriNets with the Declare language[1]. This tool allows for both imperative and declarative constructs in one layer. Also, the ways to use declarative sub-processes are circuitous.

In chapter 4 the notation and semantics of the hybrid modeling language are described. Thereby the first part of the design phase is completed. In this chapter, the second phase of the design phase starts. In this chapter a method is presented which helps the user, step-by-step, to develop a compact, simple and readable model using that hybrid modeling language. The steps of the method are explained and some of them are supported with a set of guidelines. These guidelines, for example, will help the user to decide when to use either the declarative or the imperative part of the hybrid modeling language.

In section 5.1 a case study is introduced. This example is used throughout this chapter to demonstrate the different steps of the method. In section 5.2 the method is presented, as well as, a global overview of the different steps. In sections 5.3-5.6 more detailed descriptions of the steps of the method are given.

5.1 Case study

This section introduces an example process description which is used throughout the chapter to demonstrate the different guidelines and principles. Eventually a hybrid process model is created of this process description. The process description is taken from an assignment of the course Process Modeling for Information Systems (1BP05) taught at the Eindhoven University of Technology. In order to improve its applicability for the scope of this thesis, the assignment has been slightly modified. For readability reasons, the activities, discovered in the process description, are highlighted with a bold font. The process description considers a part of a fictive diagnosis process of patients having a severe cough.

“First an appointment is made with the specialist, this is followed by an actual visit to the specialist. The specialist records the symptoms and decides that (1) there is no need to worry and the process is ended, (2) the patient should make a new appointment to visit the specialist later or (3) the symptoms require an X-ray to be made. When a new appointment has to be made, the same choice needs to be made. If the specialist decides an X-ray needs to be made of a patient, the patient is referred to a radiologist who takes the X-ray photograph. That X-ray is sent back to the specialist and, based on that X-ray the specialist may decide to do more tests: (1) perform another X-ray, (2) do a blood test and/or (3) perform a CT-scan. For example, when another X-ray is performed the choice is made again to do more tests or not. Every test result must be sent to the specialist and multiple tests can
be done in parallel. The specialist evaluates each test separately i.e. the specialist does not wait until all test results are available because this could put the patient in danger. Whenever a test result indicates a serious problem, an emergency treatment is started. When no serious problems are detected the normal treatment is started. Note that multiple emergency treatments can be started, depending on the number of serious problems detected. The process ends when the normal treatment is started or all necessary emergency treatment(s) is/are started.”

5.2 Method to create hybrid process model

This section describes a method to create a hybrid process model from a process description. It uses a top-down approach because it is important to keep a clear overview of the process. With every step more details are added to that overview. An advantage of this method is that it can be used in an iterative way. When new insights have been gathered from other steps, the model can be adjusted and improved. The steps of the method are shown in Figure 5.1.

![Figure 5.1: Iterative process of creating a hybrid process model](image)

The steps are explained in more detail in the following sections. Before focusing on the details, an overview is presented by briefly explaining the steps of the method. A process description is used as a starting point for applying this method, but perhaps it can also be applied for other starting points.

Define groups: A group is a subset of the set of activities defined in the process description. How to define these groups and what characteristics should be taken into account when defining groups is explained in section 5.3.

Define blocks: A block is also a subset of the set of activities defined in the process description. The difference between a group and a block is that in a block either exclusively imperative or exclusively declarative constructs are allowed. A group can contain both constructs and thus may contain one or more blocks. Since the method is iterative, the groups can be adjusted after defining blocks (if needed). The guidelines developed for defining both types of blocks can be found in section 5.4.

Define relations between blocks: The different block combinations and ways to connect or combine them are explained in section 5.5. After this step, it is also allowed to redefine blocks (and even groups) if necessary.

Define relations between activities: When the overall structure of the process is defined, the imperative and declarative constructs should be added to the activities inside the blocks. Guidelines for this step are developed and presented in section 5.6. As Figure 5.1 shows, from this step it is also allowed to adjust groups, blocks or relations between blocks.

As mentioned before, it is not only possible to iterate over the different steps, it is also advised. It may, for example, be possible that a firstly discovered declarative block contains an activity which is better modeled imperatively.
5.3 Define groups

Identifying groups is the first step towards creating a hybrid process model. Recall that, a group is a subset of the set of activities defined in the process description. By introducing groups, the process description is divided into smaller parts. Especially when bigger process descriptions are considered, groups can positively influence the overview a user has of the process.

5.3.1 Characteristics

Process descriptions can be divided into numerous different groups because of various reasons. One of these reasons can be the size of the set of activities. Another reason is the level of detail a user needs to model a process. There are no generally applicable rules to define groups. However, a group does have characteristics that can be taken into account when creating groups. The activities within a group are both related and connected. The terms related and connected are described in more detail below.

Connected: Every activity in a group should be connected to at least one other activity in the group. Connected means that each activity in the group should have at least one other activity in the group as its direct predecessor for at least one case. The exception is the first activity of a sequence (called the starting or initial activity, if any). These activities can not be preceded by another activity in the group.

Related: All activities in a group should be related to each other. Whether or not two activities are connected can be computed using a computer, but for deciding whether or not activities are related, human assessment is needed. Activities can be related for several reasons. Three important reasons are:

1. Activities that are executed by the same type of resource (e.g. put activities that should be executed by a secretary in a different group than activities that should be executed by a manager).

2. Activities that concern the same business object (e.g. put activities that concern cleaning a floor in a different group than activities that concern cleaning a bathtub).

3. Activities that are in the same logical unit of work (e.g. group activities together that make sure particular data is loaded, changed and saved).

Figure 5.2: Example of defining groups
An example is shown in Figure 5.2. If a process description describes a sequence of activities A, B and C, then there are three possibilities to group these activities:

- If activities A, B and C are related to each other, they can be put in one group \((G_1)\). A, B and C are also connected since A is the initial activity, B is connected to A and C is connected to B.

- If activities A and B are related but they are both not related to activity C, then activities A and B can be grouped together \((G_2)\) and activity C can be put in a different group \((G_3)\). Activity B is still connected to activity A. The same reasoning holds for the situation where activities B and C are related and both not related to activity C.

- If activities A and C are related and they are both not related to activity B, e.g. because of the resources that should execute these activities, then all three activities should be put in three different groups \((G_4 \text{ up until } G_6)\). Activity A is not connected to activity C. Because A is not the direct predecessor of C, they cannot be put in one group. Therefore group \(G_7\) is incorrect. \((G_7, G_8)\).

In addition to the structure created by the introduction of groups, these groups can also be used as a starting point for identifying blocks. Since all activities inside a group are related and connected, these groups are also candidates for blocks. A group can contain one or more blocks, groups can also be combined into one block.

### 5.3.2 Concepts applied to the case study

In this subsection, the activities of the case study (section 5.1) are divided into groups. The characteristics of groups are taken into account. Firstly related activities are grouped together. Secondly, it is tested whether the activities of those groups are connected.

Eleven activities can be distinguished from the example. The highlighted activities ‘Take X-ray’ and ‘Perform X-ray’ are combined into ‘Perform X-ray’. The following five groups can be identified taking into account the characteristic related:

- \(G_1\): (‘Make an appointment’; ‘Make a new appointment’). Both activities are executed by the same resource, e.g. a secretary.
- \(G_2\): (‘Visit the specialist’; ‘Record the symptoms’). A specialist is involved with both activities.
- \(G_3\): (‘Perform X-ray’; ‘Perform blood test’; ‘Perform CT-scan’). All three activities concern the business object, performing tests.
- \(G_4\): (‘Send test results’; ‘Evaluate test results’). Both activities are about test results.
- \(G_5\): (‘Start normal treatment’; ‘Start emergency treatment’). Both activities concern a treatment.

Taking into account the connected characteristic, there is one group which violates this property. The activities in group \(G_1\) are not connected. Although activity ‘Make an appointment’ can be seen as the starting activity, it is not a predecessor of the activity ‘Make a new appointment’. The other way around, activity ‘Make a new appointment’ is also not a predecessor of activity ‘Make an appointment’.

Therefore group \(G_1\) is not a well-defined group. A solution is to combine groups \(G_1\) and \(G_2\). Activity ‘Make an appointment’ is still the starting activity. It is also the predecessor of activity ‘Visit the specialist’. Activity ‘Visit the specialist’ is in its turn the predecessor of activity ‘Record the symptoms’. At least one case exists where activity ‘Record the symptoms’ is the predecessor of activity ‘Make a new appointment’.

Therefore the connected characteristic is satisfied. The related characteristic is be satisfied by the explanation that all activities are steps in the examination of the complaints of the patient. Now that both characteristics are satisfied, group \(G_{12}\) is a well-defined group. Figure 5.3 shows the final division of the activities into groups.
5.4 Define blocks

By deciding on the groups, a high-level overview of process has been obtained. The next step, is to define blocks which include activities that are either exclusively modeled with the imperative or declarative part of the hybrid modeling language. The starting point for this step is the groups that were defined in the previous step.

This section presents two sets of guidelines. One will help to discover imperative blocks (subsection 5.4.1), the other will help to discover declarative blocks (subsection 5.4.2). Both the set of imperative guidelines (IBG) and the set of declarative guidelines (DBG) indicate characteristics of the behavior of activities that are typical for the corresponding type of modeling language. The guidelines are applied to the case study at the end of this section (subsection 5.4.3).

5.4.1 Discover imperative blocks

The imperative modeling approach is the most used modeling approach. Paper [10] show that people think in a more sequence-based way. Even though it is almost second nature for experienced modelers to develop models with an imperative language, this subsection presents four guidelines to discover imperative blocks in a process description. The guidelines are labeled with ‘IBG’ (i.e. ‘Imperative Block Guideline’) followed by a number.

These guidelines are based on the characteristics of the imperative modeling elements that differentiate the language from the declarative modeling language. These are: XOR gateway (IBG1), Parallel gateway (IBG3) and the Sequence Flow (IBG4). IBG2 is a counterpart for declarative guideline DBG4 and IBG3 is a counterpart for declarative guideline DBG3.

**IBG1** “If the process description indicates that a clear choice has to be made between multiple activities, then model these activities within an imperative block.”

**IBG2** “If the process description indicates a long-term dependency between two activities and the activities in between are following a specified order, then model these activities within an imperative block.”

**IBG3** “If the process description indicates that certain sequences of activities must be executed in parallel and between the different activities there are no other dependencies, then model these activities within an imperative block.”

**IBG4** “If the process description indicates that activities must be executed in a defined sequence and cannot happen more than once independent of each other, then model these activities within an imperative block.”
5.4.2 Discover declarative blocks

Identifying declarative blocks, compared to identifying imperative blocks, is a more difficult task to do. This subsection provides guidelines that will help to complete that task. The guidelines are explained and an example is used to motivate the advantages of declarative languages over imperative languages in the specified situation. The guidelines are labeled with ‘DBG’ (i.e. ‘Declarative Block Guideline’) followed by a number.

The guidelines are based on the characteristics of the declarative modeling language. Activities modeled with a declarative language are allowed to happen more than once (DBG1) and in different sequences (DBG2). Since it is a constraint based modeling language, long-term dependencies can be introduced (DBG4). Activities do not have to follow each other directly and in a predefined order. There are even activities allowed that do not have any constraints attached to them. They can be executed regardless of the execution of other activities (DBG5). DBG3 is the counterpart of the imperative guideline IBG3.

One general remark about declarative process modeling is that, it is often used when the process requires user input for deciding on which activities to execute. The user, for example, can decide how often activities may occur when that is not specified explicitly. Another example is that, the user can decide when the process or sub-process ends if there are multiple end activities.

DBG1 “If the process description indicates a subset of activities which are in the same group and are allowed to be executed more than once, then this subset of activities should be modeled within a declarative block.”

When an activity must happen exactly \( n \) times and it is modeled with a declarative language, the exactly-constraint is used. When the same behavior is modeled with an imperative language a ‘loop’ construct can be used. Using this construct makes it hard to determine when the \( n^{th} \) time has passed. Mimicking this behavior using an imperative language is hard.

Example: Consider the models of Figure 5.4. Both models model the behavior of first one activity A, which is directly followed by either one, two or three activities B which is directly followed by one activity C. Model (a) models this behavior using an imperative language. It uses XOR-constructs to choose between all possible sequences of activities B. Model (b) is the Declare-version which starts with the init-activity A which may only happen once. Then activity B can happen at most three times. After that last-activity C must happen once. The declarative model even covers more behavior. After activity B is executed for the first time, it can still make the choice to execute activity B another zero, one or two times. Even modeling this small example with an imperative language already creates a bigger and more complex model than modeling it with a declarative language.

![Figure 5.4: Imperative (a) and declarative (b) way of modeling the sequence A, maximum of 3 B’s, C](image)

DBG2 “If the process description indicates a subset of activities which are in the same group and allowed
5.4. DEFINE BLOCKS

to be executed in multiple, yet not all different sequences, then this subset of activities should be modeled within a declarative block.

The situation described in this guideline is when there are activities which can be executed in different orders, yet not all possible sequences.

**Example:** Assume three activities A, B and C are allowed to happen in different orders. It is not possible to execute activity B somewhere after activity C has been executed. Figure 5.5 shows two models which express this behavior. Model (a) presents this behavior modeled with an imperative language. All paths are explicitly modeled and at the beginning the choice for one particular path has to be made. Model (b) uses a declarative language and models the three activities with a not succession-constraint between C and B. Also exactly-1-constraints are added to forbid all three activities to occur more than once. As it can be seen, for this small example the declarative model is much smaller. In the declarative model even more behavior is possible. After the first activity is executed the choice of the next activity can still be made. In the imperative model, on the other hand, the choice for the whole sequence is made in the beginning.

![Figure 5.5: Imperative and declarative way of modeling](image)

**DBG3** “If the process description indicates a subset of activities which must be executed in parallel but some restrictions are applicable to these activities, then this subset of activities should be modeled within a declarative block.”

**Example:** Assume a process that has three activities: A, B and C. They all can happen in parallel, but it is not allowed for activity B to be executed directly after activity A (note that the sequence <B, A> is still possible). Figure 5.6 shows three models. Model (a) shows how activities A, B and C can be modeled in parallel using an imperative language. However in this way it is not possible to model that activity A cannot be executed directly before activity B. Hence (a) is not well defined. All different possible sequences should be modeled using an XOR-element in model (b). Model (c), the declarative model, shows only the three activities with one relational-constraint between A and B. To model the exact same possible sequences, all three activities also have an exactly-constraint where n equals one.

The difference between this guideline and guideline [DBG2](#) is how the behavior is explained in a process description. For guideline [DBG2](#) parallelism is not mentioned, just that activities can happen in multiple sequences but not all. On the other hand, for guideline [DBG3](#) parallelism is explicitly mentioned along with possible restrictions on the order of the execution of the activities.

**DBG4** “If the process description indicates a long-term dependency between two activities and it is not exactly specified what happens in between and in what order, then this subset of activities should be modeled within a declarative block.”

The long-term dependency mentioned in the guideline can be modeled with many Declare constraints. It can, for example, indicate that activity A should always be followed by activity B. This does not have to happen directly, there can be different activities in between. When this behavior is modeled with an
imperative language, a decision has to be made about when to execute activity B exactly. Such decision does not have to be made up front when modeling such a process with a declarative language.

**DBG5** “If the process description indicates an activity which can be executed regardless of the execution of other activities, then this activity should be modeled within a declarative block.”

This guideline describes an activity that is not part of any sequence, and therefore can happen anytime. There is one exception: when an *init*- or *last*-activity is present, there are bounds to when the activity can be executed.

*Example:* An activity, as described in DBG5, can be ‘Add items to the list’, in a process about making a to-do list and executing all activities of that list. It can be imagined that it does not have any relation to other activities. With an imperative modeling language, this is possible through modeling the activity within an ad-hoc sub-process. This approach is a bit circuitous, because a sub-process needs to be introduced for modeling just one activity. Using the declarative approach, only that activity can be modeled.

### 5.4.3 Concepts applied to the case study

This subsection explains in which blocks the activities of the example (section 5.1) can be divided.

Concerning group $G_{12}$: (‘Make an appointment’; ‘Visit the specialist’; ‘Record the symptoms’; ‘Make a new appointment’), the following two blocks can be defined:

$B_1$: [‘Make an appointment’; ‘Visit the specialist’]

Activity ‘Make an appointment’ can only be executed once and is always followed by activity ‘Visit the specialist’. Using IBG4 it can be concluded that these two activities are placed in one imperative block.

$B_2$: [‘Record the symptoms’; ‘Make a new appointment’]

The process description contains a clear choice moment: a decision, based on the recorded symptoms, has to be made. The options to choose from are: ending the process, making another appointment or requiring an X-ray. Only ending the process or requiring an X-ray are no activities itself. Using guideline IBG1 it can be concluded that activities ‘Record the symptoms’ and ‘Make a new appointment’ must be modeled in an imperative block.

Concerning group $G_3$: (Perform X-ray; Perform blood test; Perform CT-scan), the following block can be defined:
5.5. DEFINE RELATIONS BETWEEN BLOCKS

$B_3$: [(‘Perform X-ray’; ‘Perform blood test’; ‘Perform CT-scan’)]

The process description describes that after an X-ray is performed, three possible follow up procedures can occur and they can happen in different orders. For example, an X-ray can be followed by another X-ray or a blood test and/or a CT-scan. Using guideline DBG2 it can be concluded that all three activities ‘Perform X-ray’, ‘Perform blood test’ and ‘Perform CT-scan’ must be modeled in the same declarative block.

Concerning groups $G_4$: (‘Send test results’; ‘Evaluate test results’) and $G_5$: (‘Start normal treatment’; ‘Start emergency treatment’), the following block can be defined:

$B_4$: [(‘Send test results’; ‘Evaluate test results’; ‘Start normal treatment’; ‘Start emergency treatment’)]

The activities ‘Send test results’ and ‘Evaluate test results’ follow each other but the specialist does not wait until all results have been sent to start evaluating. This implies that ‘Send test results’ and ‘Evaluate test results’ can happen more than once and in different sequences. Given the fact that they are also in the same group, guideline DBG1 can be used, and these two activities are placed inside the same declarative block.

After evaluating the results, emergency treatments can be started. Activity ‘Start emergency treatment’ can be a successor of activity ‘Evaluate test results’ and can happen more than once. Again, guideline DBG1 can be used and activity ‘Start emergency treatment’ is placed in the same declarative block as activities ‘Send test results’ and ‘Evaluate test results’.

After analyzing the activities using the guidelines, only one activity has not been classified yet: ‘Start normal treatment’. This activity can also be a successor of activity ‘Evaluate test results’ but can only happen once. If it does not occur, then activity ‘Start emergency treatment’ must happen. For the time being this activity is placed inside the same declarative block as activity ‘Start emergency treatment’.

After analyzing the relations between the activities it can turn out that this activity should be modeled using an imperative approach. In general, it is advised to iterate the three steps to make sure the end model is correct. Figure 5.7 shows the different blocks identified above including their activities.

![Figure 5.7: The first attempt to divide of the activities into blocks](image-url)

5.5 Define relations between blocks

After the blocks are identified and the groups are adjusted (if needed), the structure of the main or parent-process is defined. This is done by defining relations between the identified blocks. In this section, a closer look is taken at how blocks can behave towards each other and how that can be modeled using the elements of the hybrid modeling language. At the end of this section, the principles presented are applied to the case study.
CHAPTER 5. METHOD AND GUIDELINES

The starting point for this step is to look at two blocks and determine the behavior towards each other. If more than two blocks are identified, first determine the behavior between two of them. Then iterate the step by applying the principles presented in this section to a third block and the result of the two blocks. If none of the options fit for a pair of blocks, it can be the case that the blocks are not related. In this case, iterate this step with another combination of blocks (if any). Note that, if an imperative parent-process is identified, all blocks should be connected eventually. On the other hand, if a declarative parent-process is considered, blocks do not have to be connected.

There are three possible combinations of two blocks, considering imperative and declarative blocks: two imperative blocks, two declarative blocks or one imperative and one declarative block. The combination of an imperative and a declarative block is considered the same as the other way around. The blocks can be either connected to or combined with each other. The options of how and when to connect or combine two blocks are presented in this section. Table 5.1 shows which option suits what combination of blocks.

<table>
<thead>
<tr>
<th></th>
<th>Connect A</th>
<th>Connect B</th>
<th>Combine C</th>
<th>Combine D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Two declarative blocks</td>
<td>v</td>
<td>v</td>
<td>v</td>
<td>v</td>
</tr>
<tr>
<td>Two imperative blocks</td>
<td>v</td>
<td>v</td>
<td>v</td>
<td>v</td>
</tr>
<tr>
<td>One imperative &amp; one declarative block</td>
<td>v</td>
<td>v</td>
<td>x</td>
<td>v</td>
</tr>
</tbody>
</table>

Table 5.1: Possibilities of modeling different blocks

Recall the semantics of the hybrid modeling language (chapter 4): a block (as a whole) behaves as an activity of its parent-process. Only when the block is executed, the activities inside are allowed to be executed.

5.5.1 Connecting two blocks

This subsection describes two ways to connect two blocks. Each option is described and supported with an example. If there is a difference in applying the option to different types of blocks, it is indicated and explained.

A. Use sequence flow

A sequence flow can be used to connect two blocks in the following situation: a process description describes that a process modeled in block1 as a whole must be fully completed before any activity of the process modeled in block2 may be executed. Also neither of these blocks is allowed to be executed more than once. Furthermore relations between activities inside these two blocks are not allowed.

Example: The registration process for an online shop must be fully completed before a customer could order an item from that shop. In this example, a block considering the process of registration and a block considering the process of ordering an item can be identified. These blocks are not allowed to happen multiple times. The activities inside these blocks can be modeled either with an imperative or declarative language. The way to model the behavior between these blocks is by using a sequence flow.

B. Use a Declare constraint

A Declare constraint can be used to connect two blocks if one of the rules of the Declare constraints apply to both blocks as a whole.

Example: A disease can be treated by undergoing either treatment1 or treatment2. Whatever treatment is chosen, the treatment should be executed more than once. Both these treatments have several internal
steps. These can be modeled either imperatively or declaratively. A Declare constraint that satisfies this behavior is the **not co-existence** constraint. This constraint applies to the treatments as a whole and does not consider the internal steps. Therefore this constraint can be used to connect the two blocks.

### 5.5.2 Combining two blocks

This subsection describes two ways to combine two blocks. Each option is explained and supported by an example. If there is a difference in applying these options to different types of blocks, it is indicated and explained.

#### C. Merge the two blocks

For this option, the activities inside the blocks are considered rather than the blocks as a whole.

It is not possible to combine an imperative and declarative block into one. The purpose of introducing the concept of a block was to separate the modeling elements of the imperative and declarative modeling languages. Therefore the combination of these two blocks is not allowed.

Two blocks of the same type can be merged into one. This is allowed if a process description prescribes that a mix of activities from both blocks must be able to be executed in one sequence. The following two examples will make more clear what happens with each type of blocks.

**Example of two declarative blocks:** Assume block 1 contains activities A, B and C and block 2 contains activities D and E. If it should be allowed to execute the sequence <A, D, B>, then block 1 and block 2 should be merged into one block.

**Example of two imperative blocks:** Consider the model in Figure 5.8. The first block (block 1) contains a sequence of activities A, B and C and the second block (block 2) of activities D and E. The dashed lines represent relations between the activities of the two blocks. The dashed line between activity C and D can be captured by connecting block 1 to block 2 with a sequence flow. The other dashed line cannot be captured by using a sequence flow or Declare constraint between the blocks. Therefore these two blocks should be combined into one block in order to allow for all desired behavior.

![Figure 5.8: An example of two blocks which need to be combined](image)

#### D. Place one block inside the other block

With this option, a whole block is placed into another block while the activities stay inside their own block. One of these blocks, e.g. block 1, is acting like the parent-process of block 2. Block 1 can be either a declarative or imperative modeled process. Regardless of its type, it contains a process which needs to be modeled separately. This process is modeled inside block 2.

All other options assume that both blocks are on the same layer of the process model. This option, however, creates hierarchy in the process model. An example of this option is given in the next subsection 5.5.3.

### 5.5.3 Concepts applied to case study

Recall that there are four blocks distinguished by using the guidelines of section 5.4 (Figure 5.7). This subsection explains the relations between these blocks by using the principles of this section.
Determining the relation between the two imperative blocks $B_1$ and $B_2$, Table 5.1 shows that there are four possible ways to do so. Looking at the process description, it states that after recording the symptoms and the decision is made to let the patient make a new appointment, the patient visits the specialist again. This pattern can be repeated a couple of times. The key point here, is that activity ‘Visit the specialist’ is related to both ‘Record the symptoms’ and ‘Make a new appointment’. This relation cannot be captured with a single sequence flow (A) or a Declare constraint (B). Nevertheless, it fits the description of merging the two blocks (C) perfectly, since a mix of activities of the two blocks is required. Therefore the imperative blocks $B_1$ and $B_2$ are combined into the new imperative block $B_{12}$.

Determining the relation between blocks $B_3$ and $B_4$, Table 5.1 points out that there are four possibilities for relating these two declarative blocks. Considering the option of using a sequence flow (A), it should be the case that all the testing activities must be done before the test results are sent to the specialist and treatments can start. According to the process description, this is not the desired behavior. Therefore the option of using a sequence flow to connect the two block is cannot be used.

Looking at option B, a Declare constraint could be able to capture the desired behavior specified in the process description. A desired sequence, for example, is to run tests, send the results, run more tests, evaluate results of these previous tests and decide to start the normal treatment. After the normal treatment is started, no more tests can be done. This behavior is too complex to be captured by a Declare constraint. Hence, the option of using a Declare constraint cannot be used either.

So, for the desired behavior, is a mix of activities of both blocks is required in one sequence. This behavior fits the description of merging two blocks (C). Therefore the declarative blocks $B_3$ and $B_4$ are combined into one new declarative block $B_{34}$.

Looking at the newly created imperative block $B_{12}$ and the newly composed declarative block $B_{34}$, Table 5.1 shows that there are only three possibilities to relate them. Recalling the process description, all the activities of the declarative block may only occur when the decision is made to require an X-ray. Since this is not the only option to choose from, this behavior cannot be modeled with a sequence flow (A) or Declare constraint (B). Merging the two blocks (C) is not an option, therefore block $B_{34}$ is placed inside block $B_{12}$ (D). Block $B_{12}$ acts like the parent-process of block $B_{34}$. Figure 5.9 shows the new distribution of the blocks.

5.6 Define relations between activities

After the block structure is created and the relations between the blocks are determined, the relations between the different activities can be determined. Imperative blocks require the sequence in which the activities must happen. Relations between activities in declarative blocks should be discovered and translated into constraints. For both these actions a subsection is dedicated. Subsection 5.6.1 explains how a sequence is discovered for an imperative block. Subsection 5.6.2 presents guidelines to discover
5.6. DEFINE RELATIONS BETWEEN ACTIVITIES

Declarative relations between activities and suggests constraints to model these relations. Subsection 5.6.3 continues with modeling the case study (section 5.1) using the specified guidelines. At the end, an actual hybrid model is presented.

5.6.1 Discover imperative relations

After the imperative blocks are discovered, the relations between the activities must be identified. This subsection will not have a list of guidelines to guide the user in different situations. The imperative way of modeling is commonly used and the basic constructs are not new. In general: Find the sequence flow and recognize what activities instantiate choices or parallel sequences and which activities belong to these choices or parallel sequences. Guiding the user to model the flow when all activities are already given is not part of the scope of this project.

5.6.2 Discover declarative relations

This subsection provides guidelines to identify relations between already identified declarative activities. It is assumed that the activities mentioned in the guidelines are declarative activities. A declarative activity is an activity that can happen multiple times if there are no constraints attached to the activity. The guidelines point the user in a certain direction to use a specific (group of) constraint(s). Every guideline is accompanied by an example which shows different ways to use the constraints. The guidelines are labeled with ‘DRG’ followed by a number, ‘DRG’ stands for ‘Declarative Relation Guideline’.

DRG1 “If the process description indicates an activity that should be the first or the last activity to be executed in the execution, then model these activities using the init or last activity respectively.”

The init- and last- activity are described in subsection 2.3.1. Only one of both types is allowed in a process. The reasoning behind this is that it is not possible to have two different activities be the first or last activity in a sequence. Another property of these activities is that they still can occur more than once, as long as they are the first or last activity to be executed. To prevent this repetition, the exactly-constraint can be used.

Example: Consider the models shown in Figure 5.10, they both have a first and last activity. When executing model (a), activity A should be executed first, after that it is not directly possible to execute C. Activity B can be executed at any time after activity A is executed and makes it possible for activity C to be enabled.

Model (b) shows an example with the last activity (C), which also has an exactly-constraint with \( n \) equal to one. For the behavior this means, whenever activity C happens, none of the activities are allowed to happen anymore and the model is finished. Also only traces where both A and B are executed are valid because of the co-existence-constraint.

Figure 5.10: Two models including the init and last activity

DRG2 “If the process description indicates an order between two activities which is only effective the first time of their execution, then model the relation between these activities using the precedence constraint.”
The precedence-constraint is the most used constraint according to the empirical investigation presented in [3.2.2]. It is the sequence flow of the declarative language. Whenever used with the exactly-constraint \((n = 1)\) the actual sequence flow can be mimicked.

**Example:** Consider the models in Figure [5.11]. Two situations are shown: One where the precedence-constraint is combined with the exactly-constraint and one shows the same situation without the exactly-constraint.

Model (a) shows behavior which is similar to a model with an XOR between C, A and B. Activity C must happen before either A or B can happen (precedence-constraint) when C happens, it can never happen again (exactly-1-constraint). Then either A or B may happen (not co-existence-constraint) and when that choice is made that activity can also happen only once (exactly-1-constraint). This is also exactly what happens when an XOR-gateway is placed in between A, B and C.

Model (b) does not have the exactly-constraints, there as well holds that C must happen before either A or B can happen, when C happens it is still allowed to occur more times. When a choice is made between A and B, also that activity can happen multiple times.

![Figure 5.11: Two possible models using precedence and exactly constraints](image)

**DRG3** “If the process description indicates an activity or sub-process which can be executed a specific number of times, then model this activity or sub-process using one of the existence constraints (exactly, existence, absence).”

With the existence constraints it can be specified that an activity can be executed an exact number of times or must fall within a specified range. The existence-constraint specifies that an activity must happen at lest \(n\) times while the absence-constraint specifies that an activity may happen at most \(n\) times. The exactly-constraint specifies that an activity should happen exactly \(n\) times. These constraints can also influence other activities which are related to the ones with one of these constraints.

**Example:** Consider the two examples shown in Figure [5.12] which uses the exactly- and absence-constraint in combination with other constraints.

Model (a) shows the use of the exactly-constraint in combination with the last-activity. In this model, activity C is executed last and once which means that after that execution none of the other activities can be executed. Without the exactly-constraint a last-activity can happen multiple times but always has to be the last one.

Model (b) repeats the sequence \(<A, C, B>\) with a maximum of three times. The constraints used are alternate succession (between A and B) and chain succession (between A and C). Initially A has to happen because succession-constraints are used, afterward C must happen because of the direct relation (chain succession-constraint). Thereafter only B can happen because it has to occur before A can happen again (alternate succession-constraint). When B has happened only A can be executed and the sequence starts again, it can be repeated at most three times.

**DRG4** “If the process description indicates that certain activities can happen multiple times until another activity has happened, then model the relations between these activities using the not succession constraint.”

The not succession-constraint ensures that an activity cannot be executed after another activity. This constraint can be used to prevent a loop e.g. prevent a system to place two orders.
5.6. DEFINE RELATIONS BETWEEN ACTIVITIES

Example: Consider the examples shown in Figure 5.12 where the not succession-constraint is used. Model (a) allows activities A and B to happen in any order and any number of times until activity C is executed, thereafter both A and B are not allowed to be executed anymore. Model (b) shows another variant of the not succession-constraint, the not chain succession-constraint. This constraint is more relaxed with respect to the not succession-constraint. In this example activities A and B can happen any number of times, and in any order, and when C is executed, B cannot be executed directly after (A can). When, after activity C is executed, activity A or C is executed it is again allowed to execute B. This constraint only captures what happens exactly after an activity is executed.

Figure 5.12: Two possible models using the exactly and absence constraints

Example: Consider the examples shown in Figure 5.13 where the not co-existence-constraint is used. Model (a) allows activities A and B to happen in any order and any number of times until activity C is executed, thereafter both A and B are not allowed to be executed anymore. Model (b) shows another variant of the not co-existence-constraint, the not chain co-existence-constraint. This constraint is more relaxed with respect to the not co-existence-constraint. In this example activities A and B can happen any number of times, and in any order, and when C is executed, B cannot be executed directly after (A can). When, after activity C is executed, activity A or C is executed it is again allowed to execute B. This constraint only captures what happens exactly after an activity is executed.

Figure 5.13: Two possible models using ‘not (chain) succession’ constraints

DRG5 “If the process description indicates two activities that are not allowed to happen together, then model the relation between these activities using the not co-existence constraints.”

The not co-existence-constraint does not capture restrictions such as that one of the two activities can only happen once and the other activity multiple times more than one.

Example: Consider the examples shown in Figure 5.14 where the not co-existence-constraint is combined with other constraints. Model (a) requires that C should happen before the choice is made to execute A or B. A can only be executed once, when A is chosen both A and B cannot be executed anymore. It is still possible to execute C. When B is chosen, B and C can be executed any number of times and in any order. Model (b) does not restrict the number of times an activity can be executed. In this model the choice between A and B influences the activities which can be executed after them. When A is chosen, C and D can still be executed. When B is chosen, only D can be executed after that. This is due to the fact that A and C are connected with a precedence-constraint, which means that before the first time C can happen A should have happened at least once. This model shows that the not co-existence-constraint can also have the effect of excluding two activities from the execution.

Figure 5.14: Two different models including the not co-existence constraint.

DRG6 “If the process description indicates that an activity cannot happen twice in an ordering without a
specific other activity somewhere in between, then model the relations between these activities using one of the alternate constraints (alternate precedence, alternate response, alternate succession).”

The three alternate constraints permit different activities to be executed twice in an ordering.

Example: Consider the models in Figure 5.15 they show examples of all three constraints which sequences are permitted when using that constraint.

In model (a) it is not possible to execute two B’s after each other without an A somewhere in between.
In model (b) it is not possible to execute two A’s after each other without a B somewhere in between.
In model (c) it is not possible to execute two A’s after each other without a B somewhere in between and it is not possible to execute two B’s after each other without a A somewhere in between. It is always possible for activity C to happen in any of the models. For more explanation about these constraints see also subsection 2.3.1.

Figure 5.15: The three different alternate constraints

5.6.3 Concepts applied to the case study

This subsection continues with the example described in section 5.1. Two blocks were identified, one imperative and one declarative. The relation between these blocks was that the declarative block should be seen as a sub-process of the imperative block. In this subsection the detached activities within the blocks are connected to each other using declarative constraints or the sequence flow. The guidelines introduced in subsection 5.6.2 are used to identify the relations between the declarative activities.

Starting with the imperative block, a choice construct was already identified. Activity ‘Record the symptoms’ initiates the choice between ending the process, the activity ‘Make a new appointment’ and the declarative block (called ‘Require an X-ray’ in this subsection in stead of B34). The process description provides the information that this choice is an exclusive choice.

Following the process description this choice can only happen if the patient has visited the specialist, and that is only possible when an appointment is made. Making an appointment is also the first activity of the process.

According to the process description: when the choice is made to let the patient make a new appointment, the specialist must be visited. When the choice is made to end the process, the process must be ended. It also mentions that after a normal treatment is finished, or the required amount of emergency treatments are performed, the process must end. Figure 5.16 shows the imperative block of the model including one start and one end event.

According to the process description an X-ray is the first test that should happen. Using guideline DRG1 activity ‘Perform X-ray’ is an init-activity. That same guideline can be used to identify that ‘Start normal treatment’ and ‘Start emergency treatment’ should be last-activities. However only one last-activity can exist in a declarative block. Whenever one of the two becomes a last-activity, that activity should always happen. Since a choice has to be made between the two activities this behavior is not desired. To conclude, none of the two is modeled as a last-activity.

The activity ‘Perform X-ray’ is the first activity of the declarative block. According to the process
5.6. DEFINE RELATIONS BETWEEN ACTIVITIES

Figure 5.16: The model of the imperative block

description there are three possible follow-up tests to perform: perform another X-ray or perform a blood-test and/or perform a CT-scan. Whenever a new X-ray is made the same choice stays applicable. It is not possible to perform two blood-tests after each other without performing an X-ray in between. The same holds for the CT-scan. According to guideline DRG6, one of the alternate constraints has to be used. In this particular case the alternate precedence constraint is the one to use, since it should not be possible to perform two blood-tests or CT-scans (B in the example) without performing an X-ray in between (A in the example).

Before a test result can be sent to the specialist, at least one test should have been performed. It does not matter which test, but since the X-ray is performed first, it is most convenient to take that test to be the one which has to be performed before a result can be sent. This relation is only effective the first time, after that, also results of blood tests and CT-scans should be send to the specialist. In this situation, guideline DRG2 can be used to identify that the precedence constraint should be used.

According to the process description, at least one test result should be sent to be evaluated and, after that, more results can be sent and evaluated in any order. For example, one test result can be evaluated more than once. It is also not specified that one test cannot be sent more than once. In this situation also guideline DRG2 can be used to identify that the precedence constraint should be used between activities ‘Send test results’ and ‘Evaluate test results’.

As mentioned earlier only one of the activities ‘Start normal treatment’ and ‘Start emergency treatment’ may be performed. Using guideline DRG4 it is concluded that a not co-existence constraint must be used to model this behavior. Activity ‘Start normal treatment’ may only happen once in this process, using guideline DRG3 it is decided to use the exactly constraint with \( n \) equals 1.

The process description states that before any treatment is started either a serious problem should be indicated during the evaluating of the test results, or after evaluating all test results no serious problem could be identified. So at least one test result should be evaluated before any treatment may ever start. Whenever one test result is evaluated, depending on whether or not a serious problem is detected, treatment(s) can be started. According to guideline DRG2 the precedence constraint should be used.

At this moment almost all behavior is captured by the model except whenever a normal treatment is started no new test can be performed, or test results can be sent or evaluated. In principle, all activities are allowed to happen, but if a normal treatment is started none of them may happen ever again. According to guideline DRG4 this behavior should be modeled with the not succession constraint. In this case, a not succession constraint should be modeled between ‘Start normal treatment’ and all other activities except ‘Start emergency treatment’. Since the model becomes less readable when this is accomplished, it is a typical case where a user should interact with the model. The user should terminate the declarative block (by executing the Finish activity) when ‘Start normal treatment’ or the required number of ‘Start emergency treatments’ are executed. In principle the exactly constraint with \( n \) equal to 1 is not necessary anymore, because after the activity is executed none of the other activities is allowed to be executed.
Figure 5.17 shows the final hybrid model created from the process description (section 5.1) using guidelines specified in sections 5.4 and 5.6. The graphical notation for a declarative block introduced in section 4.2 is used.
In the previous chapters the analysis of the problem and the design of the approach were established. In this chapter, the implementation and evaluation phase is described. An empirical evaluation of a set of guidelines was conducted by means of an experiment. The guidelines that were tested, are the ones that help the user to choose between placing activities in an imperative or declarative block.

The experiment is described in the following sections. Section 6.1 elaborates on the planning of the experiment (e.g. which different tasks are created, which group of subjects is approached, what type of tool is used). Section 6.2 captures the preparation, execution and data validation of the experiment. The analysis of the data is displayed in section 6.3 followed by the interpretation of the results in section 6.4. Finally, the conclusions drawn are presented in section 6.5.

### 6.1 Experimental definition and planning

In this section all aspects that are included in the experimental design are described. Basically, an overview is given of how the experiment is put together and what is tested. Figure 6.1 shows the independent variables and the dependent variables of this experiment.

**Figure 6.1: Variables tested in the experiment**

**Factor and Factor levels.** The factor considered in this experiment is presence of guidelines with two factor levels. The factor levels considered are guidelines versus no guidelines.

**Subjects.** The subjects of this experiment are students enrolled in classes on business process management (1BM05) at Eindhoven University of Technology (TU/e). The experiment subjects are expected to
be familiar with creating process models from process descriptions using an imperative language. It is not expected that they have heard of, read or created declarative models.

**Objects.** Three process descriptions are used as the objects for this experiment. Two of the three descriptions are based on (simplified) real life situations and one of them is based on a declarative model of the case study developed in paper [9]. All these three process descriptions contain both declarative and imperative elements since the purpose of the experiment is to identify both imperative and declarative blocks.

**Tasks.** A task is created for each process description. Figure 6.2 shows how such a task looks like. Every process description task starts with a counting sub-task. This sub-task is followed by four groups of sub-tasks. Each group consists of the three sub-tasks shown in Figure 6.2 on the right. A differentiation is made between the subjects with the guidelines and the subjects without the guidelines regarding the content of the group of sub-tasks. Summarizing, each subject receives three process description tasks. These consist of a counting sub-task, followed by four groups of different sub-tasks. The next enumeration explains the content of each sub-task created for the experiment.

1. **Counting sub-task.** This should be a very simple sub-task about counting the number of activities of a particular (sub-)process. There is one of these sub-tasks for every process description. The purpose of this sub-task is to let the participants read the process description carefully. An example of a counting sub-task is the following: “How many activities can happen during a meeting?”, with the next four answers to choose from: 3, 4, 5 and 7.

2. **Classification sub-task (imp/decl).** This is the most important sub-task of the experiment. Four of these are created per process description. Such a sub-task contains a group of activities which has to be classified as either an ‘imperative block’ or a ‘declarative block’. An extra option ‘I don’t know’ is possible to avoid guessing. An example of such a task is the following: “To which type of block does the group [Send meeting request, Send agenda, Hold meeting, Plan another meeting] belong?”. The three answers to choose from are: ‘imperative’, ‘declarative’, ‘I don’t know’.

3. **Classification sub-task (main/sub).** This is the ‘dummy’ sub-task for the group without the guidelines. They have to classify a group of activities as either part of the main process, part of a sub-process or representing a sub-process. To give an example of such a classification sub-task: the question asked is “Please finish the following sentence: Activity ‘Hold meeting’...”. The four answer options are ‘belongs to the main process’, ‘belongs to a sub-process’, ‘is representing a sub-process’ and ‘I don’t know’.

4. **Mental effort sub-task.** After each classification sub-task (imp/decl), the participants are asked to

---

1 All material used during the experiment can be found in A.6.3
6.1. EXPERIMENTAL DEFINITION AND PLANNING

rate their mental effort for performing that task. The answer can be chosen from a 7-point Likert scale (1: very simple, 7: very difficult).

5. **Guideline specification sub-task.** After each classification sub-task (imp/decl), the group with the access to the guidelines should specify which guideline(s) have been used to answer that sub-task. Besides the different guidelines also the option ‘None’ is possible.

**Response variables.** To measure the effect of using the guidelines while placing activities in imperative or declarative blocks, three response variables are used. These are:

1. **Accuracy.** This is measured by the number of correctly answered classification sub-tasks (imp/decl).

2. **Perceived usefulness.** This is measured by a post task questionnaire. This questionnaire holds several questions about the usefulness of the guidelines. To be able to test the consistency of the subjects answers, questions are stated slightly different but are asking the same. The questions in three categories were asked: perceived ease of use (PEOU), perceived usefulness (PU) and Intention to use (ItU). This response variable is only measured by the group subjects who had access to the guidelines.

3. **Mental effort.** This is measured by the subjects’ ratings on how difficult they found the classification sub-task (imp/decl).

**Hypotheses.** For the response variables accuracy and mental effort, a null hypothesis is associated. There is no null hypothesis for the response variable perceived usefulness because it is only measured by the group who had access to the guidelines.

1. **Null Hypothesis** $H_{0,0}$: There is no significant difference (in terms of accuracy) in whether or not using the guidelines by completing the classification sub-tasks (imp/decl).

2. **Null Hypothesis** $H_{0,1}$: There is no significant difference (in terms of mental effort) in whether or not using the guidelines by completing the mental effort sub-tasks.

**Parameters.** In addition to the described factors other variables can affect the response variables under examination and therefore need to be controlled. For this experiment, the three main parameters that can influence the outcome are: understanding of the English language, modeling experience and other personal factors. All these three parameters are dealt with using a questionnaire at the beginning or end of the experiment including questions addressed to the three parameters mentioned.

**Experimental design** The design type of this experiment is *One factor with two treatments*. Within this design the subjects are randomly assigned to the treatment, therefore it is also completely randomized. Moreover the subjects are evenly distributed over the two treatments, which makes the design balanced. Summarizing, the experiment is a randomized, balanced design with one factor and two treatments.

**Instrumentation and data collection procedure** The participants received an envelop with a scrap paper, a written explanation about the experiment and an introduction to imperative and declarative modeling. Half of the participants received the guidelines and an explanation of each one (group 2). The other half received an explanation about main and sub-processes (group 1). This extra information was also received on a paper inside the envelope. The actual experiment was conducted using the Cheetah Experimental Platform[^2] which guided them through the different tasks and questionnaires. The tool also automatically logged the given answers, as well as the time that spent on answering the questionnaire questions.

[^2]: [http://bpm.q-e.at/?page_id=56]
6.2 Experimental operation

The aspects of the content of the experiment are determined and presented in the previous section. In this section, the preparation, execution of the experiment are described. Also, a small part is dedicated to data validation.

**Experimental Preparation.** Three process descriptions were created for the empirical test. It was made sure that every process description had both imperative as declarative activity groups (blocks). Figure 6.3 presents the structure of the experiment. Group 1 is the group without the guidelines and group 2 the group with the guidelines. The first flowchart shows the activities that group 1 need to execute and the second flowchart shows the activities that are created for group 2. To ensure the overall understandability of the experimental setup, to estimate the total time of the experiment and to make sure the process descriptions were not too simple or too difficult a pre-test was conducted. In this pre-test both versions were tested.

![Figure 6.3: Structure of the experiment](image)

**Experimental Execution.** The experiment was conducted on the 18th of December, 2013. A total of 142 students from Eindhoven University of Technology participated in this experiment. The participants had a time-frame of 105 minutes to complete the experiment, though the majority finished the experiment within the hour.

**Data Validation.** Not all 142 subjects were able to finish the experiment properly or send their data to the right location. Four students could not deliver their data due to several technical problems. After all, data from 138 students was collected for the data analysis. Of these 138 students, 70 did not have access to the guidelines (group 1) and 68 had access to the guidelines.

6.3 Data analysis

This section describes the data analysis of the data gathered throughout the experiment. In subsection 6.3.1 the process of data cleansing is discussed. Subsection 6.3.2 presents the results found for the process description tasks, which include the classification sub-tasks (imp/decl) as well as the specification and mental effort sub-tasks. Lastly, subsection 6.3.3 describes the results of the post-task questions. The calculations and statistic test are performed with Microsoft Excel 2010 and IBM SPSS Statistics version 22.
6.3. DATA ANALYSIS

6.3.1 Data cleansing and demographics

Data cleansing is applied to maintain a set of subjects and tasks that fit to the predefined profiles (as described in [15]). Outliers should be removed from the data set, as well as sub-tasks that are not satisfying the task-profile. This subsection presents the task-profile and which question is removed from the data set and why. It also explains which outliers are found and removed for what reason.

Task profile and removed sub-tasks

Tasks, that are included in the results, should have the following profile: Every classification sub-task must have only one correct answer and at least one guideline must exist which leads to the correct answer. In hindsight, the second classification sub-task of the first process description (question 2) did not satisfy the profile. This question can be found in Appendix A.6.3. The group of activities presented in that question could be modeled with the imperative as well as the declarative modeling languages. There was also no guideline which led to the correct answer. Therefore, the data of this sub-task has been removed from the data set.

Outliers based on modeling experience

Recalling the profile for the subjects, presented in section 6.1, they should be familiar with creating imperative process models and are not supposed to have any experience with declarative modeling languages. The purpose of the pre-task questionnaire is to identify subjects that do not satisfy this profile. Table 6.1 shows that there were no imperative process modeling experts attending the experiment. In total there were three participants who had ever created declarative models. One of group 1 (created five declarative models) and two of group 2 (created 30 and four models respectively). The participant who created 30 declarative models in the last year gave the feedback (in the feedback dialog at the end of the experiment) that he knew too much about Declare to make the experiment become a real experiment. Therefore, this participant is marked as an outlier and the scores of this participant are removed from the set. (Though his results were not exceptionally good compared to the other participants.)

<table>
<thead>
<tr>
<th>No guidelines (1)</th>
<th>Guidelines (2)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Years of modeling experience</td>
<td>0</td>
</tr>
<tr>
<td>Nr. process models read*</td>
<td>0</td>
</tr>
<tr>
<td>Nr. process models created*</td>
<td>0</td>
</tr>
<tr>
<td>Average nr. activities**</td>
<td>0</td>
</tr>
</tbody>
</table>

* these numbers are based on the last twelve months.
** these numbers represent the average number of activities the read and created process models have.

Table 6.1: Modeling experience

Outliers based on answers to the classification sub-task (imp/decl)

Subjects, who did not understand the questions at all, have to be identified and their results have to be removed from the data set. However it is hard to identify these subjects. One method for identifying these subjects is to count the number of ‘I don’t know’ answers they filled in. One subject stood out in this category, 7 out of 11 classification sub-tasks were answered with ‘I don’t know’. It is safe to conclude that this subject did not understand the tasks of the experiment properly enough. Therefore the data of this subject has been removed from the data set.

43
After the data cleansing there are 11 questions, 66 subjects that received guidelines and 70 subjects that did not receive guidelines left.

### 6.3.2 Process description tasks

This subsection is about the main part of the experiment. This comprises the classification sub-tasks (imp/decl), the guideline specification sub-tasks and the mental effort tasks. A number of statistical tests were performed on the data. Their outcomes and the reason for using these are explained.

#### Correct vs. Incorrect

The correct answers to all questions has been set before the experiment. The same holds for the guideline(s) used to come to the correct answer. These answers, specified before the experiment, are used to classify the answers of the participants as ‘correct’ or ‘incorrect’. A total score (from 0 till 10) has been computed for each participant by dividing the number of correctly answers questions by the total number of questions (11). The averages of the groups are as follows: the group without the guidelines scored a 7.3, while the group with the guidelines scored a 7.5.

Table 6.2 presents percentages of the participants who answered the specific questions correctly (participants who answered with ‘I don’t know’ are excluded from the total). The tables show these percentages for both groups (with and without guidelines). The correct answers are stated with ‘imp’ for imperative and ‘decl’ for declarative.

<table>
<thead>
<tr>
<th>Question</th>
<th>Process description 1</th>
<th>Process description 2</th>
<th>Process description 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 3 4 5 6 7 8 9 10 11 12</td>
<td>5 6 7 8 9 10 11 12</td>
<td>9 10 11 12</td>
</tr>
<tr>
<td>Answer</td>
<td>imp decl decl imp imp decl decl decl imp imp imp</td>
<td>imp imp decl decl decl</td>
<td>decl imp imp imp</td>
</tr>
<tr>
<td>No guidelines (1)</td>
<td>80% 75% 71%</td>
<td>88% 72% 62% 71%</td>
<td>67% 76% 77% 87%</td>
</tr>
<tr>
<td>Guidelines (2)</td>
<td>80% 95% 67%</td>
<td>77% 68% 65% 66%</td>
<td>72% 81% 82% 88%</td>
</tr>
<tr>
<td>Difference</td>
<td>0% 20% 4%</td>
<td>11% 4% 3% 4%</td>
<td>5% 5% 5% 1%</td>
</tr>
<tr>
<td>Significant?</td>
<td>No Yes No</td>
<td>No No No No</td>
<td>No No No No</td>
</tr>
</tbody>
</table>

Table 6.2: Guidelines vs. No Guidelines

Also the differences between the percentages per question of the two groups are calculated. A binomial test is used to test whether this difference is significant or not. A significance level of 0.05 was taken into account and the hypotheses used for this test are:

\[H_0 : p_{\text{guidelines}} = p_{\text{no guidelines}}\]  

\[H_1 : p_{\text{guidelines}} > p_{\text{no guidelines}}\]

As the table shows, there is only one question which is significantly answered better by the group with the guidelines (2) compared to the other group. Overall, group 2 answered six questions better than group 1. However, the Wilcoxon (p-value of 0.646) and sign-test (p-value of 0.754) did not reveal that the difference between the overall results of the two groups is significant. Therefore the null hypothesis defined before the experiment (Section 6.1) cannot be rejected.
6.3. DATA ANALYSIS

Level of Difficulty

The participants had to rate their mental effort after each classification sub-task (imp/decl) on a scale
from 1 (very difficult) till 7 (very simple). Table 6.3 shows the averages per question for both groups.

<table>
<thead>
<tr>
<th>Question</th>
<th>Process description 1</th>
<th>Process description 2</th>
<th>Process description 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 3 4</td>
<td>5 6 7 8</td>
<td>9 10 11 12</td>
</tr>
<tr>
<td>Answer</td>
<td>imp decl decl</td>
<td>imp imp decl decl</td>
<td>decl imp imp imp</td>
</tr>
<tr>
<td>No guidelines (1)</td>
<td>4.26 4.04 4.23</td>
<td>4.17 4.29 4.57 4.29</td>
<td>4.79 4.33 5.44 5.47</td>
</tr>
<tr>
<td>Guidelines (2)</td>
<td>4.27 4.41 4.59</td>
<td>4.11 4.59 4.20 4.14</td>
<td>4.02 4.00 4.03 4.55</td>
</tr>
</tbody>
</table>

Table 6.3: Level of difficulty

Since the Shapiro-Wilk test revealed that the data is not normally distributed, a Wilcoxon and sign tests
have been conducted between group 1 and 2. There is no significant difference found with a significance
level of 0.05 (p-values of 0.209 and 0.774 respectively). Therefore the second hypothesis defined in section
6.1 is also rejected.

A Wilcoxon test has been conducted for the separate averages to see whether they differ significantly from
the neutral value (4). A significance level of 0.05 has been used. Whenever a value is significantly different
from the neutral value, the cell containing that value is colored green. The corresponding p-values can
be found in appendix D.

As the table shows, group 1 (without guidelines) found all the questions of the third process description
easier than ‘neutral’. The averages of the last two questions are even significantly different from the value
5 (rather simple). Therefore, these questions were found even easier than the other two by the subjects.
This is, however, not in line with the percentage that answered the question correctly.

For more insight in the data about the level of difficulty, Tables 6.4 and 6.5 have been created. They
present the mental effort rates of the participants sorted on whether they answered the question correctly or not.
The group without guidelines has a considerably bigger number of significantly different averages when answering correctly compared to the averages shown in Table 6.3. Question 12 is also
still significantly different from the rather simple value (5). It can also be seen that the reason for the
slightly negatively rated questions (10, 11) of Table 6.3 is the rating of the participants who incorrectly
answered the questions (or even answered with ‘I don’t know’). The rating of the participants who incorrectly
answered question 11 is even significantly less than the neutral value (4). There are no significance
calculations conducted for the ‘I don’t know’-answers, since the maximum number of ‘I don’t know’s per
question was four and an average of less than 0.5.

<table>
<thead>
<tr>
<th>No guidelines</th>
<th>Process description 1</th>
<th>Process description 2</th>
<th>Process description 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Question</td>
<td>1 3 4</td>
<td>5 6 7 8</td>
<td>9 10 11 12</td>
</tr>
<tr>
<td>Answer</td>
<td>imp decl decl</td>
<td>imp imp decl decl</td>
<td>decl imp imp imp</td>
</tr>
<tr>
<td>Correct</td>
<td>4.32 4.10 4.33</td>
<td>4.07 4.29 4.55 4.23</td>
<td>4.79 4.32 5.47 5.53</td>
</tr>
<tr>
<td>Incorrect</td>
<td>4.00 4.00 4.10</td>
<td>5.13 4.32 4.55 4.46</td>
<td>4.78 4.35 5.31 5.33</td>
</tr>
<tr>
<td>I don’t know</td>
<td>4.25 2.00 3.00</td>
<td>3.67 4.00 6.00 2.00</td>
<td>6.00 4.50</td>
</tr>
</tbody>
</table>

Table 6.4: Level of difficulty grouped by correct and incorrect answers [no guidelines]
CHAPTER 6. EMPIRICAL EVALUATION

Usage of the guidelines

This paragraph focuses on whether or not the intended guideline was used to answer the questions. For the record, the data showed in this paragraph is only from group 1 (with guidelines). Table 6.6 shows the percentages of participants who used the intended guideline(s) (‘I don’t know’-answers excluded). It can be seen that in 4 out of 11 questions the majority used the intended guideline(s). Also the intended guidelines are shown in the table. The combination of IBG1 and IBG4 were used by the majority when needed. Based on these numbers, it cannot be concluded that a particular guideline was not understood.

Table 6.6: Usage of intended guideline

<table>
<thead>
<tr>
<th>Question</th>
<th>Process description 1</th>
<th>Process description 2</th>
<th>Process description 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Answer</td>
<td>Intended guideline(s)</td>
<td>Used</td>
</tr>
<tr>
<td></td>
<td>imp decl decl</td>
<td>IBG4 DGB1 DBG1</td>
<td>50% 66% 30%</td>
</tr>
<tr>
<td>Correct</td>
<td>4.38 4.45 4.68</td>
<td>IBG4 IBG3 DBG1 DBG5</td>
<td>47% 47% 41%</td>
</tr>
<tr>
<td>Incorrect</td>
<td>3.85 4.67 4.41</td>
<td>DBG2 DBG1 IBG4</td>
<td>45% 63% 65% 46%</td>
</tr>
<tr>
<td>I don’t know</td>
<td>1.00 1.00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6.5: Level of difficulty grouped by correct and incorrect answers [guidelines]

Another examination of the data revealed the figures of Table 6.7. It covers two groups: One that used the intended guideline(s) for the given question and one that did not use the intended guideline(s). Of these groups the percentages who answered the questions correctly are presented. To give an example: for question 1, 94% of the participants who used the intended guideline(s) answered that question correctly and 67% of the participants who did not use the intended guideline(s) answered the question correctly. Participants who answered with ‘I don’t know’ were excluded.

Table 6.7: Correctness vs. Usage of intended guideline

<table>
<thead>
<tr>
<th>Question</th>
<th>Process description 1</th>
<th>Process description 2</th>
<th>Process description 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Answer</td>
<td>Intended guideline(s)</td>
<td>Used</td>
</tr>
<tr>
<td></td>
<td>imp decl decl</td>
<td>IBG4 DGB1 DBG1</td>
<td>94% 95% 85%</td>
</tr>
<tr>
<td>Correct</td>
<td>4.38 4.45 4.68</td>
<td>IBG4 IBG3 DBG1 DBG5</td>
<td>100% 98% 92% 96%</td>
</tr>
<tr>
<td>Incorrect</td>
<td>3.85 4.67 4.41</td>
<td>DBG2 DBG1 IBG4</td>
<td>97% 95% 95% 100%</td>
</tr>
<tr>
<td>I don’t know</td>
<td>1.00 1.00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6.6: Usage of intended guideline

Table 6.7: Correctness vs. Usage of intended guideline
6.3. DATA ANALYSIS

The percentages of participants, who answered the questions correctly using the intended guideline(s), are very high. The differences between the two groups have been calculated and it has been tested whether or not these differences are significant. This is conducted by means of another binomial test. A significance level of 0.05 is taken into account and the following hypothesis are used:

\[ H_0 : p_{\text{intended guideline}} = p_{\text{no intended guideline}} \]  
\[ H_1 : p_{\text{intended guideline}} > p_{\text{no intended guideline}} \]  

Almost all questions were significantly answered better by the group who used the intended guideline. The exception is question 3, the reason is that it has been answered correctly by the majority of both groups anyway.

Moreover, a significant correlation is found between the number of correctly answered questions and the number of intended guidelines used. The Spearman’s rho correlation coefficient had a value of 0.429 which is significant at the level of 0.01 (2-tailed).

No use of guidelines

During the data analysis the observation has been made that a number of participants exceptionally often used the answer ‘None’ to a question about which guidelines the participant used. It seems that these students did not use the guidelines for answering the classification questions. Table 6.8 shows percentages of the correctly answered questions per group. The first group consists of 59 participants and the second group of 7 participants.

<table>
<thead>
<tr>
<th>Question</th>
<th>Process description 1</th>
<th>Process description 2</th>
<th>Process description 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1  3  4</td>
<td>5  6  7  8</td>
<td>9  10 11 12</td>
</tr>
<tr>
<td>Answer</td>
<td>imp  decl  decl</td>
<td>imp  imp  decl</td>
<td>decl  imp  imp  imp</td>
</tr>
<tr>
<td>&lt; 8 None’s (1)</td>
<td>81% 97% 66%</td>
<td>76% 69% 64% 68%</td>
<td>66% 80% 80% 88%</td>
</tr>
<tr>
<td>≥ 8 None’s (2)</td>
<td>71% 71% 71%</td>
<td>71% 57% 71% 29%</td>
<td>100% 71% 86% 71%</td>
</tr>
<tr>
<td>Difference</td>
<td>10% 25% 5%</td>
<td>5% 12% 7% 39%</td>
<td>34% 8% 6% 17%</td>
</tr>
<tr>
<td>Significant?</td>
<td>Yes Yes No</td>
<td>No Yes No Yes</td>
<td>Yes Yes No Yes</td>
</tr>
</tbody>
</table>

Table 6.8: Group ‘< 8 # None’s’ vs. group ‘≥ 8 # None’s’ correctness

A normality test (Shapkiro-Wilk) revealed that both data sets are not following a normal distribution. Therefore the Wilcoxon test has been performed instead of the one sample t-test. This test indicated no significant difference between the two sets (level of significance is 0.05 and the p-value is 0.197). So, the two groups did not score significantly different from each other over the eleven questions. The level of significance per question was again calculated using a binomial test (significance level 0.05). The following hypotheses were used:

\[ H_0 : p_1 = p_2 \]  
\[ H_1 : p_1 > p_2 \]  

Where \( p_1 \) is the success rate of the group who indicated to use the guidelines and \( p_2 \) is the success rate of the group who did not use the guidelines enough. As the table shows, in 6 out of 11 cases the group who used the guidelines did a better job and in one case the group who did not use the guidelines scored better.
Table 6.9 gives an insight into how these groups rated their mental effort. Only the group who said to use the guidelines has scored significantly different from the neutral value in some cases. This was again calculated with a Wilcoxon test. Questions 3, 6, 9 and 12 were found the easiest by the group who did not use guidelines.

<table>
<thead>
<tr>
<th>Question</th>
<th>Process description 1</th>
<th>Process description 2</th>
<th>Process description 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>imp decl decl</td>
<td>imp imp decl decl</td>
<td>decl imp imp imp imp</td>
</tr>
<tr>
<td>&lt; 8 None’s</td>
<td>4.32 4.43 4.58</td>
<td>4.10 4.58 4.19 4.20</td>
<td>3.92 3.95 4.00 4.53</td>
</tr>
<tr>
<td>≥ 8 None’s</td>
<td>3.86 4.29 4.71</td>
<td>4.14 4.71 4.29 3.57</td>
<td>4.86 4.43 4.29 4.71</td>
</tr>
</tbody>
</table>

Table 6.9: Group '< 8 # None’s' vs. group '≥ 8 # None’s' difficulty

6.3.3 Post-task questionnaire results

This subsection is about the results of the post-task questions. At the end of the experiment questions were asked about two topics: the guidelines and the process descriptions. These questions were posed positively and negatively, but they were all asking the same. The purpose of this approach was to test whether or not the participants were reading the questions well enough. The questions could be answered on a scale from 1 (strongly disagree) till 7 (strongly agree). To be able to calculate with the answers to these questions, the answers to the negatively asked questions were first transformed into the same scale as the positively asked questions.

The Cronbach’s alpha is used to calculate the reliability of the results of this questionnaire. The values found for the questions about the guidelines were quit positive (see Table 6.10). A value above 0.70 is considered as ‘good’. It can be said that the results are reliable. That means the results can be taken seriously. The calculated means are all above the neutral value (4) and two are significantly different from that neutral value. The answers to these questions scored negative on a normality test (both Kolmogorov-Smirnov and Shapiro-Wilk). Therefore a Wilcoxon signed ranks test was performed to calculate the level of significance.

<table>
<thead>
<tr>
<th>Category</th>
<th>Cronbach’s alpha</th>
<th>Mean</th>
<th>Significant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perceived ease of use</td>
<td>0.834</td>
<td>4.1</td>
<td>Undecided (0.656)</td>
</tr>
<tr>
<td>Perceived usefulness</td>
<td>0.787</td>
<td>4.7</td>
<td>Yes (0.000)</td>
</tr>
<tr>
<td>Intention to use</td>
<td>0.873</td>
<td>4.5</td>
<td>Yes (0.001)</td>
</tr>
</tbody>
</table>

Table 6.10: Usefulness guidelines

From the category perceived ease of use, one question was removed from the set. This question (Q14) scored relatively low compared to the other questions. It was a negatively asked question and probably the hardest question in terms of understandability of the whole set. The overall reliability should not suffer from removing a question. The Cronbach’s alpha value increases (from 0.827 to 0.834) after removing Q14 from the set. Therefore it is a legitimate action to remove that question from the set. There are other questions (of other categories) which, when they are removed, increase the Cronbach’s alpha value. However, these questions have not been answered extremely high or low or seemed difficulty put. Therefore none of the other questions have been removed from their sets.
6.4 Discussion

Besides questions about how useful the guidelines were, also questions are asked to the participants about the clearness and ambiguousness of the process descriptions. These questions have been asked to both groups and have been answered quite positively (see Table 6.11). However, the calculated Cronbach’s alpha’s are too low to consider the answers as reliable.

<table>
<thead>
<tr>
<th>Group</th>
<th>Cronbach’s alpha</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>No guidelines</td>
<td>0.491</td>
<td>5.4</td>
</tr>
<tr>
<td>Guidelines</td>
<td>0.532</td>
<td>5.6</td>
</tr>
</tbody>
</table>

Table 6.11: Clearness process descriptions

6.4 Discussion

This section discusses the interpretation of the results presented in section 6.3. The section is split into two parts. The first part presents the interpretations about the subjective data and the second about the objective data of the experiment.

6.4.1 Subjective data

The subjective data covers the answers to the pre-task, post-task and mental effort questions. Starting with the pre-task questions, it can be said that the audience was as skilled as expected. There was one exception, but its results are removed.

Looking at how the participants rated the difficulty of the classification questions it can be concluded that the group without the guidelines found the questions easier in comparison with the group with the guidelines. Even when the questions were answered incorrectly, they thought of it as rather simple. An explanation can be that the group with the guidelines had more information to process by answering a question. It is plausible that they read the question and all the guidelines with their examples to decide which one is the correct answer. Because they had more information and more to process, it could be the case that they found the questions more difficult compared to the other group. The latter group did not have any information that could help them find the right answer.

Comparing the data of Tables 6.2 and 6.3, it can be seen that the questions that were rated as easy by group 1 do not correspond to the questions that were answered better compared to group 2. Group 1 even rated the questions that were answered better by group 2 as easy. Summarizing, even if the participants indicated that they found the tasks easy, they made mistakes.

Another remarkable observation is that questions 10 and 11 of the third process description were found rather difficult compared to the scores by the group with the guidelines. Around 80% of the participants answered the questions correctly but they were still rated rather low. An explanation (for the difficulty rate) could be that there was some confusion in these questions (the questions can be found in appendix A.6.3). The activity group described is a sequences of activities followed by a choice moment, between using ‘liquid clog remover’ or not. However, the activity ‘do not use liquid clog remover’ was not part of the activity group. This has probably caused confusion. Often, when CPN-tools is used, both activities should be present to have a correct model but when BPMN is used such activity can be skipped by a direct arrow to the end event (see also Figure 6.4). Since it should be possible to choose whether or not to execute that activity, participants could have seen some declarative aspect in it. So, some confusion could have occurred with these questions, but it did not result in incorrect answers.
Concerning the post-task questions it can be concluded that the guidelines were found useful. An explanation the perceived ease of use was rated lower than the perceived usefulness could be that the guidelines were not written clear enough. So, the idea of having a set of guidelines during these type of questions seems pleasant and the participants have even said that they intend to use the guidelines more often.

### 6.4.2 Objective data

The group with the guidelines (group 2) answered 6 out of 11 questions better than the other group (group 1). This difference is not found to be significant. A notable observation, which can be made, is that question 3 was answered significantly better by the group with the guidelines. Regardless of the exact guideline that was used, 95% of the participants of group 2 answered it correctly. When looking at the question, it is a very typical declarative group of activities. However, the question was not rated as extremely simple.

By looking only at the differences in Table 6.2 it seems that the guidelines do not have any effect. As Table 6.7 shows, if a participant used the intended guideline the question is almost always answered correctly. This result is backed up with a significant correlation between the number of correctly answered questions and the number of intended guidelines used. The problem, however, is that the percentages of participants that used the right guideline are rather low. By only 4 out of 12 questions more than 50% chose the intended guideline. So, the participants seem to have a problem with picking the right guideline for the question, but if they managed to pick the right guideline, they answered the question correctly (in most cases).

Another distinction that has been made within group 2 is based on the number of questions the participant indicated to have used a guideline. A participant should have indicated to use a guideline in at least seven questions, to be sure that the participant actually used the guidelines. Surprisingly, the group who did not indicated to use the guidelines very often scored quite well. However, 6 out of 11 questions are answered significantly better when the participants indicated to use guidelines. One question was answered significantly better by the group who claimed that they did not use guidelines. When looking at this question it is a typical declarative group of activities (see Appendix A.6.3). The guidelines which should lead to this correct answer are very general.

Both types of data gave somehow the same results. It seems that there were problems with finding the right guideline for a question. This can explain the fact that the perceived ease of use was scored low. A possible cause can be the (possible) ambiguity of the guidelines. On the other hand when the right guideline was picked the questions were answered correctly. This can explain the high score of the perceived usefulness. Finally, a positive fact is that ‘intention to use’ scored significantly above the neutral value. It is important that the participants are willing to use the guidelines in the near future, it shows that they think of them as useful.
6.5 Conclusions & improvements

In this section the conclusions regarding the retrieved data and results of the experiment are presented. There are also suggestions given to improve the guidelines.

6.5.1 Conclusions

Both null hypotheses, stated beforehand, are not rejected. This implies that there is no reason to believe that there is a significant difference (in terms of accuracy and mental effort) between the group with the guidelines and the group without them. A conclusion based only on this fact would not cover the results of the experiment. Other interesting facts came up.

The guidelines are found useful by the participants, they even intend to use the guidelines with other similar case in the future. Whenever the guidelines were used, the answers to the questions were correct. Actually a significant correlation, between the number of correctly answered questions and the number of intended guidelines used, is found. However, the intended guidelines were not always found. Based on the judgement of the participants the guidelines were not that easy to use. It can be concluded that the guidelines are too ambiguous but helpful when understood.

6.5.2 Improvements

Improvements of the clearness and formulation of the guidelines could improve the ease of use. Two suggestions for improvements to the guidelines are proposed. The guidelines of interest are the ones for deciding between using an imperative or a declarative modeling language.

Currently, the guidelines consist of two sub-sentences e.g. IBG1: ‘If the process description indicates that a clear choice has to be made between more than one activity, then model these activities within an imperative block.’ The first part is the ‘if-condition’ and is the important part of the guideline. It states what the process description should describe to apply the guideline. The second part is always the same, it either recommends the user to model the activities imperatively or declaratively. This can be improved by just stating the important part of the guideline. In this way the essence of the guideline is immediately shown. The newly formulated guidelines should present a statement which can be true or false for activities of a process description. If the statement is true, the activities should be placed in the corresponding block. As a reference, the guidelines developed in [16] are also succinctly formulated. The essence is presented in a short sentence.

Another suggestion for an improvement is to give the guidelines a meaningful name. By reading the name, it should be clear what the guideline is about, e.g. the guideline IBG1: ‘If the process description indicates that a clear choice has to be made between more than one activity, then model these activities within an imperative block.’ could be named ‘Choice moment’. As a reference, the guidelines presented in [2] also have meaningful names, e.g. ‘guideline of correctness’ already suggests the correctness of the process model is concerned. Another benefit of this improvement is that it makes it easier to identify an ‘opposite’ guideline, e.g. guidelines IBG3 and DBG3 are each others counterpart but this is not noticeable from the names. When they are named, for example, ‘Parallel without other dependencies’ and ‘Parallel with other dependencies’ it can be seen that they are opposing guidelines.
In this chapter the sub objectives of chapter 1 are revisited and it is discussed to which extent they are achieved (section 7.1). Also the requirements established for the hybrid process modeling approach in chapter 3 are validated. In section 7.2 suggestions for future work are given.

7.1 Conclusion

The main objective of this thesis is the development of a new modeling approach: the hybrid process modeling approach. This approach consists of three components. One of these components is also evaluated in this thesis.

**Sub-objective 1.** Develop a hybrid modeling language.

For the development of the hybrid modeling language, an imperative and declarative modeling language are combined. The, already existing, modeling languages BPMN 2.0 and Declare are combined to create a hybrid modeling language. An extra element, called *block*, is introduced to avoid the problem of having both imperative and declarative modeling elements on one layer of the process. A block may only contain modeling elements of one type of modeling language (imperative or declarative). All possible combinations of blocks inside blocks are allowed.

Given the developed hybrid modeling language, requirements R1.1 - R1.3 are satisfied. R1.4 was not part of the scope, this one is not satisfied. There are, however, suggestions stated of how to implement the new modeling language. It is difficult to judge whether R1.5 is satisfied without an evaluation. Two actions are undertaken for trying to satisfy this requirement. The first one is that two well-known modeling languages are used instead of developing a whole new modeling language. The other one is that two members of the UX (User Experience) team, of Perceptive Software, looked at the Declare languages and made several graphical improvements.

**Sub-objective 2.** Develop a method for creating a hybrid process model.

A method for creating a hybrid process model is developed. This method consists of four steps. The first step is to define groups. A group is a subset of the set of activities, specified in a process
description, which are connected and related. Groups are made to divide the set of activities in smaller parts, to create a clear overview of what the process is about. The second step is to define blocks. As mentioned before, a block can contain exclusively imperative or declarative activities. At the end of this step, all activities are put inside one block. The third step is to look at the relations between the different blocks. There are four possible ways to combine blocks, these are not all applicable to all block combinations. The fourth step is to relate the activities within each block. It is an iterative method, so adjustments can be made by going back to a previous step.

Both requirements R2.1 and R2.2 are satisfied. There are four well-defined steps and the method is iterative.

**Sub-objective 3.** Develop a set of guidelines which supports the method and helps the user to create a hybrid process model.

For every step of the method, guidance is given. Characteristics are given which can be taken into account when defining groups. A set of guidelines is developed which helps to decide whether the imperative or the declarative part of the hybrid modeling language should be used. The guidelines are based on characteristics of the corresponding language. The different ways to connect or combine two blocks are given as a guidance for the definition of the relations between the different blocks. Another set of guidelines is developed to help find the right declarative constraint to apply. These guidelines are an extra support for the user to model with a declarative modeling language.

Requirement R3.1 is clearly satisfied, since such set of guidelines is developed. It is difficult to judge whether requirements R3.2 and R3.3 are satisfied without an empirical evaluation. However, one set of guidelines is evaluated empirically. Whether or not these requirements are satisfied for this particular set of guidelines is discussed in the following paragraph.

**Sub-objective 4.** Evaluate the set of guidelines by means of an empirical experiment.

An empirical experiment was conducted with students from Eindhoven University of Technology to test whether the set of guidelines, for deciding which type of modeling language should be used, helped them to classify groups of activities as imperative or declarative. Two groups were formed: One had access to the guidelines and the other had not. Findings that emerged from the experiment were that, based only on the number of correctly answered questions of both groups, the guidelines did not help them. However, other statistics showed that when the intended guideline was used to answer a particular question, the answer was correct most of the time. A significant correlation, between the number of correct answers and the number of times the intended guideline was used, was found. Moreover, the subjective results pointed out that the guidelines were found useful and that participants intend to use the guidelines in the future. The easy to use aspect of the guidelines was found slightly positive, which could explain the fact that the participants found it difficult to find the intended guideline for the question. Hence, improvements can be made to the guidelines. The guidelines can be formulated differently and more intuitive names can be given.

It is still difficult to state whether or not the requirements R3.2 and R3.3 are satisfied. The evaluation shows that both aspects were rated positively, but improvements can be made.

### 7.2 Future work

There are a lot of possible future research projects regarding this topic, some of them are presented in this section.
Executable hybrid modeling language

The hybrid process modeling approach developed in this thesis could be refined and turned into an actually executable modeling language. To achieve this, new (declarative) elements should be added to the BPMN 2.0 standard. Also an executable sub-process element need to be added to the Declare language, since only a conceptual version exists. Developing such executable modeling language could be a new research project in itself. When this is achieved, verification techniques could be applied to the hybrid models and these models could be tested on correctness.

Further improvement of the guidelines

The guidelines for discovering blocks could be further improved. Also another evaluation could be conducted to test these improved guidelines. It can also be tested whether or not the guidelines were actually improved by using them both in an experiment.

Staying on this topic, subjects could be asked about how they deal or work with such guidelines in an experiment. Maybe more aspects of useful guidelines will be found in such experiments. Also a thinking-out-loud experiment would be a good suggestion to test whether or not the guidelines were understood.

The second set of guidelines presented in this thesis could be tested by the means of an empirical experiment. These guidelines should lead the user to a complete hybrid model, including the constraints between the declarative activities. The subjects participating in this experiment should be familiar with the Declare language to really test these guidelines.

Understandability of hybrid models

Reasons can be found why hybrid models should be more convenient than just imperative or declarative models. But at the moment, no research has been done on the understandability of hybrid models. This would be interesting. It could also be tested if people are able to use a hybrid modeling language and if guidelines could help them in that process. Another interesting aspect that can be measured is whether or not people can develop a hybrid process model faster than an imperative or declarative model. These persons should be familiar with the hybrid modeling language.
A.1 General information

Dear participant, welcome to this experiment!
The experiment is about the early phases of process modeling. In particular, the phase where you read a process description and try to divide the activities into sub-processes. I have created three process descriptions and prepared a set of questions about the process descriptions. Your task will be to answer the questions. I will collect your answers for further data analysis, all data is stored anonymously and no answers can be tracked back to you. After some questions, I will ask about the difficulty of that question. Please be honest about how difficult that question was to you. Indicating that you found it difficult is no sign of weakness. On the contrary, some questions have been designed to be quite challenging.

As explained before, the experiment is about dividing activities of a process description into sub-processes. I will refer to sub-processes with the term blocks from now on. There are two different type of blocks: imperative (procedural) and declarative.

Imperative block
An imperative block is a block in which the activities are modeled using an imperative modeling language. Examples of imperative modeling languages are BPMN, Petri Nets, BPEL etc. This is the approach I assume you have modeled almost all models you've made during the years of your study. This approach models every path explicitly; also all exceptions need to be modeled explicitly. Figure A.1 shows a process modeled with CPN Tools.

![Figure A.1: Imperative model](image)
The example shows the process where A is the start activity, followed by either B or C. After which D must be executed and E and F can be executed in parallel, eventually followed by activity G which leads to the end state.

**Declarative block**

A declarative block is a block in which the activities are modeled using a declarative modeling language. Examples of this type of language are Declare and DCR Graphs. You probably will be less familiar with this type of approach. The declarative approach is a constraint-based approach which is specifically useful for flexible processes. With flexible processes, it is meant that there exist activities which can happen multiple times and/or in multiple orders. A declarative model describes under what circumstances activities may happen. It explains the relations between the activities by defining constraints between activities. Figure A.2 is an example which illustrates this type of approach.

![Declarative model](image)

Figure A.2: Declarative model

![Legend of the constraints](image)

Figure A.3: Legend of the constraints of A.2

This process describes the activities A to E. The constraints that are added are: A has to happen before C; E can only happen once; D has to be the last activity executed in this process; and either C or E can be executed. All other traces are possible if they comply with the rules, for example activity B can happen any time and any number of times. For those who have not worked with CPN Tools; All activities with a thicker, green border are enabled at this moment.

Stop reading until it is requested to start reading again

**A.2 Explanation group with guidelines**

I developed guidelines to help identifying the different blocks from a process description. There are guidelines to help identifying both declarative (DBG) and imperative (IBG) blocks. These are presented and
explained below. You are recommended to use these guidelines when answering the questions.

**DBG1:** If the specification describes a subset of activities which are related to each other and are allowed to be executed more than once, then this subset of activities should be modeled within a declarative block.

Think of the process of ordering and receiving drinks in a restaurant. Generally, it is possible to order multiple drinks which eventually will be received. So the activities order a drink and receive a drink are two related activities because first an order of a drink should arrive before that drink can be received. Both order a drink and receive a drink can happen more than once, so using this guideline these two activities are modeled within a declarative block.

**DBG2:** If the specification describes a subset of activities which are related and allowed to be executed in multiple sequences, yet not all possible different sequences, then this subset of activities should be modeled within a declarative block.

Again imagine the process of ordering and receiving drinks in a restaurant, although for this example the activity ‘receive bill is added. This guideline is about the sequences in which the activities can happen. When the amount of sequences is more than one and not the same as all possible sequences this guideline is applicable. The three activities can happen in different sequences but order drink’ should be the first and after receive bill has happened it is not possible to order a drink anymore (it is still possible to receive previously ordered drinks). So multiple sequences may be created (such as: order two drinks, receive them separately after which the bill is received, or: order two drinks, receive the bill and receive the two drinks together) but not all of the possible sequences due to the restrictions mentioned earlier. This implies that the guideline is applicable to this example and the activities should be modeled in a declarative block.

**DBG3:** If the specification describes a subset of activities which must be executed in parallel but some restrictions are applicable to these activities, then this subset of activities should be modeled within a declarative block.

All drinks which are on the menu can be ordered in parallel. To give an example: The activities order coffee, order baileys and order lemon juice can happen in parallel. Assume a customer orders these three drinks. The restaurant has two rules concerning the ordering of drinks. The first is that it is not allowed to order more than 5 alcoholic drinks; the second is that it is not allowed to order a lemon juice and a baileys at the same time by the same person (because of the weird chemical interaction between these liquids). In this example, there are a couple of activities which are executed in parallel and some restrictions which are applicable to these activities, so these activities should be modeled within a declarative block.

**DBG4:** If the specification describes a long-term dependency between two activities and it is not exactly specified what happens in between and in what order, then this subset of activities should be modeled within a declarative block.

In general, if someone orders a drink in a restaurant, that person will eventually have to pay for that drink. In the meantime the customer can, for example, order some food, eat some food, order some more drinks, ask the menu, receive the bill. These activities can, for example, happen in different sequences or not happen at all. The point is that it is not known exactly what happens in between, the only thing that is certain is that when a drink is ordered a bill needs to be paid for that drink. The guideline is therefore applicable on the activities ‘order drink’ and ‘pay’. These should be modeled within a declarative block.

**DBG5:** If the specification describes an activity which can be executed regardless of the ex-
execution of other activities, then this activity should be modeled within a declarative block. Again imagine a simple restaurant process i.e. sit at a table, order food, eat food, receive a bill and pay. For this example another activity is added to that process: ‘go to the toilet’. In principle there are no restrictions for this activity to happen. It can for example happen between ordering and eating food, after the bill is paid or even before you sit at the table. To be more specific, the activity ‘go to the toilet’ can happen regardless of any activity. This guideline applies to this activity in this process and therefore this activity should be modeled within a declarative block.

Below the guidelines to identify imperative (or procedural) blocks are presented. There is no example present to explain the guidelines. It is assumed that you are somewhat familiar with imperative process modeling.

IBG1: If the specification indicates that a clear choice has to be made between more than one activity or sequence, then model these activities within an imperative block.

IBG2: If the specification describes a long-term dependency between two activities and the activities in between are following a specified order, then model these activities within an imperative block.

IBG3: If the specification describes that certain sequences of activities must be executed in parallel and between the different activities there are no other dependencies, then model these activities within an imperative block.

IBG4: If the specification describes that activities must be executed in a defined sequence and cannot happen more than once independent of each other, then model these activities within an imperative block.

Stop reading until it is requested to start reading again

A.3 Explanation group without guidelines

This experiment will also ask questions about how the structure of the process is described. This means, for example, how the sub-processes fit into the main process or whether there are activities which represent a sub-process in the main process. To make this clearer to you, a couple of different situations are explained below. It is allowed to use these examples to answer the questions.

The process in Figure A.4 is an example of an imperative main process which consists of the activities A, B and C. Activity B represents a declarative sub-process.

The process in Figure A.5 is an example of a declarative main process which consists of the activities A, B, C and D. Activity D represents an imperative sub-process.
A.4 Pre-task questions

General modeling experience questions

1. Which description matches best your current work status? (student, academic, professional)
2. How many years ago did you start process modeling?
3. How many process specifications did you read in the last 12 months?
4. How many process models have you analyzed or read within the last 12 months? (A year has about 250 work days. In case you read one model per day, this would sum up to 250 models per year.)
5. How many process models have you created or edited within the last 12 months?
6. How many activities did all these models have on average?
7. How many work days of formal training on process modeling have you received within the last 12 months? (This including e.g. university lectures, certification courses, training courses. 15 weeks of a 90 minute university lecture is roughly 3 work days.)
8. How many work days of self-education have you made within the last 12 months? (This includes e.g. learning-by-doing, learning-on-the-fly, self-study of textbooks or specifications.)
9. Which process model languages have you used before? (Aris, BPEL, BPMN, BPMone, PetriNets/Colored PetriNets/CPN Tools, Tibco/COSA, Workflow nets/WoPeD)

English understanding questions

1. I consider myself being a process modeling expert (1-7)
2. I have troubles reading English text (1-7)
3. I have troubles understanding English text (1-7)
APPENDIX A. EXPERIMENT MATERIAL

Declarative modeling experience questions

1. Have you ever heard of declarative process modeling (before today)? (Yes, No, I don’t know)
2. Have you ever seen a declarative process model (before today)? (Yes, No, I don’t know)
3. Have you ever made a model with a declarative language such as Declare or DCR-Graphs? (Yes, No, I don’t know)
4. How many declarative models have you made in the last 12 months?
5. How difficult do you find it to model with a declarative language? (1-7)

A.5 Post-task questions

Perceived Ease of Use
Q1. I found the procedure for applying the guidelines complex and difficult to follow.
Q4. Overall, I found the guidelines difficult to use.
Q6. I found the guidelines easy to make sense of.
Q9. I found it difficult to apply the guidelines to the example process descriptions.
Q11. I found the individual guidelines clear and easy to understand.
Q14. I am not confident that I am now competent to apply these guidelines in practice.

Perceived Usefulness
Q2. I believe that the use of the guidelines would reduce the effort required to discover imperative and declarative blocks in a process description.
Q3. The different blocks discovered using the guidelines would be difficult for the users to understand.
Q5. The guidelines would help users to verify whether the discovered imperative and declarative blocks are correct.
Q7. Overall, I found the guidelines to be useful.
Q8. The process models created using the guidelines would be difficult to maintain.
Q12. Overall, I think these guidelines do not provide an effective solution to the problem of discovering imperative and declarative from process descriptions.
Q13. The use of the guidelines would make it easier to communicate about process models with imperative and declarative blocks to end users than without them.
Q15. Overall, I think these guidelines are an improvement to the situation without modeling guidelines.

Intention to Use
Q10. I would definitely not use the guidelines to discover imperative and declarative blocks from a process description.
Q16. I intend to use these guidelines if I have to discover imperative and declarative blocks from process descriptions.
Q17. I would recommend my peers to use the guidelines to discover imperative and declarative blocks from process descriptions.
A.6 Main part questions

This section contains the three process descriptions accompanied with questions about it. Answers to the questions are written in bold immediately after the question. There are also some recurring questions, which are indicated with a dot instead of a number.

A.6.1 Process description 1

“When an employee wants to schedule a meeting, a secretary sends a meeting request for the meeting. For simplicity reasons, the process of finding a date when all participants are available is not taken into account. An agenda is needed for the meeting and is sent to all participants before the meeting is held. It includes the time, date, place and items up for discussion for the meeting. When the meeting is held, a couple of activities can take place. First of all, the chairman opens the meeting and eventually closes it at the end. During the meeting, items can be added and removed from the agenda. Obviously at least one item should be discussed; otherwise the meeting was not necessary. It is possible for the participants to ask questions about the items, these will be answered by other participants. If necessary, the secretary plans a new meeting. Otherwise the process just ends.”

1. How many activities can happen during a meeting? 7 (or 5 if you did not count open/close meeting)
2. To which type of block does the group [Send meeting request, Send agenda, Hold meeting, Plan another meeting] belong? Imperative
3. Which of the guidelines did you use to classify the group of activities? IBG4 (clear sequence)
4. To which type of block does the group [Open meeting, Close meeting] belong? Declarative
5. Which of the guidelines did you use to classify the group of activities? DBG4 (long term not knowing what happens in between)
6. To which type of block does the group [Add item to agenda, Remove item from agenda, Discuss item of agenda] belong? Declarative
7. Which of the guidelines did you use to classify the group of activities? DBG1 (happen more than once) DBG2 (different sequences possible)
8. To which type of block does the group [Ask question, Answer question] belong? Declarative
9. Which of the guidelines did you use to classify the group of activities? DBG1 (happen more than once)

• How difficult did you find it to answer the previous question? 1 (very difficult) 7 (very simple)
• Did you use more than one guideline, please fill in the identifiers of these guidelines (i.e. DBG1).

A.6.2 Process description 2

“When someone wants to buy a house, their financial situation must accommodate for this. To check if that is the case the following a couple of activities must be performed. First the financial situation of the potential buyers needs to be checked. To check this, three activities need to be executed: looking at their salary, looking at their mandatory expenses and looking at their savings. These checks can be done simultaneously. Afterwards, the amount which they are able to spend is calculated. Followed by
a visit to the bank to make arrangements about a credit or mortgage, in case they find an attractive house.

After the financial assessment has been completed, the "search-and-buy-a-house" process can start. First of all, this process includes the search for an attractive house. Websites such as funda.nl can be consulted to search for available houses, but other approaches are also possible. If an appealing house is found, mostly the house is viewed before any negotiation starts. When the house is viewed and the buyers are still interested, an initial offer is made by the sellers of the house. This offer can be accepted (and the house will be bought immediately) or the negotiations about the price will start. The house can be visited more than once before accepting or declining an offer. The buyers are free to stop negotiating and search for another house at any point.

A broker can be hired to help the customer anywhere in the process. A broker, for example, can be hired to help with finding an attractive house or to support the negotiations with the sellers of the house. He can even be hired when the negotiation has already started. Only if a broker is hired, a discussion with him can be started about the price of the house or any other subject."

1. Checking the financial situation of the buyers consist of how many tasks? 3
2. To which type of block does the group [Check own financial situation, Calculate affordable amount, Check finances with bank, Search and buy a house] belong? Imperative
3. Which of the guidelines did you use to classify the group of activities? IBG1 (clear sequence)
4. To which type of block does the group [Look at salary, Look at mandatory expenses, Look at savings] belong? Imperative
5. Which of the guidelines did you use to classify the group of activities? IBG3 (parallel without dependencies to each other)
6. To which type of block does the group [Search for house, View house, Decide on offer, Negotiate] belong? Declarative
7. Which of the guidelines did you use to classify the group of activities? DBG1 (negotiation part), DBG2 (stop at any point, start at beginning search for house - )
8. To which type of block does the group [Hire broker, Discuss with broker] belong? Declarative
9. Which of the guidelines did you use to classify the group of activities? DBG5 (can happen regardless of the state the process is in) DBG1 (discuss with broker can happen more than once)
   - How difficult did you find it to answer the previous question? 1 (very difficult) 7 (very simple)
   - Did you use more than one guideline, please fill in the identifiers of these guidelines (i.e. DBG1).

A.6.3 Process description 3

“A bathroom needs to be cleaned. This includes: cleaning the bathtub, cleaning the sink, cleaning the floor and dusting the furniture of the room. There are some rules which need to be followed in achieving the goal:

- Before every cleaning job (cleaning bathtub, sink or floor) you need fresh water with some cleanser.
- The dusting needs to be done before the vacuuming.
• Before the bathtub and the sink can be cleaned, they need to be free of dust.
• The mopping has to happen at the end because you do not want to clean anything in the room when the floor is wet.

Every cleaning job has its own little process. When cleaning the bathtub, first the shower curtains (if any) need to be put out of the bad. The bathtub needs to be cleaned with a cloth and a scouring sponge after which it must be rinsed. At the end, the choice can be made whether or not to use liquid clog remover. The process for cleaning the sink is almost the same as for cleaning the bathtub. The only difference is that instead of putting the shower curtains out, all items on the sink have to be removed. For cleaning the floor, first all items have to be removed from the floor. After which the floor can be vacuumed and mopped and all items can be put back.”

1. Cleaning the floor consists of how many activities? 4
2. To which type of block does the group (Clean bathtub, Clean sink, Clean floor, Dust, Change water) belong? Declarative
3. Which of the guidelines did you use to classify the group of activities? DBG1 (can happen more than once) DBG2 (can happen in multiple sequences)
4. To which type of block does the group (Put shower curtains out, Clean with cloth, Clean with scouring sponge, Rinse, Use liquid clog remover) belong? Imperative
5. Which of the guidelines did you use to classify the group of activities? IBG1 (choice between using or not using LCR) IBG4 (clear sequence)
6. To which type of block does the group (Remove items from sink, Clean with cloth, Clean with scouring sponge, Rinse, Use liquid clog remover) belong? Imperative
7. Which of the guidelines did you use to classify the group of activities? IBG1 (choice between using or not using LCR) IBG4 (clear sequence)
8. To which type of block does the group (Remove items from floor, Vacuum, Mop, Put items back on the floor) belong? Imperative
9. Which of the guidelines did you use to classify the group of activities? IBG4 (clear sequences)
   • How difficult did you find it to answer the previous question? 1 (very difficult) 7 (very simple)
   • Did you use more than one guideline, please fill in the identifiers of these guidelines (i.e. DBG1).
Both the modeling languages BPMN and Declare contain a lot of different elements. Most of these elements are used in more advanced or specific types of models. The purpose of the hybrid concept is to develop communication models. Recalling chapter 2, communication models are models which are created for the customer to understand their processes. Preferably these models should be syntactically correct, but they don’t always are. For creating communication models, not all existing elements of both languages are needed. This section describes which set of elements of both BPMN and Declare could be used to start the introduction of the hybrid approach.

### B.1 BPMN

BPMN has a wide range of elements. Although practitioners are generally more involved with BPMN, also a basic set of elements is created. To decide which elements are included in the basic set, a software engineer with expertise on BPMN (working at Perceptive Software) was approached. Her opinion is taken into account by creating the initial set of BPMN elements. The initial set includes the following elements:

**Events** Start, Start (Message), Start (Time). Intermediate, Intermediate (Message), Intermediate (Time), End, End (Message), End (Time).

**Activities** Task and Sub-process (collapsed).

**Gateways** Normal, Data-based Exclusive and Parallel Gateways.

**Connection objects** Sequence flow.

In paper [35] BPMN models from consulting, Internet searches and educational seminars were used to count the frequency of different BPMN elements. Relations were obtained between specific BPMN elements and common BPMN subsets of elements were found among the obtained models. According to the paper there are two main sets the BPMN models appear to fall into. One which is likely used by consultants, it includes pools and lanes. The other one which is likely used by designers and analysts doesn’t include lanes and pools but includes different kind of gateways. Since the initial set already includes those gateways and it should be used by practitioners (consultants), also pools and lanes are
B.2. DECLARE

included. All other elements of the initial set were according to Fig 1 of the paper also frequently used. The new set includes the following elements:


**Activities** Task and Sub-process (collapsed).

**Gateways** Normal, Data-based Exclusive and Parallel Gateways.

**Connection objects** Sequence flow, Message flow.

**Swimlanes** Pool, Lane.

Note that the connection object ‘message flow’ is also included. Otherwise communication between the lanes and pools is not possible.

### B.2 Declare

The Declare language consist of a lot of constraints. To use this language for people who have never seen it before, it is quite a challenge. To create a less challenge also a basic set of constraints is created. With these constraint it should be possible to create the basic type of models. Whenever a more advanced model has to be created, all other constraints are also allowed.

#### B.2.1 Literature study

To decide which constraints will be included in the basic set a literature study was conducted. Papers were searched and found regarding the practical use of Declare. ‘Declare’, ‘practical use’, ‘declarative modeling language’ are some of the keywords used. Authors of these papers were i.a. Fabrizio M. Maggi and Christian Stahl. These authors were approached and their opinions have been taken into account by creating the basic set.

Besides the constraints, the Declare language has three types of activities as described in 2.3, all three activities are included in the basic set. This section describes which constraints are included in the basic set and a reasoning why. Once again the division between the four categories of constraints is made.

**Existence constraints**

- Exactly(A,n)
- Existence(A,n)
- Absence(A,n)

All three existence constraints are included as their graphical representation is very intuitive and their meaning is straight forward. Even if the constraints will not be used very often it is not unpleasant to include them in the basic set.

**Relation constraints**

- Responded Existence(A,B)
- Response(A,B)
• Precedence(A,B)
• Alternate Precedence(A,B)

This category contains the most constraints but only the above four constraints selected. These are the least constraining constraints but have a useful functionality. Following, there will be an explanation why the other constraints are not included in the basic set.

The succession constraint is not included because it is a combination of the response and precedence constraints. But whenever it is the case that a lot of response and precedence constraints are used between the same activities, it could be more clear to replace them with the succession constraints.

The chain constraints are not used because they are quite similar to the normal sequence flow of BPMN. When a chain constraint is needed, it may be good to look at the problem again and see if it probably more convenient to model it using BPMN.

From experience it is nice to have the alternate precedence constraints. It has some useful behavior, for example a customer in a restaurant can order and receive drinks, when using the alternate precedence constraints between the activity ‘order drink’ and ‘receive drink’ it is possible to order several times and make sure there will be a receive moment for the drinks.

Negation constraints
• Not co-Existence(A,B)
• Not Succession(A,B)
• Not Chain Succession(A,B)

All negation constraints stated above are included. The behavior these constraints express can also be expressed with BPMN but there are much more elements needed.

Choice constraints
None of the choice constraints are included in the basic set. Most of the time when a declarative language is used, these choice operators aren’t used. Since the same functionality is available in BPMN, these constraints are not included. Most of the time a construction is possible where the BPMN choice operator is used between different declarative elements.

B.2.2 Self conducted experiment

To decide which constraints of the constraint collection of Declare are used most, an empirical research similar to [35] is conducted. This research focuses on finding the most frequent used constraints. In order to conduct this research, a set of Declare models must be collected. To find Declare models to create this collection, the consideration was made to use the same method as J. Recker and M. zur Meuhlen to find BPMN models [35]. They collected models from three different sources: several Internet search engines, consulting projects and educational seminars. Since Declare is not widely known, the Internet research only yielded three models (using different search engines with the term ‘Declare model’). Besides, there are only a couple of people researching the field of declarative modeling (or mining) making use of the Declare language. Also hardly any educational seminars on Declare exist.

These reasons drove me to use another method to find Declare models. This method contains first finding the researchers who are working with Declare and search for their papers, questionnaires, and manuals etc. which contain Declare models. Also referred papers were looked into. The authors and data sources
found are shown in Appendix C. In total 59 were models obtained of which 24 were constructed by the same person and thus were considered separately and five were duplicates. These 24 models were created by performing a case study specified in [9]. A couple of these models were rearranged, corrected and used for different questionnaires. The models created for these questionnaires were included in the final set of models. Overall a collection of Declare models with varied size and number of different constraints used is created.

Taking into consideration that a constraints should at least be present in 40% of the models, the following constraints were found:

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Init</td>
<td>50%</td>
</tr>
<tr>
<td>Exactly</td>
<td>50%</td>
</tr>
<tr>
<td>Response</td>
<td>43%</td>
</tr>
<tr>
<td>Precedence</td>
<td>87%</td>
</tr>
<tr>
<td>Not succession</td>
<td>57%</td>
</tr>
<tr>
<td>Not co-existence</td>
<td>53%</td>
</tr>
</tbody>
</table>

Table B.1: Most used constraints of Declare

When only the models which consisted of ten or more constraints were considered, only the constraints ‘precedence’ and ‘exactly’ were used with a percentage more than 40%.
At the University of Innsbruck a group of researchers is involved with declarative modeling. Their website is: [http://bpm.q-e.at](http://bpm.q-e.at). This website contained papers, questionnaires and master theses containing Declare models. The authors within this group involved with the Declare models are:

- Barbara Weber ([http://bpm.q-e.at/?page_id=263](http://bpm.q-e.at/?page_id=263)),
- Stefan Zugal ([http://bpm.q-e.at/?page_id=271](http://bpm.q-e.at/?page_id=271)),
- Jakob Pinggera ([http://bpm.q-e.at/?page_id=267](http://bpm.q-e.at/?page_id=267)),
- Cornelia Haisjackl ([http://bpm.q-e.at/?page_id=1157](http://bpm.q-e.at/?page_id=1157)), and
- Paul Pichler

Besides, the following (former) professors and (former) PhD students of the University of Technology Eindhoven wrote different papers containing Declare models.

- M. Pesic,
- F.M. Maggi,
- M. Montali,

The following papers contain the used Declare models: [34], [33], [18], [29], [29], [28], [23], [24], [17], [5], [4], [3], [14], [27].

These are the Internet sites and questionnaires containing the used Declare models:

One model is obtained by the example application ‘complaint.xml’ of the Declare application.
This appendix presents tables with the p-values of the Wilcoxon tests conducted in section 6.3.

Table D.1 gives the p-values of the results presented in Table 6.3.

<table>
<thead>
<tr>
<th>Question</th>
<th>Process description 1</th>
<th>Process description 2</th>
<th>Process description 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 3 4</td>
<td>5 6 7 8</td>
<td>9 10 11 12</td>
</tr>
<tr>
<td>Answer</td>
<td>imp decl decl</td>
<td>imp imp decl decl</td>
<td>decl imp imp imp</td>
</tr>
<tr>
<td>No guidelines(1)</td>
<td></td>
<td>(0.00)</td>
<td>(0.00)</td>
</tr>
<tr>
<td>Guidelines(2)</td>
<td>(0.00)</td>
<td>(0.00)</td>
<td></td>
</tr>
</tbody>
</table>

Table D.1: P-values of the level of difficulty values compared to the neutral value 4.

Table D.2 gives the p-values of the results presented in Table 6.4.

<table>
<thead>
<tr>
<th>Question</th>
<th>Process description 1</th>
<th>Process description 2</th>
<th>Process description 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 3 4</td>
<td>5 6 7 8</td>
<td>9 10 11 12</td>
</tr>
<tr>
<td>Answer</td>
<td>imp decl decl</td>
<td>imp imp decl decl</td>
<td>decl imp imp imp</td>
</tr>
<tr>
<td>Correct</td>
<td>(0.03) (0.02)</td>
<td>(0.00) (0.03)</td>
<td>(0.00) (0.04) (0.00)</td>
</tr>
<tr>
<td>Incorrect</td>
<td>(0.04)</td>
<td>(0.04) (0.03)</td>
<td>(0.00) (0.00)</td>
</tr>
</tbody>
</table>

Table D.2: P-values of the level of difficulty values compared to the neutral value 4 sorted on correctness (no guidelines).

Table D.3 gives the p-values of the results presented in Table 6.5.
Table D.3: P-values of the level of difficulty values compared to the neutral value 4 sorted on correctness (guidelines).
APPENDIX E

Changes to Declare

This appendix shows and explains the reasons why some of the Declare elements are changed. These changes were developed by senior interaction designer Hiendaž “Gena” Drahun and UI designer & developer Emiliano “Max” Martínez of the UX team of Perceptive Software.

E.1 Init and last activity

The init and last activity of Declare are disassociated from a normal activity by a house-shape symbol on top of the activity. Inside this symbol the words ‘init’ or ‘last’ were written. According to the experts, this is not obvious enough. When a user looks at a model, these two activities should stand out, because of their important role. Therefore they came up with two new layouts of these activities. The ‘old’ and ‘new’ ones are shown in Figure E.1.

![Figure E.1: Old and new version of the init and last activities.](image)

The shape of the init activity was chosen in such a way that it should look like a starting point. It looks like an activity with an outgoing arrow. The shape of the last activity was chosen in such a way that it should look like an ending point. Therefore the gap in the activity-shape.

E.2 Existence constraints

The existence constraints are as well as the init and last activity disassociated from a normal activity by a house-shape symbol on top of it. Inside this symbol a number or a range identifies the type of constraints. The experts were not in favor of the house-shaped symbols and decided to move the number...
E.3. NEGATION CONSTRAINTS

and ranges to the inside of the activity shape. Figure E.2 shows the ‘old’ and ‘new’ situation.

![Figure E.2: Old and new version of the existence constraints](image)

The experts were also not charmed by the way the ranges were written down. They thought it would be more convenient to express the existence and absence ranges as ‘min n’ and ‘max n’ respectively instead of ‘n...*’ and ‘0...n’ respectively.

E.3 Negation constraints

The negation constraints are characterized by the single line with two small lines perpendicular to that line. The experts thought on this type of line was that when this line should connect to activities which are of a certain distance from each other, these two small lines were not visible enough. Therefore they came up with a dotted line for the negation constraints. An example is shown in Figure E.3.

![Figure E.3: Old and new version of the not-constraints](image)

The combination of dots and arrowheads will stay the same for all negation constraints, only the line between them is dashed.

E.4 Chain constraints

A chain constraint in Declare is depicted by a triple line as shown in Figure E.4. The experts’ thoughts on this triple line was that they could be confused with the double line of the alternate constraints. Therefore a simple but effective solution was founded: a thick line. An example of this line can also be found in Figure E.4.

![Figure E.4: Old and new version of the chain-constraints](image)
Bibliography


