An Empirical Study on Simplification of Business Process Modeling Languages

Eric Umuhoza, Marco Brambilla, Davide Ripamonti

Politecnico di Milano. Dipartimento di Elettronica, Informazione e Bioingegneria
Piazza L. Da Vinci 32. I-20133 Milan, Italy
[firstname].[lastname]@polimi.it

Jordi Cabot

ICREA - UOC. Internet Interdisciplinary Institute.
Av. Carl Friedrich Gauss, 5. Ed. B3. E08860
Castelldefels, Spain
jordi.cabot@icrea.cat

Abstract

The adaptation, specially by means of a simplification process, of modeling languages is a common practice due to the overwhelming complexity of most standard languages (like UML or BPMN), not needed for typical usage scenarios while at the same time companies don't want to go to the extremes of defining a brand new domain specific language. Unfortunately, there is a lack of examples of such simplification experiences that can be used as a reference for future projects. In this paper we report on a field study aimed at the simplification of a business process modeling language (namely, BPMN) for making it suitable to end users. Our simplification process relies on a set of steps that encompass the selection of the language elements to simplify, generation of a set of language variants for them, measurement of effectiveness of the variants through user modeling sessions and extraction of quantitative and qualitative data for guiding the selection of the best language refinement. We describe the experimental setting, the output of the various steps of the analysis, and the results we obtained from users. Finally, we conclude with an outlook towards the generalization of the approach and consolidation of a language simplification method.

Categories and Subject Descriptors D.2.1 [*Software Engineering*]: Requirements/Specifications—Methodologies; D.2.2 [*Software Engineering*]: Design Tools and Techniques—Computer-aided software engineering (CASE); D.2.6 [*Software Engineering*]: Programming Environments—Graphical environments

General Terms Languages, Design, End-users

Keywords Language simplification, Domain-specific modeling languages, Business process modeling languages, Personal process modeling

1. Introduction

In recent years, the increased adoption of domain specific languages (DSLs) and of general purpose modeling languages by practitioners with different role and expertise has paved the way to the need of providing ways to simplify such languages to adapt to the expectations of practitioners.

Indeed, in many cases adopters complain about complexity or impreciseness of notations, when facing the challenge of starting using new languages. In some cases, the language they are asked to use is not exactly matching the domain or need, while in other cases the language complexity in terms of notation (number of symbols and syntax rules) or semantics, is overwhelming with respect to what is actually needed.

Therefore adaptation of modeling languages, especially in terms of simplification, has become a common need in companies. This avoids creating of completely new domain specific languages [20] or further increasing the size of existing language to match the specific needs (e.g., through profile-like mechanisms), without dropping useless parts [14]. Unfortunately, neither precise processes nor practices for addressing this problem are studied, and very few examples of language simplification experiences are reported[9]. In existing experiences users are often involved in the initial phases of the process [10], are asked to provide examples of models [5, 19], or to setup discussion sessions [3], but users rarely take an active role along the whole language definition process [10, 15].

In this paper we report on a field study aimed at the simplification of a domain-specific modeling language by means of direct involvement of users through quantitative and qualitative analysis of their modeling experience.

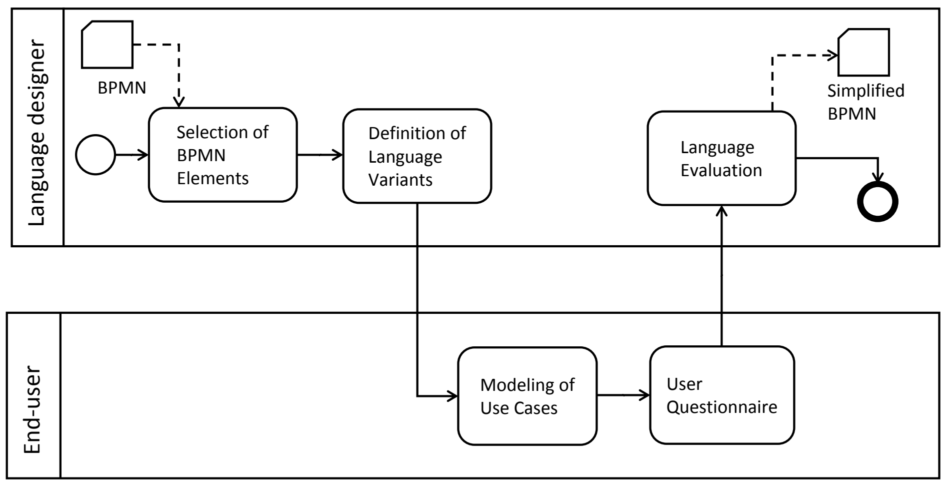


Figure 1: Empirical study on simplification of BPMN: At the beginning of the process, we select (the language designers) the BPMN elements to simplify; we then generate the language variants allowing us to assess the effectiveness of the language through the modeling sessions performed by end-users, in the third phase. In the fourth phase, we collect users' feedback through a Questionnaire while in the last phase we evaluate the language variants based on the data gathered in previous phases. The outcome is a simplified BPMN, suitable for end-users.

In our specific case, we consider the problem of simplifying a standard business process modeling language for making it suitable to end users. General purpose business process modeling languages are widely known and adopted. Despite their expressive power and notations vary a lot, they share some criticisms about their overwhelming complexity with respect to the typical usage scenarios and intended adopters. The most prominent example is the Business Process Modeling Notation (BPMN). It offers a vast range of modeling constructs which turns it in an overly complex language [18] [13], which makes it an under-used language in most situations and by most practitioners. Indeed, business analysts frequently use arbitrary subsets of BPMN. Michael zur Muehlen and Jan Recker [16] raised the question "*How much language is enough?*" and evaluated quantitatively which of the modeling constructs provided by BPMN are used regularly. Their findings showed that less than 20% of BPMN constructs are used regularly. However the usability study alone is not enough to determine which fraction of the language should be preserved or removed, since this depends a lot on the intended use and therefore it might not be enough to rely on the most used elements only.

Thus, simplification is not an easy process, and needs to be applied carefully, considering the concrete objectives of the simplification. We studied the case of *personal process management*. Personal process management refers to the application of BPM techniques and tools to personal task management, to-do lists, and small work plans shared with friends and personal contacts. The introduction of the concept of process and execution flow in personal, everyday life tasks could allow users to manage their activities in a more structured, coherent and consistent way. We call this *Personal Process Management (PPM)* [1]. The language for modeling such processes should be complete enough for describing basic processes but also simple enough to let people understand, accept and use them in their everyday life [22].

Our simplification process relies on a set of steps that encompass: (i) the selection of the language elements to simplify; (ii) definition of a set of language variants for those elements; (iii) measurement of effectiveness of the variants through modeling sessions performed by end-users and explicit questionnaires; and (iv) Extraction of quantitative and qualitative data for guiding the selection of the best language refinement. Notice that in this work we do not focus on the graphical notation to be used (aka., concrete syntax, that is the graphical symbols to be adopted), but only on the expressive power of the language.

The paper is organized as follows: in Section 2 we present the overview of our experience, by describing the steps of the simplification process; and then in Sections 3-7 we go through each step and describe the experience and the results obtained in each of them; in Section 8 we provide an outlook on the outlook towards the generalization of the approach and consolidation of a language simplification method; in Section 9 we discuss the related work and in Section 10 we conclude.

2. Overview

In this section, we describe our study on simplification of BPMN with the aim to make a suitable language for personal, every life tasks modeling. In this work, we use *Personal Process Modeling Language* or equivalently, *a Notation for Business Process Modeling* to refer to that simplified version of BPMN. Our simplification process relies on the following phases (Figure1):

1. *Selection of BPMN elements to be simplified.* In this phase, we identify the elements of the BPMN that are suitable for the modeling of personal processes. During the elements identification we consider both the nature of personal tasks and the users' needs, which differ from the needs of organizations for which the BPMN was designed. In fact, an investigation conducted in[1] revealed that the users don't want to deal with complex decision points, involving definition of conditional expressions, complex event management, or exceedingly complex process structures. The output of this phase is a set of language elements with their full description in terms of relevance and relationships (Table 1);
2. *Generation of the language variants.* In this phase, we take in input the elements defined in the previous phase and we produce a set of alternative syntaxes, the language variants. Language variants are generated from a sub set of the language elements.

   To measure the effectiveness of the designed syntax we need it to be tested by intended users. The challenge is how to submit the elements to the users so that they can evaluate them in the best way. A unique syntax with all elements, for all users, is not possible. In fact, some elements (like alternative elements) cannot be used at the same time. Moreover, considering that we are modeling the experiment for end users, all elements maybe too confusing. A possible solution is to fix a number of elements for each variant and then take all possible combinations, but this creates a problem of finding people available to test a such high number of alternatives.
3. *Modeling of use cases by end-users.* In this phase, the users test the language by modeling the assigned use case (a pair of a personal process with the language variant to be used to model it). During this phase, a logging system logs the users' actions (such as the creation, modification, and deletion of an element) useful to derive quantitative data allowing to compare and evaluate the language variants. We also monitor the users by taking the notes on how they model the assigned processes, by direct observation. Those qualitative information complete the data gathered through a logging system for our analysis.
4. *User Questionnaire.* For users profiling and segmentation, the users are asked to answer a questionnaire about their personal information: knowledge in the computer science field and about the ease of understanding the scenarios and modeling them with the assigned syntaxes. The questionnaire contains a free form field allowing the users to report their general feedback like the lack of elements or functionality in the syntaxes.
5. *Language evaluation.* In this phase, we analyze the data collected in previous phases (the execution data stored in the logs and the notes taken during the use cases modeling phase, and the users' evaluations given in the Questionnaire) to make an informed decision of the best variant based on those quantitative and qualitative data.

3. Selection of Language Elements

In this section we describe the selection of the elements which compose a notation for personal process modeling. The elements of this notation are a *small subset* of BPMN allowing to model the personal processes.

Considering the characteristics of such processes and the needs of end-users, we came up with the following elements: Task, Sequence, Parallel Execution, Conditional Execution, Events, Loop Execution, and Parameters. To understand how those elements can be used together to model a personal process, we have characterized each elements by specifying:

- the relevance (expressed as mandatory or not). The mandatory elements must be included in all the language variants otherwise the variant is invalid;
- the alternatives elements. For each element we enumerate the different options of a such element. For example the element *Parameter* can be a *global parameter*, a *one local parameter* or a *multiple local parameter*; and
- the dependencies. For each element we identify the elements on which it depends on, if any. For example, when modeling a personal process the *conditional execution* uses the value of the *parameter*. Thus, conditional execution depends on parameter.

Table 1: Characterization of the elements of personal process modeling notation

Element	Mandatory	Alternative	Dependencies
Task	Yes	N/A	N/A
Event	Yes	Start	N/A
		End	
Event	No	Wait For	N/A
		Wait Till	
Sequence	Yes	N/A	Source and Target
Parallel Execution	No	N/A	N/A
Conditional Execution	No	N/A	Parameters
Parameter	No	Global	N/A
		One Local	
		Multiple Local	
Loop Execution	No	N/A	Conditional Execution

Task Is an atomic activity included within a Process flow. A *task* is used when the work in the process cannot be broken down to a finer level of detail and it must be assigned to one or more *actors*.

Sequence Sequence flow of execution is represented by arrows. An arrow connecting two tasks means that the task at the start of the arrow must be completed before starting the execution of the task at the end.

Parallel Execution *Parallel Gateways* create parallel flows for a simultaneous execution of tasks.

Parameters The *parameters* are used to store values submitted by *actors*. These values can be used in *conditional gateways*, printed in *tasks* bodies or just stored. We have identified three possible ways to use parameters:

- Global Parameters, available in all the processes directly or indirectly connected after the creating *task*;
- One-Local Parameters, available only in the *tasks* to which they have been propagated; and
- Multiple-Local Parameters. Similarly to *one-local parameters*, the *multiple-local parameters* are only available in the *tasks* to which they have been propagated. But, in this case, more than one parameter can be propagated.

Conditional Execution The only way to make conditional execution is to create conditional parallel flows using *conditional gateway*. This notation allows using only the simple conditions. Therefore, the arithmetic operations and comparisons between parameters are not supported. The *conditional gateway* uses *parameters* to evaluate the condition.

Loop The *loop* is a specific tool to create backward flows. It requires the *conditional gateway* to examine the loop conditions. The expressed conditions are mutually exclusive.

Events An event is something that happens during the course of a process. The events are grouped in the following three categories:

- Start Event. It indicates where a process will start;
- Intermediate Event. An intermediate event indicates where something happens, an event, somewhere between the start and end of a *process*. In BPMN, those events are used to model events such as message-based communication among actors, flow control through exception handling, and delays expected within the business process. While those events are important to model business processes, some of them (e.g message-based communication) are not relevant in personal processes.

  The timer is the only relevant intermediate event in personal process. In fact, it could be useful to impose time constraints on the execution flow. We have introduced two time events: *Wait For* and *Wait Till* which express, respectively, the waiting for a generic time and the waiting of a specific date or time;and
- End Event. It indicates where a process will end.

The start and end event are considered mandatory elements, because processes need to have a start and end point. Conversely, intermediate events (Wait For and Wait Till) are not mandatory (Table 1).

Table 2: The language variants

Element	**Alternative**	**Variant 1**	**Variant 2**	**Variant 3**	**Variant 4**
Event	Start	×	×	×	×
	End	×	×	×	×
Task		×	×	×	×
Parameter	Global	×	×		
	One Local			×	
	Multiple Local				×
Intermediate Event		×		×	×
Sequence		×	×	×	×
Parallel Execution		×	×	×	
Conditional Execution			×	×	×
Loop			×		

4. Definition of Language Variants

To measure the effectiveness of the notation we need it to be tested by end-users through the modeling sessions. The challenge is how to submit the elements of the language to the users so that they can evaluate them in an efficient way. In our approach, this done through a set of language variants: alternative syntaxes generated considering only a subset of the language elements. In the next paragraphs, we describe how we generate those variants.

We use the language variants, which use a reduced set of language elements, because it is easier (for end-users) learn and use a language composed by few elements rather than a complex one. Furthermore, it not always possible to test all the elements of a language together. For example, the alternative elements cannot be included in the same syntax at the same time.

However, testing all valid combinations of the elements becomes unfeasible due to the high number of the variants and consequently, the required high number of users to test them. Therefore, the dimension of the language variants (the number of the elements to include in each variant and the number of variants) is a compromise between the possible combinations of the elements and the number of users available to test them[1].

In our case, a good compromise between the number of elements and the number of the users available for the test was the following *four variants* (Table 2):

- Variant One: it makes use of sequence and parallel flows, events, global parameters, but no loops and no conditional flows.
- Variant Two: it makes use of sequence, parallel and conditional flows, global parameters, loops but no events.
- Variant Three: it makes use of sequence, parallel and conditional flows, events, one local parameter, but no loops.

[1] Indeed, we face similar trade-offs as those heavily studied in software testing. As such, a good strategy when defining the variants should be to maximize the coverage both in terms of combination of elements to tests and in terms of coverage of relevant usage scenarios for the language

- Variant Four: it makes use of sequence and conditional flows, events, multiple local parameters, but no loops and no parallel flows.

All the four language variants have the basic elements in common: the start and the end events, the task and the sequence routing. Those are *mandatory* elements for the *personal process modeling language*. We built a linear *variant* with global parameters, events, parallel routing but not conditional and loop gateways. We also built a more complex *variant* with multiple local parameters, conditional routing but no parallel gateway. The other two *variants* were instead more balanced.

5. Modeling of Use Cases by End-Users

In previous section, we have defined the elements of a notation for personal process modeling and we have identified its four variants. The next step of our approach is to submit these variants to the end-users and conduct the experiment in which users model the assigned use cases. A use case represents a couple of a personal process with the language variant to be used to model it.

5.1 Scenarios

We have designed three scenarios based on the real life situations. Every scenario describes a process with the suggested activities and the general constraints.

Scenario A - Holiday with friends You are organizing holiday with friends. You need to decide the transport, choose the hotel and plan possible tours. Then you have to draw up the budget and if it is included in the planned costs, you can proceed with reservations. The week before the departure, you will meet together to define the last details.

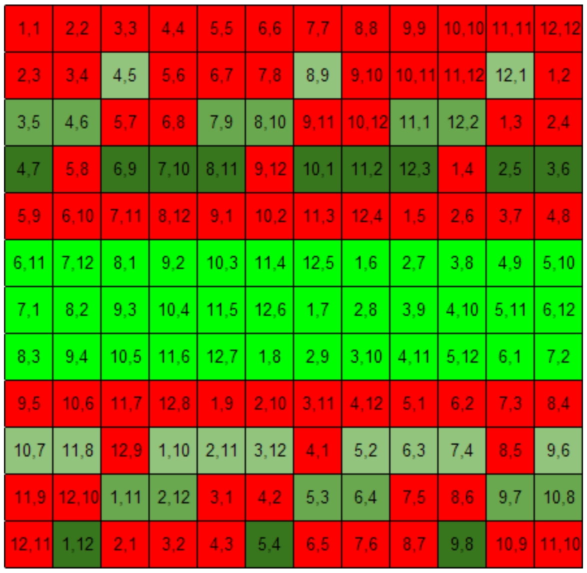
Scenario B - Association party You are organizing a free-entrance party of your association. You need to publicize the event, and ask the authorization to the municipality if you registered at least 50 reservations. Then you can contact the catering service and engage the band. The event will start with the dinner at 8:00 pm followed by the concert at 9:30 pm if the dinner has finished.

Scenario C - Warehouse management Your goal is to optimize the buying and selling of the raw materials, maintaining always 1000 units constant.Every week, the quantity of material in warehouse must be checked and then, according to the amount, you need to buy new material or sell it. The order must be approved by the accounting department. Once arrived, new stock must be placed and cataloged.

In addition, we have developed a *test scenario*, used at the beginning of the modeling sessions to explain to the users how the tool works and to let them get confidence with it.
Scenario Test - Basket tournament You are organizing a basket tournament. You need to collect the registrations, schedule the matches, rent the playgrounds and find the sponsors.

Table 3: Use cases

Use case	**Language variant**	**Scenario**
1	1	A
2	2	A
3	3	A
4	4	A
5	1	B
6	2	B
7	3	B
8	4	B
9	1	C
10	2	C
11	3	C
12	4	C



1,1	2,2	3,3	4,4	5,5	6,6	7,7	8,8	9,9	10,10	11,11	12,12
2,3	3,4	4,5	5,6	6,7	7,8	8,9	9,10	10,11	11,12	12,1	1,2
3,5	4,6	5,7	6,8	7,9	8,10	9,11	10,12	11,1	12,2	1,3	2,4
4,7	5,8	6,9	7,10	8,11	9,12	10,1	11,2	12,3	1,4	2,5	3,6
5,9	6,10	7,11	8,12	9,1	10,2	11,3	12,4	1,5	2,6	3,7	4,8
6,11	7,12	8,1	9,2	10,3	11,4	12,5	1,6	2,7	3,8	4,9	5,10
7,1	8,2	9,3	10,4	11,5	12,6	1,7	2,8	3,9	4,10	5,11	6,12
8,3	9,4	10,5	11,6	12,7	1,8	2,9	3,10	4,11	5,12	6,1	7,2
9,5	10,6	11,7	12,8	1,9	2,10	3,11	4,12	5,1	6,2	7,3	8,4
10,7	11,8	12,9	1,10	2,11	3,12	4,1	5,2	6,3	7,4	8,5	9,6
11,9	12,10	1,11	2,12	3,1	4,2	5,3	6,4	7,5	8,6	9,7	10,8
12,11	1,12	2,1	3,2	4,3	5,4	6,5	7,6	8,7	9,8	10,9	11,10

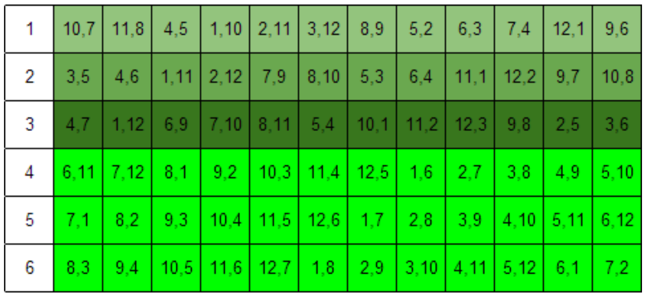
Figure 2: Graeco-Latin square of use case pairs

5.2 Use Cases Definition

A use case represents a pair of a scenario with a language variant to be used to model it. The use cases are based on the relevant scenarios of the domain. Having 4 language variants and 3 scenarios, overall there are 12 possible pairs, use cases (Table 3).

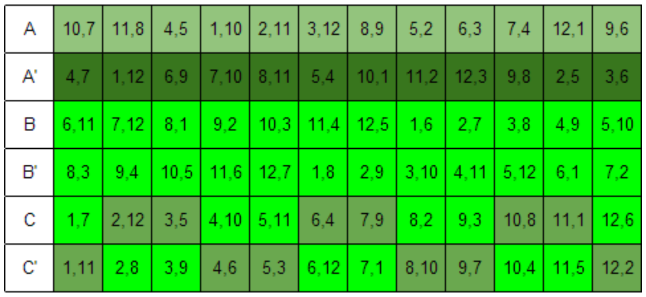
5.3 Assigning Use Cases to Users

We have decided to submit two use cases to each user. The submission order of the two use cases is important because in the modeling of the second use case, the user will apply the knowledge acquired in the first one. In order to overcome unintended effects and to have balanced experiments, all the possible use cases pairs have to be studied considering both the *order* and the *scenario-variant coupling*. For that purpose, we use the Graeco-Latin square theory[11]. Applying the Graeco-Latin square theory to the use cases, we obtained the result shown in Figure 2.



1	10,7	11,8	4,5	1,10	2,11	3,12	8,9	5,2	6,3	7,4	12,1	9,6
2	3,5	4,6	1,11	2,12	7,9	8,10	5,3	6,4	11,1	12,2	9,7	10,8
3	4,7	1,12	6,9	7,10	8,11	5,4	10,1	11,2	12,3	9,8	2,5	3,6
4	6,11	7,12	8,1	9,2	10,3	11,4	12,5	1,6	2,7	3,8	4,9	5,10
5	7,1	8,2	9,3	10,4	11,5	12,6	1,7	2,8	3,9	4,10	5,11	6,12
6	8,3	9,4	10,5	11,6	12,7	1,8	2,9	3,10	4,11	5,12	6,1	7,2

Figure 3: Valid experiments



A	10,7	11,8	4,5	1,10	2,11	3,12	8,9	5,2	6,3	7,4	12,1	9,6
A'	4,7	1,12	6,9	7,10	8,11	5,4	10,1	11,2	12,3	9,8	2,5	3,6
B	6,11	7,12	8,1	9,2	10,3	11,4	12,5	1,6	2,7	3,8	4,9	5,10
B'	8,3	9,4	10,5	11,6	12,7	1,8	2,9	3,10	4,11	5,12	6,1	7,2
C	1,7	2,12	3,5	4,10	5,11	6,4	7,9	8,2	9,3	10,8	11,1	12,6
C'	1,11	2,8	3,9	4,6	5,3	6,12	7,1	8,10	9,7	10,4	11,5	12,2

Figure 4: Valid experiments without duplicates

Each cell of the table, depicted in Figure 2, represents a single experiment made by *two use cases* to be assigned to *a single user*. In each row of the table, every variant and every scenario appears the same number of times, and every use case appears one time in the first position and one time in second position. The *invalid experiments* are highlighted in red. We consider invalid an experiment which has the same scenario and/or the same variant in both tests. For example, the experiment [2,3] is invalid because of the repetition of scenario A, while the experiment [5,9] is invalid because of the repetition of the first variant. Excluding the red cells, the remaining green cells can be grouped as shown in Figure 3.

The second and the fifth rows have six cells each that are mirrored repetitions of the other six cells ([3,5] and [5,3], [4,6] and [6,4], etc). Let's call the remaining rows A, A', B, B', C and C' (Figure 4). We can see that A' has the same couplings of A but with use cases in the inverse order. The same holds for B'-B and for C'-C.

5.4 Modeling Sessions

Each modeling session of the use cases by end-users has been conducted following this exact procedure in five steps:

1. *Introduction*. First, we have introduced the modeling tool to the users and we have presented them its purpose. Then we have explained them the objective of the experiment: to test the effectiveness and ease of use of the adopted language variant to model the small processes, in daily life or in small and medium-sized enterprises.
2. *Registration*. The users logged in to the tool using one of their social accounts.
3. *Instruction*. The users read the main manual of the editor which describes how to make the basic actions of creation, modification and deletion of elements. Then, a short test of 3 minutes took place under our supervision in order to take confidence with the basic functionality of the tool. This was done using a test scenario, equal for all the users.
4. *Experiment*. Each user had to model two use cases. In this step the user could read the manual of the first assigned variant followed by the description of the coupled scenario. At this point he could start modeling the process. Once the first process was modeled, the user could proceed with the second use case in the same way described before.
5. *Closure*. At the end of the test, after the user had submitted both models, he was asked to answer a questionnaire about his personal information, his knowledge in the computer science field, the ease of understanding the scenarios and modeling them with the assigned variants. He was also asked to report the lack of elements or functionality in the assigned language variant.

5.5 Monitoring

During the experiments, we monitored users' actions in order to collect the data indispensable to assess language variants. The monitoring was done through: direct observation, noting every user's activity and comments made while modeling the assigned processes; and through a logging system. The logging system was limited to the graphical editor and each log rows contains a code name for the tracked action, a field used for additional info, and the target of the action. The logs have been stored in a structured database from which the desired data are extracted by means of SQL queries. We have prepared a set of basic queries that retrieve the:

- time required to create each process,
- number of times an element has been edited,
- number of elements created,
- number of times an element has been moved,
- number of delete,
- number of process savings,
- number of validation requests, and
- number of invalid connections. Although the tool checked for correctness of the model and did not allow to create wrong models, it recorded the number of invalid connections as the number of times the users attempted at drawing wrong connections between elements.

Starting from these basic queries, we have built the composite queries extracting derived data used to compare and to

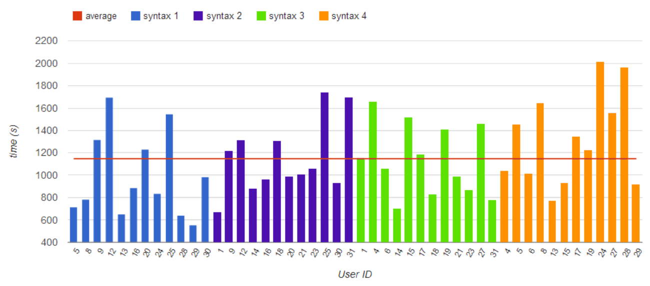


Figure 5: Experiments duration per single user

evaluate the language variants. All these data, together with the opinions and feedback collected from the questionnaire, must be analyzed to understand each variant pros and cons and determine a good compromise.

5.6 Execution Data Analysis

The experiment involved 24 users. 21 are men, while only 3 are women. 19 users are in the 18-30 years range, 2 users in the 31-50 range and 3 users in the 51-70 range. Half of the users are students, 7 have high skills in computer science while 9 users define themselves as basic PC users. Overall, the users are not experts in BPMN. In fact, only 2 users claim to use it regularly, 8 know it but they do not use it while the remaining 14 have never heard about it.

5.6.1 Durations

From the analysis of the times of the single experiments (Figure 5) we notice that processes modeled with the *first variant* are those done *faster*, with an average duration of 16’27”. On the contrary the *fourth variant* is the *slower*, with an average time of 22’02”, about 34% slower than the first (5’35” more). The second and the third variants are instead quite similar between them, about 16% and 15% slower than the first. Looking at the standard deviations, the fourth and first variants have higher values. All the delivered processes are correctly validated except one made with the second variant. It is interesting to notice that 19 times over 24, the second test has been modeled in less time than the first one, with an average of 7’30” less. This is probably due to the fact that during the first test users need more time to take confidence with the editor. In the remaining 5 times, users took an average of 3’20” more. Also the variants play a role in the difference of time between the two tests. In fact, users who play first with the fourth variant, and then with the first variant, take on average 12’43” less, while users who first used the first variant and then used the second variant, take on average 1’27” less. It seems that the fourth variant is *more powerful* then the first but is also *heavier to use*, so it takes more time. The first variant is *simpler*, offers minor possibilities and is *faster to use*.

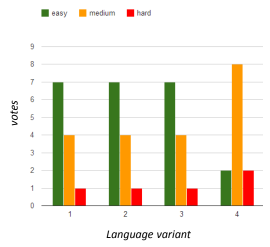


Figure 7: Variants difficulty

5.6.2 Number of Elements Creations and Deletions

The processes modeled with Variant One have few elements, thus they are smaller and simpler. Instead, Variant Four has the most number of creations and is the richer one. Variant Two and Variant Four seem to have same number of elements used (Figure 6.a). The number of deletions is quite constant for all the variants, a little less in the first and a little more in the second.

The most used elements are of course tasks and connections. Those most used elements are omitted in (Figure 6.b) to better highlight other elements variations. The *Wait For* events are rarely used and have a high percentage of deletions in all the variants in which they appear, respectively 60%, 80% and 67%, while the *Wait Till* events are preferred and more used. The *Parallel Gateway* is less used than the *Conditional Gateway*. In the first variant there is only the *Parallel Gateway*, in the fourth there is only the *Conditional Gateway*, while in the other two they appear both. In the second variant the *Parallel Gateway* is used twice as often as the Conditional. In relation to these data we also need to point out that variants three and four are those with a greater number of created elements overall. The *Loop*, with an average use of 1.33 per process, is the most used element in the processes where it is available, namely those modeled with variant two.

5.6.3 Validation Requests

During all the tests, users have used the validation button to check the processes correctness. Generally, the processes were correct but sometimes errors were found and users had to validate again the process. Globally, all the variants have almost the same number of validations. The highest percentage of validations with errors is the one related to the Variant One, immediately followed by Variant Three. Variants One and Variant Four are those with the lowest percentages of false validations.

5.6.4 Editing Elements

The average time per process spent by users editing elements is approximately the same for all the variants except for the Variant One that is significantly smaller. On average, the time spent modifying single elements is more in the Variant One, though with higher standard deviation, while less time was spent on Variant Four. On the other hand, the number of

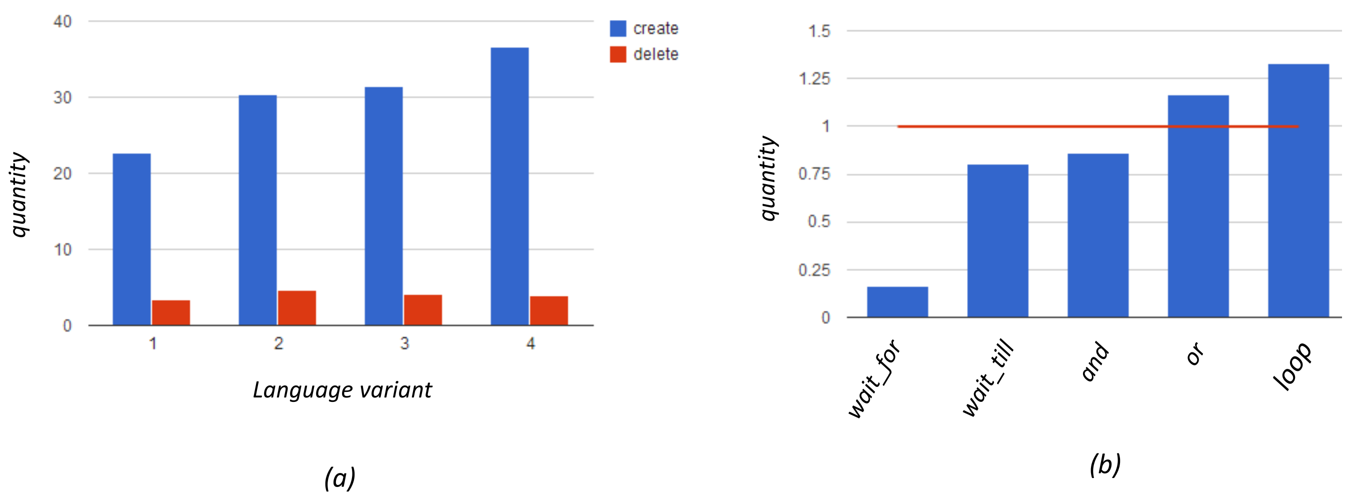


Figure 6: *(a)* Shows the average number of creations and deletions per process while *(b)* shows the average number of elements per process

element changes is greater in Variant Four, while it is smaller for the Variant One. The high number of changes for variants with *local parameters* characterized however by short editing times. These characteristics should be caused by the need to manually propagate local parameters. The data related to the *events* showed that they were edited practically in the same way. *Conditions* required on average, more time to be configured with Variant Four.

6. User Questionnaire

This section resumes the answers gathered from the 24 users involved in the experiments. The users' feedback complement quantitative measures presented in Section 5.6 Execution Data Analysis.

Scenarios difficulty Each scenario was tested sixteen times. Both scenarios A and B were judged mainly easy with 13 and 12 votes each and the remaining votes on medium level. Scenario C resulted to be the hardest with only 8 votes on easy level, 7 on medium level and 1 on hard level.

Language variants difficulty Each variant was tested twelve times. Variants One, Two and Three were equally judged with 7 votes on easy, 4 on medium and 1 on hard difficulty. Instead, Variant Four is perceived harder since users gave only 2 votes on easy, 8 on medium and 2 on hard (Figure 7).

Cases difficulty The easiest combination of variants and scenarios resulted to be Variant Three and Variant Four with scenario A, and Variant One with scenario B. The worst was Variant Three with scenario C.

Variants deficiencies The most noticed deficiency was the lack of the *Loop* element. In fact, in 15 tests, users claimed the necessity to have the *loop* to correctly model the process they had in mind. Particularly, its absence was felt more in the third variant.
The *Conditional Execution* is included in all the variants except for the first, and right in this, in 8 tests over 12, users wanted it.
The *Parallel Execution* does not exist only in the fourth variant end its absence was notice only in 3 tests over 12 tests. The lack of the events has not gone unnoticed in Variant Two.
Users noticed also deficiencies on *parameters types*: someone claimed Boolean parameters but then he resolved using the textual type; other users would have preferred to create lists of parameters and treat them as arrays. In some cases it would have been useful to compare two parameters inside a condition, or sum them or compare them with a value. Using the third variant, in two cases users would have needed to propagate more than one local parameter, while in other two cases they would have preferred to have global parameters that are more comfortable to use. In the fourth variant instead, in three tests the propagation of multiple local parameters was found too heavy.

After all, even if every variant has its pro and cons, all the users have succeeded to model the assigned processes, sometimes using alternative, weird but functional methods to bypass the restrictions.

7. Language Evaluation

By analyzing the data related to the different aspects of the modeled processes during the experiments (the users evaluations given in the questionnaire, the notes taken during the experiments, and the execution data), we can state that there is no language variant which is clearly better than the others in absolute terms. This is quite common in any realistic setting, and it's the reason why the problem of selecting *the best language* is not a trivial task at all (notably,

Table 4: Comparing the elements number of the initial language, BPMN, with its simplified version, the language variants. BPMN provides 52 modeling elements while the four variants use 7, 8, 8, 7 elements respectively.

Language	BPMN	Variant One	Variant Two	Variant Three	Variant Four
Number of element	52	7	8	8	7

it cannot be reduced to simply choosing the variant with the most used concepts), also because different relevance or priority may need to be put on one aspect or the other when finalizing the choice for the given vertical scenario.

However, the information collected and combined together through the various means proposed in our approach, provide evidence and quantitative data allowing the decision the be made more objectively and in an informed way.

The main methodological guideline consists in defining two dimensions in the decision making process: (i) *Language evaluation*. Besides evaluating independently each single language element, it's also important to consider how they actually integrate with each other in a comprehensive tool, i.e., the language variant. (ii) *Element evaluation*. This dimensions suggests to evaluate each element in the language independently, so as to compare the different available options for its implementation and get oriented towards the potentially best selection of elements.

There is no prescribed order or ranking of importance between these two aspects, as the final decision may very well require some looping over the two, until a satisfactory selection is reached for both. We report here the evaluations and analysis done in our case study, so as to provide hints on the possible aspects to be taken into account in the decision.

7.1 Language Variants Evaluation

Each language variant exhibits a set of pros and cons, which again can be distilled by analyzing both the quantitative measures taken during the experiments, and the feedback of the users through the questionnaire. Probably the best variant would be the one achievable choosing the best options of each element based on the previous analysis. However, overall language characteristics must be considered too.

All the four language variants designed in this work are simpler, in terms of the *number of used elements*, with respect to BPMN. In fact, Variant One and Variant Four use only 13.5% of BPMN elements while Variant Two and Variant Three use 15.4% of BPMN elements. In our specific case, the number of used elements is not a strong discriminating factor since all variants use more or less the same number of elements (Table 4). Here follow the resuming considerations about the variants, along with a pros and cons list of each one.

7.1.1 Variant One

It is the quickest and leanest in the processes creation, but in many cases it has turned out to be too poor and ineffective. It is more suitable to describe very easy and linear processes and it is too limited if there is the need to increase the modeling detail. Its pros and cons are reported in Table 5.

Table 5: Language Variants Evaluation: Pros and Cons of Variant One

Pros	**Cons**
- Small number of elements to learn and a few rules to use them. - Processes are modeled faster with a smaller number of elements and deletions. - Shorter process design time. - No connection errors were done during the experiments. - A few validation errors were done during the experiments. - Smaller number of validation requests and lower percentage of wrong models submitted for validation.	- Lack of specific tools to express the conditional execution. - Harder in medium complex and complex processes modeling. - Longer task editing time. - Poor expressive power.

7.1.2 Variant Two

It is the only variant which presents all the gateway types: particularly the Loop Gateway has proved to be really useful in the modeling of the more complex passages of the processes. Its pros and cons are reported in Table 6.

Table 6: Language Variants Evaluation: Pros and Cons of Variant Two

Pros	**Cons**
- The loop gateway makes possible to model iterative execution - Average modeling time - Number of used elements below the average. - Small number of connection errors	- Lack of specific tools to express time events. - Higher percentage of validation errors. - Higher number of validation requests.

7.1.3 Variant Three

This variant represents a good compromise between ease of use and descriptive power but the simplicity of a single parameter is, in practice, a great limit. Its pros and cons are reported in Table 7.

7.1.4 Variant Four

This variant turned out to be the worse one, and was the one more criticized by users. Its pros and cons are reported in Table 8.

Table 7: Language Variants Evaluation: Pros and Cons of Variant Three

Pros	Cons
- Lower Wait Till and Conditions editing times. - Small number of Wait Till and Conditions changes. - Lower number of users suggestions in the questionnaire.	- Number of connection errors over the average. - Each time a parameter is added it must be manually propagated, if needed.

Table 8: Language Variants Evaluation: Pros and Cons of Variant Four

Pros	Cons
- Small number of validation errors - Lower number of validation request with lower percentage of invalidity.	- Connection errors above the average. - Each time a parameter is added it must be manually propagated, if needed. - Bigger number of elements used. - Longer processes creation times. - Longer elements editing times. - Higher number of editing on elements. - The hardest to use according to questionnaire results. - Greater number of suggestions in the questionnaire.

7.2 Elements Evaluation

Single element evaluation should take into account quantitative performance of the element in the experiment, as well as hints collected explicitly from users. Indeed, data collected through the logging of the experimental modeling phase may have little meaning if not complemented by the users opinions and descriptions of the problems. Therefore, also the users evaluations given in the questionnaire and the notes taken during the experiments must be taken in consideration. Keeping in mind all these aspects, we can give a resuming evaluation of the single elements of the four variants in our scenario.

Wait For Event It has been by far the least used element and the most deleted. Sometimes, instead of using it, users preferred to use the time constraints inside the tasks, misinterpreting their meaning.

Wait Till Event Compared to the Wait For event, the Wait Till event has been used and appreciated by the users, and it has been useful to resolve the proposed scenarios.

Parallel Gateway It is a basic element that has been frequently used when it was available and missed in the variant where it was not. Sometimes it has been replaced by the Conditional Gateway using equal conditions on the branches.

Conditional Gateway It has turned out to be fundamental for its great descriptive power. Users have felt the lack of it when it was not available.

Loop Gateway Looking at the collected data, the Loop has been the most relatively used element, proving to be very useful to resolve the critical aspects of the modeling. When available, it has been really appreciated, while, when missing, users claimed its need and have found harder the process modeling. Conversely, the Loop element has introduced some more errors, so it requires a little more attention in its use.

Global Parameters Overall, global parameters were easy to understand and much appreciated, despite an initial difficulty of some users that were not confident with the concept of variable and parameter in computer environment (this consideration applies also to local parameters).

One Local Parameter It has shown its limit when there was the need to receive more than a parameter in a task. Users found not intuitive to propagate and receive the parameter.

Multiple Local Parameters They have been judged more useful than the one local parameter because they increased the modeling efficiency, but on the other hand, they were criticized because of the heaviness they introduce in the notation.

7.3 Language Variant Selection

The selection of the best language variant depends highly on the domain needs. However, the winning variant should be the one having the following properties:

- Simplicity. The language variant should be simple to understand and easy to use. We assess this property through the analysis of the quantitative data (such as required time to create the process, number of connection errors and validation requests) gathered during modeling sessions and qualitative information collected through user questionnaire and notes.

- Completeness with respect to domain requirements. The winning variant should allow the modeling of all relevant aspects of the domain. We assess this property through analysis pros and cons of each variant and the evaluation of single elements.

The language Variant One fits the most the above-mentioned properties.
Simplicity. The analysis of pros and cons showed that processes modeled with Variant One were done faster, using

small number of elements and they do not presented any connection error. Furthermore, the validation ratio (number of validation errors/number of validation requests) was low. *Completeness.* Single elements evaluation showed that all the elements of Variant One are well understood and users required them when missing. The pros and cons analysis showed that variant One has poor expressive power. However, this is not a problem in personal process. As said so far, the selection of the best variant depends on the domain. In next paragraph, we illustrate how the winning variant could have been another one given the same settings but changing only the domain.

Let us consider that (i) we are simplifying BPMN for an organization which needs to model its structured business processes; and (ii) the variants and the single elements evaluation still the same. The suitable variant in this context is variant Two since is the most complete with respect to domain. Its high complexity compared to variant One is not a big problem for a company since companies usually hire modelers with required knowledge to model their processes.

8. Generalization of the Approach

In this section, and based on the experience reported so far, we provide some hints on how the described simplification process can be generalized to be useful in the simplification/personalization of other modeling languages.

The generalization involves mainly the initial parts of the process where we need to reduce the scope of the problem by first choosing a base language to simplify and the set of relevant elements to evaluate. Once this is done, the rest of process follows the same schema of Figure 1.

While in the case study described in this paper, the choice of BPMN as the starting point was an obvious choice as the standard and widely adopted language in the process modeling area, but things may not the same in other domains. If there are several candidate languages (to be found, for instance, by looking at systematic reviews available for the domain), the selection of the base language to simplify must be done objectively based on some predefined criteria, the selection dimensions. While these dimensions may be decided by each company, clearly, aspects like the popularity of the language, its tool support or the availability of predefined extension mechanisms should be taken into account since they will for sure facilitate the simplification process.

Then, you should proceed with the identification of the language elements to evaluate in the language variants step. In general, the key to do well this phase is to have a deep understanding of the needs of the end-users: what are their requirements? what language elements are more likely to fit those requirements? Those elements would constitute the starting set of candidate elements to focus on. Next we should explore how they are related in the original language metamodel among themselves and with other (auxiliary) elements to enrich that set. Once this identification phase is completed we can proceed with the definition of the variants.

Once this is done, the rest of the process described in the paper can be easily adapted to any language. For instance, to define the language variants, the only language-specific part consists in the identification of the dependencies between the language elements to evaluate (captured by looking at the metamodel definition as indicated in the previous step). Similarly, the process to design the use cases should follow the same recommendations as for the BPMN scenario (mainly to make sure that the use cases cover the variety of situations we want to evaluate). The data analysis part is not language-dependent so its application is straightforward.

9. Related Work

The active participation of end-users in the early phases of the software development life-cycle is key when developing software [6, 7, 17]. Among other benefits, the collaboration promotes a continual validation of the software to be build [8], thus guaranteeing that the final software will satisfy the users' needs.

This is also true when the goal of the development process is to produce a (modeling) language to suit the needs of a user community. So far, this has been typically addressed by either creating a brand new language for that community (what it is known as a Domain-Specific Modeling Languages (DSML), a modeling language specifically designed to perform a task in a certain domain [20]) or by extending an existing base language (usually by applying some kind of profile-like mechanism [2, 14]). Our approach differs from both scenarios since we want to start from a base language (much less time consuming and error prone than trying to build a DSL from scratch) but we do not want to extend that language but rather simplify it.

All approaches recognize the need to involve domain experts in the language design process [10, 15, 21] but unfortunately this is not yet a common practice. Participation of end-users is still mostly restricted to the initial set of interviews which hinders the process performance [10] since end-users must wait until the end to see if designers correctly understood all the intricacies of the domain. Several works have tried to facilitate this participation by proposing to derive a first language definition from user examples [5, 12, 19]. An alternative approach promotes quick iteration and discussion cycles between domain experts and language developers from the beginning within a collaborative environment [3] which could also include the use of example models [4]. While all these are valid practices, we propose a more hands-on approach where end-users not only provide a few examples or give opinions on how the language is built but actually test its variations while being monitored and the collected data is then used to make more informed decisions based on real data.

10. Conclusions

In this paper, we reported on a field study aimed at the simplification of a business process model language for making it suitable to end-users.

Our simplification process relies on: (i) the selection of the language elements to simplify, (ii) generation of a set of language variants for those elements, (iii) measurement of effectiveness of the variants through modeling sessions performed by end-users, and (iv) extraction of quantitative and qualitative data for guiding the selection of the best language refinement. We described the experimental setting, the output of the various phases of the analysis, and the results we obtained from users.

The results of our study, showed that the followed approach provides evidence and qualitative and quantitative data enabling the selection of the best language variant to be taken more objectively and in an informed way.
Future works will include the analysis of the concrete syntax (e.g., the graphical symbols to be used) as part of the simplification process, the application of the proposed approach when the modeling needs evolve (e.g., language maintenance and evolution) and, more importantly, the replication of this study in other domains to advance in the generalization of our language simplification process.

Acknowledgements. We thank Pierfilippo Bianchi for helping us to prepare the experiments. This work was partially funded by the AutoMobile EU 7th FP SME Research project.

References

[1] Marco Brambilla. Application and simplification of bpm techniques for personal process management. In *BPM Workshops*, pages 227–233. 2013.

[2] Marco Brambilla, Andrea Mauri, and Eric Umuhoza. Extending the Interaction Flow Modeling Language (IFML) for Model Driven Development of Mobile Applications Front End. In *MobiWIS*, pages 176–191, 2014.

[3] Javier L. Cánovas Izquierdo and Jordi Cabot. Enabling the Collaborative Definition of DSMLs. In *CAiSE conf.*, pages 272–287, 2013.

[4] Javier L. Cánovas Izquierdo, Jordi Cabot, Jesús J. López-Fernández, Jesús Sánchez Cuadrado, Esther Guerra, and Juan de Lara. Engaging End-Users in the Collaborative Development of Domain-Specific Modelling Languages. In *CDVE conf.*, pages 101–110, 2013.

[5] Hyun Cho, Jeff Gray, and Eugene Syriani. Creating Visual Domain-Specific Modeling Languages from End-User Demonstration. In *MiSE workshop*, pages 29–35, 2012.

[6] Kevin Dullemond, Ben van Gameren, and Rini van Solingen. Collaboration Spaces for Virtual Software Teams. *IEEE Softw.*, 31(6):47–53, 2014.

[7] Les Hatton and Michiel van Genuchten. Early Design Decisions. *IEEE Softw.*, 29(1):87–89, 2012.

[8] Tobias Hildenbrand, Franz Rothlauf, Michael Geisser, Armin Heinzl, and Thomas Kude. Approaches to Collaborative Software Development. In *FOSE conf.*, pages 523–528, 2008.

[9] Swiss e-government standards: Modelling Businesses using BPMN. http://www.ech.ch/vechweb/page?p=dossier&documentNumber=eCH-0074&documentVersion=2.00. Accessed: 2015-08-01.

[10] Steven Kelly and Risto Pohjonen. Worst practices for domain-specific modeling. *IEEE Softw.*, 26(4):22 –29, 2009.

[11] Dominic Klyve and Lee Stemkoski. Graeco-Latin Squares and a Mistaken Conjecture of Euler. *The College Mathematics Journal*, 37(1), 2006.

[12] Marco Kuhrmann. User Assistance during Domain-specific Language Design. In *FlexiTools workshop*, 2011.

[13] Matthias Kunze, Alexander Luebbe, Matthias Weidlich, and Mathias Weske. Towards understanding process modeling - the case of the BPM academic initiative. In *BPMN 2011*, pages 44–58, 2011.

[14] Philip Langer, Konrad Wieland, Manuel Wimmer, and Jordi Cabot. EMF profiles: A lightweight extension approach for EMF models. *Journal of Object Technology*, 11(1):1–29, 2012.

[15] Marjan Mernik, Jan Heering, and Anthony M. Sloane. When and How to Develop Domain-specific Languages. *ACM Comput. Surv.*, 37(4):316–344, 2005.

[16] Michael Zur Muehlen and Jan Recker. How much language is enough? theoretical and practical use of the business process modeling notation. In *CAiSE '08*, pages 465–479, 2008.

[17] John Rooksby and Nozomi Ikeya. Collaboration in Formative Design: Working Together. *IEEE Softw.*, 29(1):56–60, 2012.

[18] Michael Rosemann, Jan Recker, Marta Indulska, and Peter F. Green. A study of the evolution of the representational capabilities of process modeling grammars. In *CAiSE 2006*, pages 447–461, 2006.

[19] Jesús Sánchez Cuadrado, Juan de Lara, and Esther Guerra. Bottom-up Meta-Modelling: an Interactive Approach. In *MODELS conf.*, pages 1–17, 2012.

[20] Jesús Sánchez Cuadrado and Jesús García Molina. Building Domain-specific Languages for Model-driven Development. *IEEE softw.*, 24(5):48–55, 2007.

[21] Markus Völter. MD*/DSL Best Practices, 2011.

[22] Ingo Weber, Hye-Young Paik, Boualem Benatallah, Corren Vorwerk, Zifei Gong, Liangliang Zheng, and Sung Wook Kim. Personal process management: Design and execution for end-users. Technical Report UNSW-CSE-TR-1018, 2010.