Improving the Developer Experience with a Low-Code Process Modelling Language

Henrique Henriques, Hugo Lourenço
OutSystems
Linda-a-Velha, Portugal
(henrique.henriques|hugo.lourenco)@outsystems.com

Vasco Amaral, Miguel Goulão
NOVA LINCS, DI, FCT/UNL
Lisboa, Portugal
(vma|mgoul)@fct.unl.pt

ABSTRACT

Context: The OutSystems Platform is a development environment composed of several DSLs, used to specify, quickly build and validate web and mobile applications. The DSLs allow users to model different perspectives such as interfaces and data models, define custom business logic and construct process models. **Problem:** The DSL for process modelling (Business Process Technology (BPT)), has a low adoption rate and is perceived as having usability problems hampering its adoption. This is problematic given the language maintenance costs. **Method:** We used a combination of interviews, a critical review of BPT using the "Physics of Notation" and empirical evaluations of BPT using the System Usability Scale (SUS) and the NASA Task Load indeX (TLX), to develop a new version of BPT, taking these inputs and Outsystems' engineers culture into account. **Results:** Evaluations conducted with 25 professional software engineers showed an increase of the semantic transparency on the new version, from 31% to 69%, an increase in the correctness of responses, from 51% to 89%, an increase in the SUS score, from 42.25 to 64.78, and a decrease of the TLX score, from 36.50 to 20.78. These differences were statistically significant. **Conclusions:** These results suggest the new version of BPT significantly improved the developer experience of the previous version. The end users background with OutSystems had a relevant impact on the final concrete syntax choices and achieved usability indicators.

CCS CONCEPTS

• **Software and its engineering → Software usability**; **Domain specific languages**; **Visual languages**;

KEYWORDS

Low-Code Languages, Developer Experience

ACM Reference Format:
Henrique Henriques, Hugo Lourenço and Vasco Amaral, Miguel Goulão. 2018. Improving the Developer Experience with a Low-Code Process Modelling Language. In *ACM/IEEE 21th International Conference on Model Driven Engineering Languages and Systems (MODELS '18), October 14–19, 2018, Copenhagen, Denmark.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3239372.3239387

MODELS '18, October 14–19, 2018, Copenhagen, Denmark

ACM ISBN 978-1-4503-4949-9/18/10…$15.00
https://doi.org/10.1145/3239372.3239387

1 INTRODUCTION

Modelling Languages are increasingly adopted in industry. Improving the developer experience with those languages has a potential economic impact both by facilitating their adoption and by making developers more productive. Moody's seminal work on the "Physics of Notations" [23] has raised awareness to the importance of effective visual notations. However, there is scarce evidence and examples of industry-strength studies highlighting these benefits.

The *OutSystems Platform* is used to create web and mobile applications with a set of integrated Domain-Specific Languages (DSLs). These DSLs are visual modelling languages that allow developing applications at a high abstraction level, hiding low-level details about creating and publishing those applications. This allows significantly faster development times and a higher quality result when compared to general-purpose languages [29]. The platform is used both internally and by external organizations, free-lancers, and even end-users, which develop their projects using this technology.

OutSystems includes a DSL called *Business Process Technology* (BPT) for process modelling. BPT is used by developers with programming and process modelling knowledge and as a communication medium with business managers. Through interviews with OutSystems developers and data collected from recent projects, we found that BPT was not having the expected adoption rate (less targeted languages within OutSystems are used for process modelling) and was being used for purposes other than process modelling. Maintaining BPT has an associated cost. It was important to identify possible flaws in the language and make any necessary changes to raise its value for the company and its customers.

We used a combination of techniques for developing an improved version of BPT, including *a)* an analysis based on the "Physics of Notations" [23], *b)* interviews with professional BPT users, *c)* a "crowdsourced" approach to the production of an improved version of BPT's concrete syntax and *d)* its evaluation in terms of semantic transparency [2], along with *e)* a usability evaluation using the System Usability Scale (SUS) [1] and *f)* a NASA Task Load indeX (NASA TLX) [8] assessment of the effort involved in using BPT.

This combined methods approach has allowed for the production and implementation of a significantly improved version of BPT, in terms of its usability. The process is abstract enough to be applied to other visual modelling languages. The whole evaluation and improvement process of BPT was used as a testbed by OutSystems for the combination of these techniques to support language evolution.

We introduce BPT (section 2), the language evaluation process for BPT (section 3), and the new BPT proposal and the usability experiment (section 4). On section 5 we discuss results and implications for practice. We then present related work (section 6) and summarise the main conclusions (section 7).

2 BUSINESS PROCESS TECHNOLOGY

2.1 OutSystems Platform

The architecture [16] is divided into three main components: *Service Studio*, *Platform Server* and *Application Server* (Figure 1).

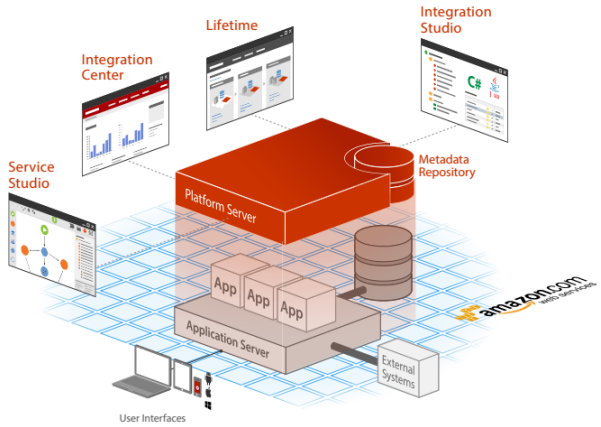

Figure 1: *OutSystems Platform* architecture [7].

Service Studio is the development environment for all the DSLs supported by OutSystems. When the developer publishes an application, *Service Studio* saves a document with the application's model and sends it to the *Platform Server*. The IDE is divided into four main views: process modelling, interface flows, custom logic/APIs access and database modelling.

Platform Server synthesises code given a particular stack (e.g., for a Windows Server [21] using SQL Server [20] this will be ASP.Net [18] and SQL code). The compiled application is then deployed to the *Application Server*. The *Platform Server* also includes a *Scheduler Service*. This service manages the execution of steps within process models developed using BPT and also of scheduled jobs resulting from *Timers*.

Application Server runs on top of *Oracle WebLogic* [26], *JBOSS* [10] or *IIS* [19]. The server then stores and runs the developed application which is connected to a relational database management system, which can be *SQL Server*, *Oracle* [28] or *MySQL* [27]. The SQL code generated by the *Platform Server* is specific to the selected database management system.

2.2 Current BPT concrete syntax

BPT users design, execute and manage processes which are fully integrated with applications built with the *OutSystems Platform*. Figure 2 summarises BPT's concrete syntax.

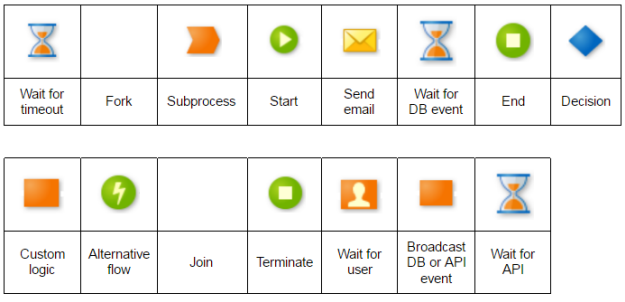

Figure 2: BPT development environment.

Start starts the process flow. There can only be one Start in a process. An *Alternative flow* is used to start a new parallel flow in the process. It has an attribute called *Launch On* where the user defines what condition triggers the flow (e.g. a data-base event or an API call). *End* and *Terminate* share the same symbol, to either terminate the whole process or the particular flow it is connected to, respectively. *Subprocess* calls another process. *Wait for user* (also known as *Human Activity*) is linked to a pre-developed *Web Screen* (an Interface designed using another part of the *OutSystems Platform*) and pauses the flow waiting for the user to trigger an action on that *Web Screen*. An *Automatic Activity* contains an action flow which is defined in a separate window. The action flow can include *Custom Logic*, event broadcasts via the database (*Broadcast DB*), or API calls (*API Event*). *Wait* pauses the process flow. The flow can then be resumed by a specific API call (*Wait for API*), a database event (*Wait for DB*) or an associated timeout (*Wait for Timeout*). *Send Email* is associated to a pre-developed email screen (which can contain dynamic data values). When the flow reaches this node it sends the email to the email addresses entered in the node's attribute. The *Decision* node has *n* outgoing flows and the chosen flow is decided based on custom logic defined in a separate window. Finally, although the language semantically supports parallelism, with a similar semantics of activity diagrams, there are no symbols for the concepts of *Fork* and *Join*.

3 BPT EVALUATION

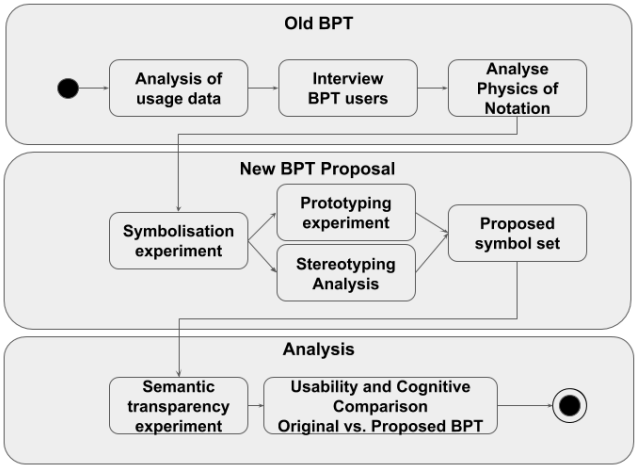

Figure 3: BPT evaluation process.

As depicted in Figure 3, the process starts by evaluating the existing BPT, in order to identify improvement opportunities. The evaluation consists of an *analysis of usage data from Outsystem's Platform logs*, *interviews with BPT users*, an *analysis of BPT using the "Physics of Notations"*, and a *usability analysis*.

3.1 Analysis of usage data from the OutSystem's platform

Procedure We analysed a repository of real application models (i.e. deployed in customers) and collected metrics concerning the percentage of those that actually use BPT, the percentage of BPT models that use process metadata (a bad smell in BPT, due to severe

time performance implications), and the average number of nodes of BPT models.

Results From a repository of 5145 OutSystems application models available in the OutSystems platform, only 179 (around 3%) used BPT. There were a total of 353 BPT models, of which 120 used process metadata. The 353 BPT models had an average of 18.7 nodes per BPT.

3.2 Interviews with BPT users

Procedure We interviewed 6 developers (2 Senior and 4 Lead Engineers) with at least 3 years of experience working with BPT developing applications for OutSystems clients. The interviews followed the Design Thinking philosophy of empathy interviews [30], where the interviewee tells a story, from which one can collect more insights than those which would normally be available through answers to direct questions. The goal is to get to the root of problems by applying the *five whys* technique [3]. This is an iterative interrogative technique to explore cause-and-effect relationships underlying a particular problem by repeatedly asking *why* (at least *five* times). The interviews covered the following topics:

- **The context BPT is being used in.** Although BPT is a DSL it is possible that BPT is being used out of its domain. Usability issues can occur when a language is used in a domain that it was not designed for.
- **Why BPT was chosen.** This brings up the strengths of BPT and the features that the interviewee likes, and may help to start a conversation about things that can be further improved in said features.
- **What features are less used and why.** Features that are not getting much use may need to be changed, removed or better explained with training, documentation, etc. It is possible that the users do not use a certain feature because they do not know enough about it or its potential usefulness.
- **What features are missing.** With the daily usage of the language does the expert feel like there is something missing? Are there use-cases within the language's domain that can not be answered?

Results The following insights about BPT were extracted from the interviews:

- Development teams liked using BPT but were "scared" to do so due to low-level *nuances*. They preferred to fall back to what they were used to (Timers).
- Parallelism was hard to model and so was identifying synchronization bugs.
- New team members could not start working with BPT without specialised training.
- Some clients explicitly requested the use of BPT.
- Maintaining a project with BPT was difficult and costly.
- BPT was often used outside of its domain. While BPT was designed to be a process modelling language it was also being used for event handling. These event handlers are very small processes (normally around three or four nodes). They start automatically in response to an event (like an API call), perform a small automatic action and then end. The problem is that the language runtime was not designed for this behaviour and, as such, does not perform well.

3.3 BPT analysis using the *"Physics of Notations"*

Procedure We conducted a critical analysis of BPT, following the Physics of Notations (PoN) principles. We checked if BPT complies with each of the PoN nine principles, to determine the extent to which BPT's concrete syntax adheres to them and, in that process, identify concrete syntax improvement opportunities that would mitigate the identified non-conformities to those principles.

Results In this section, we outline the main conclusions of our analysis of the BPT concrete syntax using the PoN as a reference framework. We do this by analysing each of the PoN 9 principles.

Semiotic clarity: There should be a one-to-one relationship between semantic constructs and concrete syntax. We found cases of *symbol deficit*: while there is semantic support for using *Forks* and *Joins* to model parallelism, there were no special symbols for these. This symbol deficit may explain why our interviewees reported difficulties with identifying synchronization bugs and modelling parallelism (see section 3.2). We also found cases of *symbol overload*: the *Timeout* symbol is used for three types of *waits* (all with different behaviours); the *End* symbol is the same used for *Terminate*; there are several ways of triggering a process but the *Start* symbol does not reflect this. There are also some issues with platform consistency which may be regarded as a symbol overload problem when considering the other OutSystems DSLs. The *Custom logic* and *Broadcast DB or API event* constructs are represented by an orange ball in other OutSystems DSLs.

Perceptual discriminability: Symbols with higher visual distance are easier to distinguish. No symbol had a visual distance greater than 2 visual variables. Colour and shape were the prevalent variables present in all the symbols. BPT uses textual differentiation, with all but the *End* symbol having a label defined by the developer. The *End* symbol has a label defining its behaviour: *End* or *Terminate*. There were no mechanisms for redundant coding.

Semantic transparency: Semantic transparency is the extent to which the meaning of a symbol can be inferred from its appearance. We made two complementary assessments of semantic transparency: a critical analysis of the concrete syntax, reported here, and an experimental evaluation, comparing the semantic transparency of the original BPT with the one of the proposed improved syntax of BPT (section 4). In our critical analysis, we detected three visually opaque symbols. *Call Subprocess* and *Automatic Activity* have a shape that is not related to the corresponding notions. The *Conditional start* symbol is represented with a lightning symbol which is not related to the notion of starting. However, this is mitigated by both *Start* and *Conditional start* sharing the same colour.

Complexity management: BPT has two mechanisms for dealing with complexity: the *Call subprocess* construct and the *Automatic Activity*. The subprocess construct calls another process and only proceeds after all flows of the subprocess finish. The automatic activity encapsulates logic but does not allow it to be reused. The *Subprocess* construct is a good mechanism for managing complexity. It promotes reusable code and allows for several hierarchical levels. The *Automatic Activity* could be improved by allowing re-usability.

Cognitive Integration: A language should include some explicit mechanisms to support the integration of information from

different diagrams. Two relevant mechanisms are *Conceptual integration* and *Perceptual integration*. Conceptual integration provides mechanisms (e.g. a summary diagram) to help the reader assemble information from separate diagrams into a coherent mental representation of the system. However, BPT does not support such mechanisms. Perceptual integration offers cues to simplify the navigation between diagrams. BPT supports *orientation* and *destination recognition* through the labelling of diagrams, but no explicit support for *route choice* and *monitoring*.

Visual expressiveness: The number of visual variables used in a notation defines its visual expressiveness. BPT has a visual expressiveness of 2. It only uses *shape* and *colour* as information carrying variables. *Horizontal* and *vertical position*, *size*, *brightness*, *texture*, and *orientation* are free variables in BPT.

Dual coding: Textual encoding should supplement, rather than substitute graphics. However, BPT uses text as a way to distinguish symbols (e.g. the *Custom logic* and the *Broadcast DB or API event* are indistinguishable without textual labels). On a more positive note, BPT supports a text annotation construct, so that developers can add optional text to the process.

Graphic economy: The number of symbols in the language should be manageable. BPT has 9 different symbols, which is over the recommended upper limit of 6 [23]. This shortcoming is mitigated by the fact that BPT is normally used within *Service Studio*, where a toolbar also serves, in practice, as a key for the BPT diagrams. This mitigates the potential difficulty in remembering what each symbol means, which is in general more challenging for novices when understanding software engineering diagrams [25].

Cognitive Fit: The cognitive fitness principle suggests that different representations of information are suitable for different tasks, audiences, and media. As is common in most Software Engineering languages, BPT uses a single visual representation for all purposes and audiences. However, this is not perceived as a significant shortcoming, as BPT's notation is relatively small and simple to understand. Having separate dialects for experts and novices seems unlikely to bring significant benefits. Further research would be required to assess the potential impact of specific tasks on the usability of BPT. The notation is not easy to sketch since the language was designed to be used only within *Service Studio*.

4 NEW BPT PROPOSAL

4.1 Research questions

During the analysis of the current version of BPT, we concluded that one of the areas that could be improved was its concrete syntax. Our goal was to create a new set of symbols that had a one to one relation between symbols and semantic constructs, and with a high level of semantic transparency. This was done by applying a modified method adapted from the work by Caire *et al.* [2]. That said, the results of the Physics of Notations analysis should not be ignored and other factors (such as consistency) also needed to be considered. Three research questions guided our quasi-experiments on the semantic transparency of BPT:

- **RQ1.** Is the original BPT concrete syntax semantically opaque?
- **RQ2.** Can participants unfamiliar with BPT design more semantically transparent symbols for BPT than the original?
- **RQ3.** Which concrete syntax (original, stereotype, prototype, proposed) is more semantically transparent?

After conducting the semantic transparency evaluation of different versions of BPT, we further compared the *usability* and *cognitive effort* of the *original BPT* with the BPT version proposed in this paper. This lead to three additional research questions:

- **RQ4.** Which concrete syntax (*original*, *proposed*) leads to a better understandability of BPT models in the context of interpretation tasks?
- **RQ5.** Which concrete syntax (*original*, *proposed*) leads to a better understandability of BPT models, as perceived by practitioners after performing model interpretation tasks?
- **RQ6.** Which concrete syntax (*original*, *proposed*) leads to a lower cognitive effort, as perceived by practitioners while performing BPT model interpretation tasks?

4.2 Research design

The research design consisted of 6 related empirical studies, where the results of the earlier studies provide inputs to the later studies

We conducted five interrelated studies. Three were experiments involving novices. The participants used in each of the experiments were exclusive to that experiment, so that they would not be influenced by their own participation in other experiments. Additional material on all these empirical studies can be found in this paper's companion site [11].

a) **symbolization experiment:** novices were asked to draw a set of symbols that they thought best represented the language constructs;
b) **Stereotyping analysis:** a set of symbols was built based on *the most common symbols drawn by the novices*;
c) **Prototyping experiment:** a different group of novices was asked to identify the best symbol for each construct - *the most frequently selected symbol for each construct* was chosen;
d) **Proposed symbol set:** a set of symbols was built, taking into account the results from the stereotyping and prototyping experiments, the interviews with users and the eye-tracking usability tests;
e) **Semantic transparency experiment:** a third group of novices were asked to infer the meaning of each symbol. This was done for the *original*, the *stereotype*, the *prototype* and the *proposed* symbol sets. This last empirical study is the one where we finally evaluated the three research questions presented in section 4.1.
f) **Usability and cognitive effort comparison:** The original and the proposed BPT were compared in terms of their usability and of the cognitive effort associated in using them.

4.3 symbolization experiment

i) Goal: Semantic transparency is achieved when users can infer the meaning of a symbol. A possible way to achieve an acceptable level of transparency is to have members of the target audience generate a set of symbols for the language. This was done applying the sign production technique [12], where novices were asked to draw symbols that best represent each of BPT's semantic constructs.
ii) Participants and materials: The 24 participants were software developers (half of them were students, the other half professionals)

between 18 and 32 years old. None of them had prior experience with BPT. The sign production questionnaire contains:

- a cover page with information about the study, a disclaimer and an out of context example to exemplify the expected answer format;
- fifteen questions (one for each of the semantic constructs) and respective answer box;
- a final page with a series of screening and demographic questions.

iii) Procedure: The participants received a printed questionnaire and were asked to answer them. They took from 30 to 40 minutes to complete the questionnaire. To process the questionnaires, we developed an application that receives as an input the questionnaire in digital format, cuts the answers into separate images, saves and indexes them. This allows viewing all the answers of a specific questionnaire or viewing all the answers for one specific construct.
iv) Results: The outcome of this activity was a dataset of 24x15 symbol proposals for an improved BPT concrete syntax.

4.4 Stereotyping analysis

i) Goal: The stereotyping analysis builds on the assumption that, if several participants think of the same visual metaphor when proposing a representation for a given construct, this metaphor is likely to be easily recognizable by others. The goal of this task was to build a stereotype concrete syntax based on the most drawn metaphors for each construct.
ii) Materials: The input for this analysis was the symbol set collected from the questionnaires produced in the symbolization experiment, described in section 4.3.
iii) Procedure: The analysis of the drawings generated by the sign production technique was done using the judges' ranking method [14]. The symbols were first classified into categories based on their conceptual similarity. Then, we chose the symbol that was most representative of the most frequent category.

For example, the symbols for the semantic construct *Start process* were divided into five categories: *media play button*, a *text*, a *power button*, a *traffic light* and an *on switch*. Of the total symbols, 11 were placed in the *media play button* category, 6 in *text*, 2 in *power button*, 1 in *traffic light* and 2 in *on switch*. The remaining symbols were not categorised because they did not make sense or were unreadable. So, for the *Start process* construct the chosen symbol was the symbol that best-represented *media play button*. This process was repeated for each semantic construct.
iv) Results: The stereotyping analysis resulted in a set of 15 symbols, one for each construct (Figure 4). None of the symbols had an absolute majority. The large variety of symbols, and in some cases the lack of answer, illustrates the difficulty in creating a concrete representation for the constructs.

4.5 Prototyping experiment

i) Goal: A potential shortcoming of the stereotype symbol set is that the most drawn symbols are not necessarily those that better convey BPT's constructs. A visual metaphor may be a mnemonic of a construct's name but not a good representation of the concept itself [2, 14]. As such, we conducted a prototyping experiment to identify which icons were better metaphors for the concepts, rather than for their names. **ii) Participants and materials:** We had a

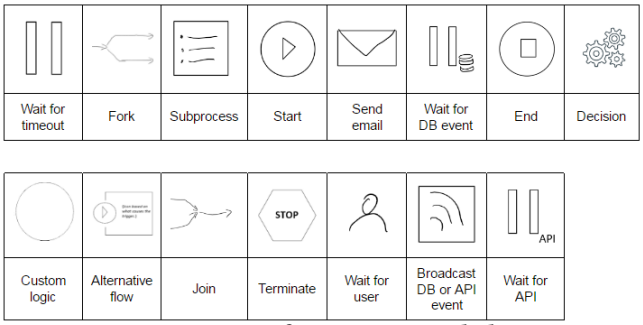

Figure 4: Set of stereotype symbols.

mixture of students and professionals, all software developers without previous knowledge of BPT, participating in the prototyping experiment. Most of the 16 participants were recruited through a digital third-party platform for usability tests called *UsabilityHub* [34].

We created a questionnaire where each question had a description of a semantic construct and a set of possible symbols for that construct. The possible choices for each construct were a symbol from each category previously defined in 4.4. There were two versions of the questionnaire: one on paper (which can be found at the companion site) and another made available through *UsabilityHub*.
iii) Procedure: Participants were asked to choose the best symbol to represent the description of the semantic construct. Most of the answers were collected through *UsabilityHub*. The data exported from that platform was then complemented with the data collected from the few participants using the paper version of the questionnaire.
iv) Results: The set of most frequently chosen symbols for each construct is our *prototype symbol set* (Figure 5). Again none of the symbols were selected by an absolute majority. While a few symbols matched the Stereotyping analysis, others were radically different. The *Subprocess* construct is especially noteworthy since it has a dynamic effect: it is a real-time representation of the process being called which expands when the user mouse hovers it.

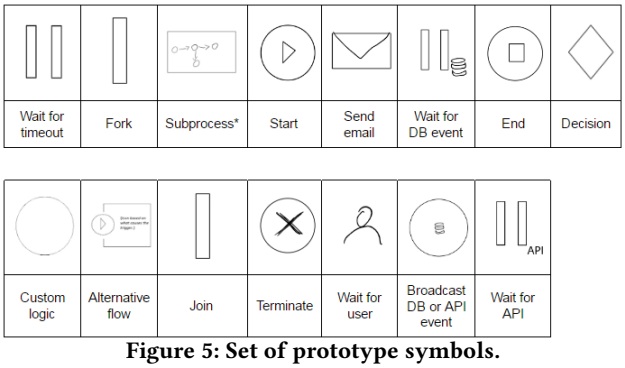

Figure 5: Set of prototype symbols.

4.6 Proposed BPT

i) Goal: A shortcoming of the stereotyping and prototyping experiments is that each symbol is generated independently. This neglects the development environment and the need for consistency with the rest of the OutSystems platform. Our goal was to propose an alternative BPT that would leverage this consistency concern, adding

it to the knowledge gathered with the symbolization, stereotype and prototype experiments.

ii) Materials: We used as inputs the sets of stereotype and prototype symbols, as well as the feedback collected through interviews with OutSystems developers (already described in 3.2) and our knowledge of the rest of the OutSystems DSLs.

iii) Procedure: We proposed a new concrete syntax that takes into account consistency with the other OutSystems DSLs, while remaining as close as possible to the set of prototype symbols, as we expected these to be the most semantically transparent alternatives. However, while some changes were direct (simply changing the icon), others required changes to how the flows work and therefore go beyond the concrete syntax. This section goes over each of the changes made to BPT and how they were implemented. Note that, in the present stage, all these changes were developed as a prototype with the goal of testing the proposed concrete syntax. Further development is still necessary to make them part of the actual OutSystems platform.

Adding new elements, updating symbols and syntactic rules. The original BPT set of symbols was comprised of nine symbols while the new one has a total of fifteen different symbols. So, in order not to increase the language's complexity by adding six new symbols to the toolbar, certain symbols were placed in groups and the symbol changes based on attribute values. As such, only two new symbols had to be added to the toolbar: *Fork* and *Join*.

The BPT language is defined by a meta-model which contains all the syntactical rules and constraints. This is consumed by a compiler which then generates a series of partial classes which can then be completed with the language's semantics. To add the new elements, nodes were created in the meta-model but their semantics were not touched at this stage given that the changes were just to prototype the syntax.

To view what causes a process to start, one has to view the properties of the process. During the usability tests, 100% of the participants first looked for that information on the Start button. As such, the property was replicated to the Start button and any change made on button is updated in the process's property. This ensures there will not be any conflicts even though there is redundancy.

Some symbols have the same behaviour as the original ones. The only change made was the replacement of the original icon with the new one. This was the case for the *Decision* and *Wait* nodes.

As the language now has explicit symbols for parallelism, the syntactical rules for outgoing arrows had to be updated. Originally any node could have *N* outgoing and incoming arrows. We changed this to ensure that only the *Decision* (number of outgoing arrows is based on the condition) and *Fork* can have *N* outgoing arrows.

Symbol groups. When adequate, we grouped symbols together in order to reduce complexity. For each of these groups, we chose a symbol to represent the group. This symbol is then decorated with an overlay based on properties. We created the following groups:

- **Waits.** This group includes the *Wait for API call*, *Wait for DB event* and the *Wait for timeout* constructs. The symbol used to represent the group is the *Wait for API call* symbol since it is the most generic of the three.
- **End and Terminate.** This group contains the *End* and *Terminate* symbols. They both have similar functionality but the *End* is most commonly used, as such it was chosen to represent the group.
- **Start.** The launch of a process can be done in different ways. The default symbol for *Start* is the one presented in 8. However, if the process is launched via a DB event then the symbol is updated with a small overlay.
- **Conditional start.** The proposed *Conditional Start* symbol has an overlay representative of a DB event. But, the conditional start can also be triggered by an API call. As such, the default symbol used does not have an overlay but if the user chooses a DB event as a trigger then the overlay is applied.

Using Actions in BPT. An Action is a piece of reusable code. There are different types of Actions: *created by users* which can include any type of custom logic, a set of *system actions* provided by *Service Studio*, *entity actions* that are used to manipulate the database, and *API actions* which interact with triggers and APIs.

Originally in BPT, all action calls and logic had to be encapsulated in an *Automatic Activity* (Figure 6). These were not reusable and at times created unnecessary complexity. Changes had to be made to ServiceStudio's syntax to allow Actions to be used in the BPT flow. With this, Actions have now replaced the *Automatic Activity* (Figure 7) since they provide generalization and re-usability while not creating unnecessary complexity.

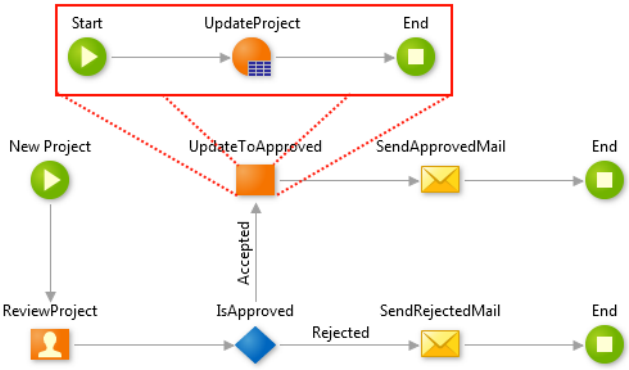

Figure 6: Automatic activity (original BPT)

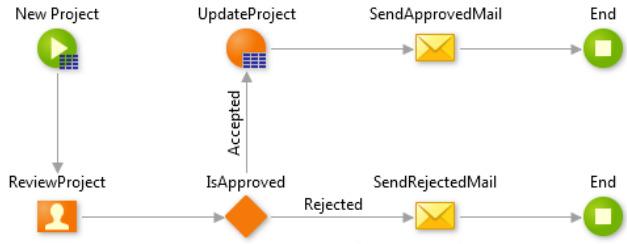

Figure 7: Actions (new BPT)

v) Results: Figure 8 presents the proposed concrete syntax for BPT.

The *End* and *Terminate* remain green (even though that could be considered semantically perverse) because *End* is green on the rest of the platform. Changing it in the rest of the platform would have a large impact on established users so it was decided that it would be best to keep the green in these elements.

The most important changes when compared to the prototype symbol set concern the database related symbols (*Wait for DB event* and *Broadcast DB or API event*), the *Wait for timeout* and *Wait for API*, the *Fork*, the *Join* and the *Alternative flow*:

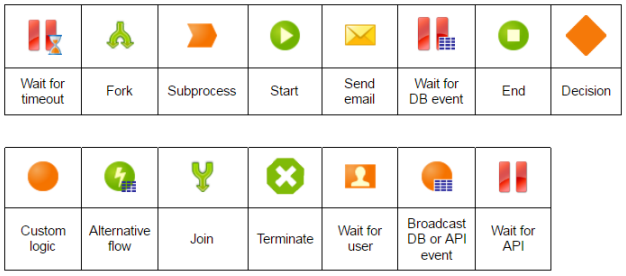

Figure 8: Set of proposed symbols.

- **The database related symbols:** Both the stereotype and prototype symbol sets have use three cylinders to represent any activity related to a database. This is a common representation for databases. However, *Service Studio* uses a different representation (a blue table). As such, all representations related to data were changed to maintain consistency.
- **Wait for timeout or API:** These were switched because waiting for an API call is an unconditional pause. A timeout is a condition. In order to be consistent with the database wait (which is also conditional), it was decided it would be best have an overlay on the conditions.
- **Fork and Join:** Arrows in ServiceStudio only specify flows and have no semantic definition. As such, the fork and join in the stereotype symbol would be a drastic change to the common behaviour of arrows. The metaphor from the prototype symbol set was not used because it makes it harder to differentiate forks from joins. The symbols proposed came in second on the prototyping experiment.
- **Alternative flow:** the symbols in both *stereotype* and *prototype* were too similar to the *Start* symbol. We decided to maintain the current symbol and add an overlay depending on what triggers the alternative flow.

4.7 Semantic transparency experiment

i) Goal: The goal of this study was to **evaluate the semantic transparency of the 4 alternative symbol sets for BPT**: *original BPT* (Figure 2), *stereotype BPT* (Figure 4), *prototype BPT* (Figure 5), and *proposed BPT* (Figure 8). We conducted a blind interpretation study where participants had to infer the construct associated with each symbol. Comprehension tests [36] are commonly used to measure the symbol's transparency and is recommended by the International Organization for Standardization (ISO) for *testing the comprehensibility of graphical symbols* [13].

ii) Participants and materials: The participants were 20 MSc students from Universidade Nova de Lisboa (UNL) from the Informatics course. None of them had previous knowledge of BPT. We created a survey for each of the 4 different sets of symbols. Each survey had a question for each of the symbols in each set, followed by a list with all of the language's semantic constructs. Participants were able to select one or more constructs that, in their opinion, were best represented by the corresponding symbol. The surveys were created in digital form. We developed a web application that asked the participant to input their (academic) email, and prevented answers from repeated emails. The application randomly redirected the participant to one of the four surveys, while keeping a balanced distribution of respondents among alternative notations (in practice, for every 4 participants, one would be randomly assigned to each of the alternative notations). This ensured that a tester could not answer more than one survey, to prevent bias. In the end, we had 5 respondents per notation.

iii) Procedure: Each participants received a link to one of the 4 alternative questionnaires and filled it in. There was no fixed time limit for this task, but the estimated time for completion was *"no longer than 15 minutes"*.

iv) Hypotheses, parameters and variables: The independent variable was the concrete syntax (i.e. *original*, *stereotype*, *prototype* or *proposed* BPT). The dependent variable was the symbol's hit rate, used as an indicator of symbols comprehension (a proxy for semantic transparency). We hypothesized that the hit rate for the *Original BPT* would be outperformed by all the alternatives, that the *stereotype* would be outperformed by the *prototype*, which, in turn, would be outperformed by the *proposed BPT*:
$OriginalBPT < StereotypeBPT < PrototypeBPT < ProposedBPT$

v) Results: The results in Table 2 show that, on average, the *Prototype* concrete syntax is the most semantically transparent set of symbols, with 79% of hit ratio. The proposed BPT concrete syntax comes next, with around 69% of hit ratio. Both are above the ISO threshold for comprehensibility (67%) [13]. However, only the *Prototype BPT* has the mode above that threshold (Figure 9). Although with a similar median value, the Stereotype BPT had a lower mean hit rate, when compared to the *Prototype* and the Proposed BPT. The *Original BPT* obtained the lowest mean hit rate. In spite of not being the notation with the best hit rate, the *proposed BPT* was selected in the end for further analysis, as it is the one that best fits into the remaining OutSystems DSLs landscape. As such further comparisons will focus on the original BPT and the *proposed BPT*.

Table 1: Hit rate

Construct	Original	Stereotype	Prototype	Proposed
Wait for timeout	**1.00**	.60	**1.00**	**.80**
Run subprocess	.40	.20	.60	.20
Start process	.40	**1.00**	**.80**	.60
Send email	**.80**	**1.00**	**1.00**	**1.00**
Wait for DB event	.20	**1.00**	**1.00**	**.80**
End process	.40	.20	**1.00**	.60
Decision	.40	.0	.60	.60
Custom logic	.0	.0	.60	.40
Alternative flow	.0	.40	.40	.60
Fork	.0	**1.00**	.40	**1.00**
Join	.0	**.80**	.40	**1.00**
Terminate	.20	**1.00**	**1.00**	**.80**
Wait for user	.60	.60	**1.00**	**.80**
Wait for API call	.20	.60	**1.00**	.60
Broadcast DB event	.0	**.80**	**1.00**	.60
Mean Hit Rate	.31	.61	**.79**	**.69**
Standard Deviation	.31	.37	.26	.23

4.8 Usability and cognitive effort comparison

i) Goal: Even if the language has a high transparency rating, assessed here indirectly through the *hit rate*, it is important to conduct more usability experiments. The evaluations described in the previous sections evaluated symbols individually. This does not ensure a high language usability rating. We followed the method described in section 4.8 to further assess the *proposed BPT* in terms of its usability and of the cognitive effort required to use it, contrasting it with the *original BPT*.

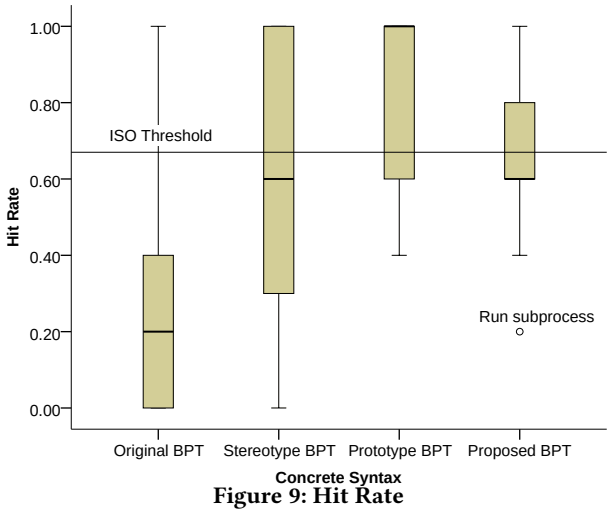

Figure 9: Hit Rate

ii) Participants and materials: A total of 25 subjects participated in this evaluation. They were professional software developers aged between 23 and 40 years old and experienced with the OutSystems platform. None of them had previous experience with BPT.

We created two versions of the evaluation material, including three BPT models with varying levels of complexity that covered all the BPT language constructs. Semantically equivalent models were represented with the *original BPT*, in the first version, and the *proposed BPT*, in the second version. For each of those models, there was a set of questions, presented on a side window with a Google forms questionnaire. We asked participants to answer those questions, to assess their level of understanding about each model. The questions were objective (e.g. *"What elements of the language interact with the database?"*) rather than subjective (e.g. *"What do you think of X?"*), so that we could objectively assess whether the answer was correct or not. Overall, the questionnaires contained 9 questions. We also used a Simple Usability Scale (SUS) [1] questionnaire and a NASA TLX questionnaire [8] to assess the usability and cognitive effort required to answer the interpretation questions, respectively, as perceived by the participants.

The setup included a single-screen computer, equipped with an eye tracker, which was calibrated for each participant, at the beginning of the evaluation session. The OutSystems development environment was open, with a solution built with BPT.

iii) Procedure: We conducted one-on-one experiments with developers. 16 participants answered questions about models built with the *original BPT*. Another group of 9 participants answered questions about the same models built with the proposed BPT. During the evaluation session, we recorded the contents of the screen, the eye tracking data of the participant while performing the tasks, and the voice of the participant. We encouraged the participant to follow a *"think aloud"* protocol so that we would obtain richer data for analysis. After answering this questionnaire, the participant also answered a System Usability Scale (SUS) [1] and a NASA TLX [9] questionnaire. Finally, there was a short open discussion where the participant would talk about the issues he had with the language and, in most cases, suggested ideas for mitigating those issues.

iv) Hypotheses, parameters, and variables: The independent variable is the concrete syntax (*Original BPT*, *Proposed BPT*). The dependent variables are the answer correction rate, the SUS score, and the NASA TLX score. For these three variables, we hypothesized that the *Propose BPT* would outperform the *Original BPT* in terms of understanding tasks with BPT models, the perceived usability of BPT and the perceived cognitive effort spent while using it.

v) Results: Table 2 presents the success rate for each question, for *Original BPT* and *Proposed BPT*. The right column presents a short comment concerning how the change of the concrete syntax affected the corresponding answer, as perceived from the observation of the participation of our subjects in the usability experiment (including the eye tracking data) and from their own feedback.

Table 3 contains descriptive statistics for the SUS and TLX data collected while performing the usability tests. The table is grouped by score type and each type contains statistics for the original BPT language (BPT) and the new BPT language (New BPT).

We used the Welch's t-test for testing the differences in the correctness of answers, SUS and TLX scores between the *Original BPT* and the *Proposed BPT*. The Welch t-test is robust to different sample sizes even in the presence of deviations from normality [15]. Table 4 contains: the means for the average correction of answers, the SUS and the TLX scores; the difference between the original and new BPT means; the 95% confidence interval of the difference; *t*, *df* and p-values. We hypothesised that the proposed version of BPT would lead to more correct interpretations of models, have a higher usability rating when compared to the original version and require a lower cognitive effort to be understood in its usage.

Participants using the *proposed BPT* provided more correct answers, with a statistically significant improvement of 0.375. This supports the hypothesis that the *proposed BPT* is easier to interpret than the original BPT. Participants using the *proposed BPT* gave it a higher score, with a statistically significant improvement of 22.53, supporting the hypothesis that the *proposed BPT* leads to improved usability. Finally, participants using the *proposed BPT* reported a statistically significant lower NASA TLX score, with less 15.72 points, supporting the hypothesis that the proposed BPT requires a lower cognitive effort to be understood. Figures 10, 11 and 12 present the distributions of correctness, SUS and TLX, respectively.

5 DISCUSSION

5.1 Evaluation of results

Overall, the *proposed BPT* significantly outperforms the *original BPT* in terms of its actual and perceived usability and, therefore, has the potential for improving the developer experience with it.

RQ1. Is the *original BPT* semantically opaque? Yes. The *original BPT* has a mean hit rate well below the ISO standards requirements for symbol recognisability. This is very common in software design languages [23]. Along with the other shortcomings of the original BPT, identified through interviews with practitioners, the PoN evaluation and the scarce actual usage of BPT in projects, this observation suggests that improving BPT has the potential to improve developer and other stakeholders' experience by making BPT symbols easier to recognize and remember.

Table 2: Results of usability tests

Question	Correct Answer Rate (%) Original	Proposed	Proposed BPT Comments
1. What causes the process to start?	6.25	100	Participants had no problem to find the information since it is now present on the Start button (which was the first place to be checked).
2. Which nodes require human interaction?	100	100	Same result as the Original BPT test. This was expected since the symbol was not changed.
3. What nodes send e-mails?	100	100	This is the same case as the question above.
4. Can this process fail?	62.5	100	Testers had a much easier time understanding the difference between *End* and *Terminate*.
5. Why can this process fail?	31.25	66.66	There is still some confusion about the scope of Terminate. It is not clear if the Terminate in the sub-process also kills the parent.
6. What node finishes a process flow?	31.25	100	With different symbols for each construct testers no longer confuse the two.
7. What node finishes all process flows?	31.25	100	Same as the above.
8. Who are the participants (actors) in this process?	0	33.33	No changes were made to this node so identifying who interacts with the process is still a problem.
9. Who is responsible for each node?	100	100	The testers matched easily the actors (due to the intuitive labels on the nodes). Without labels the results would be much worse.

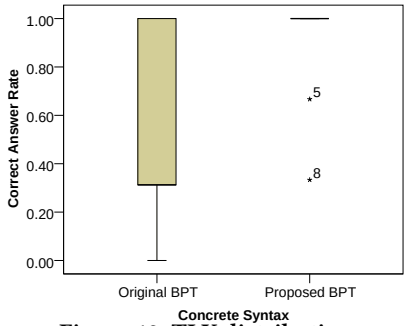

Figure 10: TLX distribution

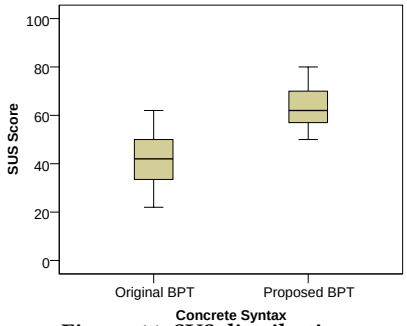

Figure 11: SUS distribution

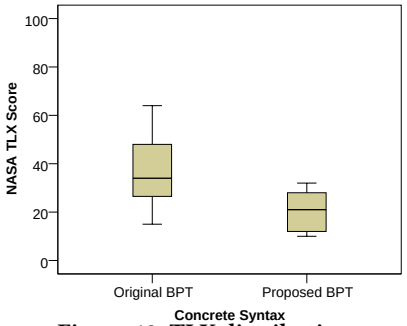

Figure 12: TLX distribution

Table 3: Descriptive statistics

	Language	N	Mean	Std. Dev.	Skew.	Kurt.	S-W
Correct	Original BPT	9	0.51	0.40	0.227	-1.739	0.085
	Proposed BPT	9	0.89	0.24	-2.121	4.001	0.000
SUS	Original BPT	16	42.25	11.72	-0.021	-0.67	0.896
	Proposed BPT	9	64.78	10.02	0.213	-1.05	0.697
TLX	Original BPT	16	36.5	14.47	0.421	-0.61	0.686
	Proposed BPT	9	20.78	8.72	-0.118	-1.81	0.245

Table 4: Welch's t-test scores

	Orig BPT mean	Prop BPT mean	Diff	95% Dif. CI Lower	95% Dif. CI Upper	t	df	p-value
Corr	0.51	0.89	-0.38	-0.71	-0.04	-2.402	12.87	0.032
SUS	42.25	64.78	-22.53	-31.83	-13.22	-5.07	19.01	0.000
TLX	36.50	20.78	15.72	6.12	25.32	3.39	22.80	0.003

RQ2. Can participants unfamiliar with BPT design more semantically transparent symbols for BPT than the original? Yes. Asking participants unfamiliar with BPT to propose alternative representations for the BPT constructs has allowed obtaining more semantically transparent alternatives to BPT. Both the *stereotype* and the *prototype* alternatives have achieved a significantly better semantic transparency. These results reinforce others where notations produced by novices consistently outperform those produced by experts, in terms of symbol recognizability [2, 32]. This suggests that symbolization experiments such as ours are a viable way of developing better concrete syntaxes. A relevant difference from [2] is that instead of producing a PoN-powered alternative for a new concrete syntax *a priori*, we proposed the improved BPT notation *after* analyzing the results from the symbolization experiments. The proposed syntax was inspired by the alternatives previously produced (particularly the *prototype BPT*), combining them it with other concerns, such as the overall coherence of the concrete syntax and how it relates to other existing DSLs in OutSystems.

RQ3. Which concrete syntax (*original, stereotype, prototype, proposed*) is more semantically transparent? The *Prototype BPT* is the alternative with the best semantic transparency. However, its difference to the *proposed BPT* and *stereotype BPT* is not statistically significant. The three alternatives are significantly better than the baseline *original BPT*. Again, this is somewhat similar to what was observed in other evaluations (see section 6).

RQ4. Which concrete syntax (*original, proposed*) leads to a better understandability of BPT models in the context of interpretation tasks? The *proposed BPT* lead to more correct interpretations in the model interpretation experiment. There were noticeable improvements: understanding where a process starts; whether and if a process fails; and, concerning the nodes finishing process flows. The insights collected in this evaluation can be used for further iterations in BPT, in particular for those details that participants still struggled with (Table 2). More importantly, the approach itself is reusable for other languages. With some variations, it has been applied to other languages (see section 6).

RQ5. Which concrete syntax (*original, proposed*) leads to a better understandability of BPT models, as perceived by practitioners, while performing model interpretation tasks? The *proposed BPT* obtained a significantly higher SUS score, denoting that the perceived usability has improved.

RQ6. Which concrete syntax (*original, proposed*) leads to a lower cognitive effort, as perceived by practitioners, while performing BPT model interpretation tasks? Consistently with the perception of increased usability, the perceived cognitive effort has significantly decreased with the *proposed BPT*.

5.2 Implications for practice

The method followed in this paper is applicable to other languages. Indeed, we have partially done so elsewhere [22, 32]. As shown in Table 1, symbols created by our participants were more transparent than those created by language engineers for the *original BPT*. This is consistent with findings in other contexts [2, 22, 32]. We introduced an important variant. Rather than using students as subjects [2, 22, 32], we had OutSystems professional developers as subjects. They are experts in the target platform, although inexperienced with BPT. This has facilitated the creation of the *proposed BPT* as a visual language that is consistent with the rest of the OutSystems platform. Having participants with the same profile as the intended end users, but with a fresh look on the language concepts so they were not influenced by the current syntax of the language being evolved helped to achieve better results than those achieved with "less informed" participants in the symbolization experiments [33].

The results of the usability experiments stress the importance of having a one-to-one relationship between the concrete and semantic constructs. This corroborates the Physics of Notations [23].

As expected, the **TLX** score decreases as the **SUS** score increases. This suggests a negative correlation between language usability and the perceived cognitive effort using it.

Our participants could not identify several constructs, in the *original BPT*. This was one of the problems related to symbol overload or deficit, reinforcing the need to have a one-to-one relationship between the semantic constructs and the concrete syntax.

The changes introduced in the *proposed BPT*, when compared to the *prototype BPT*, in order to preserve consistency with the platform resulted in a minor, statistically insignificant, decrease in transparency. The *proposed BPT* has still a mean hit rate above the ISO threshold. This illustrates the importance of context in language engineering (e.g. the colour choices for termination symbols only make sense for the context of the OutSystems platform). Understanding what works for the actual end users was key.

5.3 Threats to validity

We need to consider potential validity threats [35]. Population selection is a threat, as, due to resource constraints, all the participants used in the usability experiments were members of OutSystems (not part of the **BPT** development team). Ideally, there would also be representatives of business managers, as they are also stakeholders for BPT. Further research is required to assess the *proposed BPT* with those stakeholders. Also, due to the strict time availability of the participants (as is common in these experiments), the usability experiments were limited to three BPT diagrams of varying complexity. While those diagrams were selected for being as representative as possible of BPT, there is always the potential for increasing the external validity of these results by performing replications of this evaluation with different BPT diagrams.

6 RELATED WORK

Moody *et al.* evaluated the i^* concrete syntax using PoN and proposed a new symbol set for it [24]. Caire *et al.* compared Moody's proposed concrete syntax with alternatives produced by novices (a stereotype and a prototype concrete syntaxes) and the standard i^* concrete syntax [2]. We adapted Caire's protocol. Instead of having previously defined a PoN-informed concrete syntax for BPT, we also used the concrete syntaxes proposed through a symbolization experiment as input for the development of the *proposed BPT*.

PoN was used to evaluate and identify improvement opportunities for several modelling languages, such as BPMN 2.0 [5], Use Case Maps [4], WebML [6], and misuse cases [31]. In general, these studies reached conclusions similar to those advanced by Moody concerning the challenges in most visual notations, including UML, from a PoN perspective [23].

Matulevičius *et al.* used interviews, models creation, and evaluation of those models and the modelling language for assessing the i^* and KAOS modelling languages [17] and found clarity problems in those languages semantics definition.

7 CONCLUSIONS AND FUTURE WORK

While running into maintenance costs, and having identified the need for improving the usability of the commercial business process modelling language (BPT) at OutSystems, we have designed and put forward a systematic process to identify and fix usability issues.

We identified issues on syntactic and semantic constructs, and proposed a new notation. After evolving BPT within OutSystem's development environment, we applied the evaluation process to the proposed BPT and observed a significant increase in usability.

The comparison analysis between the original and the proposed version of the BPT confirmed that the process is effective and that the new notation has a higher usability rating. We could also conclude that: Semantic transparency has a large impact on usability; users create more semantic transparent symbols than language engineers (which goes in line with what Caire *et al.* concluded [2]); it is extremely important to have a one-to-one relationship between the concrete syntax and the semantic constructs.

To be generalizable, as future work, the evaluation process proposed in this work can be expanded with further techniques and also modified to apply to textual languages. The evaluation process identifies issues with a language's concrete and semantic constructs. However, it only provides methods to improve the language's concrete syntax. The process should be further expanded with techniques that help language engineers design semantic constructs from the ground up that answer the needs of the users while having a high usability rating. This could be achieved with the addition of techniques from Requirements Engineering, but further research is required. The proposed usability evaluation process should also make more use of the eye-tracking data. The current process only uses the eye-tracker to manually revisit recordings of the usability tests and to identify possible reasons for wrong answers. However, there are several metrics that can be extracted from the eye-tracking results that provide objective information about the tester (states of confusion, being lost in the interface, etc.). This, added to the SUS score provides a more accurate overview of the language's usability. During the analysis of the comparison between the original BPT notation and the new one, we noted that there appears to be a negative correlation between a language's SUS and NASA TLX. This should be further researched.

ACKNOWLEDGMENTS

The authors would like to thank NOVA LINCS Research Laboratory (Grant: FCT/MCTES PEst UID/ CEC/04516/2013) and DSML4MAS Project (Grant: FCT/MCTES TUBITAK/0008/2014).

REFERENCES

[1] John Brooke. 1996. SUS-A quick and dirty usability scale. In *Usability evaluation in industry*, Patrick W. Jordan, Bruce Thomas, Bernard A. Weerdmeester, and Ian L. McClelland (Eds.). Taylor & Francis, London, Chapter 21, 189–194.

[2] Patrice Caire, Nicolas Genon, Patrick Heymans, and Daniel L Moody. 2013. Visual notation design 2.0: Towards user comprehensible requirements engineering notations. In *21st IEEE InternationalRequirements Engineering Conference (RE 2013)*. IEEE, Rio de Janeiro, Brazil, 115–124. https://doi.org/10.1109/RE.2013.6636711

[3] Roger Dawson. 2012. *Secrets of Power Problem Solving*. Career Press, NJ, USA.

[4] Nicolas Genon, Daniel Amyot, and Patrick Heymans. 2010. Analysing the Cognitive Effectiveness of the UCM Visual Notation. In *International Workshop on System Analysis and Modeling (SAM 2010)*, Frank Alexander Kraemer and Peter Herrmann (Eds.). Springer, Berlin, Heidelberg, Oslo, Norway, 221–240. https://doi.org/10.1007/978-3-642-21652-7_14

[5] Nicolas Genon, Patrick Heymans, and Daniel Amyot. 2011. Analysing the cognitive effectiveness of the BPMN 2.0 visual notation. In *Software Language Engineering (SLE 2010)*, Brian Malloy, Steffen Staab, and Mark van den Brand (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 377–396. https://doi.org/10.1007/978-3-642-19440-5_25

[6] David Granada, Juan Manuel Vara, Marco Brambilla, Verónica Bollati, and Esperanza Marcos. 2017. Analysing the Cognitive Effectiveness of the webml Visual Notation. *Software & Systems Modeling* 16, 1 (2017), 195–227. https://doi.org/10.1007/s10270-014-0447-8

[7] Bruno Grácio. 2015. *Agregado: Compilar Sistemas NoSQL na Plataforma OutSystems*. Master's thesis. Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa.

[8] Sandra G Hart. 2006. NASA-task load index (NASA-TLX); 20 years later. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* 50, 9 (2006), 904–908. https://doi.org/10.1177/154193120605000909

[9] Sandra G Hart and Lowell E Staveland. 1988. Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research. *Advances in psychology* 52 (1988), 139–183. https://doi.org/10.1016/S0166-4115(08)62386-9

[10] Red Hat. 2016. JBoss Developer. (2016). http://www.jboss.org/

[11] Henrique Henriques, Hugo Lourenço, Vasco Amaral, and Miguel Goulão. 2018. Improving the Developer Experience with a Low-Code ProcessModelling Language: Companion site. (2018). https://doi.org/10.5281/zenodo.1318719

[12] William C Howell and Alfred H Fuchs. 1968. Population stereotypy in code design. *Organizational Behavior and Human Performance* 3, 3 (1968), 310–339.

[13] ISO. 2014. *Graphical symbols – Test methods – Part 1: Method for testing comprehensibility*. Standard. International Organization for Standardization, ISO/TC 145 Graphical symbols, Geneva, CH.

[14] Sheila Jones. 1983. Stereotypy in pictograms of abstract concepts. *Ergonomics* 26, 6 (1983), 605–611.

[15] Barbara Kitchenham, Lech Madeyski, Pearl Brereton, Stuart Charters, Shirley Gibbs, and Amnart Pohthong. 2016. Robust Statistical Methods for Empirical Software Engineering. *Empirical Software Engineering* 22, 2 (2016), 579–630. https://doi.org/10.1007/s10664-016-9437-5

[16] A. Lima. 2013. *OutSystems Platform - Architecture and Infrastructure Overview*. Technical Report. OutSystems. https://www.outsystems.com/home/document-download/178/8/0/0

[17] Raimundas Matulevičius and Patrick Heymans. 2007. Comparing goal modelling languages: An experiment. In *International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2007)*. Springer, Berlin, Heidelberg, Trondheim, Norway, 18–32. https://doi.org/10.1007/978-3-540-73031-6_2

[18] Microsoft. 2016. ASP.NET. (2016). http://www.asp.net/

[19] Microsoft. 2016. ISS. (2016). https://www.iis.net/

[20] Microsoft. 2016. SQL Server. (2016). http://www.microsoft.com/en-us/server-cloud/products/sql-server-2016/

[21] Microsoft. 2016. Windows Server 2016. (2016). http://www.microsoft.com/en-us/server-cloud/products/windows-server-2016/

[22] Tomás Miranda, Moharram Challenger, Baris Tekin Tezel, Ömer Faruk Alaca, Vasco Amaral, Miguel Goulão, and Geylani Kardas. 2018. Improving the Usability of a MAS DSML. In *6th International Workshop on Engineering Multi-Agent Systems (EMAS 2018)*. Springer, Stockholm, Sweden, 16 pp.

[23] Daniel L Moody. 2009. The "physics" of notations: toward a scientific basis for constructing visual notations in software engineering. *Software Engineering, IEEE Transactions on* 35, 6 (2009), 756–779. https://doi.org/10.1109/TSE.2009.67

[24] Daniel L Moody, Patrick Heymans, and Raimundas Matulevičius. 2010. Visual syntax does matter: improving the cognitive effectiveness of the i* visual notation. *Requirements Engineering* 15, 2 (2010), 141–175.

[25] Joan C Nordbotten and Martha E Crosby. 1999. The effect of graphic style on data model interpretation. *Information Systems Journal* 9, 2 (1999), 139–155.

[26] Oracle. 2015. Oracle WebLogic Server. (2015). http://www.oracle.com/technetwork/middleware/weblogic/overview/index-085209.html

[27] Oracle. 2016. MySQL. (2016). https://www.mysql.com/

[28] Oracle. 2016. Oracle. (2016). http://www.oracle.com/index.html

[29] OutSystems. 2013. *OutByNumbers - Benchmark Overview Report*. Technical Report. OutSystems. http://www.outsystems.com/res/OutbyNumbers-DataSheet

[30] Hasso Plattner, Christoph Meinel, and Larry Leifer. 2011. *Design thinking: Understand–Improve–Apply*. Springer-Verlag Berlin Heidelberg, Germany. https://doi.org/10.1007/978-3-642-13757-0

[31] Faisal Saleh and Mohamed El-Attar. 2015. A scientific evaluation of the misuse case diagrams visual syntax. *Information and Software Technology* 66 (2015), 73–96.

[32] Mafalda Santos, Catarina Gralha, Miguel Goulão, and João Araujo. 2018. Increasing the Semantic Transparency of the KAOS Goal Model Concrete Syntax. In *37th International Conference on Conceptual Modeling (ER 2018)*. Springer, Xi'an, China, 14 pp.

[33] Mafalda Santos, Catarina Gralha, Miguel Goulão, João Araujo, and Ana Moreira. 2018. On the Impact of Semantic Transparency on Understanding and Reviewing Social Goal Models. In *26th IEEE International Conference on Requirements Engineering (RE 2018)*. IEEE, Banff, Canada, 12 pp.

[34] UsabilityHub. 2018. UsabilityHub web site. https://usabilityhub.com/. (2018). Accessed: 2018-04-27.

[35] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in software engineering*. Springer-Verlag Berlin Heidelberg, Germany. https://doi.org/10.1007/978-3-642-29044-2

[36] HJ Zwaga and T Boersema. 1983. Evaluation of a set of graphic symbols. *Applied Ergonomics* 14, 1 (1983), 43–54.