Henrique Gabriel Henriques

Bachelor in Computer Science

Domain Specific Language Evaluation: OutSystems' Business Process Technology

Dissertation submitted in partial fulfillment
of the requirements for the degree of

Master of Science in
Computer Science and Engineering

Adviser:	Vasco Miguel Moreira do Amaral, Assistant Professor, Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa
Co-advisers:	Miguel Carlos Pacheco Afonso Goulão, Assistant Professor, Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa
	Hugo Miguel Ramos Lourenço, Software Engineer, OutSystems

Examination Committee

Chairperson:	Prof. Pedro Medeiros
Raporteurs:	Prof. Fernando Brito e Abreu
Member:	Prof. Vasco Amaral

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

November, 2016

Domain Specific Language Evaluation:
OutSystems' Business Process Technology

This document was created unsing the (pdf)LaTeX processor, based in the "unlthesis" template[1], developed at the Dep. Informática of FCT-NOVA [2]. [1] https://github.com/joaomlourenco/unlthesis [2] http://www.di.fct.unl.pt

Acknowledgements

"Programming today is a race between software engineers striving to build bigger and better idiot-proof programs, and the universe trying to produce bigger and better idiots. So far, the universe is winning." - Rick Cook

I would like to thank the Faculty of Sciences and Technology from the NOVA University of Lisbon for providing amazing courses that allowed me to grow as a professional. I also thank the Informatics Department and all its staff for caring about the students and for providing a solid education. Thank you also to *OutSystems* for funding a scholarship for this dissertation.

Thank you to my advisers: Vasco Amaral, Miguel Goulão and Hugo Lourenço. They were available whenever I needed them, they steered me in the right direction and their suggestions were invaluable to the quality of this document. A special thank you to Hugo Lourenço for closely following my process, helping with whatever issue I had and for being an amazing initial bridge between myself and the rest of *OutSystems*.

I also thank everyone at *OutSystems' R&D* for making me feel welcome and for always being available to help. A special thank you goes to António Pereira for always being able to find more people for testing. I want to also thank all my colleagues in FCT for making these last five years fun and interesting. A special thank you goes to Luis Afonso Carvalho and Pedro Maroco, we worked on a lot of projects together and your outstanding skills at *git*, good management of time and artistic pyrotechnics ensured everything went smoothly.

Finally, I want to thank my parents: José Gabriel Henriques and Janet Henriques. Without them I could not be where I am now. I also want to thank my old friend José Fábio Jesus for our gaming sessions and for being an expert in lore. Last but not least a very special thank you goes to Ana Rita Dias. For her daily dosage of love and support, for the constructive criticism about my writing and for cooking amazing meals, ensuring that I remained well nourished during the development of this dissertation.

Abstract

Domain Specific Languages (DSL) are created with the intent of making problem solving easier via abstractions and notations that are closer to the domain users' way of thinking. However, in order to fulfil this intent the language must be considered usable by its target audience, which requires using a principled approach, contrary to an ad-hoc philosophy. Unfortunately, language engineers are not always concerned by usability. Usability techniques developed by Human Computer Interaction (HCI) experts are generally focused solely on application interfaces that were not designed to be directly applied to languages. However, previous studies have shown that languages are in essence interfaces between users and the computational system, so applying HCI techniques is not incorrect. However, to the best of our knowledge, there is no simple evaluation process that allows developers to validate their languages, both syntactically and semantically.

The *OutSystems Platform* is a development environment composed of several domain specific languages. It is used to quickly build and validate web and mobile applications. The languages allow users to build interfaces and data models, define custom business logic and construct process models. Howbeit, the DSL for process modelling (Business Process Technology (BPT)), does not have the desired adoption rate and is often used for solving problems out of the intended domain it was originally designed for. This is problematic, given that the language has an associated maintenance cost.

The purpose of this dissertation is twofold: create a succinct and easy process for evaluating visual programming languages, and apply the proposed process to BPT in order to identify any usability issues that may be present in the BPT language. The process we developed includes adapted HCI evaluation techniques (such as the Systems Usability Score, Task Load Index and others). After identifying the main problems with the BPT language, the language was updated with a new syntax. We performed a comparison analyses between the original and new syntaxes, which showed that the new version is more transparent and has a significantly higher usability rating.

Keywords: Domain specific language, language evaluation, language usability, process modelling, OutSystems Platform

Resumo

Domain Specific Languages (DSL) são criadas com a intenção de tornar mais fácil a resolução de problemas através de abstrações e notações que são mais próximas da maneira de pensar dos utilizadores do domínio. Isso, no entanto, só é verdade se a linguagem for considerada utilizável pelo seu público-alvo, que requer o uso de uma abordagem baseada em princípios, ao contrário de uma filosofia ad-hoc. Infelizmente, a usabilidade não é um dos aspetos técnicos com que os engenheiros de linguagens mais se preocupam. Técnicas de usabilidade desenvolvidas por especialistas em *Human Computer Interaction* (HCI) são geralmente focadas exclusivamente em interfaces de aplicações e não foram desenvolvidas para serem diretamente aplicadas a linguagens de programação. No entanto, estudos prévios mostraram que as linguagens de programação são, na sua essência, interfaces entre os utilizadores e o sistema computacional, assim sendo a aplicação de técnicas *HCI* não é considerada incorreta. No entanto, no decorrer da nossa pesquisa, não foi encontrado nenhum processo de avaliação simples que permita aos engenheiros validar as suas linguagens tanto sintatica como semanticamente.

A *OutSystems Platform* é um ambiente de desenvolvimento composto por várias *DSL*. Este é usado para construir rapidamente aplicações web e móveis. As linguagens permitem aos utilizadores construir interfaces e modelos de dados, definir lógica de negócios personalizada, e construir modelos de processos. Todavia, a *DSL* para modelar processos (*Business Process Technology (BPT)*), não tem a taxa de adoção desejada e é utilizada em problemas fora do domínio pretendido. Isto é devido ao custo de manutenção associado.

Esta tese tem dois objetivos: criar um processo sucinto e fácil para avaliar linguagens de programação, e usar esse mesmo processo para identificar quaisquer problemas que possam estar presentes na linguagem *BPT*. Para conseguir isto, foi desenvolvido um processo simples de acompanhar que inclui técnicas *HCI*. Uma vez identificados os principais problemas com o *BPT*, foram feitas alterações à linguagem. Finalmente, foi realizada uma comparação entre a sintaxe original e a nova. Estas mostraram que a nova versão é mais transparente e tem uma classificação de usabilidade significativamente maior.

Palavras-chave: Linguagens de domínio específicos, avaliação de linguagens, usabilidade de linguagens, modelação de processos, OutSystems Platform

Contents

List of Figures

List of Tables

Acronyms

BPEL Business Process Execution Language.

BPMI Business Process Management Initiative.

BPML Business Process Modelling Language.

BPMN Business Process Model and Notation.

BPT Business Process Technology.

DCR Dynamic Condition Response.

DSL Domain Specific Languages.

HCI Human-Computer Interaction.

IDE Integrated Development Environment.

SUS System Usability Scale.

TLX Task Load Index.

UML Unified Modeling Language.

XML eXtensible Markup Language.

Chapter 1

Introduction

Domain Specific Languages (DSL) are developed to answer the needs of a specific domain with the goal of reducing development time, improving product quality and bridging the gap between domain users and software developers. However, to achieve said goals the language has to satisfy a set of requirements in order to be sufficiently usable. Otherwise it can have negative effects on development and eventually be abandoned. Unfortunately, usability is many times ignored (or undervalued) by Software Engineers and Notation Designers [44]. And when usability is a concern, there is very little to no documentation explaining the thought process behind the chosen syntax notation [29].

Programming languages in general exist to enable the efficient development of software but that efficiency is dependant on the language's usability. Yet, even so, one of the most neglected areas in programming languages research is the bridge between programming languages and Human-Computer Interaction (HCI) [47]. This dissertation includes several HCI techniques that can be used in evaluating the usability of languages. These techniques are then used to evaluate a commercial language called Business Process Technology.

1.1 Context and Description

Usability engineering is a field that focuses on human-computer interaction and is most commonly used to achieve elegant and efficient user interface design [48]. A DSL's goal is to bridge the gap between a domain expert and computational systems and can be seen as a user interface [4]. As such, it is only natural to apply usability techniques when evaluating languages, even more so taking into account that said techniques are well documented and studied.

One of the purposes of this dissertation is to evaluate a DSL called BPT developed by the software company OutSystems. The BPT language allows developers to specify and implement business processes in the context of an OutSystems application. All this is done using the Integrated Development Environment (IDE) created by OutSystems called Service Studio (more on OutSystems and Service Studio in section 2.1).

1.2 Motivation

One of the key characteristics of the industrial era was the introduction of processes, the notion that production output would increase if labour is divided, people have different roles and specific tasks within a overall process. One of the first process descriptions was Adam Smith's pin factory in 1776 [58]. He described the process of creating a pin as follows:

> *"One man draws out the wire, another straights it, a third cuts it, a fourth points it, a fifth grinds it at the top for receiving the head: to make the head requires two or three distinct operations: to put it on is a particular business, to whiten the pins is another ... "*

From the excerpt it is easy to get confused about who does what and when, and so visual notations were created in order to make process definitions easier to understand. However, studies indicate that usability still is not a high concern when it comes to process modelling languages [16] and this trend is true with other DSLs in general [47].

For a DSL to be successful it needs to be usable by the developers using the language, but when it comes to Business Processes, there is the added complexity of ensuring the readability by a business manager. The BPT language by OutSystems is used by developers with programming and process modelling knowledge, but it is also used as a communication medium for business managers.

Through interviews with members of OutSystems and data collected from recent projects developed using Service Studio, we identified that the BPT language was not having the expected adoption rate or was being used for purposes other than process modelling. Since maintaining the language has an associated cost, it is important to identify possible flaws in the language and make any necessary changes.

1.3 Objectives

This dissertation has two main objectives: creating a systematic process for evaluating visual programming languages and then applying that process to the BPT language from OutSystems.

The evaluation process is abstract enough that it can be applied to any visual programming language. It includes usability tests for the language's concrete syntax [44] and several techniques from the HCI area. The evaluation process is then applied to BPT, not only in order to improve the language, but also as a way to demonstrate the evaluation process's viability. Based on the results from the evaluation, this dissertation also proposes (and implements) changes to the language in order to improve its usability and increase its commercial value.

1.4 Key Contributions

This dissertation's main contributions are the definition of a systematic evaluation process for visual programming languages, a companion application to help manage the evaluation data, a detailed usability report (using the aforementioned process) on OutSystems' BPT and necessary changes to make the language more usable.

1.5 Structure

This document is organised, excluding the current chapter, in the following way:

- Chapter 2 - Background: includes an overview of the OutSystems Platform and its architecture. Also, offers a brief description of process and event languages, and describes different established techniques for usability evaluation;
- Chapter 3 - Related Work: a brief overview of other language evaluations;
- Chapter 4 - Evaluation Process: a proposed process for evaluating languages and an overview of the created companion application;
- Chapter 5 - Analysing BPT: an analysis of OutSystems' BPT following the proposed process;
- Chapter 6 - New version of BPT: construction of BPT's new syntax following methods in the proposed process and semantic transparency analysis;
- Chapter 7 - Results: an analysis of the results obtained through usability experiments of the original and new notation and possible threats to their validity;
- Chapter 8 - Conclusions and future work: an overview of what the dissertation achieved and suggestions for future work.

Chapter 2

Background

2.1 OutSystems

OutSystems is a software company which developed the *OutSystems Platform*. Said platform is used to create web and mobile applications resorting to a set of integrated DSLs. The OutSystems' visual language allows users to develop at a higher abstraction level, thus not having to worry about low level details related to creating and publishing applications. This results in significantly faster development times and a higher quality result when compared to general purpose languages [52].

The *OutSystems Platform* provides a visual development environment called *Service Studio* which allows applications to be developed (and then reused by other applications) in modules called *eSpaces*. These modules that contain process definitions, user interfaces, business logic, and the data model for applications.

2.1.1 OutSystems Platform

The OutSystems Platform architecture [36], which is represented in Figure 2.1, is divided into three main components: *Service Studio*, *Platform Studio* and *Application Server*.

2.1.1.1 Service Studio

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 model and sends it to the *Platform Server*.

The IDE is divided into four main views, one for process modelling, one for interface flows, one where you can define custom logic and access APIs and another for database modelling.

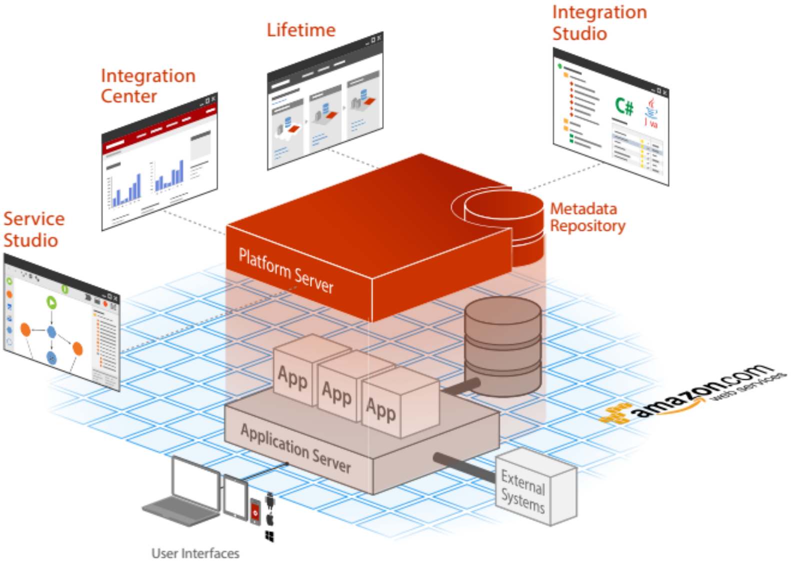


Figure 2.1: *OutSystems Platform* architecture [24].

2.1.1.2 Platform Server

The *Platform Server* uses the model to generate code that depends on the particular stack being used. E.g., for a Windows Server [42] using SQL Server [41] this will be ASP.Net [39] and SQL code. Once this process is completed the compiled application is then deployed to the *Application Server*.

The *Platform Server* also includes the *Scheduler Service*. This service manages the execution of steps within process models developed using BPT and also of scheduled jobs resulting from *Timers*.

2.1.1.3 Application Server

The *Application Server* runs on top of *Oracle WebLogic* [49], *JBOSS* [28] or *IIS* [40]. The server then stores and runs the developed application which is connected to a relational database management system, which can be *SQL Server*, *Oracle* [51] or *MySQL* [50]. Note that the SQL code generated by the *Platform Server* is specific to the selected database management system.

2.2 Process languages

Presented here are several visual languages that are used for modeling processes. Each of the presented languages includes an example process which all have the same semantic behaviour (with the exception of Petri-nets, as it would result in an unnecessary complex example). This allows for an easier interpretation. The process manages expenses: when an expense is submitted, it must be approved by a Manager and the Finance department; if either do not approve, then the expense submitter must update the expense so it can be evaluated once again; if it is approved, then payment is made; and once the payment has been processed, an email is sent to the submitter.

2.2.1 Business Process Technology

The *OutSystems Platform* includes a DSL for modelling processes called BPT. This DSL enables users to design, execute and manage processes which are fully integrated with the applications built with the *OutSystems Platform*. An eSpace can contain several different processes and a process can invoke another process. This allows for a modular development methodology which results in cleaner diagrams and enables reuse.

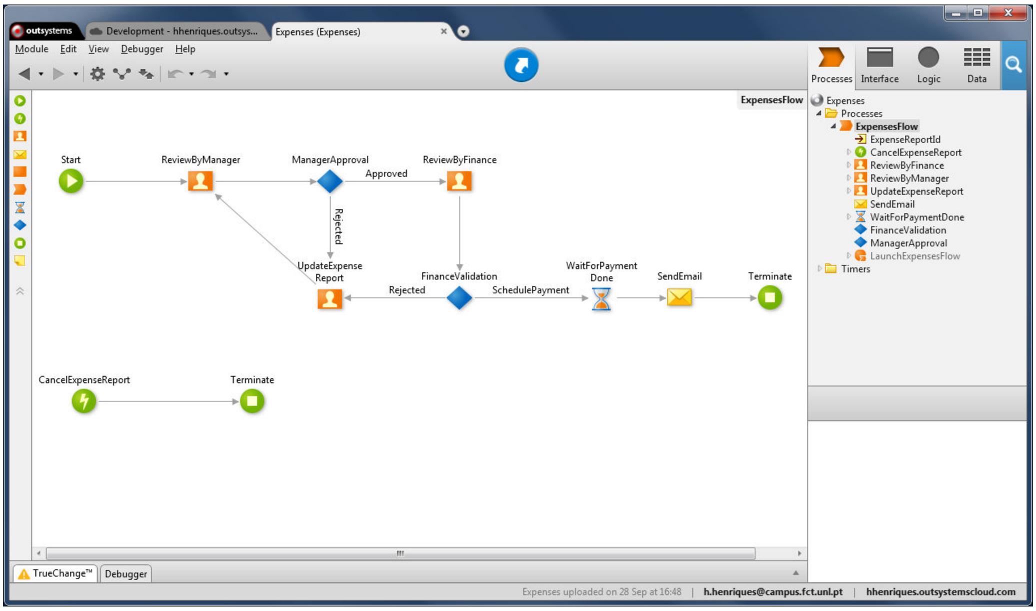


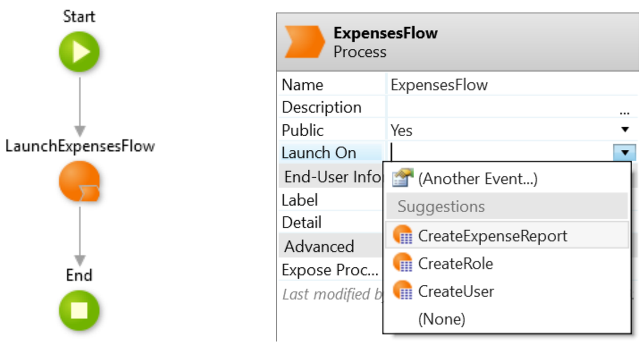
Figure 2.2: BPT development environment.

Figure 2.2 contains an example of the expenses process build with BPT. The figure also contains BPT's development environment within ServiceStudio.

2.2.1.1 Launching a Process

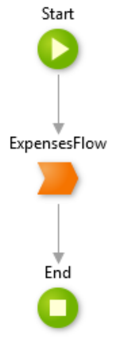
There are three ways to launch a process in BPT: it may be explicitly launched in an action flow 2.3(a), automatically launched when an entity is created 2.3(b), or launched within another process 2.3(c).

Note that even though the explicit and nested alternatives look similar (and even have the same syntax for Start and End), these occur in very different contexts. The explicit call is done in a user-defined action which is where all the application logic is implemented, such as data-base manipulation, API calls and others. A nested call always occurs in the context of a process flow.



ExpensesFlow
Process

Name	ExpensesFlow
Description	...
Public	Yes
Launch On	
End-User Info	
Label	
Detail	
Advanced	
Expose Proc...	
Last modified b	



(a) Explicit (b) Automatic (c) Nested

Figure 2.3: Launching a process.

2.2.1.2 Concrete Language Syntax

The OutSystems BPT is a visual language. As such, its syntax contains a set of symbols. Below is a short description of each element of the syntax.

- **Start**. The Start icon start the process flow, each process has to have a Start and there can only one.
- **Conditional Start**. The Conditional Start 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, said condition can be a data-base event or a API call.
- **End**. The End icon has two uses depending on a flag *Terminate* defined in the attributes. If the flag is set to *No* then it terminates the particular flow it is connected to, otherwise it terminates the whole process.
- **Process**. Calls another process.
- **Human Activity**. Human Activity is linked to a pre-developed *Web Screen*[1] and pauses the flow waiting for the user to trigger an action on said screen.
- **Automatic Activity**. Contains a action flow which is defined in a separate window. The action flow can include custom logic, event broadcasts via the database or API calls.
- **Wait**. Wait pauses the process flow. The flow can then be resumed by a specific API call, a database event or an associated timeout.
- **Send Email**. 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 emails addresses entered in the node's attribute.
- **Decision**. The Decision node has n outgoing flows and the chosen flow is decided based on custom logic defined in a separate window.

[1] Interface designed using Service Studio

- **Comment**. The Comment allows users to write text in a small frame, this frame is ignored by the compiler.

2.2.2 Business Process Model and Notation

Development of BPMN started in 2001. At the time Business Process Management Initiative (BPMI) were working on Business Process Modelling Language (BPML)L, a eXtensible Markup Language (XML) process execution language, but they quickly realised they needed a visual representation [62]. And so, BPMN was created in 2004 to answer the need of a standard notation for business processes [18].

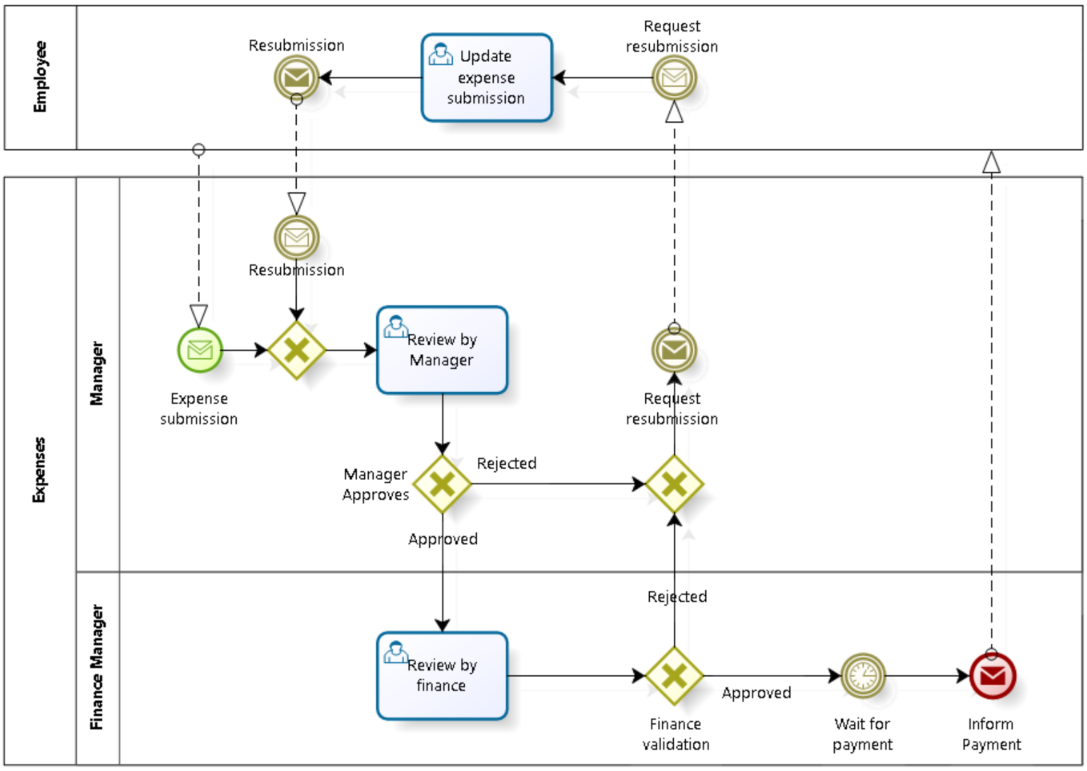


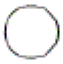
Figure 2.4: Example of a BPMN process.

BPMN has two main objectives: Standardise business process notations (this allows for consistent training since end-users would only need to know a single, agreed upon notation), and provide mechanisms to generate executable processes [62]. The process execution was originally done by BPML, being later replaced by Business Process Execution Language (BPEL) [1].

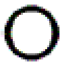
Figure 2.4 contains the expenses process example build with BPMN.

2.2.2.1 BPMN Syntax

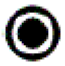
BPMN's syntax evolved and grew throughout a series of iterations. Below is a sample of the syntax which contains all the elements supported by OutSystems' BPT:



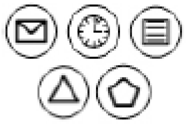
Start. Initiates the process;



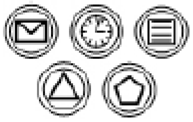
End. Finishes the process flow. The process may continue in other flows;



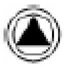
Terminate. Ends all the process flows;



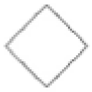
Start with condition. Initiates the process based on a specific event;



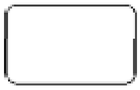
Intermediate event with catching trigger. Pauses the flow and waits for a trigger.



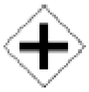
Intermediate event with a throwing trigger. Used to broadcast an event.



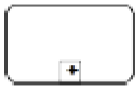
Decision. The flow continues based on logic.



Task. Can be several things, such as: sending requests and emails, and waiting for a manual action or service;



Fork & Join. Used to split and merge the process flow;



Subprocess. Direct the flow to a reusable process. The parent process resumes when the subprocess finishes;

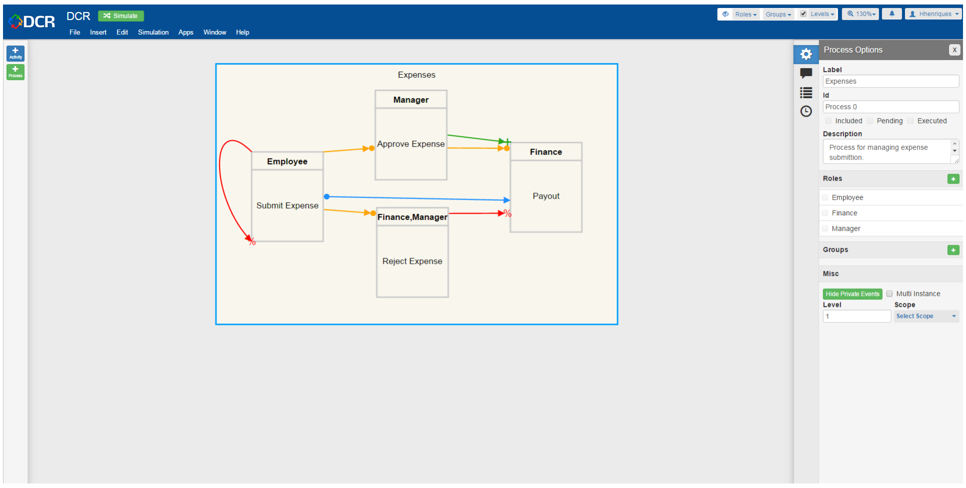
Sequence Flow. Defines the order of tasks and events

2.2.3 Dynamic Condition Response

Dynamic Condition Response (DCR) was developed in a collaboration between the IT University of Denmark and Exformatics A/S to answer the needs of the Danish mortgage credit institutes. These institutes were using a small subset of BPMN but were unhappy with it [13], and so, DCR was created which is described as a formal Adaptive Case Management Workflow notation [61].

DCR has a web based IDE (Figure 2.5). The language is composed by Processes which contain Activities, each Activity has a set of assigned Roles and are interconnected by Connectors.

Figure 2.5 contains the expenses example build with DCR.



Label: Expenses

Id: Process 0

□ Included □ Pending □ Executed

Description: Process for managing expense submittion.

Roles
- □ Employee
- □ Finance
- □ Manager

Groups

Misc

Hide Private Events □ Multi Instance

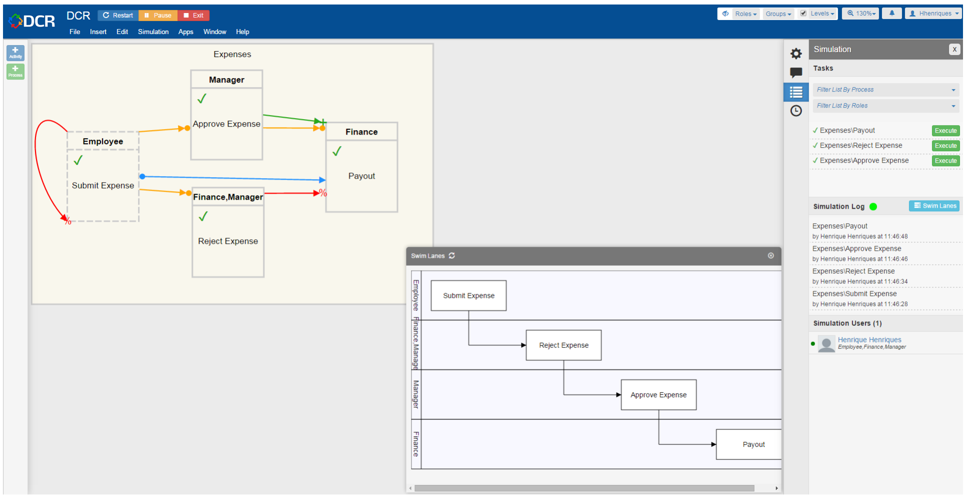
Level: 1

Scope: Select Scope

Figure 2.5: DCR development environment.

2.2.3.1 Simulation

One of the DCR's major features is its ability to simulate a process, assigning one or more roles to a user. When running a simulation, the interface of the IDE presents the simulation users with information about what activities were executed by whom and swim lanes demonstrating the process flow.



✓ Expenses\Payout Execute

✓ Expenses\Reject Expense Execute

✓ Expenses\Approve Expense Execute

Expenses\Payout
by Henrique Henriques at 11:46:48

Expenses\Approve Expense
by Henrique Henriques at 11:46:46

Expenses\Reject Expense
by Henrique Henriques at 11:46:34

Expenses\Submit Expense
by Henrique Henriques at 11:46:28

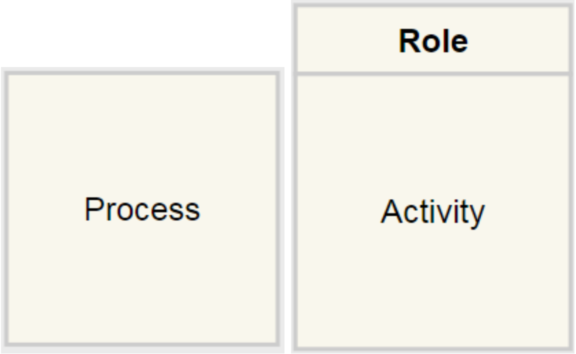
Henrique Henriques
Employee,Finance,Manager

Figure 2.6: DCR simulation.

Figure 2.6 demonstrates DCR's process simulation feature.

2.2.3.2 Concrete Language Syntax

DCR's syntax can be divided into two groups: Canvas elements (Figure 2.7) and Connectors. Canvas elements include Activities and Processes, where Activities are nested in Processes and each Activity can have one or more Roles associated to it.



(a) Process (b) Activity

Figure 2.7: DCR Canvas Elements

The Connectors are used to connect two Activities and these control the flow of the process based on the type of connector used. Figure 2.7 demonstrates DCR's canvas elements and below is a description of the available connector types and respective visual syntax.

→• **Condition**. Creates a relation between an activity A and B such that B can only occur if A has occurred first.

•→ **Response**. Creates a relation between an activity A and B such that if A occurs then, at some point, B has to occur.

→+ **Include**. Creates a relation between an activity A and B such that the occurrence of A makes the occurrence of B possible.

→% **Exclude**. Creates a relation between an activity A and B such that B cannot occur if A has occurred.

→◇ **Milestone**. Creates a relation between an activity A and B such that B can occur initially, but if A's status becomes pending while waiting for a response connection by an activity C, then B cannot occur until A has finished.

→* **Spawn**. Creates a relation between an activity A and a sub-activity B, when A occurs a new instance of B is created.

2.2.4 Petri Nets

The execution semantics of many process modelling languages are defined by enabling and firing elements based on a token-game [25]. Petri Nets have that same behaviour. As such, even if Petri Nets are not considered a Process Language, understanding Petri Nets gives a general idea of the process modelling languages' semantics.

A Petri Net is a mathematical modelling language used to aid in designing and analysing concurrent systems. It has application in several different areas in Computer Science. These include Software Engineering, communication protocols, socio-technical systems and others [7]. Petri Nets have several characteristics that sets them apart from other languages. One such characteristic is its the ability to represent systems at different levels of abstraction without having to change the description language, which has a high usability impact [55].

2.2.4.1 Language Description

Petri Nets can be seen as a particular kind of directed graphs and consist of two types of nodes: places and transitions [46]. The net has an initial marking (M_0) which consists of having k tokens in p places, this is represented by the pth component of M. Places and Transitions are connected by arcs, these arcs are labelled with their weights (W). The places from which an arc runs to a transition are called input arcs and similarly places from which an arc runs from a transition are that transitions output places. A transition may fire if there are sufficient tokens in all of that transitions input places. When it fires W tokens are consumed from the input places and the same W tokens are created in the output places. Figure 2.8 contains a small example of a Petri-net where two tokens are consumed from P0, one is consumed from P1 and two are created in P2.

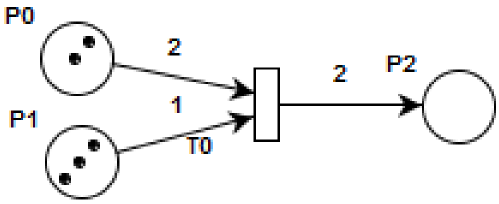


Figure 2.8: Petri Net example

2.2.4.2 Abstract Syntax

Petri nets are a 5-tuple, $PN = (P, T, F, W, M_0)$ [46] where:

$P = \{p_1, p_2, \cdots, p_m\}$ is a finite set of places,
$T = \{t_1, t_2, \cdots, t_n\}$ is a finite set of transitions,
$F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs,
$W : F \longrightarrow \{1, 2, 3, \cdots\}$ is weight function,
$M_0 : P \longrightarrow \{0, 1, 2, 3, \cdots\}$ is the initial marking,
$P \cap T = \emptyset$ and $P \cup T \neq \emptyset$.

There are other definitions of the language that are simplifications of the above syntax, for example a Petri Net without a specific initial marking is denoted by N, where $N = (P, T, F, W)$.

2.2.4.3 Concrete Syntax

A Petri Net can be used mathematically but is most commonly used with a graphical representation. These are the elements of its syntax:

○ **Place**. Contains Tokens and has Arcs outgoing to Transitions and incoming from Transitions. Two places cannot be directly connected with an Arc.

● **Token**. These are contained in Places. Tokens are consumed and created based on the weight on the arc.

→ **Arc**. Connect Places to Transitions. Arcs are labelled with weights.

▯ **Transition**. Connected with Arcs from and to Places. When a Transition is fired the tokens in the incoming places are consumed and then created in the outgoing Place, based on the Arc's weight.

2.2.5 UML - Activity Diagram

Activity Diagram is similar to a flowchart that represent the flow from a system operation to another. These operations are called activities. However, unlike flowcharts, Activity Diagrams can be used to show parallel, branched and concurrent flows [2].

Activity Diagrams have several features that make them stand out when compared to other work flow modelling languages. They support signal sending and receiving at a conceptual level and support waiting and processing states [14].

Figure 2.9 contains the expenses process example build with Activity Diagrams.

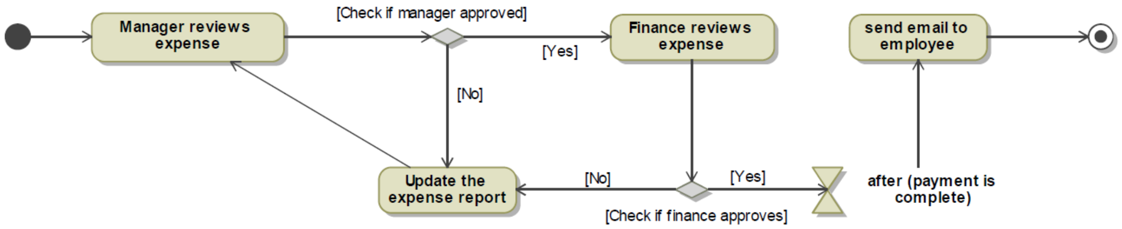

Manager reviews expense
[Check if manager approved]
[Yes]
Finance reviews expense
send email to employee
[No]
Update the expense report
[No]
[Yes]
[Check if finance approves]
after (payment is complete)

Figure 2.9: Activity Diagram example.

2.2.5.1 Concrete Syntax

Activity Diagram's syntax is a visual notation, as expected considering it is used for flow management. An Activity is a network of nodes connected by edges [2]:

● **Initial node**. Indicates the start of the activities' flow;

◉ **Final node**. Terminates an activity;

▭ **Action node**. Represents a single atomic step with the activity. The name of the action is a verb or noun with some explanation. As an example: "Review expense";

◇ **Decision**. The chosen outgoing edge is chosen based on a guard condition;

▬ **Fork & Join**. Splits and synchronises multiple concurrent flows;

⧖ **Wait**. Interrupts the activity and waits for a certain amount of time. The time it waits is based on a label;

➡ **Edge**. Used to control the flow of the activity.

The above notation is a sample of the Activity Diagram syntax and using it allows the creating of low complexity diagrams.

2.3 Human-computer interaction techniques

This section goes over several HCI techniques. Even though these techniques were not originally meant for language evaluation they can be adapted to do so. Note that this section describes the original techniques without any alterations.

2.3.1 System Usability Scale

System Usability Scale (SUS) was developed in 1986 by John Brooke. It was created as a "quick and dirty" scale for measuring usability and consists of ten questions [8].

The questions are given to the respondent (*i.e.* user) after the he/she completes a test but before any debriefing or discussion. It is also best if the respondent gives an immediate response to each item, rather than thinking about it for too long.

If the respondent can not answer to one of the questions then the centre point on the scale should be marked, as this will result in a neutral value when calculating the final score.

2.3.1.1 The questionnaire

SUS questionnaire contains ten items:

1. I think that I would like to use this system frequently.

2. I found the system unnecessarily complex.

3. I thought the system was easy to use.

4. I think that I would need the support of a technical person to be able to use this system.

5. I found the various functions in this system were well integrated.
6. I thought there was too much inconsistency in this system.
7. I would imagine that most people would learn to use this system very quickly.
8. I found the system very cumbersome to use.
9. I felt very confident using the system.
10. I needed to learn a lot of things before I could get going with this system.

The item order is important since it alternates between positive and negative. This forces the respondent to pay attention to each statement and prevents a response bias [8]. The answer box is shown in Figure 2.10.

The original questions were meant for systems but these can be easily adapted for other studies.

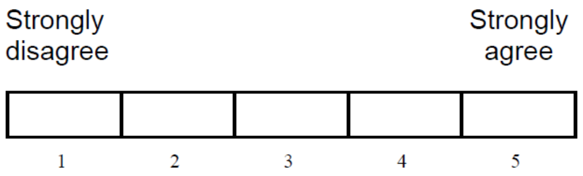


Figure 2.10: SUS answer format

2.3.1.2 Calculating the SUS score

The result of the SUS is a single number which represents a measure of the system's usability.

To calculate the final score first look at the score of the uneven items and subtract 1 from it and then look at the even ones and subtract the score from 5. Following that, sum the result of the previous step and multiply that by 2,5. This grants the final score which will always range between 0 and 100.

2.3.1.3 Interpreting the SUS score

Even though the SUS score is a value between 0 and 100, it should not be interpreted as a percentage. As an example, assume a system gets a SUS score of 60, it would be correct to say that it represents 60% of the maximum possible score. This suggests that the score is at the 60th percentile, which can be interpreted as above average. However, in 2,324 surveys the mean SUS score was of 70,14 [3], meaning the example system is actually below average.

In order to avoid miss-interpretations it is best to normalise the score to produce a percentile ranking. So taking into consideration the 70,14 average score, normalising a systems with a score of 60 means its usability percentage is 42,8%. Meaning it is a below average system.

2.3.2 Cognitive Dimensions

Cognitive Dimensions is a framework developed to help non-HCI specialists. It is mostly used to evaluate programming language usability (even though it can be applied to a wide range of interactive systems) [21]. Considering it is aimed at non-specialists, the framework does not require a lengthy detailed analysis, but rather, uses a checklist approach that ensures serious problems are not overlooked [22].

2.3.2.1 The dimensions

Most HCI techniques are designed to concentrate on the physical, low-level details of interaction between a user and a device. Cognitive Dimensions does not provide a final usability score but rather its fourteen "dimensions" (questions) aim to spark discussions that will help identify usability issues [20]. The dimensions are:

Abstraction gradient. What are the minimum and maximum levels of abstraction exposed by the notation? Can details be encapsulated?

Closeness of mapping. How closely does the notation correspond to the problem world?

Consistency. After part of the notation has been learnt, how much of the rest can be successfully guessed?

Diffuseness / terseness. How many symbols or how much space does the notation require to produce a certain result or express a meaning?

Error-proneness. To what extent does the notation influence the likelihood of the user making a mistake?

Hard mental operations. How much hard mental processing lies at the notational level, rather than at the semantic level? Are there places where the user needs to resort to fingers or pencilled annotation to keep track of what's happening?

Hidden dependencies. Are dependencies between entities in the notation visible or hidden? Is every dependency indicated in both directions? Does a change in one area of the notation lead to unexpected consequences?

Juxtaposability. Can different parts of the notation be compared side-by-side at the same time?

Premature commitment. Are there strong constraints on the order with which tasks must be accomplished? Are there decisions that must be made before all the necessary information is available? Can those decisions be reversed or corrected later?

Progressive evaluation. How easy is it to evaluate and obtain feedback on an incomplete solution?

Role-expressiveness. How obvious is the role of each component of the notation in the solution as a whole?

Secondary notation and escape from formalism. Can the notation carry extra information by means not related to syntax, such as layout, colour, or other cues?

Viscosity. Are there in the notation any inherent barriers to change? How much effort is required to make a change to a program expressed in the notation? This dimension can be further classified into the following types:

- **Knock-on viscosity**: a change in the code violates internal constraints in the program, whose resolution may violate further internal constraints;
- **Repetition viscosity**: a single action within the user's conceptual model requires many, repetitive device actions;
- **Scope viscosity**: a change in the size of the input data set requires changes to the program structure itself.

Visibility. How readily can required parts of the notation be identified, accessed and made visible?

2.3.2.2 Expanded dimensions

Even though the framework contains quite a few dimensions, its creators still consider it incomplete [20]. As such, there are numerous dimensions that have been created by the community [6]. Of note, given the context of this dissertation:

Detail in context. Is it possible to see how elements relate to others within the same notational layer?

2.3.2.3 Issues with Cognitive Dimensions

Even though Cognitive Dimensions is widely used in Visual Programming Language usability, there is evidence showing that it is not the best technique to use when it comes to specifying visual notations [43].

Some of theoretical and practical limitations of Cognitive Dimensions include: its exclusion of issues related to visual representations (since it is based solely on structural properties), the dimensions are not design guidelines and it lacks evaluation procedures or metrics (making it very subjective) [44]. The aforementioned limitations mean it does not provide a scientific basis for evaluating and designing visual notations.

2.3.3 The Physics of Notations

The Physics of Notations provides a framework for evaluating visual notations and does not have the limitations present in Cognitive Dimensions. The framework is a set of nine principles (called Prescriptive Theory). These were synthesised from theory and empirical evidence about cognitive effectiveness of visual representations [44].

2.3.3.1 Visual Variables

In order to analyse visual notations, the prescriptive theory breaks down the symbols to their atomic characteristics. Bertin [5] divided the symbols into eight characteristics (Figure 2.11): horizontal position, vertical position, shape, brightness, size, orientation, colour and texture.

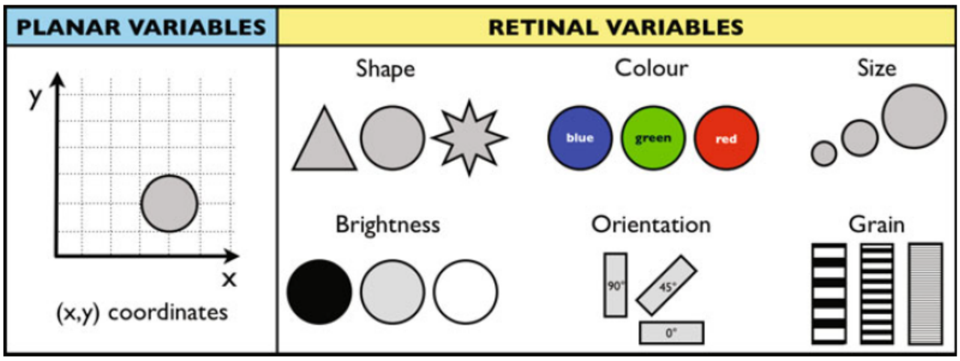


Figure 2.11: Visual Variables [18]

2.3.3.2 Prescriptive Theory

A good visual notation should follow these nine principles:

Semiotic Clarity. There should be a one-to-one correspondence between semantic constructs and graphical symbols.
This is required in order to be considered a notational system [19]. If there is not a one-to-one correspondence then one of the following can occur (Figure 2.12 helps in understanding these concepts):

- Symbol deficit: there is not a symbol that represents a specific construct;
- Symbol redundancy: there are several symbols that all represent one construct;
- Symbol overload: more than one construct is represented by a single symbol;
- Symbol excess: a symbol that does not have a corresponding construct.

In order to achieve semiotic clarity, a notation can not have any symbols that fit in the above categories [17].

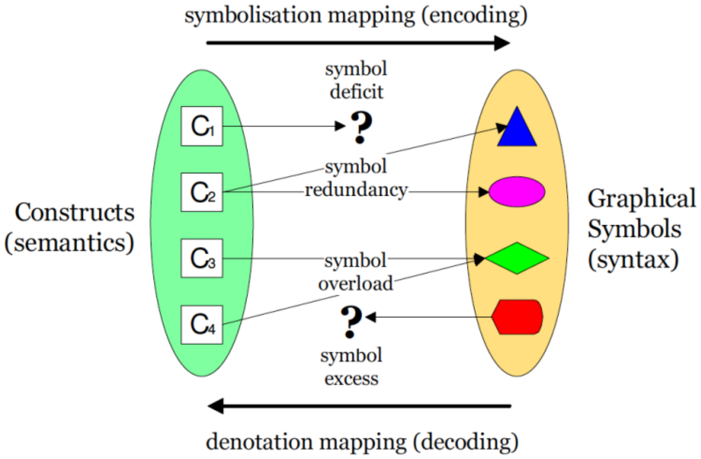


Figure 2.12: Understanding Semiotic Clarity [17]

Perceptual Discriminability: Symbols should be clearly distinguishable.
It is very important that symbols are easy to distinguish. Large visual distance between symbols allows for faster and more accurate interpretations of diagrams. When the difference is subtle there is a higher chance of error. This is especially true for novice users [44].

Semantic Transparency: Use symbols whose appearance is evocative.
Symbols should provide cues to their meaning [44]. When analysing notations, symbols can be divided into four levels of transparency:

- Semantically immediate: a novice can infer its meaning without help;
- Semantically translucent: a symbol between semantic immediacy and opacity, it provides a cue for the meaning but the novice can not infer it without some help.
- Semantically opaque: there is no relation between the appearance and its meaning (e.g., roundtangles in BPMN);
- Semantically perverse: makes a novice infer a different (or opposite) meaning from the symbol's appearance;

Complexity Management: Include mechanisms for handling complexity.
One of the largest problems in Software Engineering is that visual representations do not scale well [44]. Complexity Management is a way to mitigate that problem. However, there are several languages that do not take complexity into account (such as Entity Relationship diagrams). This results in diagrams with a very high level of complexity that are really hard to understand, even more so by novices. An example of this is shown on Figure 2.13.

A common solution for reducing complexity of large systems is to divide them into smaller subsystems. These subsystems are called modules which can then be

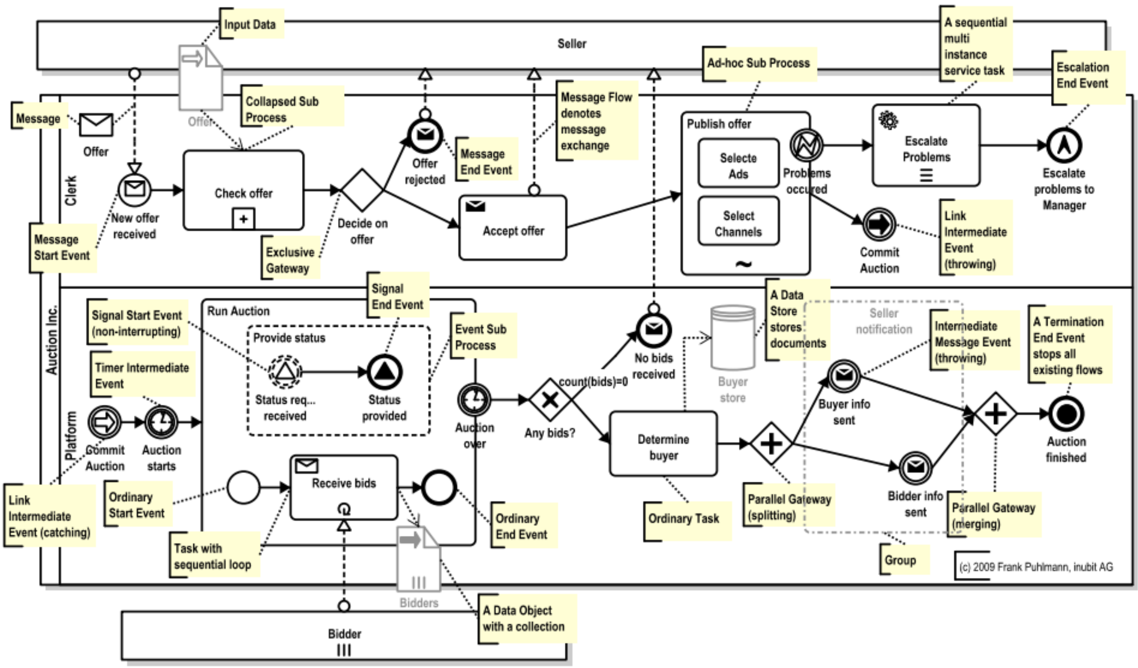


Figure 2.13: Example of a complex diagram [54].

reused. Subsystems can have their own subsystems. This creates modularisation with several levels which are called hierarchies. Hierarchies are the most effective way of organising complexity [44], allowing the system to be interpreted top-down (refinement) or bottom-up (abstraction).

Cognitive Integration: Include explicit mechanisms to support integration of information from different diagrams.
Having multiple diagrams describe a system (e.g. modularisation) requires the reader to have a mental image of the system while trying to interpret the diagram. This causes additional cognitive load [44]. There are mechanisms that can help with combating the extra load. These are Conceptual integration and Perceptual integration.

- **Conceptual integration** helps the reader interpret the different diagrams in order to better understand the system as a whole. This can be achieved by giving the readers a summary diagram, as this provides the readers with an overview of the system, or resorting to contextualisation, which adds to diagram context information (like adding all related elements from other diagrams) [44].
- **Perceptual integration** provides perceptual cues that help with navigation and transitioning between diagrams. To do this the language uses a design technique (that originated from architecture [38]) called *wayfinding* [44].

  Wayfiding is used to help users find what they want, this is done with four stages [35]:

- **Orientation**. Giving the user a current relative position to nearby objects and the final destination;
- **Route Decision**. Provide different ways for the user to get to the final destination and help the user choose the best path;
- **Route Monitoring**. Give feedback about the current path the user is on, such as path destination and current progress within the path;
- **Destination Recognition**. Make sure the destination is clear and easy to recognise. This can be achieved by placing the destination in a dead-end, preventing the user from progressing.

Placing labels on diagrams supports orientation and destination recognition. Level numbering shows users where they are in the system of diagrams and as such supports orientation. Adding navigational cues supports route choice and a navigational map that shows all the diagrams and paths between them supports orientation, route monitoring and route choice [44].

Visual Expressiveness: Use the full range and capacities of visual variables. Visual Expressiveness measures visual variation across the entire visual vocabulary [18]. It looks at how much design space and how many visual variables (see 2.3.3.1) are used in the language.

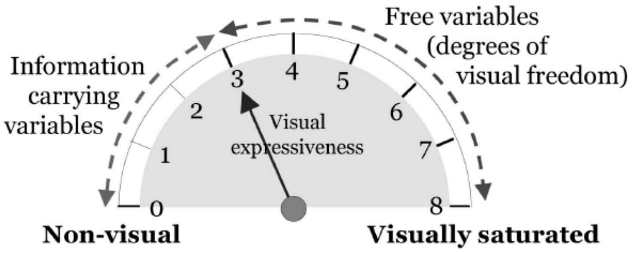

Information carrying variables
Free variables (degrees of visual freedom)
0
1
2
3
4
5
6
7
8
Visual expressiveness
Non-visual
Visually saturated

Figure 2.14: Visual expressiveness. [44].

Visual variables are divided into *Information carrying variables* (these are variables that are used by the language) and *free variables* (which are variables that are not used by the language). The number of used variables is called *expressiveness* and the number of free variables is called *degrees of visual freedom* [44] (Figure 2.11).

Notations with zero expressiveness are called *nonvisual*, or textual (*e.g.* Unified Modeling Language (UML)). Notations with eight expressiveness (or zero visual freedom) are called *visually saturated*. The higher a languages expressiveness is, the more perceptually enriched it is. This can dramatically improve the language's usability [44]. A visual representation of this is found on Figure 2.14.

Dual Coding: Enrich diagrams with textual descriptions.
Using text and graphics together is more effective than using either one of them on

their own. With this said, text should never be used as the sole basis for distinguishing between symbols, rather it should complement the graphics [44].

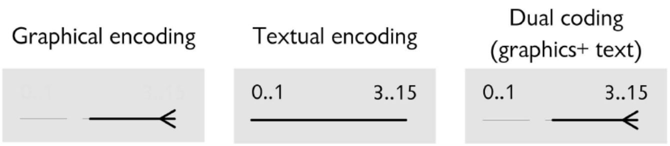


Figure 2.15: Example of dual coding. [44].

Adding text to symbols does not increase its visual distance, however it helps with interpretation by providing textual cues which increases its transparency. Figure 2.15 contains an example of dual coding.

Graphic Economy: Keep the number of different graphical symbols cognitively manageable.
Not to be confused with Complexity Management, that deals with sentence level complexity. Graphic Economy is concerned with the complexity of the language, which means it looks at the number of graphical symbols [44]. Having too many symbols in the language can become a big problem, especially for novices, since humans have a hard time working with more than six different categories of symbols (UML Class Diagrams have over forty).

There are three main ways to deal with graphic complexity:

- **Reduce semantic complexity**. The most obvious way to reduce complexity is to simplify the language's semantics. This happens because normally each construct is represented by a symbol.
- **Introduce symbol deficit**. Complexity can also be dealt with by directly reduced (without touching the semantics) by not showing some constructs graphically (though this goes against Visual Expressiveness). It is important to find a balance between graphical, textual and off-diagram encoding.
- **Increase visual expressiveness**. The six-symbol limit only applies to a single visual variable, so an easy solution is to simply increase human discrimination by using more visual variables.

Cognitive Fit: Use different visual dialects when required.
The Cognitive Fit theory (from Information Systems) states that there should different representations for different types of users [59]. Most Software Engineering notations do not follow this theory and have only one notation that is meant to be a "one size fits all". This can cause issues which can be broken down into:

- **Difference in user skill**. Developing a notation that is easy to understand by a novice but also complex enough to answer the needs of experts is nearly an impossible task. This happens because novices have difficulty in discriminating between symbols, are more effected by complexity and need to consciously remember what symbols mean. A solution for this is to have "user modes", where the novice mode has a subset of the notation (to about complexity) and more cues to help with semantic transparency [44].

- **Difference in task**. Using a notation in a software environment built for that notation is very different to using it on paper in a meeting. This happens because on paper the user has limited access to visual variables (such as colour) and symbols that are not simple (due to semantic transparency) can be hard to hand draw. To solve this there should be a secondary notation which is simplified for sketching [44].

2.3.4 NASA Task Load Index

NASA Task Load Index (NASA-Task Load Index (TLX)) is a subjective assessment tool for rating perceived workload. Its original application was aviation but is now used in a variety of different domains [56]. The term workload represents the cost of completing a task and there are many psychological definitions on how to measure it [37]. NASA-TLX measures workload by dividing it into six subclasses, known as scales: Mental, Physical, and Temporal Demands, Frustration, Effort and Performance [26]. The rating is a numeric value between 5 and 100. A low rating means a low workload and a high rating meaning a high workload.

2.3.4.1 The scales

Each scale should always be presented with a description and the participant should read each description with attention before rating it.

Mental Demand. How much mental and perceptual activity was required (e.g. thinking, deciding, calculating, remembering, looking, searching, etc)? Was the task easy or demanding, simple or complex, exacting or forgiving?

Physical Demand. How much physical activity was required (e.g. pushing, pulling, turning, controlling, activating, etc)? Was the task easy or demanding, slow or brisk, slack or strenuous, restful or laborious?

Temporal Demand. How much time pressure did you feel due to the rate of pace at which the tasks or task elements occurred? Was the pace slow and leisurely or rapid and frantic?

Performance. How successful do you think you were in accomplishing the goals of the task set by the experimenter (or yourself)? How satisfied were you with your performance in accomplishing these goals?

Effort. How hard did you have to work (mentally and physically) to accomplish your level of performance?

Frustration. How insecure, discouraged, irritated, stressed and annoyed versus secure, gratified, content, relaxed and complacent did you feel during the task?

2.3.4.2 The questionnaire

The NASA-TLX questionnaire is given to the participant after he/she completes a task. It is divided into two phases.

In the first phase participant is presented with the six scales (with an accompanying description) and is asked to rate each scale within a 100-point range with a 5-point step [27]. An example is shown in Figure 2.16.

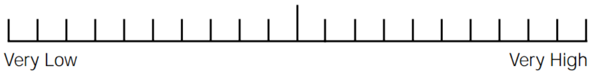


Figure 2.16: NASA-TLX answer format

The second phase asks the participant to weigh each scale's importance. With the six scales there are 15 possible pairwise comparisons, each of these are presented to the participant who chooses which scale he/she considers to be more relevant for the task. With this each scale has a weight that ranges from 0 (not at all relevant) to 5 (more relevant that any other scale) [23].

To then calculate the final score, for each scale multiply its rating with its weight, divide that value by 15 and finally sum all the scales. The score will always be a value between 0 and 100 [23].

2.3.4.3 Raw-TLX

There is a version of NASA-TLX called Raw-TLX which removes the weighing process and the ratings are simply averaged. When using Raw-TLX, scales can be removed if they are considering irrelevant to the task.

There are studies defending that Raw-TLX is better than the original [9], others defend that the original is better and others saying they are the same. With this, the choice comes down to personal preference [26].

Chapter 3

Related work

3.1 Using Physics of Notations to evaluate BPMN 2.0

In 2011 Genon *et al.* used the Physics of Notations to study BPMN 2.0's visual notation [18]. In that study they go through the nine principles from the Physics of Notations and for each they analyse BPMN 2.0 according to said principle. This section contains how the analysis was done and the result for each principle.

3.1.1 Semiotic Clarity

This principle states that there should be a one to one correspondence between semantic constructs and graphical symbols. To check this, a set of metaclasses was selected from the BPMN 2.0 metamodel. This set was then mapped with the set of BPMN symbols.

A mapping function more complex than a simple *semantic construct* ↔ *symbol* was used. This function made four considerations:

- only concrete metaclasses that are not enumerations were considered;
- expanded/collapsed representations were only considered as a single symbol;
- metaclasses with attributes that cause variation in the symbol are considered distinct semantic constructs;
- if the multiplicity of a role modifies the representation of the metaclass, then it is considered a different semantic construct.

The result of the mapping can be found on table 3.1.

Defect	Occurrences
Symbol Excess	1
Symbol Deficit	57
Symbol Redundancy	1
Symbol Overload	13

Table 3.1: Result of Semiotic Clarity BPMN 2.0 evaluation [18].

3.1.2 Perceptual Discriminability

The principle covers the need for visual distance, symbols should be easy to distinguish which in turn increases the language's expressiveness.

In order to analyse BPMN 2.0's perceptual discriminability each symbol was broken down to its visual variable values and for each value it was identified if that value was a semantic carrier.

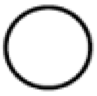
Metaclass	Symbol	Visual variable value	Semantics carrier
Event		**(x,y)**: variable	no
		shape:circle	yes
		colour: black/white	no
		brightness: N.A.	no
		size: variable	no
		orientation: N.A.	no
		texture: thin border line	yes

Table 3.2: Example of a symbol's visual variable analysis [18].

The result of the analysis indicates that no symbol has a visual distance greater than two visual variables. The language does not support redundant coding, perceptual popout or textual differentiation.

Table 3.2 contains an example of a symbol's visual variable analysis.

3.1.3 Semantics transparency

A visual notation should be transparent, symbols should be evocative and help with the interpretation of their meaning.

The only way to analyse transparency is to go through all the symbols and for each one decide on its transparency. The study covers the symbol's marker and shape, it classifies them on their transparency and provides a justification for the given classification.

The results indicate that the majority of the symbols are not at all transparent, with a few semantic perverse cases.

3.1.4 Complexity Management

The two solutions within BPMN for managing complexity are modularisation and hierarchies. The analysis of complexity management was done by reviewing what semantic constructs exist that help with complexity, these are: subProcess, linkEvent and callActivity.

- **SubProcess** can be used to decompose large diagrams into a set of sub-diagrams and can also represent different levels of detail. As such, subProcess contributes to the diagram's hierarchy and modularisation.
- **LinkEvent** is a construct that allows navigation through diagrams by indicating go-to points. It can be used to connect two sections of a Process and also as a off-page connector for printing across multiple pages. This improves modularisation.
- **CallActivity** is a wrapper for the invocation of a global task. As such, it helps with modularisation.

The results indicate that BPMN 2.0 has some mechanics to improve complexity management, though these could be improved (such as allowing subProcesses to reference another process, rather than having the subProcess depicted inside the parent).

3.1.5 Cognitive Integration

A language should have mechanisms to help users interpret diagrams. The study analyses this by trying to identify mechanisms within BPMN 2.0's notation that answer the need for perceptual and conceptual integration.

- **Perceptual integration**. BPMN 2.0 does not have most techniques that help with wayfinding. It does not have a dedicated artefact for displaying the diagram's name. Considering hierarchies are embedded in the parent level, numbering is not required. There is only one navigational cue (Link Event) but it requires the use of a TextNotation to specify what diagram is being referenced. There is not a navigation map.
- **Conceptual integration**. This can be achieved by providing the user with a summary diagram or with contextualisation. BPMN 2.0 does not support either of these mechanics.

3.1.6 Visual Expressiveness

The notation's visual expressiveness is measured by the number of visual variables used. After analysing BPMN 2.0's notation the study concludes that BPMN 2.0's visual expressiveness is 4 ((x,y), shape, colour and texture).

3.1.7 Dual Coding

The principle states that text can be used to complement graphics in order to help interpretation. BPMN 2.0 achieves this by using the construct *Text Annotation* which can allows the user to attach text to any other symbol (Figure 3.1).

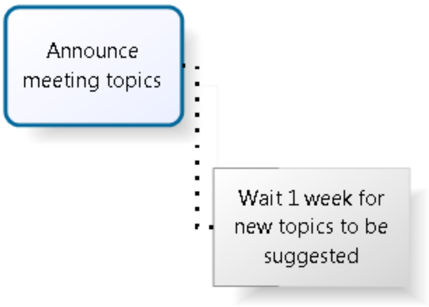


Figure 3.1: Example of good dual coding usage.

However, there are cases where textual descriptions are mandatory and symbols do not make sense without them. This goes against the Dual Coding principle. As an example, a process starting with an event has to have a an accompanying message, otherwise it does not make sense (Figure 3.2).

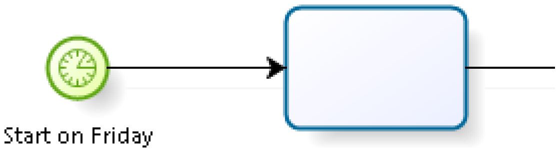


Figure 3.2: Example of bad dual coding usage.

3.1.8 Graphic Economy

The principle states that humans have a hard time working with more than six different categories of symbol. The size of BPMN 2.0's visual vocabulary is one hundred and seventy one. The size of the vocabulary is due to the large amount of markers which can then be combined with the different shapes. Taking only into account the shapes, then there are six elements in the vocabulary, but without the markers the language does not have enough expressiveness to be usable.

The Physics of Notations proposes three strategies to reduce graphic complexity:

- **Reduce semantic complexity**. This is very hard to achieve. Reducing the language semantics would also reduce its expressiveness and would prove too simple for its requirements;

- **Introduce symbol deficit**. This could work but the information that would be removed by the deficit would have to be added somewhere off-diagram. A link to that information would then need to be placed in the diagram;
- **Increase visual expressiveness**. The language still have four free visual variables, making this an appropriate strategy.

3.1.9 Cognitive Fit

The principle supports that there should be more than one notation, depending on user skill and task. BPMN 2.0 has only one notation. The notation can be very hard for novices as some symbols are not easy to discriminate due to double line borders and contain markers. Also, the visual vocabulary is very large, making it hard to remember.

The notation is easy to sketch since it does not include the colour visual variable. That said, it can be complicated for novices because drawing double lines and thick borders takes time to be well sketched.

3.2 Using Visual Notation Design to improve i*

It has already been established that notations play a critical part in language usability [44]. As such, its important to understand what is the best way to design a notation. Moody *et al.* published in 2010 a paper that evaluates the *i** notation using Physics of Notations and proposes a new symbol set [45]. Caire *et al.* in 2013 published a paper that proposes some new design methods and applies them to *i**. The study then compares them to the notation's standard, the notation proposed by Moody *et al.* and the result of the new methods [10].

3.2.1 The *i** standard and Physics of Notations

The *i** language [15] is considered one of requirements engineering most influential notations [10]. The standard *i** notation is shown in Figure 3.3.

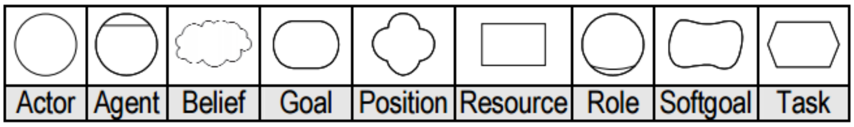


Figure 3.3: Standard *i** notation [10][15].

In 2010 Moody *et al.* published a paper evaluating the *i** using the principles from the Physics of Notations, the study concluded that the *i** notation was semantically opaque (there is no connection between the symbol and its meaning) [45]. The study also proposes changes to the current notation (Figure 3.4). These changes follow the Physics of

Notations principles and are made by experts in the field.

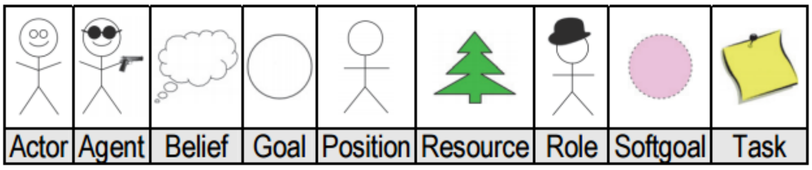


Figure 3.4: Proposed notation by Moody *et al.* [10][45].

3.2.2 Stereotyping

Caire *et al.* [10] proposes that perhaps the best way to develop a visual notation is to ask the end users for help. This was done applying the Sign Production Technique [30]. The technique asks members of the target audience to generate symbols that represent concepts in the language. The researchers asked 104 naive participants to draw a symbol for each of the nine *i** constructs.

From the result of the Sign Production Technique, a judge's ranking method [32] was used to create a population stereotype (an average for each construct). The resulting nine symbols can be seen in Figure 3.5.

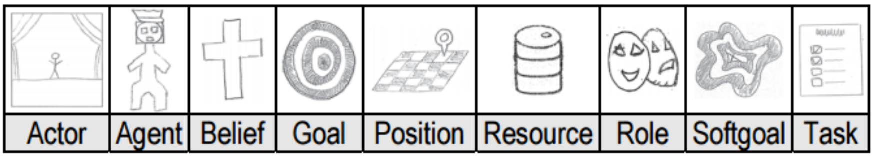


Figure 3.5: Stereotype symbol set [10].

3.2.3 Prototyping

Another method used was prototyping. This was done by asking 30 naive users to go through the symbols resulting from the Sign Production Technique and choose the symbol that best represents the construct. The resulting set of symbols can bee seen in Figure 3.6.

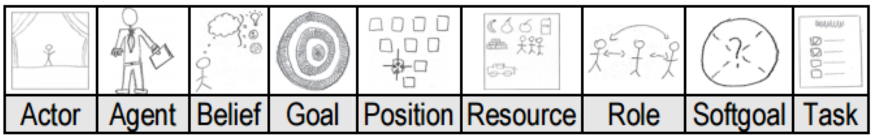


Figure 3.6: Prototype symbol set [10].

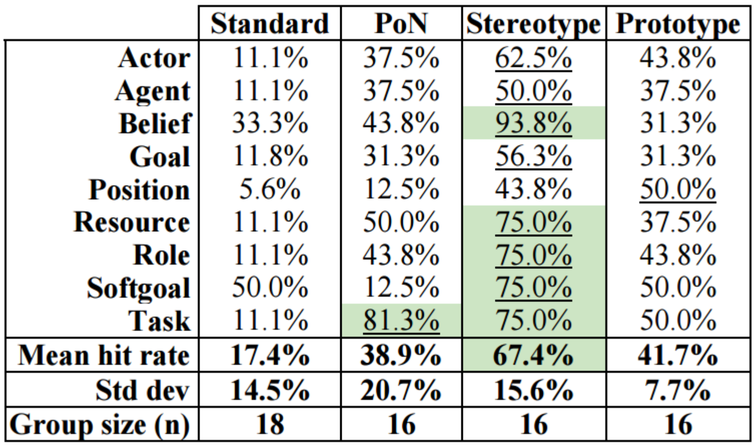
This resulted in only a third of the symbols matching the stereotype counterparts. The other symbols won with a high level of consensus.

3.2.4 Choosing the best symbol set

An experiment was done to determine which of the following notations was the most comprehensible:

- **Standard i***. developed by experts using intuition;
- **Physics of Notations i***. Created by experts following the Physics of Notations principles;
- **Stereotype i***. The most common symbols created by novices;
- **Prototype i***. The symbols with the most votes, judged by novices.

The experiment was a blind interpretation study. It is a common method for evaluating comprehensibility of graphics symbols and is used for testing ISO standard symbols prior to release.



	Standard	PoN	Stereotype	Prototype
Actor	11.1%	37.5%	62.5%	43.8%
Agent	11.1%	37.5%	50.0%	37.5%
Belief	33.3%	43.8%	93.8%	31.3%
Goal	11.8%	31.3%	56.3%	31.3%
Position	5.6%	12.5%	43.8%	50.0%
Resource	11.1%	50.0%	75.0%	37.5%
Role	11.1%	43.8%	75.0%	43.8%
Softgoal	50.0%	12.5%	75.0%	50.0%
Task	11.1%	81.3%	75.0%	50.0%
Mean hit rate	**17.4%**	**38.9%**	**67.4%**	**41.7%**
Std dev	**14.5%**	**20.7%**	**15.6%**	**7.7%**
Group size (n)	**18**	**16**	**16**	**16**

Figure 3.7: Result of the blind interpretation study. [10].

The results can be seen in Figure 3.7. In green are values that are over ISO's compressibility threshold and the best values for each construct are underlined.

The results indicate that novice generated symbols are more semantically transparent than symbols generated by experts. Only the stereotypical notation was able to achieve an average higher ISO's minimum threshold of 67%, with the standard i* notation being the worst of the four.

3.3 Eye-tracking in language evaluation

In the 1800s cognitive psychology researchers first used eye-tracking in order to better answer questions regarding reading. That research led to the conclusion that eyes do not traverse text in a smooth matter [31]. Rather, the movement of the eyes can be broken down to a series short stops (called fixations) and of *saccades* (quick, simultaneous movements of both eyes between fixations) [11].

The first time eye-tracking was used for language usability was in 1990, it was used to study code comprehension. In 2015 Sharafi *et al.* publishing a Systematic Literature Review (SLR) about the usage of eye-tracking in software engineering [57]. That paper concluded that eye-tracking was not commonly used, having found only 36 relevant software engineering studies.

3.3.1 Technology

There are several different ways for eye-trackers to measure the rotation of the eyes. The SLR by Sharafi divides the techniques into two relevant categories: intrusive and non intrusive eye-trackers.

Intrusive eye-trackers typically contain three miniature cameras. These are mounted on a padded headband which is worn by the participants during the usability test. Two of the cameras capture eye-movements using infrared lights and the third (optional) is used to track head movement. Given that the tracker is intrusive it may worry the the participants and thus influence the results.

Non-intrusive eye-trackers are divided into two generations. Both have cameras mounted near the computer screen. The older generation has one camera which beams light to the participant's eyes. The light is then reflected and captured by the camera on the way back. The newer generation requires two cameras which track the participants head using eye-brows, noses and lips. In order to distinguish between head and eye-movement, the cameras use infrared corneal reflection and pupil centring.

3.3.2 Limitations

Sharafi's SLR discusses some limitations to using eye-tracking.

The accuracy reported by eye-tracking manufactures are the result of tests conducted in ideal conditions. The test subjects do not wear anything that can effect the accuracy (such as glasses or lenses) and they never move their head. This scenario is not realistic as participants of usability studies will not always have the same iris size, can require seeing glasses and tend to move their head while conducting the test. To combat these limitations researchers can increase the size of the models and text to compensate for the lack of precision. Researchers should also calibrate the eye-trackers regularly during the test.

Results indicate that there is a gradual decrease in accuracy as time progresses. This is called drift. This is caused by physiological changes to the eyes (like wetness) and can be mitigated by ensuring that light conditions remain stable. The duration of the tests should take drift into account, shorter tests are recommended and calibrations should be repeated regularly.

The last limitation discussed by Sharafi is called the Hawthorn effect. Throughout the duration of the study the researcher is responsible for guiding the participants, calibrating the eye-tracker and monitoring the recording to ensure that the tracker is tracking correctly. The presence of the researcher can cause a bias, given that the participants feel that they are being watched. Some studies combat this limitation by sitting researchers away from the participants. This ensures that there is not any interaction between the participants and researchers. Studies also indicate that at the start of the session the researcher must explain that the eye-tracker only records records eye-movement, no video is being recorded and that all data is anonymous.

3.3.3 Data analysis

Eye-tracking studies produce a large amount of data. Manually processing that data would be unwise since it is prone to error and would require a lot of time. There are several automated tools for analysing eye-tracking data. The choice of what software to use depends on what kind of data the study wants analysed. The different metrics can be divided into five categories: Number of fixations, duration of fixations, saccades, scan-paths and gaze.

3.4 The OutSystems method

This section goes over the methodology used at OutSystems for usability evaluation. The method is an adapted version from the method described by Krug in *Rocket surgery made easy* [34].

3.4.1 The method

The method has a one month cycle and is broken down in the following items:

- There is a priority queue with features that need to be tested, as features are developed they go into queue. The position of the feature in the queue is based on its perceived importance;

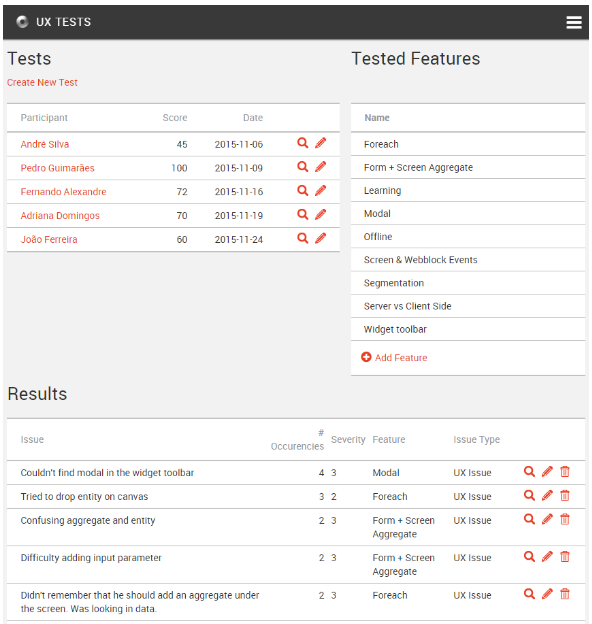
- A set of features are chosen from the queue to be tested. The number of features chosen is based on how long it takes for a participant to test them. Tests are planned to take no more than one hour and a half;

- Tests are mostly made to novices. This happens because experts have a bias for the old version, though some specific features are also tested with experts to evaluate acceptance;

- Screen and voice is captured and participants are asked to think out loud;

- At the end of test, the participant is asked to fill in a SUS questionnaire;

- At the end of month there is a meeting with key members from different teams to discuss the encountered issues. A list of common issues is created. The video recording is used to understand what was happening and possible solutions are discussed;
- A feature is flagged for change if it is considered critical and few users have trouble with it, or if it is not critical but a lot of users have trouble with it;
- A report is written and there is a meeting with the leader of the team responsible for the feature in order to discuss possible changes.



Tests

Create New Test

Participant	Score	Date
André Silva	45	2015-11-06
Pedro Guimarães	100	2015-11-09
Fernando Alexandre	72	2015-11-16
Adriana Domingos	70	2015-11-19
João Ferreira	60	2015-11-24

Tested Features

Name
Foreach
Form + Screen Aggregate
Learning
Modal
Offline
Screen & Webblock Events
Segmentation
Server vs Client Side
Widget toolbar

Results

Issue	# Occurencies	Severity	Feature	Issue Type
Couldn't find modal in the widget toolbar	4	3	Modal	UX Issue
Tried to drop entity on canvas	3	2	Foreach	UX Issue
Confusing aggregate and entity	2	3	Form + Screen Aggregate	UX Issue
Difficulty adding input parameter	2	3	Form + Screen Aggregate	UX Issue
Didn't remember that he should add an aggregate under the screen. Was looking in data.	2	3	Foreach	UX Issue

Figure 3.8: Tool used to manage usability tests.

The usability tests are done with a single participant at a time and there is always a member of OutSystems present. It follows these steps:

1. There is a short briefing about the goal of the test and some basic notions of how the usability test works;
2. A set of questions are made to define the user's profile;
3. The participant watches a small video introduction to the OutSystems platform. The length is approximately seven minutes;
4. Participant is asked to execute some predefined tasks;
5. Product feedback questions are asked, such as "*What did you like?*", "*What would you change?*" and "*Where did you struggle?*";
6. The participant is asked to fill in a SUS questionnaire.

Chapter 4

Evaluation process

4.1 The process

The process for evaluating languages is divided into two main parts: identifying issues with the language and developing improvements for said language. Some of the steps in the process require interaction with people experienced with the language being evaluated, while others require conducting tests with people who have never interacted with the language. A visual representation of the process can be found in Figure 4.1.

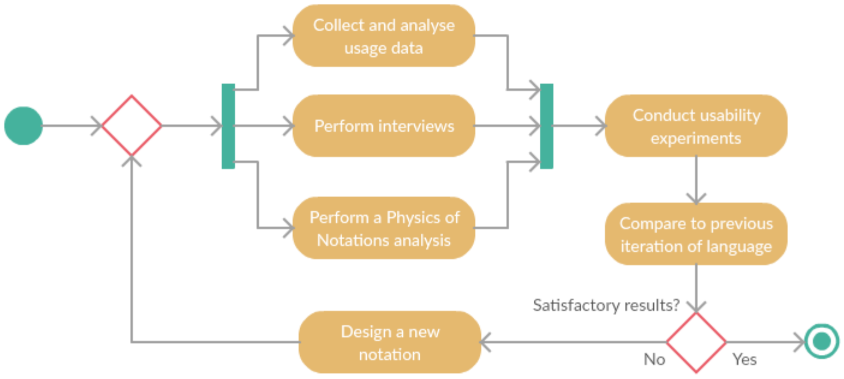

Collect and analyse usage data
Perform interviews
Perform a Physics of Notations analysis
Conduct usability experiments
Compare to previous iteration of language
Satisfactory results?
No
Yes
Design a new notation

Figure 4.1: Evaluation process activity diagram.

4.1.1 Perform interviews

Conduct interviews with people who currently use the language. The interviews should follow the Design Thinking philosophy of empathy interviews [53]. This kind of interview allows the interviewee to tell a story. A lot more insights can be collected from those stories than with direct questions. The interviews should cover the following topics:

- **In what context is the language being used in.** Even if the language is a DSL it is possible that the language 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 was the language chosen.** This brings up the strengths of the language. Possible features that the expert likes and may 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 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?

4.1.2 Collect and analyse usage data

If possible collect everyday data while the users are using the language to develop solutions. This can then be used to extract usage metrics, such as: how long do users take to perform a certain task; how complex are the projects developed using the language; do users get "lost" while using a certain language feature; and how frequently are features used.

4.1.3 Physics of Notations

Analyse the language following the Physics of Notations principles. Check if the language is complying with each of the nine principles and when making changes to the language ensure that the changes solve any issues discovered in this analysis.

4.1.4 Usability experiments

Conduct usability tests where the tester is given an example of a solution developed using the language being evaluated and is asked a series of questions related to the example. The example should use all language constructs and the questions should not be subjective (*e.g.*: "What do you think of X?"). An example of a good question is: "What elements of the language interact with the a database?". The answer to that will be objective and will either be correct or not;

These experiments should record the screen and capture the voice of the tester (if permission is given). The tester should be asked to think out loud so that it is easier to analyse at a later date. Eye-tracking can also be used as a supplement. This makes the tester's line of thought while trying to answer the questions easier to follow at a later date.

At the end of the experiment the tester should be asked to answer a System Usability Scale and Task Load Index questionnaire.

4.1.5 Designing a new notation

The new notation should be created following *Caire et al.*'s Visual Notation Design. This requires a large number of novices but has shown to produce the best results. That said, the results of the Physics of Notations analysis should not be ignored and other factors (such as consistency) also need to be considered.

4.1.6 Analyse the new notation

This should be done by conducting a semantic transparency study. However, even if the language has a high transparency rating it is important to conduct more usability experiments (following the same method as in 4.1.4). This happens because semantic transparency evaluates symbols individually, which does not ensure a high language usability rating.

4.2 Usability Tool

We developed an application to help with the evaluation of languages. The goal is to speed up the evaluations, keep the information centralised and automate the process as much as possible.

The application allows users to manage different languages, their usability tests and process Sign Production results. The application was built using the OutSystems Platform.

4.2.1 Languages

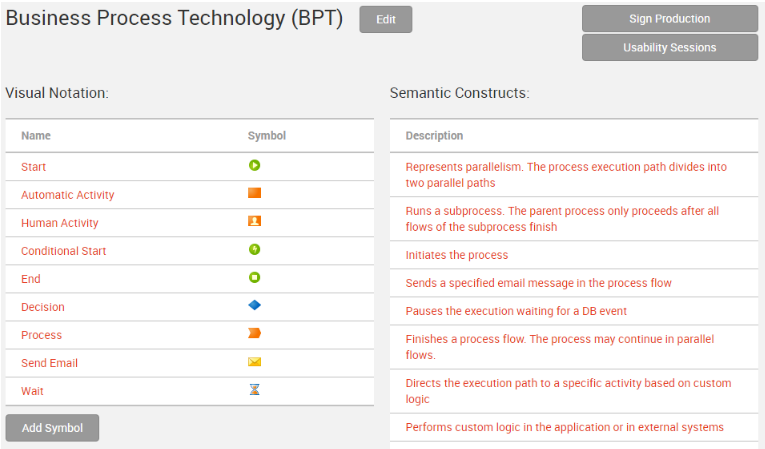
Each user can have a set of languages. Each language has its own page, where the user adds the language's semantic constructs and visual notation (Figure 4.2). This is then used for the analysis of the Sign Production results.

4.2.2 Usability Sessions

This section of the application manages usability tests with study participants. Each study saves data about the tester's profile, URL to the survey used for the test, possible URL to a recording of the session and surveys completed by the tester (Figure 4.3).

4.2.2.1 Surveys

UsabilityTool supports SUS and TLX surveys. A link to each of the surveys is placed on the table which contains the evaluations. If they tester did not complete the survey then it contains a link to an empty survey, otherwise it shows the tester's score which links



Business Process Technology (BPT)

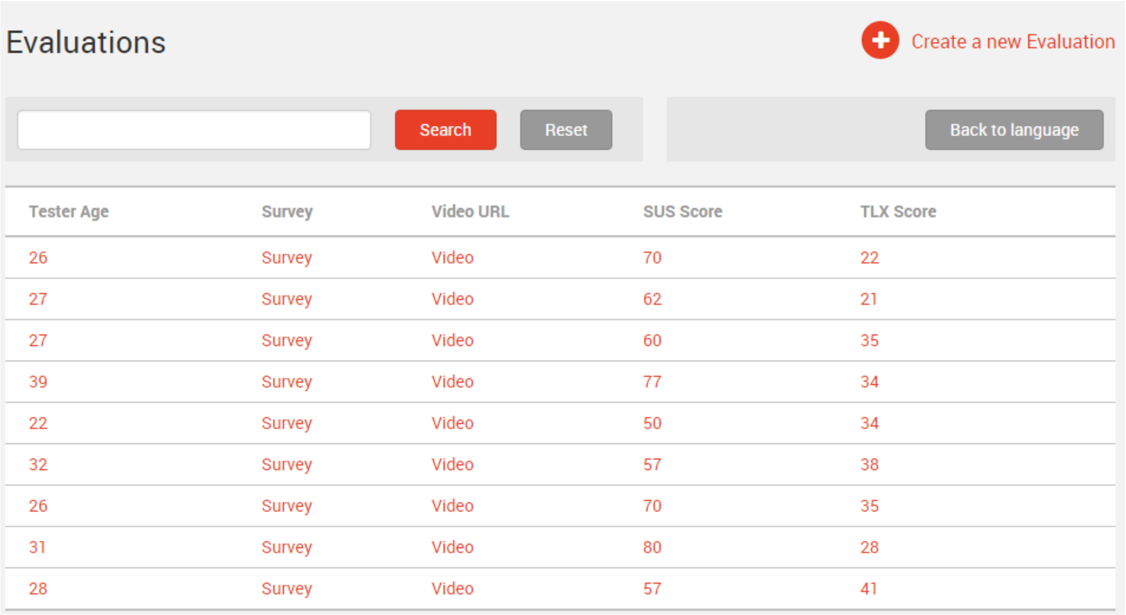
Visual Notation:

Name	Symbol
Start	▶
Automatic Activity	■
Human Activity	👤
Conditional Start	⚡
End	⏹
Decision	◆
Process	➤
Send Email	✉
Wait	⌛

Semantic Constructs:

Description
Represents parallelism. The process execution path divides into two parallel paths
Runs a subprocess. The parent process only proceeds after all flows of the subprocess finish
Initiates the process
Sends a specified email message in the process flow
Pauses the execution waiting for a DB event
Finishes a process flow. The process may continue in parallel flows.
Directs the execution path to a specific activity based on custom logic
Performs custom logic in the application or in external systems

Figure 4.2: Example of a language page.



Evaluations

Tester Age	Survey	Video URL	SUS Score	TLX Score
26	Survey	Video	70	22
27	Survey	Video	62	21
27	Survey	Video	60	35
39	Survey	Video	77	34
22	Survey	Video	50	34
32	Survey	Video	57	38
26	Survey	Video	70	35
31	Survey	Video	80	28
28	Survey	Video	57	41

Figure 4.3: Example of a Usability Sessions page.

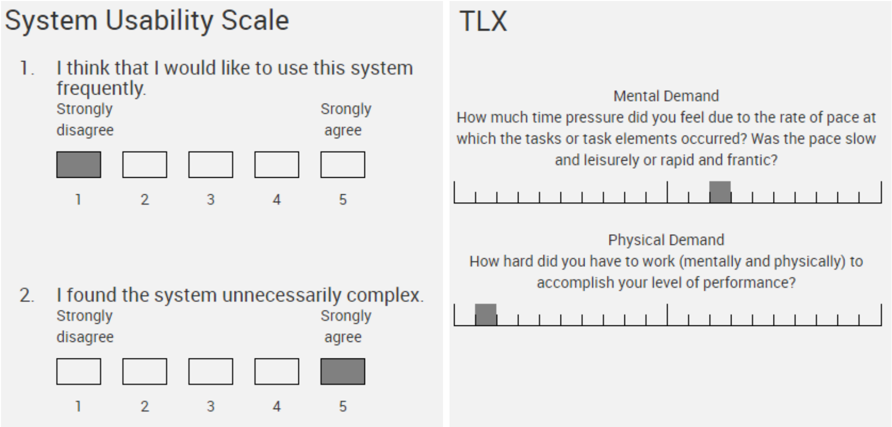
to the filled in survey. The two surveys were created to be as close as possible as their original versions (Figure 4.4).

4.2.3 Sign Production

This is a feature created to help users with the technique presented in 3.2. Here the users can upload and analyse the results from the paper questionnaire.

4.2.3.1 Uploading results

The results can be uploaded manually one by one or in a batch. To upload the questionnaires manually the user creates a new study and then uploads the results of each of the



System Usability Scale

1. I think that I would like to use this system frequently.

Strongly disagree ■ 1 □ 2 □ 3 □ 4 □ 5 Strongly agree

2. I found the system unnecessarily complex.

Strongly disagree □ 1 □ 2 □ 3 □ 4 ■ 5 Strongly agree

TLX

Mental Demand

How much time pressure did you feel due to the rate of pace at which the tasks or task elements occurred? Was the pace slow and leisurely or rapid and frantic?

Physical Demand

How hard did you have to work (mentally and physically) to accomplish your level of performance?

(a) SUS (b) Raw TLX

Figure 4.4: Example of surveys supported by UsabilityTool

language constructs. A console application called "Sign Production Cutter" was created in C# and has two features: cut and organise the questionnaire answers and upload the answers to UsabilityTool.

To cut and organise the questionnaires, Sign Production Cutter receives as input the scanned pages from the questionnaire (all of them, not only the answer pages). It then cuts boxes where the answers are and saves them in the local directory. The file organisation is done in two ways: one where a folder is created for each questionnaire and all the answers are saved in that folder with a final image containing the tester's profile; and another where a folder is created for each construct and all the answers relating to that construct are saved. The questionnaire folders were created to make it easier to upload the answers to the UsabilityTool. The construct folders make it easier to analyse the results without uploading them to UsabilityTool. This means Sign Production Cutter is useful regardless if UsabilityTool is being used or not.

If the user chooses to, the application uploads the answers by calling API functions created in UsabilityTool. However, the user must fill in a text file with the tester's profile before uploading. Since the profile contains open answers it would be very hard to automate this process.

4.2.3.2 Analysing results

The analysis of the questionnaires can be done in the local directory after using Sign Production Cutter to process the answers or on UsabilityTool after uploading the answers. UsabilityTool organises the answers the same way that Sign Production Cutter does: by questionnaire and by constructs. The page of a questionnaire contains all the answers of that questionnaire and the profile information of the tester. The construct page contains all the answers for that specific construct, making it easier to compare the results.

CHAPTER 5

Analysing BPT

The first step in analysing BPT was identifying that there was indeed a problem with the language. By analysing data from projects developed using OutSystems Platform we concluded that BPT has a low adoption rate. The next step was identifying what was the problem. This was done by using Physics of Notations to ensure that the language is considered usable according to the principles presented in 2.3.3 and conduct evaluations with testers to understand where users were having trouble with the language.

5.1 Interviews

A series of interviews were made to developers within OutSystems who use the language to create solutions for clients. This was done in order to better understand why OutSystems' BPT does not have the desired adoption rate.

5.1.1 Interview questions

The interviews did not follow a predetermined script, rather the interview followed the Design Thinking philosophy of empathy interviews [53]. The goal was to find the root of the problem by getting the expert to tell a story and applying the *five whys* [12] technique. The *five whys* is an iterative interrogative technique used to explore the cause-and-effect relationships underlying a particular problem. This is done by asking *why* at least five times to each answer. That said the main topics were:

- "In what scenarios do you use BPT to develop solutions?";

- "Do clients request the use of BPT?";

- "Do you like using BPT?";

- "What would you change in BPT?";
- "What would you add to BPT?".

Even though some of the topics are binary questions (which goes against the Design Thinking philosophy) by using the *five whys* we were able to expand the topic and collect insights about it.

5.1.2 Insight analysis

The following insights about BPT were extracted from the interviews:

- Development teams like using BPT but are scared to do so due to low level nuances, as such they fallback to what they are used to (*Timers*);
- Parallelism is hard to model and so is identifying synchronisation bugs;
- New team members can not start working with BPT without specialised training;
- Some clients explicitly request the use of BPT;
- Maintaining a project developed with BPT is difficult and costly; there are performance issues.

Another insight was that BPT is often used outside of its domain. While BPT was designed to be a process modelling language it is 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.

5.2 Data Collection

OutSystems has a repository of projects developed with the OutSystems Platform that contains 5145 *eSpaces*. OutSystems' R&D group has developed an internal application called *QueryGrabber* that allows data to be gathered from *eSpaces*. *QueryGrabber* receives as inputs the path with the *eSpaces* and a *C#* class file with a function (*processor*) that is called for each *eSpace*. In essence, *QueryGrabber* can be seen as a *foreach* that iterates the *eSpaces* and for each *eSpace* it calls the provided *processor* function.

We developed a *processor* function that collects the following information about each *eSpace*:

- Number of processes built with BPT;
- Total number of BPT nodes;
- Number of flows within the process;

- Number of interfaces, total number of actions;
- Total number of widgets with in the interfaces;
- Number of occurrences of each construct provided by BPT;
- Number of process calls;
- Information pertaining to complexity (like number of arrows pointing to nodes and number of arrows exiting nodes);
- Existence of infinite cycles within a process.

Table 5.1 contains the most informative metrics collected. The values show us that only about 3% of the eSpaces use BPT. This is a very small number. The eSpaces that use BPT have in average two processes each and are quite complex with an average of 18.7 nodes per process. This shows that BPT is being used in complex systems. However, 34% of the processes uses its metadata, which is saved in system entities. This means it manually queries the database and uses information like process scheduling in its logic. This is a bad practice as it makes the process much slower.

Table 5.1: Important collected metrics

Metric	Value
Total eSpaces	5145
Total eSpaces using BPT	179
Total BPTs	353
Average Nodes per BPT	18.7
Number of BPT that use process metadata	120

5.3 Physics of Notations

A Physics of Notations study was performed on BPT. We went through the nine principles from the Physics of Notations and for each we analysed BPT according to said principle.

5.3.1 Semiotic Clarity

We verified if there is a one-to-one relationship between semantic constructs and concrete syntax. This was done by mapping each of the constructs with a language symbol and we found cases of symbol deficit and overload. This happens when there is no symbol that matches a specific construct (deficit) and if a symbol has more than one construct pointing to it (overload).

- **Deficit:** The language semantically supports Forks and Joins for modelling parallelism. However, there are no symbols for these two constructs. This explains why on the interviews users were complaining about the difficulty to identify synchronisation bugs and model parallelism.

- **Overload:** There are several cases of symbol overload: The Timeout symbol which is used for three types of waits (which all have different behaviours); The End symbol is used for the End construct but also for the Terminate; and there are several ways of triggering a process, but the Start symbol does not reflect that.

5.3.2 Perceptual Discriminability

Symbols with high visual distance are easier to distinguish. We broke down each of BPT's symbols into their base visual variables and defined whether they were semantic carriers or not. An example of how this was done can be seen in Table 5.2.

Table 5.2: Example of a symbol's visual variable analysis.

Metaclass	Symbol	Visual variable value	Semantics carrier
Automatic Activity	🟧	**(x,y)**: variable	yes
		shape:rectangle	yes
		colour: orange	yes
		brightness: N.A.	no
		size: static	no
		orientation: N.A.	no
		border grain: N.A.	no

The results showed us that no symbol has a visual distance greater than three visual variables. Colour and shape were the prevalent variables being present in all the symbols. The language uses textual differentiation, each symbol has a label defined by the user with the exception of the End symbol. In this case the label is defined on the symbol's behaviour (End versus Terminate). The language does not have any mechanisms for reduntant coding.

5.3.3 Semantic Transparency

There are two ways of defining a language's transparency: ask a set of novices to match a symbol to its semantic construct or by iterating each of the constructs and decide on its transparency. The first method is used in 6.1.6 as part of the production of a new set of symbols. However, by simply analysing the symbol we can also reach some conclusions. Below are the symbols that we consider not to be transparent:

- **Conditional Start**. The lighting symbol is often used to represent events, which fits the conditional element. However, the symbol is in no way related to "starting". This is mitigated by sharing a colour visual variable with the Start symbol but can still be considered semantically opaque;

- **Call Subprocess**. This shape is in no way related to the notion of "Process".
- **Automatic Activity**. This shape is in no way related to the notion of "Automatic Activity".

5.3.4 Complexity Management

BPT has two mechanics for dealing with complexity: the "Call subprocess" construct and the "Automatic Activity". The subprocess construct calls a another process and then 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 mechanic for managing complexity, since it promotes reusable code and allows for several hierarchical levels. The Automatic Activity can be improved by allowing re-usability.

5.3.5 Cognitive Integration

A language should have mechanisms to help users interpret diagrams. There are two ways to achieve this:

- **Perceptual integration.** All BPT processes require a name which helps the interpretation of the diagram. Subprocess nodes have a label with the process being called and to view the process the user simply opens the subprocess node. Since BPT is always contained within ServiceStudio, there is no need for page numbers;
- **Conceptual integration.** Can be achieved by providing the user a summary or some context. This to remove the need to try to interpret the diagram to understand its use. BPT does not have any mechanism that help with conceptual integration.

5.3.6 Visual Expressiveness

Visual expressiveness is related to the number of visual variables the language uses. An analysis of BPT's syntax we conclude that BPT's visual expressiveness is 3 ((x,y), colour and shape). This value may seem low but it is enough, colour is a very strong visual variable and considering the language is always used within the Service Studio IDE there is no need to consider use cases where colour is removed due to printing in black and white.

5.3.7 Dual Coding

According to this principle text can be used to complement graphics to help interpretation. However, there are several symbols in BPT that do not make sense without text.

In Figure 5.1 it is clear that without the text it would be impossible to know what Automatic Activity, Human Activity, Email or Decision are doing. After analysing BPT's

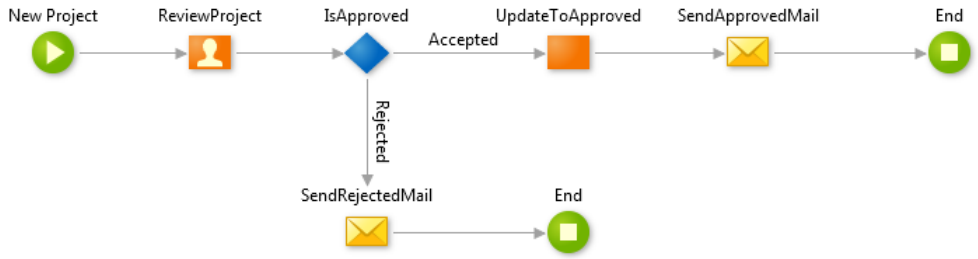


Figure 5.1: Example of dual coding in BPT.

symbols we concluded that only the End symbol does not require text to be easily interpreted.

BPT also has a text annotation construct. This allows users to add non-mandatory text to the process.

5.3.8 Graphic Economy

Physics of Notations recommends only having six different categories and BPT has nine different symbols. However, this is not an issue since the language is always used within ServiceStudio which has a sidebar with all symbols present. This removes the need for users to memorise the symbols.

5.3.9 Cognitive Fit

BPT's notation is small and simple to understand. Having separate notations for experts and novices does not make much sense. The notation is not easy to sketch since the language was designed to be used within ServiceStudio.

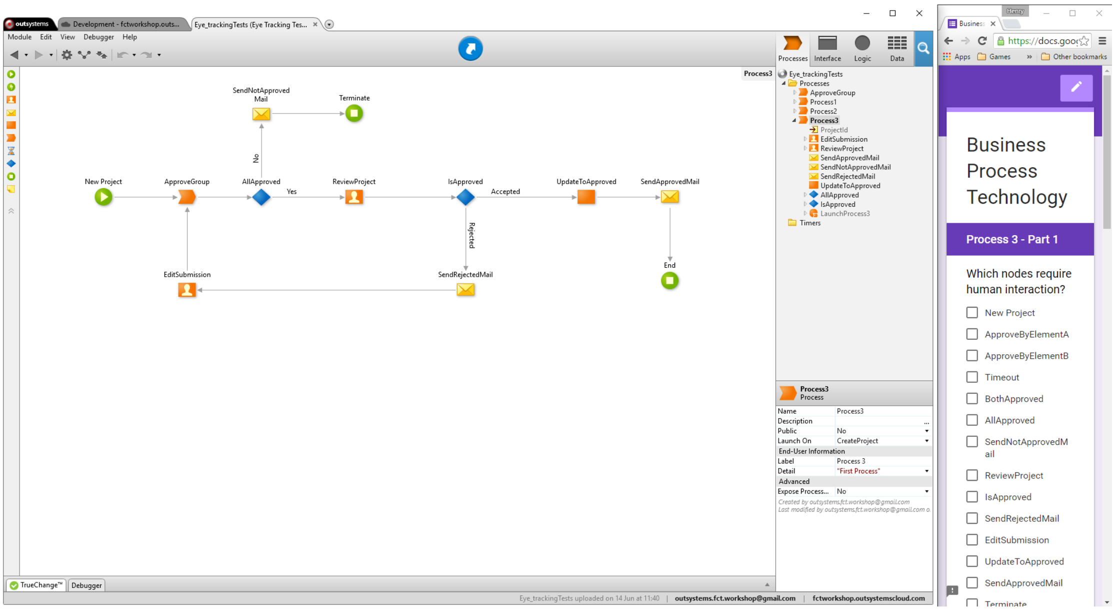
5.4 Usability experiments

One-on-one experiments were performed with inexperienced users. These were done with the help of an eye-tacker, the software Ogama [60] and the developed UsabilityTool. The objective was to determine not only syntax issues but also semantic ones.

5.4.1 Experiment protocol

The experiment was set up on a single screen computer with the eye tracker under the screen. Open on the screen was ServiceStudio with the process that was being tested open and on the side a set of questions related to that specific process (Figure 5.2). The testers were all from the Computer Science area and experienced with the OutSystems Platform but not with BPT.

The first step of the usability experiment was calibrating the eye-tracker using Ogama. Once that was done the user started answering the questions. Three processes were used



Process3
Process

Name	Process3
Description	...
Public	No
Launch On	CreateProject
End-User Information	
Label	Process 3
Detail	"First Process"
Advanced	
Expose Process...	No

Created by outsystems.fct.workshop@gmail.com
Last modified by outsystems.fct.workshop@gmail.com o

Business Process Technology

Process 3 - Part 1

Which nodes require human interaction?

□ New Project
□ ApproveByElementA
□ ApproveByElementB
□ Timeout
□ BothApproved
□ AllApproved
□ SendNotApprovedMail
□ ReviewProject
□ IsApproved
□ SendRejectedMail
□ EditSubmission
□ UpdateToApproved
□ SendApprovedMail
□ Terminate

Figure 5.2: Example of Usability Experiment screen.

with varying levels of complexity and covered all the language constructs. After the completion of the questions relating to the three process, the user answered a SUS and TLX questionnaire on the UsabilityTool.

At the end of the experiment, there was a small discussion where the user would talk about the issues they had with the language and also gave ideas for possible solutions.

5.4.2 Experiment analysis

The results of the analysis was mostly a manual process. For each of the experiments the answers were checked and if a user answered incorrectly then the eye-tracking recording was used to understand where the user was looking and what caused the error.

A list of issues was compiled and for each a possible solution. Most of the issues were related to confusion created by the syntax. Most notably were:

- **Parallelism**. None of the users was able to understand if a node with more than one incoming flow would wait for all flows and then merge, or would just continue as a parallel flow;
- **End and Terminate**. A common issue encountered was either not noticing that the End and Terminate were not the same, or not understanding the behavioural difference between the two;
- **Decision**. A minority of the users did not notice that the Decision node could be expanded and as such did not understand its semantic behaviour and thought it was a simple *If* node;

- **Automatic Activity**. Most the users expressed confusion about the existence of this node and defended that it should be an *Action*. This happens because Automatic Activity behaves the same way, but without the benefit of being reusable;

- **Start**. All the users checked the Start node properties for information relating to what causes the process to start. However, that information is present in the process properties and not in the Start properties;

- **Call subprocess**. Approximately half the users defended that the process would be much easier to interpret if the subprocess node had information about the process being called. Some recommend a preview of the process being called when moving the mouse over the node;

- **Human Activity**. All the users expressed confusion when trying to identify who were the actors (end-users) that interacted with process. Most defend that there should be a way to identify the target of a Human Activity simply by looking at the process and not having to search the properties.

Chapter 6

New version of BPT

The first step in creating a new version of BPT was to define its new concrete syntax. The next step was the implementation of the new syntax which was followed by usability tests to check if the changes were indeed an improvement over the original syntax.

6.1 Sign Production

During the analysis of the current version of BPT it was concluded that one of the areas that could be improved was its concrete syntax. The goal was to create a new set of symbols that had a one to one relation between the concrete 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.* [10].

The performed research consists of five interrelated studies (three of which are experiments involving novices). Figure 6.1 summarises the research steps and shows how the different experiments are related.

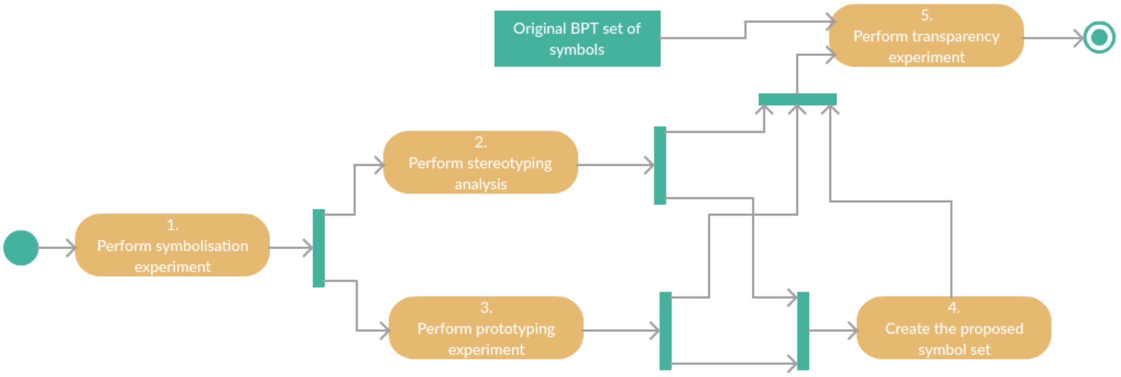

Original BPT set of symbols
5. Perform transparency experiment
2. Perform stereotyping analysis
1. Perform symbolisation experiment
3. Perform prototyping experiment
4. Create the proposed symbol set

Figure 6.1: Research steps.

1. **Symbolisation experiment:** novices are asked to draw a set of symbols that they think best represents the language constructs;
2. **Stereotyping analysis:** a set of symbols is built based on the most common symbols drawn by the novices;
3. **Prototyping experiment:** a new group of novices are asked to identify the best symbol for each construct;
4. **Proposed symbol set:** a set of symbols is built taking into account the results from the stereotyping and prototyping experiment, the interviews with users and the eye-tracking usability tests;
5. **Semantic transparency experiment:** novices are asked to infer the meaning of each symbol. This is done for original BPT symbols and all the previous sets generated from this experiment.

The participants used in each of the experiments are exclusive to that experiment. This removed the possibility unintentional bias.

6.1.1 Symbolisation experiment

Semantic transparency is achieved when users can infer the meaning of a symbol, so it is acceptable to conclude that the best 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 [30] where novices were asked to draw symbols that best represents each of BPT's semantic construct.

The questionnaire was printed out and participants were asked to answer them, taking in average 30 to 40 minutes. The participants were all from Computer Science and aged between 18 and 32 without prior experience with BPT. The sign production questionnaire contains (the full questionnaire can be found at Appendix A):

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

To process the questionnaires an application was developed that receives as an input the questionnaire in digital format, cuts the answers into separate images and saves them. To make processing even easier, the images are saved in two different directories: one where the images grouped in folders based on the questionnaire they belong to and another where they are grouped by semantic construct. Like this, it is possible to view all the answers of a specific questionnaire or view all the answers for one specific construct. The application also allows the user to upload the answers to the Usability Application.

The decision to do the questionnaire in paper was based on previous studies and because it required the least amount of upfront work. However, given the difficulty to find participants (due to the required time investment) and the extra work to process the data we concluded that it would beneficial to be able to send the participant a digital version of the questionnaire and have them draw the answers using a stylus. This conclusion was reached while processing the data and therefore was not used in this study, but for future studies it would be best if both paper and digital options were available.

6.1.2 Stereotyping analysis

The analysis of the drawings generated by the sign production technique was done using the judges' ranking method [32]. The symbols were first classified into categories based on their conceptual similarity. Then, the chosen symbol was the one that was most representative of the category with the most symbols.

As an 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 rest 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 done for each semantic construct, resulting in a set of 15 symbols, one for each construct (which can be found in Figure 6.2).

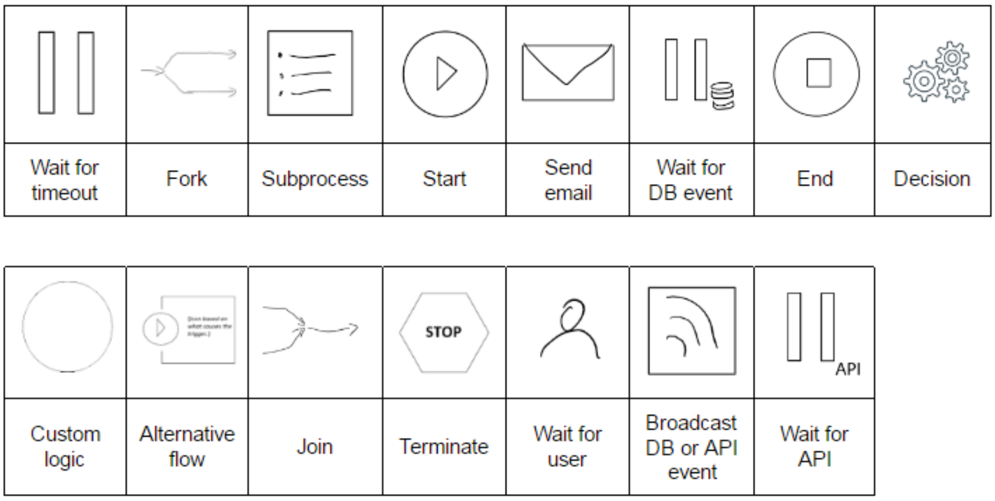


Figure 6.2: Set of stereotype symbols.

None of the symbols won with an absolute majority. The large variety of symbols, and in some cases the lack of answer, demonstrates the difficulty in creating a concrete representation for the constructs.

6.1.3 Prototyping experiment

To create a prototype set of symbols, a new questionnaire were created. This one still contains a question for each construct. Each question has a description of the semantic construct and a set of possible symbols for that construct. The possible answers are a symbol from each category previously defined in 6.1.2. Participants are asked to choose what they think is the best symbol to represent the description of the semantic construct. Two version of the questionnaire was created, one in paper (the full questionnaire can be found at Appendix B) and another using a digital third party platform for usability tests called UsabilityHub[1].

Processing the questionnaires was simplified due to the usage of a digital platform and most the answers were done digitally. UsabilityHub allows the data to exported to a spreadsheet. The latter was then completed with the answers from the paper version.

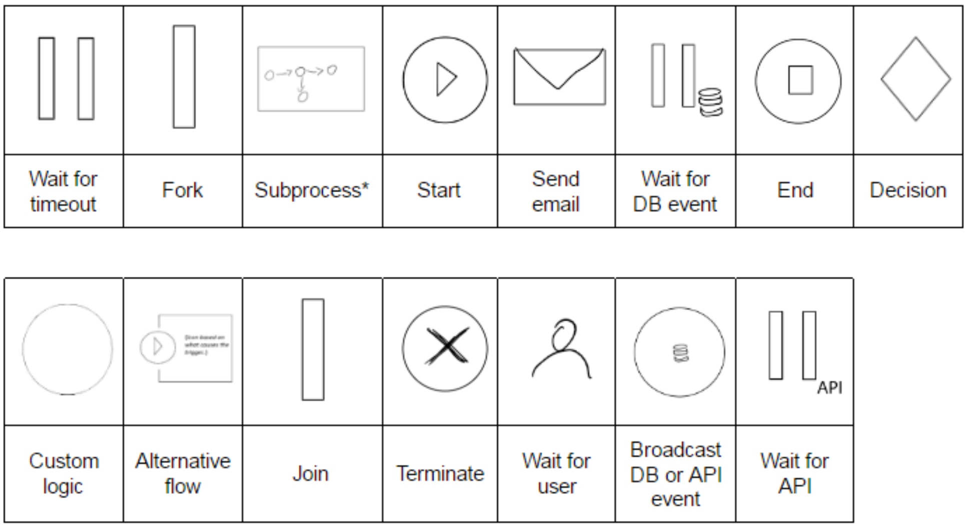


Figure 6.3: Set of prototype symbols.

The set of winning symbols can be seen in Figure 6.3. None of the symbols won with absolute majority and there are a few symbols that match the Stereotyping analysis and others that are radically different. Of note is the Subprocess construct, since this symbol has a dynamic effect as it is a real-time representation of the process being called which expands when the user mouses over it.

6.1.4 Original BPT

From a Physics of Notations point of view, the original set of symbols has several problems. Some of the symbols are semantically perverse, and there is more than one case of symbol overload and symbol deficit.

[1] http://usabilityhub.com

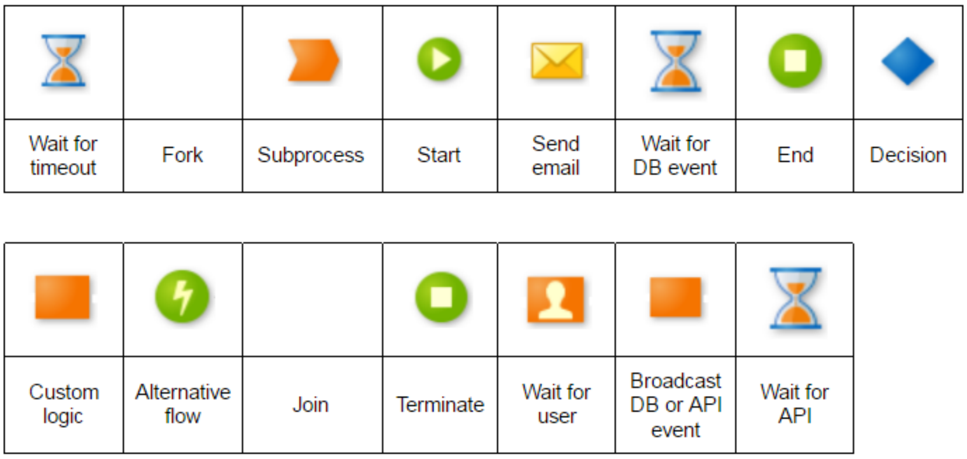


Figure 6.4: BPT's original set of symbols.

There are also some issues with platform consistency. The "Custom logic" and "Broadcast DB or API event" symbols do not match the rest of the platform. The same semantic constructs are represented by an orange ball. The full set of original symbols can be found in Figure 6.4.

6.1.5 Proposed symbol set

One of the issues with the stereotyping and prototyping experiments is that each symbol is generated independently, this means it does not consider the development environment or the need for consistency with the rest of the OutSystems platform. With this mind, a new set of symbols was created (Figure 6.5), this one takes into account consistency while trying to remain as close as possible to the set of prototype symbols.

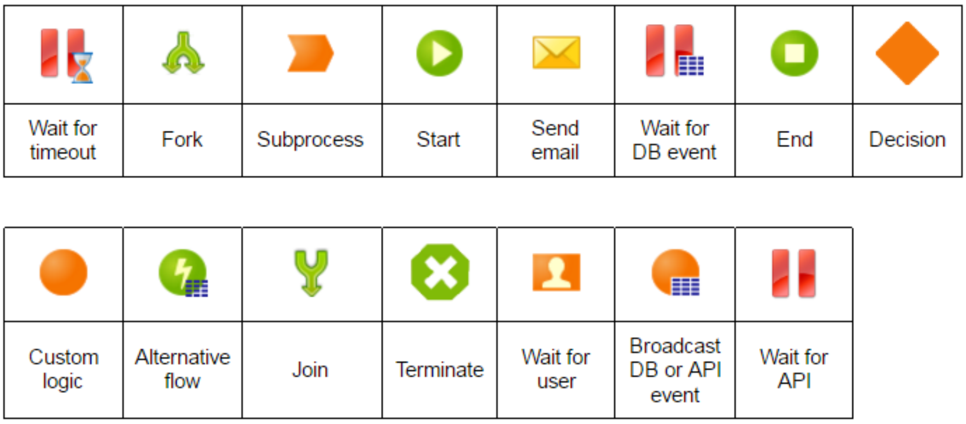


Figure 6.5: Set of proposed symbols.

The "End" and "Terminate" remain green (even though that could be considered semantically perverse), this was done because "End" is green on the rest of the platform.

Changing it on 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.

The bigger changes when compared to the prototype symbol set were to the database symbol, the timeout and API waits, the fork, the join and the alternative flow:

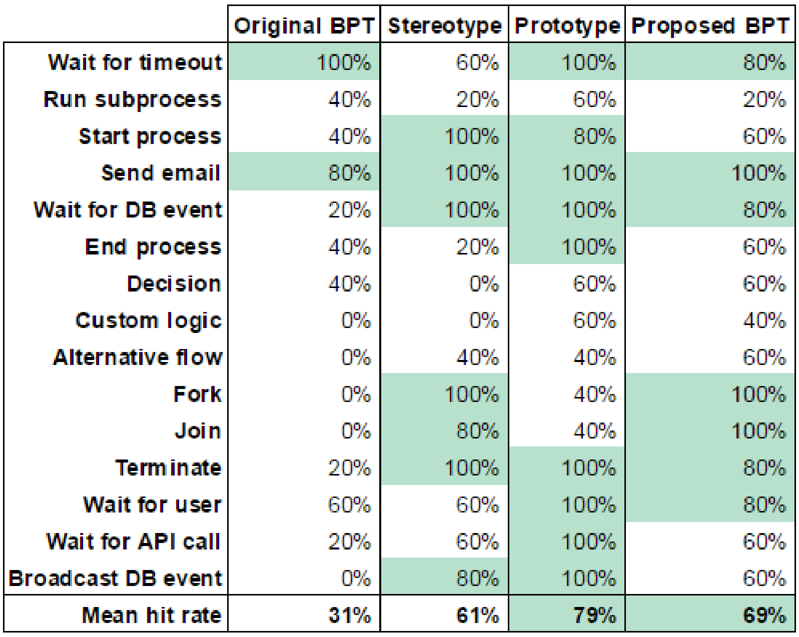
- **The database symbol:** In both the stereotype and prototype set of symbols a set of three cylinders is used to represent any activity that related to database. This makes sense since it is the most common representation for databases. However, Service Studio uses a different representation (a blue table). As such, all representations related to data were changed to maintain consistency;
- **Waits:** These were switched because waiting for an API call is unconditional pause. A timeout is a condition so 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 Service Studio are only used to 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 the arrows. The "bar" from the prototype symbol set was not used because semantically it is important to differentiate forks from joins. The symbol proposed came in second on the prototyping experiment;
- **Alternative flow:** the symbol in both stereotype and prototype were too similar to the "Start" symbol. As such, it was decided that it would be best to maintain the current symbol but add an overlay depending on what triggers the alternative flow.

6.1.6 Semantic transparency experiment

Comprehension tests [64] are commonly used to measure the symbol's transparency and is used for testing ISO standard symbols [10]. The comprehensibility survey has a question for each of the symbols in the set. Each question has one of the language's concrete symbols on top and then a list with all the language's semantic constructs. The tester then selects one or more constructs that, in their opinion, best matches the symbol. Comprehensibility is then measured by the symbol's hit rate. The testers were all Computer Science MSc students.

A survey was created for each of the different sets of symbols (a tester can only answer one of the surveys to prevent bias). The surveys were created in digital form, this allows the maximum number of novice participants to answer the survey. A web application was created that asks the tester to input their academic email, the email is checked and it is saved in the database if it has not been used yet. Then, the application randomly directs the tester to one of the four surveys. This ensures that a tester does no answer more than one survey.

The results shown in Figure 6.6 show that the Prototype set is the most semantically transparent set of symbols. This is the same conclusion Caire *et al.* [10] reached, however



	Original BPT	Stereotype	Prototype	Proposed BPT
Wait for timeout	100%	60%	100%	80%
Run subprocess	40%	20%	60%	20%
Start process	40%	100%	80%	60%
Send email	80%	100%	100%	100%
Wait for DB event	20%	100%	100%	80%
End process	40%	20%	100%	60%
Decision	40%	0%	60%	60%
Custom logic	0%	0%	60%	40%
Alternative flow	0%	40%	40%	60%
Fork	0%	100%	40%	100%
Join	0%	80%	40%	100%
Terminate	20%	100%	100%	80%
Wait for user	60%	60%	100%	80%
Wait for API call	20%	60%	100%	60%
Broadcast DB event	0%	80%	100%	60%
Mean hit rate	**31%**	**61%**	**79%**	**69%**

Figure 6.6: Semantic transparency results.

in this case there is not such a large discrepancy between the Stereotype and the Prototype sets.

Only the Prototype and Proposed meet the ISO threshold for symbol acceptance (67% [10]). The original set of symbols being much lower than the rest, this demonstrates the importance of involving the end-users in the construction of a language's concrete syntax. Also, there were several constructs with 0% transparency. These were the ones with 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 slight alterations in order to maintain consistency with the platform resulted in a minor decrease in transparency. But, it is still above the ISO threshold.

6.2 Modifying BPT

Some changes were direct (simply changing the icon) but others required changes to how the flows work. This section goes over each of the changes made to BPT and how they were implemented. Note that all these changes are a prototype and were made only to test the proposed syntax. As such, all changes are all client side and can not be compiled.

6.2.1 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 toolbar: Fork and Join.

The BPT language is defined by a meta-model which contains all the syntactical rules and constraints. This is then 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 given that the changes are just to prototype the syntax.

In order to view what causes a process to start you need to view the properties of the process. During the usability tests 100% of the participants first looked at the properties of the Start button for this information. 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, in these cases 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. The symbols present in the new toolbar can be seen in Figure 6.7.

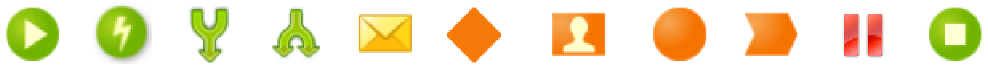
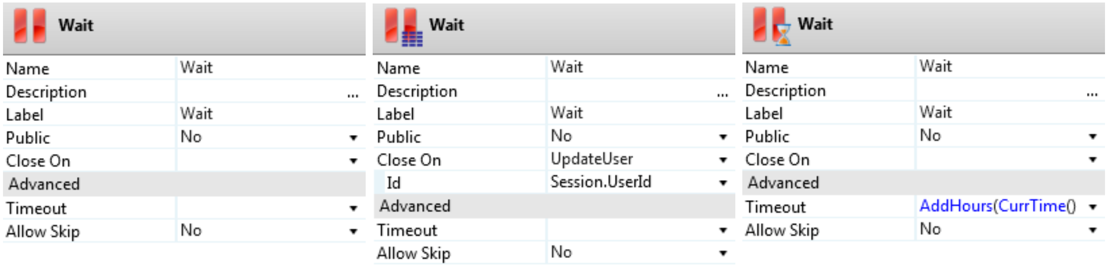


Figure 6.7: Updated toolbar.

Given that 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, changes were made to ensure that only the Decision (number of outgoing arrows is based on the condition) and Fork can have N outgoing arrows.

6.2.2 Symbol groups

Certain symbols were grouped together in order to reduce complexity. For each of these groups a symbol was chosen to represent the group and then it is updated with an overlay based on properties (Figure 6.8).



Wait

Name	Wait
Description	...
Label	Wait
Public	No ▾
Close On	▾
Advanced	
Timeout	▾
Allow Skip	No ▾

(a) API Call

Wait

Name	Wait
Description	...
Label	Wait
Public	No ▾
Close On	UpdateUser ▾
Id	Session.UserId ▾
Advanced	
Timeout	▾
Allow Skip	No ▾

(b) DB Event

Wait

Name	Wait
Description	...
Label	Wait
Public	No ▾
Close On	▾
Advanced	
Timeout	AddHours(CurrTime() ▾
Allow Skip	No ▾

(c) Timeout

Figure 6.8: Example of the *Wait* symbol group.

The following groups were created:

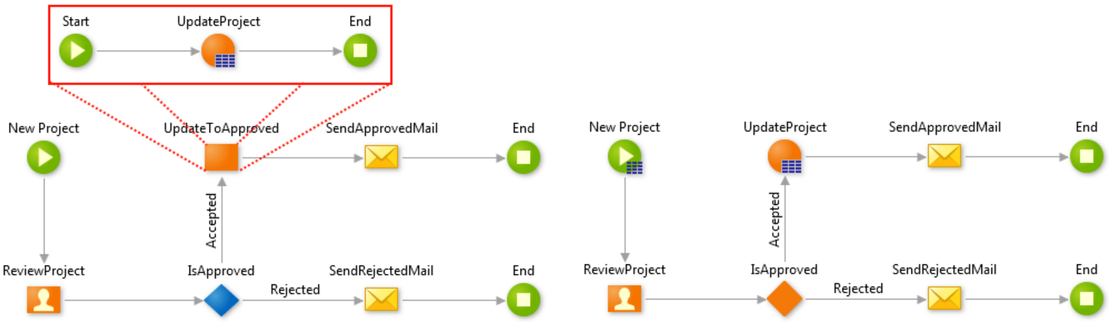
- **Waits.** The groups includes the *Wait for API call*, *Wait for DB event* and the *Wait for timeout* constructs. The symbol used to represent it is the *Wait for API call* symbols since it is the most generic of the three;
- **End and Terminate.** This group contains the *End* and *Terminate* symbols. 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 6.5, however if the process is launched via a DB event then the symbol is updated with a small overlay;
- **Conditional start.** The proposed symbol for the *Conditional Start* has an overlay representative of a DB event. However, 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.

With the aforementioned a stereotype was created. If an event symbol does not have an overlay then the event is triggered via an API call, otherwise it is triggered based on the symbol present in the overlay.

6.2.3 Using Actions in BPT

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

Originally in BPT all action calls and logic had to be encapsulated in an *Automatic Activity* (6.9(a)). These were not reusable and at times created unnecessary complexity.



(a) Automatic Activity

(b) Actions

Figure 6.9: Comparison of original Automatic Activity with new Actions.

Changes had to be made to the ServiceStudio syntax to allow Actions to be used in the BPT flow. With this, Actions have now replaced the *Automatic Activity* (6.9(b)) since they provide generalisation and re-usability while not creating unnecessary complexity.

Chapter 7

Results

The metrics used in the analysis were gathered using usability experiments presented in 4.1.4. The same set of questions was used for both languages (original and new BPT). The examples that were analysed by the testers to answer the questions were semantically the same but the syntax differed based on what version of BPT was being tested.

A total of 25 testers participated in the experiment. The testers were all Computer Scientists, 84% of them were male and 16% female, they were all aged between 23 and 40 years old, and none of them had any previous experience with BPT.

7.1 Usability experiment

Figure 7.1 contains a graphic overview of the results.

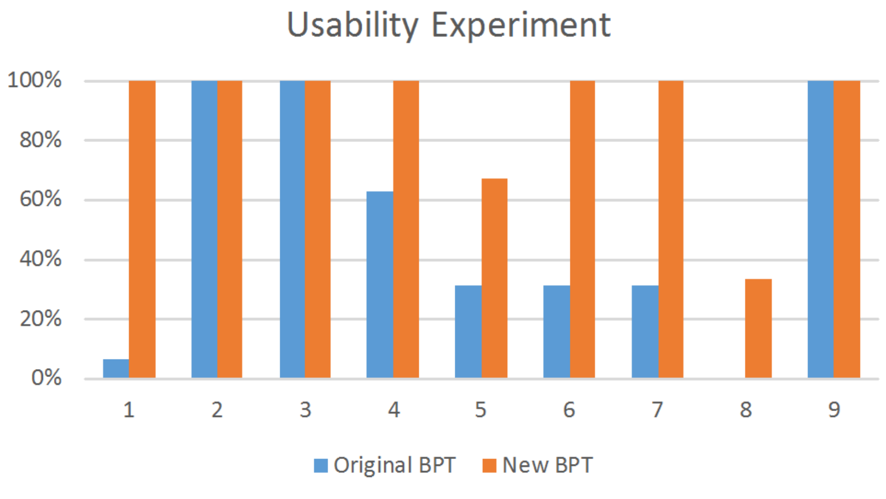


Figure 7.1: Graph of answer distribution

Each number present in Figure 7.1 corresponds to a question which can be found on table 7.1. Said table also includes a more detailed overview of the results and comments about the New BPT tests.

Table 7.1: Results of usability tests

	Correct Answer Rate (%)		
Question	Original BPT	New BPT	New BPT Comments
1. What causes the process to start.	6.25	100	The participants did not have any trouble finding 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	As with the previous experiment, once the testers were given information about the actors it was easy to match them. This is due to the intuitive labels on the nodes. If the labels were not present the results would be much worse.

7.2 SUS and TLX

To determine the language's usability we analysed the SUS and TLX scores gathered from usability experiments. By analysing these two scores we can determine whether the difference between the original and the new BPT language are significant and relevant. As a reminder, a higher SUS score is better while a lower TLX score is better.

7.2.1 Descriptive statistics

Table 7.2 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).

Table 7.2: Descriptive statistics

	Language	N	Mean	Std. Deviation	Skewness	Kurtosis	Shapiro-Wilk
SUS	BPT	16	42.25	11.72	-0.021	-0.67	0.896
	New BPT	9	64.78	10.02	0.213	-1.05	0.697
TLX	BPT	16	36.5	14.47	0.421	-0.61	0.686
	New BPT	9	20.78	8.72	-0.118	-1.81	0.245

7.2.2 Hypotheses testing

Welch's t-test was used instead of the Student's t-test for testing the difference in SUS and TLX scores between the current and the new versions of BPT. This happens because it has been shown that the Welch's t-test is better suited for studies with different sample sizes [33].

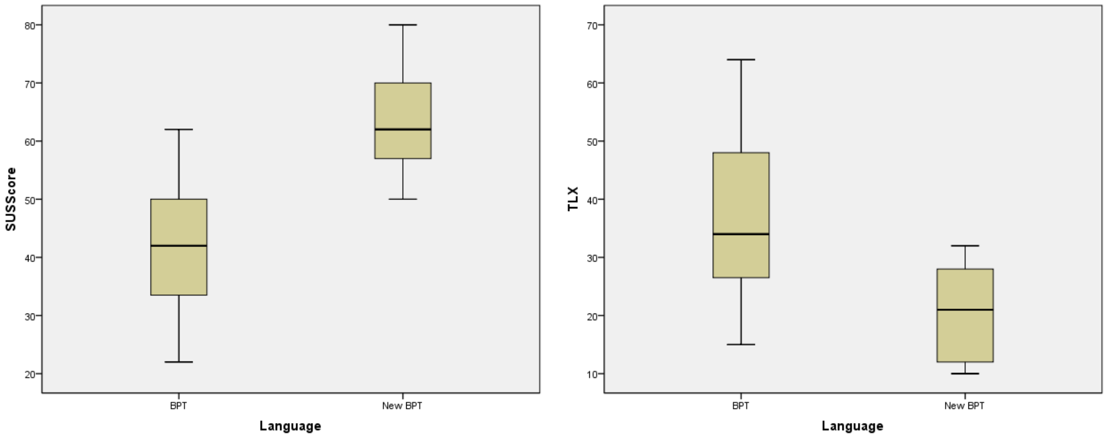
Table 7.3 contains: the means for both SUS and TLX; the difference between the original and new BPT means; the 95% confidence interval of the difference; *t*, *df* and p-values.

Table 7.3: Welch's t-test scores

	BPT mean	New BPT mean	Difference	95% Dif. CI Lower	95% Dif. CI Upper	t	df	p-value
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

We hypothesised that the new version of BPT would have a higher usability rating when compared to the original version. The SUS and TLX scores differed significantly according to Welch's t-test, $t_{SUS}(-5.07) = 19.01$, $p_{SUS} = .000$ and $t_{TLX}(3.39) = 22.80$, $p_{TLX} = .003$. On average, users who tested the original version of BPT gave it a SUS score of 42.25 and a TLX score of 36.50, while users who tested the new version of BPT gave it a SUS score of 64.78 and a TLX score of 20.78. The 95% confidence interval of the difference for the effect of the new BPT on the SUS score is between -31.83 and -13.22 and on the TLX score it is between 6.12 and 25.32. As such, these results support our hypothesis.

The above is further show in Figure 7.2 which contains a graphical display of the SUS and TLX scores.



(a) SUS Score

(b) TLX

Figure 7.2: Box plot of distribution

7.3 Discussion of results and implications

Looking at the Semantic transparency results (Figure 6.6) it is clear that symbols created by the users are far more transparent than those created by language engineers. This is consistent with Caire *et al.*'s findings [10]. This is further demonstrated in our analysis of diagrams through usability experiments (Table 7.1).

The results of the usability experiments also illustrates the importance of having a one-to-one relationship between the concrete and semantic constructs. This corroborates Moody's Physics of Notations [44].

When analysing the SUS and TLX scores (Table 7.3 and Figure 7.2) the TLX score decreases as the SUS score increases. This seems to suggest that there is a direct relationship between the language's usability and the perceived workload.

As seen in subsection 7.2.2 the results support our hypothesis. As such, we conclude that the dissertations objectives were achieved.

7.4 Threats to validity

Of the threats presented by Wohlin *et. al.* in [63], the only concerning one was the one related to the population selection. When selecting testers it is important that they provide a representative sample of the population. Unfortunately, due to resource constrains, all the testers used in the usability experiments were members of OutSystems (not part of the BPT development team). Ideally there would be representatives of OutSystems developers and business managers. This would allow us to measure the usability from a developer's perspective and also the usability from the perspective of people who simply interpret the process.

Chapter 8

Conclusions and future work

This dissertation had two primary objectives: to create a systematic process for evaluating language's usability, and to identify and fix any usability issues with a domain specific language from OutSystems called Business Process Technology.

We first started with the evaluation process since it would later be used to analyse BPT. The process begins with evaluators collecting and analysing normal everyday usage data, performing interviews with language users and performing a Physics of Notations analysis. Once those steps have been completed, the evaluator should conduct usability experiments where the testers are asked to interpret a solution developed using the language being evaluated and are asked a series of questions related to the example. With all the collected data the evaluator should then decide whether the language needs to be improved. If it doesn't then the process ends. Otherwise, the evaluator should then use the Sign Production Technique to develop a new syntax. The process should then restart using the new concrete syntax.

With the process complete, we then applied it to BPT. A few issues were identified (both with the concrete and semantic constructs) and a new notation was created. Changes were made to BPT within ServiceStudio (OutSystem's development environment) and the process was applied once again to BPT using the new notation. Performing a comparison analysis between the original and new notation showed a significant increase in usability.

The comparison analysis between and original and new version BPT confirmed that the process is effective and that the new notation has a higher usability rating. But, other conclusions were also made: 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 [10]); and it is extremely important to have a one-to-one relationship between the concrete syntax and semantic constructs.

8.1 Contributions

The main contribution of this dissertation was the process for evaluating languages. The proposed process guides language engineers through the evaluation of the language, helps them create a new version of the language's concrete syntax, which is then compared to the previous version. This process is repeated until the comparison analysis shows an increase in usability. A companion application was created that can be used to organise the data, while following the proposed process.

Another contribution was a report of usability issues with BPT and a new syntax for the language, that has a much higher semantic transparency rating. These were achieved by applying the proposed process to BPT, which also serves to demonstrate the effectiveness of the proposed process. The new syntax created for BPT using the proposed process was implemented on ServiceStudio.

8.2 Future work

This dissertation had two main objectives: the evaluation process and the improvements to BPT. Even though both objectives were achieved, they can still be improved. The evaluation process can be expanded with further techniques and modified to also be applicable to textual languages. BPT still has some issues with its semantic constructs which should be studied and resolved.

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 provides 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 direct relation between a languages system usability score and testers task load index. This should be further researched.

Lastly, more testing should also be done on the proposed process. The process and its techniques should be applied to other visual languages.

Bibliography

[1] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, et al. *Business process execution language for web services*. Tech. rep. IBM, 2003. URL: http://www-106.ibm.com/developerworks/library/ws-bpel/.

[2] J. Arlow and I. Neustadt. *UML 2 and the unified process: practical object-oriented analysis and design*. Pearson Education, 2005.

[3] A. Bangor, P. T. Kortum, and J. T. Miller. "An empirical evaluation of the system usability scale". In: vol. 24. 6. Taylor & Francis, 2008, pp. 574–594.

[4] A. Barišić, V. Amaral, M. Goulão, and B. Barroca. "Evaluating the usability of domain-specific languages". In: ed: IGI Global, 2012.

[5] J. Bertin. *Semiology of graphics: diagrams, networks, maps*. Esri Press, 1983.

[6] A. F. Blackwell. "Dealing with new cognitive dimensions". In: *Workshop on Cognitive Dimensions: Strengthening the Cognitive Dimensions Research Community., University of Hertfordshire*. 2000.

[7] W. Brauer, W. Reisig, and G. Rozenberg, eds. *Petri Nets: Applications and Relationships to Other Models of Concurrency*. Springer Berlin Heidelberg, 1987. DOI: 10.1007/3-540-17906-2. URL: http://dx.doi.org/10.1007/3-540-17906-2.

[8] J. Brooke. "SUS-A quick and dirty usability scale". In: Taylor & Francis, 1996, pp. 189–194.

[9] E. A. Bustamante and R. D. Spain. "Measurement invariance of the Nasa TLX". In: *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*. Vol. 52. 19. SAGE Publications. 2008, pp. 1522–1526.

[10] P. Caire, N. Genon, P. Heymans, and D. L. Moody. "Visual notation design 2.0: Towards user comprehensible requirements engineering notations". In: *Requirements Engineering Conference (RE), 2013 21st IEEE International*. IEEE. 2013, pp. 115–124.

[11] B. Cassin, S. Solomon, and M. L. Rubin. *Dictionary of eye terminology*. Triad Pub. Co., 1984.

[12] R. Dawson. *Secrets of power problem solving*. Open Road Media, 2012.

[13] S. Debois, T. Hildebrandt, M. Marquard, and T. Slaats. "Hybrid Process Technologies in the Financial Sector". In: CEUR, 2015. URL: http://ceur-ws.org/Vol-1439/paper9.pdf.

[14] M. Dumas and A. H. Ter Hofstede. "UML activity diagrams as a workflow specification language". In: *UML 2001—The Unified Modeling Language. Modeling Languages, Concepts, and Tools*. Springer, 2001, pp. 76–90.

[15] S. Eric. *Social modeling for requirements engineering*. Mit Press, 2011.

[16] K. Figl, J. Mendling, and M. Strembeck. "Towards a Usability Assessment of Process Modeling Languages". In: *Proc. of EPK 2009*. 2009, pp. 118–136.

[17] N. Genon, P. Heymans, D. L. Moody, and D. Amyot. *BPMN 2.0 Process Models: Analysis according to the "Physics" of Notations Principles*. Tech. rep. PReCISE - University of Namur, 2010. URL: http://www.info.fundp.ac.be/~nge/BPMN/BPMN2_PoN_Analysis.pdf.

[18] N. Genon, P. Heymans, and D. Amyot. "Analysing the cognitive effectiveness of the BPMN 2.0 visual notation". In: Springer Berlin Heidelberg, 2011, pp. 377–396.

[19] N. Goodman. *Languages of art: An approach to a theory of symbols*. Hackett publishing, 1968.

[20] T. R. G. Green and M. Petre. "Usability analysis of visual programming environments: a 'cognitive dimensions' framework". In: vol. 7. 2. Academic Press, 1996, pp. 131–174.

[21] T. R. Green. "Instructions and descriptions: some cognitive aspects of programming and similar activities". In: *Proceedings of the working conference on Advanced visual interfaces*. ACM. 2000, pp. 21–28.

[22] T. R. Green and A. Blackwell. "Cognitive dimensions of information artefacts: a tutorial". In: *BCS HCI Conference*. Vol. 98. 1998.

[23] H. P. R. Group. *Nasa Task Load Index (TLX) v. 1.0 Manual*. NASA Ames Research Center, 1986.

[24] B. Grácio. "Agregado: Compilar Sistemas NoSQL na Plataforma OutSystems". MA thesis. Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, 2015.

[25] T. Gschwind, F. Paoli, V. Gruhn, and M. Book. *Software Composition*. Springer, 2012.

[26] S. G. Hart. "NASA-task load index (NASA-TLX); 20 years later". In: *Proceedings of the human factors and ergonomics society annual meeting*. Vol. 50. 9. Sage Publications. 2006, pp. 904–908.

[27] S. G. Hart and L. E. Staveland. "Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research". In: vol. 52. Elsevier, 1988, pp. 139–183.

[28] R. Hat. *JBoss Developer*. 2016. URL: http://www.jboss.org/ (visited on Aug. 11, 2016).

[29] S. Hitchman. "The details of conceptual modelling notations are important-a comparison of relationship normative language". In: vol. 9. 1. AIS, 2002, p. 10.

[30] W. C. Howell and A. H. Fuchs. "Population stereotypy in code design". In: vol. 3. 3. Elsevier, 1968, pp. 310–339.

[31] E. B. Huey. *The psychology and pedagogy of reading*. The Macmillan Company, 1908.

[32] S. Jones. "Stereotypy in pictograms of abstract concepts". In: vol. 26. 6. Taylor & Francis, 1983, pp. 605–611.

[33] B. Kitchenham, L. Madeyski, P. Brereton, S. Charters, S. Gibbs, and A. Pohthong. "Robust Statistical Methods for Empirical Software Engineering". In: Springer, 2016. DOI: 10.1007/s10664-016-9437-5.

[34] S. Krug. *Rocket surgery made easy*. Uitgeverij Thema, 2011.

[35] W. Lidwell, K. Holden, and J. Butler. *Universal principles of design, revised and updated: 125 ways to enhance usability, influence perception, increase appeal, make better design decisions, and teach through design*. Rockport Pub, 2010.

[36] A. Lima. *OutSystems Platform - Architecture and Infrastructure Overview*. Tech. rep. OutSystems, 2013. URL: https://www.outsystems.com/home/document-download/178/8/0/0.

[37] Y. Liu and C. D. Wickens. "Mental workload and cognitive task automaticity: an evaluation of subjective and time estimation metrics". In: vol. 37. 11. Taylor & Francis Group, 1994, pp. 1843–1854.

[38] K. Lynch. *The image of the city*. MIT press, 1960.

[39] Microsoft. *ASP.NET*. 2016. URL: http://www.asp.net/ (visited on Aug. 11, 2016).

[40] Microsoft. *ISS*. 2016. URL: https://www.iis.net/ (visited on Aug. 11, 2016).

[41] Microsoft. *SQL Server*. 2016. URL: http://www.microsoft.com/en-us/server-cloud/products/sql-server-2016/ (visited on Aug. 11, 2016).

[42] Microsoft. *Windows Server 2016*. 2016. URL: http://www.microsoft.com/en-us/server-cloud/products/windows-server-2016/ (visited on Aug. 11, 2016).

[43] D. Moody. "Theory development in visual language research: Beyond the cognitive dimensions of notations". In: *Visual Languages and Human-Centric Computing, 2009. VL/HCC 2009. IEEE Symposium on*. IEEE. 2009, pp. 151–154.

[44] D. L. Moody. "The "physics" of notations: toward a scientific basis for constructing visual notations in software engineering". In: vol. 35. 6. IEEE, 2009, pp. 756–779.

[45] D. L. Moody, P. Heymans, and R. Matulevičius. "Visual syntax does matter: improving the cognitive effectiveness of the i* visual notation". In: vol. 15. 2. Springer-Verlag, 2010, pp. 141–175.

[46] T. Murata. “Petri nets: Properties, analysis and applications”. In: vol. 77. 4. IEEE, 1989, pp. 541–580.

[47] E. Murphy-Hill, S. Markstrum, and C. Anslow. “Evaluation and usability of programming languages and tools (PLATEAU)”. In: *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*. ACM. 2010, pp. 265–266.

[48] J. Nielsen. *Usability engineering*. Elsevier, 1994.

[49] Oracle. *Oracle WebLogic Server*. 2015. URL: http://www.oracle.com/technetwork/middleware/weblogic/overview/index-085209.html (visited on Aug. 11, 2016).

[50] Oracle. *MySQL*. 2016. URL: https://www.mysql.com/ (visited on Aug. 11, 2016).

[51] Oracle. *Oracle*. 2016. URL: http://www.oracle.com/index.html (visited on Aug. 11, 2016).

[52] OutSystems. *OutByNumbers - Benchmark Overview Report*. Tech. rep. OutSystems, 2013. URL: http://www.outsystems.com/res/OutbyNumbers-DataSheet.

[53] H. Plattner, C. Meinel, and L. Leifer. *Design thinking: Understand–improve–apply*. Springer Science & Business Media, 2010.

[54] F. Puhlmann. *BPMN 2.0 Wimmelbild*. 2009. URL: http://frapu.de/blog/comments.php?y=09&m=07&entry=entry090701-211320 (visited on Jan. 19, 2016).

[55] W. Reisig. *Petri nets: an introduction*. Vol. 4. Springer Science & Business Media, 2012.

[56] S. Rubio, E. Díaz, J. Martín, and J. M. Puente. “Evaluation of subjective mental workload: A comparison of SWAT, NASA-TLX, and workload profile methods”. In: vol. 53. 1. Blackwell Publishing Ltd, 2004, pp. 61–86.

[57] Z. Sharafi, Z. Soh, and Y.-G. Guéhéneuc. “A systematic literature review on the usage of eye-tracking in software engineering”. In: vol. 67. Elsevier, 2015, pp. 79–107.

[58] A. Smith and A. S. Skinner. *The wealth of nations*. Vol. 3. World Scientific, 1991.

[59] I. Vessey. “Cognitive fit: A theory-based analysis of the graphs versus tables literature”. In: vol. 22. 2. Wiley Online Library, 1991, pp. 219–240.

[60] A. Voßkühler. “OGAMA description (for Version 2.5)”. In: 2009.

[61] B. Weber, W. Wild, and R. Breu. “CBRFlow: Enabling adaptive workflow management through conversational case-based reasoning”. In: *Advances in Case-Based Reasoning*. Springer, 2004, pp. 434–448.

[62] S. A. White. *BPMN modeling and reference guide: understanding and using BPMN*. Future Strategies Inc., 2008.

[63] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[64] H. Zwaga and T Boersema. "Evaluation of a set of graphic symbols". In: vol. 14. 1. Elsevier, 1983, pp. 43–54.

Appendix A

Sign Production Technique questionnaire

Presented here is the Sign Production Technique questionnaire. The presented version is meant for novices who have not had any contact with the BPT language.

Survey on OutSystems BPT notations

Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia
Departamento de Informática
NOVA LINCS
Contact for this survey: Henrique Henriques (h.henriques@campus.fct.unl.pt)

Part 1 - Visual representations

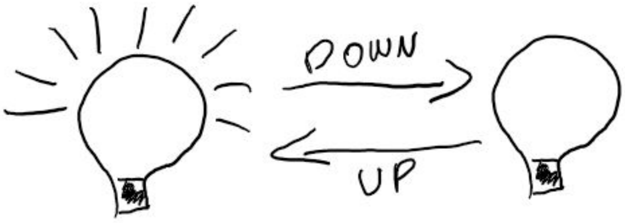
For each of these concepts, we provide you with space so that you can draw the visual representations (not necessarily a single icon) for the BPT (Business Process Technology) language in OutSystems. Please draw a representation which, in your opinion, best fits the description of the concept.

Notes:

- **Participation is voluntary and anonymous**. Although there is no fixed time limit for completing this questionnaire, we estimate that, overall, it should not exceed around 30 minutes. Thank you very much for your kind participation.
- There are no right or wrong answers, so you're invited to use your best judgment to visually represent those concepts, as best as you can.

Example from an unrelated language: Represent a light bulb switching state (on and off) based on an event.

Your suggestion:



1. Pauses the flow waiting for a timeout.

Your suggestion:

2. Represent parallelism. The process execution path divides into two parallel paths.

Your suggestion:

3. Runs a subprocess. The parent process only proceeds after all flows of the subprocess finish.

Your suggestion:

4. Initiates the process.

Your suggestion:

5. Sends a specified email message in the process flow.

Your suggestion:

6. Pauses the execution waiting for a DB event

Your suggestion:

7. Finishes a process flow. The process may continue in parallel flows.

Your suggestion:

8. Directs the execution path to a specific activity based on custom logic.

Your suggestion:

9. Performs custom logic in the application or in external systems.

Your suggestion:

10. Initiates an alternative flow and may be triggered by a DB event or an explicit API call.

Your suggestion:

11. Represent a fork and join: The main execution path is divided into two parallel paths and *after each path has finished their task*, they merge back into the main execution.

Your suggestion:

12. Terminate all the flows of the a process.

Your suggestion:

13. Waits for a user, or group, to complete the given task.

Your suggestion:

14. Used to broadcast an event (via DB) or call an API to deliver an event to a specific activity or process.

Your suggestion:

15. Pauses the execution path and resumes when a specific API action is called.

Your suggestion:

Part 2 - Background information questionnaire

Please fill in this background questionnaire:

1. Age: ____
2. Gender: Female [ ] / Male [ ]
3. Nationality:______________________________
4. Completed academic degrees: BSc [ ] / MSc [ ] / PhD [ ]
5. Degree in: Informatics [ ] / Other area [ ]; which area? _______________________________

6. Are you familiar with other process modeling techniques/languages? Yes [ ] / No [ ]
7. If you are, which are you familiar with?
   [ ] BPMN
   [ ] OutSystems BPT
   [ ] Others: ____

8. Have you ever used any process modeling language? Yes [ ] / No [ ]
9. If you have, which process modeling languages have you used before?
   [ ] BPMN
   [ ] OutSystems BPT
   [ ] Others: ____

10. How often do you use process modeling languages in your professional activities (please choose the closest match)?
    [ ] Every day
    [ ] once per week
    [ ] once per month
    [ ] once per year
    [ ] occasionally (but less than once per year)
    [ ] never

11. If you have used process modeling languages, please state in which context have you used them (please choose all that apply):
    [ ] educational
    [ ] industry
    [ ] research
    [ ] entertainment

Appendix B

Prototype symbol set questionnaire

This is the questionnaire used to create the Prototype version of the concrete syntax. Appended here is the paper version, there is a second version (digital) created using the usability testing platform called *UsabilityHub*.

Prototype Symbol Set Questionnaire

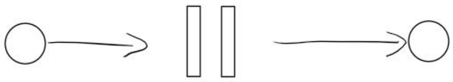
The following images are possible candidates for a specific semantic construct of a visual programming language. Please read the definition of the construct and choose the notation that you think best represents it.

Some of the representations have text in round brackets, this is a note with information to help understand the representation. Text in curly brackets is part of the representation and that text changes based on a variable (Out of context example: "{current date}").

When choosing a representation please ignore its quality, these are quick drafts. To answer just input the number present in the left corner of the visual representations.

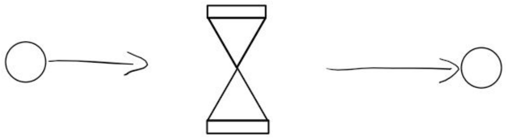
1. Pauses the flow waiting for a timeout.

1



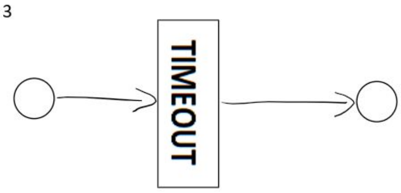
(Media pause button.)

2

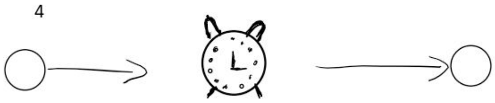


(Hourglass)

3



4

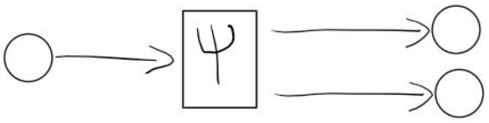


(Alarm clock.)

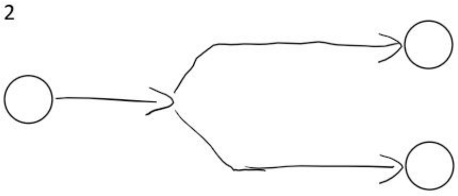
Answer: _______

2. Represent parallelism. The process execution path divides into two parallel paths.

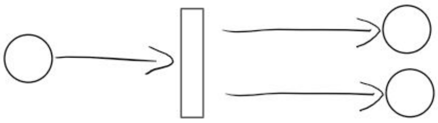
1



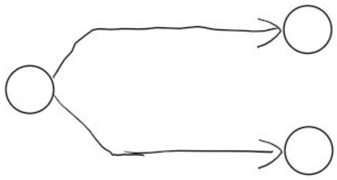
2



3



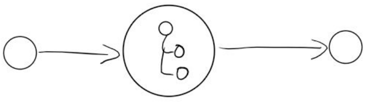
4



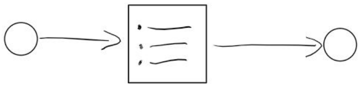
Answer: _______

3. Runs a subprocess. The parent process only proceeds after all flows of the subprocess finish.

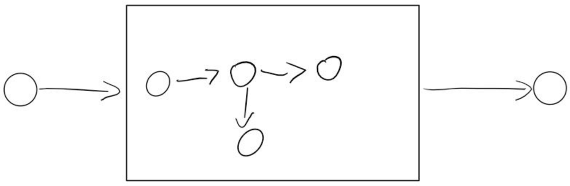
1



2

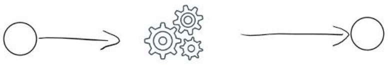


3



(Inline preview of the logic. Expanded when mouse hover.)

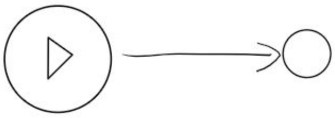
4



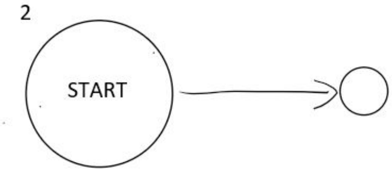
Answer: _______

4. Initiates the process.

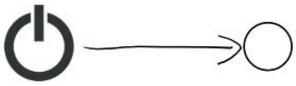
1



2

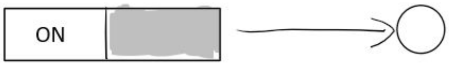


3



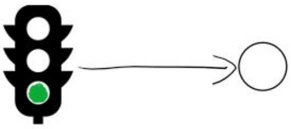
(Green power button.)

4



(On switch.)

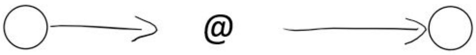
5



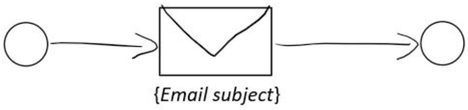
Answer: _______

5. Sends a specified email message in the process flow.

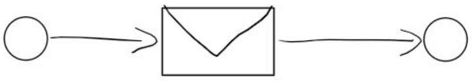
1



2



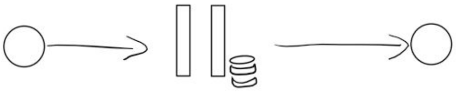
3



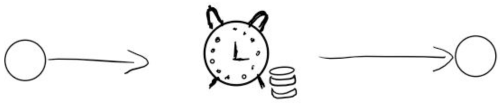
Answer: _______

6. Pauses the execution waiting for a DB event.

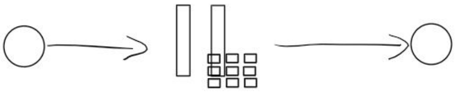
1



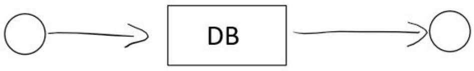
2



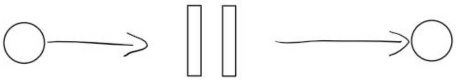
3



4



5

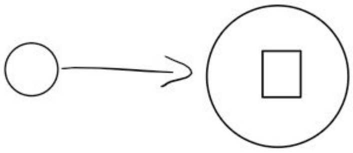


{wait condition}

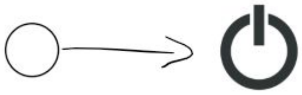
Answer: _______

7. Finishes a process flow. The process may continue in parallel flows.

1

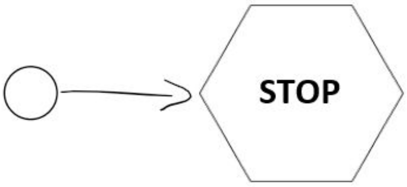


2

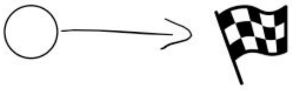


(Red power button.)

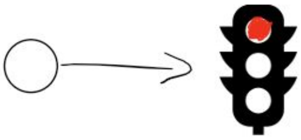
3



4



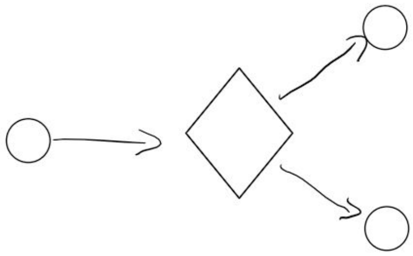
5



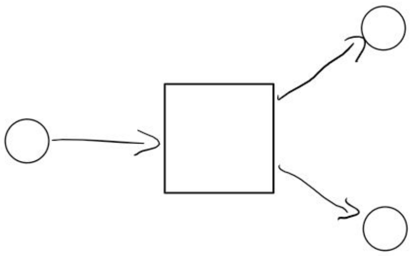
Answer: _______

8. Directs the execution path to a specific activity based on custom logic.

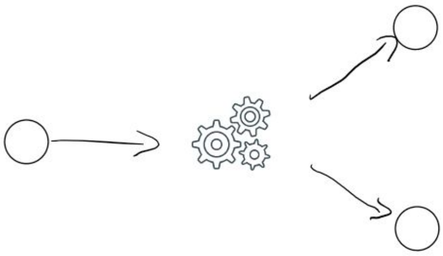
1



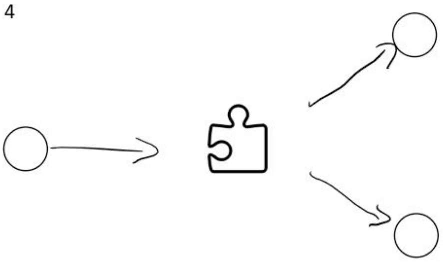
2



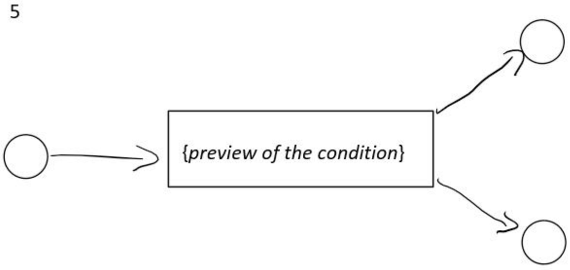
3



4



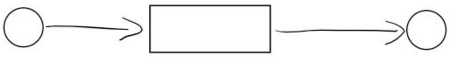
5



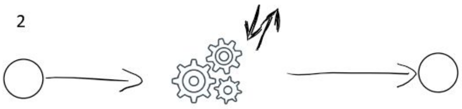
Answer: _______

9. Performs custom logic in the application or in external systems.

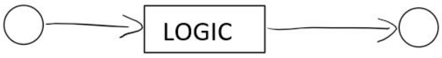
1



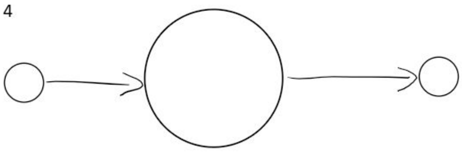
2



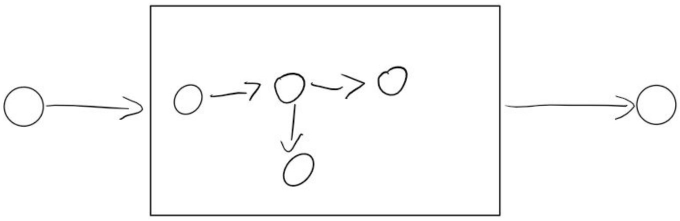
3



4



5

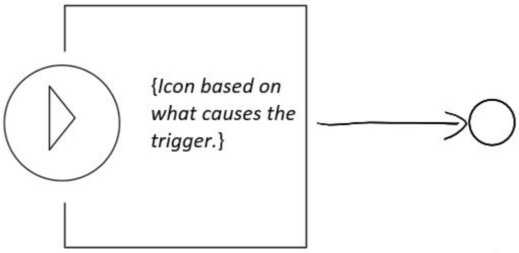


(Inline preview of the logic. Expanded when mouse hover.)

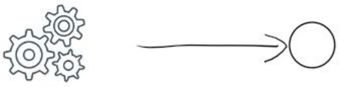
Answer: _______

10. Initiates an alternative flow and may be triggered by a DB event or an explicit API call.

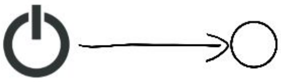
1



2

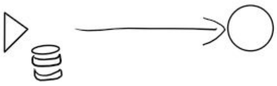


3



(Yellow power button.)

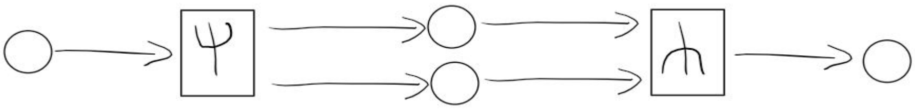
4



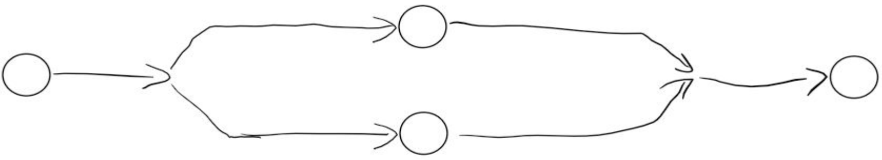
Answer: _______

11. Represent a fork and join: The main execution path is divided into two parallel paths and after each path has finished their task , they merge back into the main execution.

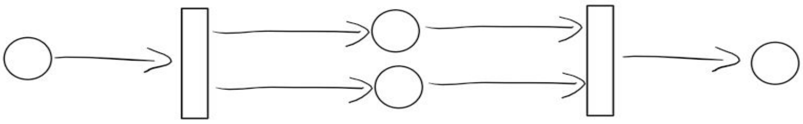
1



2



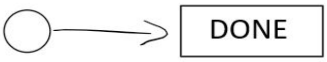
3



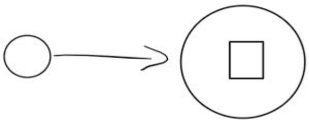
Answer: _______

12. Terminate all the flows of the a process.

1

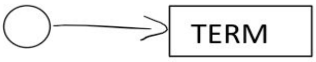


2

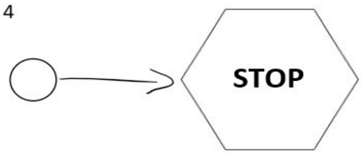


(Media stop sign in red.)

3

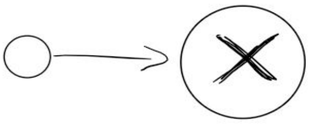


4



(Traffic stop sign.)

5

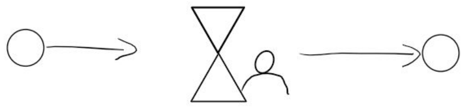


(In red.)

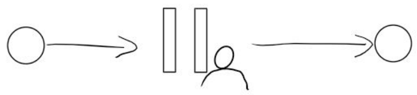
Answer: _______

13. Waits for a user, or group, to complete the given task.

1

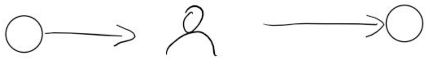


2

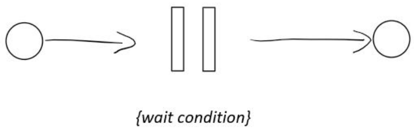


(Media pause button.)

3



4

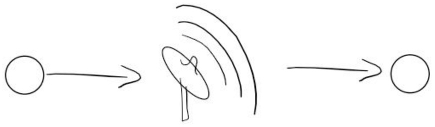


(Media pause button.)

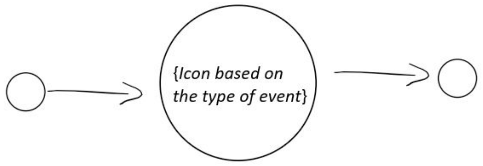
Answer: _______

14. Used to broadcast an event (via DB) or call an API to deliver an event to a specific activity or process.

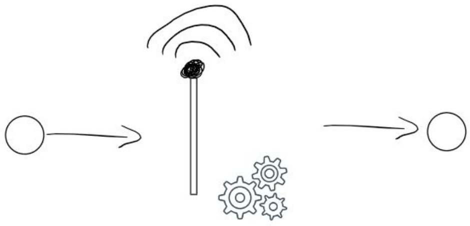
1



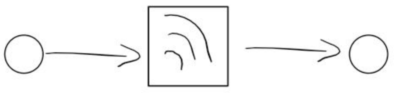
2



3



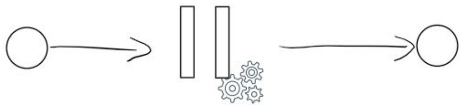
4



Answer: _______

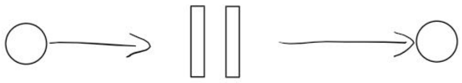
15. Pauses the execution path and resumes when a specific API action is called.

1



(Media pause button.)

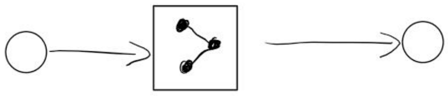
2



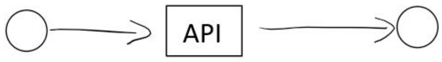
{wait condition}

(Media pause button.)

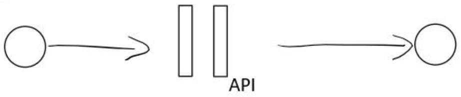
3



4



5



(Media pause button.)

Answer: _______