# Designing an Open Visual Workflow Environment

**Charles Boisvert, Chris Roast, Elizabeth Uruchurtu**
Dept. of Computing, Sheffield Hallam University
Sheffield, United Kingdom, S1 1WB
{c.boisvert | c.r.roast | e.uruchurtu}@shu.ac.uk

## Abstract

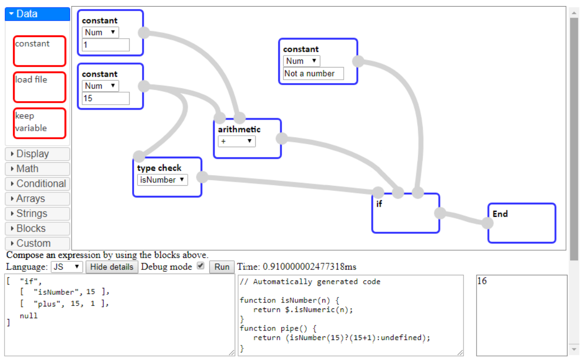
This paper presents open piping, a box-and-wire programming environment, then uses Cognitive Dimensions of Notations to analyse its interaction design and identify its weaknesses. Physics of Notations gives a complementary perspective to propose solutions which we present by example. We also discuss the respective uses and benefits of Cognitive Dimensions and Physics of Notations in this work.

**Keywords**: Computer science education; Data Science; Functional Programming; End-User Programming; Notational Design

## 1. Project background and motivations

Open Piping is an open-source visual functional programming environment, based on a box-and-wire model, intended for data processing applications.

Our ambition is to propose a graphical tool for user-defined data processes[1], with the transparency and flexibility needed to ensure that users can easily define the processes they want to operate on data, while also retaining control of these processes to use them in new environments.



Compose an expression by using the blocks above.

Time: 0.910000002477318ms

```
[  "if",
   [  "isNumber", 15 ],
   [  "plus", 15, 1 ],
   null
]
```

```
// Automatically generated code

function isNumber(n) {
   return $.isNumeric(n);
}
function pipe() {
   return (isNumber(15)?(15+1):undefined);
}
```

```
16
```

*Figure 1 – Open Piping interface with an example workflow*

Fig. 1 shows the Open Piping interface and an example data flow. A more complete description of the system is given in (Boisvert, Roast, & Uruchurtu, 2019).

Four elements motivate our work: the systematic improvement in access to programming brought by the growing ease of use and learning of programming tools; the rise of data processing, underpinned by functional programming and the growth of big data and data analytics; the possibility of modelling functional computation visually; and finally the access barriers to this visual programming paradigm.

*A systematic improvement in access to programming.* Usability breakthroughs mark the progress of all computer science, including programming. One remarkable advance is the wide range of programming learning and novice developer environments, such as MIT Scratch, using a jigsaw puzzle metaphor to represent the combination of individual statements (Resnick et al., 2009).

*The rise of data processing.* As simple applications have become more accessible, computation has shifted to new domains, and to programming languages that support multiple paradigms, like R, Clo-

[1]http://boisvert.me.uk/openpiping

jure, or Python which add functional programming to imperative, object-oriented and event based development. Yet, the jigsaw puzzle metaphor favours an imperative perspective on programming: the programming paradigms computing education tools support best, are becoming less used in practice.

*Modelling functional computation visually.* Lambda calculus' mapping to directed acyclic graphs provides a visual model, summarised table 1. The graph, or box-and-wire model, can read as a data flow.

| **Notation** | **Represents** | **Graphical equivalent** |
|---|---|---|
| $x$ | Variable | $\longrightarrow$ |
| $\lambda x.f$ | Abstraction (function $f$ has parameter $x$) | $\boxed{f}-$ |
| $fx$ | Application (function $f$ is applied to variable $x$) | $\rightarrow\boxed{f}-$ |

*Table 1 – Basic elements of untyped λ-calculus and their representation as box-and-wire*

*Access limitations to visual functional programming* Visual box-and-wire environments are common (Le-Phuoc, Polleres, Tummarello, & Morbidoni, 2008; O'Reilly, n.d.), including some in commercial (Instruments, accessed 2019) and scientific (Hull et al., 2006) use. But in many cases, the value of the tools is limited by a lack of open, accessible implementations of the processes they define and intermediate technologies. Take the case of the popular Yahoo pipes (O'Reilly, n.d.): when support ended in 2015, users only option was a complex export process.

However, data analysis applications require mastery of complex systems to apply mathematical techniques and represent information in non-trivial domains, and this needs to be supported by design.

## 2. Improving a Data Flow Visualisation

As prior research (Roast, Leitão, & Gunning, 2016; Blackwell, 2006, 2001) shows, visualisation is not easy to represent in ways that end-users spontaneously understand. Users', particularly novices, need carefully designed presentation and interaction devices.

### 2.1. Cognitive Dimensions of Notations

Cognitive Dimensions of Notations is a framework of design heuristics (Green & Petre, 1996). The common language it provides is frequently used to evaluate the usability of programming languages and interfaces (Hadhrawi, Blackwell, & Church, 2017; Ennals & Gay, 2007; Morbidoni, Polleres, Tummarello, & Le Phuoc, 2007).

Evaluating the notations used in a complex information artefact, such as Open Piping, with this framework requires a lot of judgement. As an example, let us can compare two evaluations of tools developed on principles comparable to Open Piping.

(Morbidoni et al., 2007) propose Semantic Web Pipes, a functional language and pipe editor to prototype semantic mash-ups; while (Green & Petre, 1996), introducing the framework in their analysis of visual programming environments, consider two functional environments, Labview and in Prograph.

Evaluating the viscosity of their tool, the first estimate that the underlying functional paradigm guarantees 'almost literally the principle of encapsulation and decoupling'. For the same dimension, Green and Petre instead test the viscosity by attempting a minor change to test code in each of the two functional environments, Labview and in Prograph. They choose to focus, not on the language, but on manipulating the code at the interface, explaining that 'boxes had to be jiggled about', and are not satisfied that the language supports appropriate abstractions.

The list of dimensions has varied a little since the framework was proposed. Here, we use the dimensions suggested by (Green & Petre, 1996), listed table 2 (next page).

Cognitive Dimensions provide a useful vocabulary to evaluate the usability of Open Piping, and supports identifying its limits more precisely. Below, we propose an evaluation of Open Piping against each cognitive dimensions.

| Dimension | Characteristic |
|---|---|
| Abstraction gradient | What are the minimum and maximum levels of abstraction exposed by the notation? Can details be encapsulated? |
| Closeness of Mapping | Does the notation correspond to the problem world? |
| Consistency | When some of the notation has been learnt, how much of the rest can be inferred? |
| Diffuseness / terseness | How many symbols (how much space) the notation requires to produce a certain result |
| Error-proneness | Does the notation induce user mistakes? |
| Hard mental operations | How much do the notations impose hard mental processing? |
| Hidden dependencies | Are dependencies visible or hidden? |
| Juxtaposability | Is every part of the notation visible at the same time? |
| Premature commitment | Are there strong constraints on the order in which the user must complete the tasks to use the system?<br>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 | Can the notation carry extra information by means not related to syntax (e.g. layout, colour, or other cues?) |
| Viscosity | How much effort is required to make a single change? |
| Visibility | Can required parts of the notation be identified, accessed and made visible? |

*Table 2 – Cognitive Dimensions, after (Green & Petre, 1996).*

*Abstraction gradient* Open Piping supports the re-use of code by creating new blocks, and of data by setting variables. The management of these abstractions becomes difficult if the user doesn't anticipate: that is, re-usable code identified late in a project needs to be redefined to set it as an (abstraction supporting) block.

*Closeness of Mapping and Consistency* Consistency and Closeness of Mapping are the system's strongest point, as the functional model is represented faithfully by the visual objects.

*Diffuseness / terseness* A lot of blocks can be necessary to specify even simple computations, as each function call and each operator is one block.

*Error-proneness* End-users can easily drag the wrong block, or the wrong link from output to input; though these mistakes are easily undone.

*Hard mental operations* Constructs which use expressions as input (such as lambda expressions) are very difficult to compute mentally. The number of blocks also create visual clutter and make mental operations harder.

*Hidden dependencies* Blocks represent functions and operations: end-users need to be familiar with their effect, including that of complex operations (e.g. lambda extraction).

*Juxtaposability and Role-expressiveness* How much of the computation is visible at the same time depends on its complexity: how many blocks are in use and whether it uses any user-defined blocks. Blocks are clearly annotated with the function they represent; the 'wires' relating them are more easily confused as they are not marked with information. Wires can also cross.

*Premature commitment* User-defined blocks are created within a separate window, and so the end-user needs to plan ahead that their computation belongs in a new block.

*Progressive evaluation* Solutions can easily be tested throughout the development process.

*Secondary notation* User-defined blocks can be named by the user. Spatial positioning of blocks has no incidence on the result and is also chosen by the end-user, although the blocks are presented with inputs at the top-left and outputs on the right-hand side, so blocks' disposition is intended for a top to bottom, left to right reading order.

*Viscosity* Minor changes (e.g. adding a box) require a lot of adjustment; user blocks are also difficult to redefine, as discussed above in premature commitment and abstraction gradient.

*Visibility* The notations are clear to end users, but the visualisation relies on a limited range of three objects: boxes, lines, and discs marking input or output.

Cognitive Dimensions support a useful discussion to identify of the tools' weaknesses. Following it we propose to reduce the abstraction gradient and increase viscosity by allowing a user manipulation to select a subtree in an expression, and make it into a custom block. But it is not always as clear how to address the points identified through Cognitive Dimensions. To that end, we propose to turn to another approach: Physics of Notations.

## 2.2. Physics of Notations

Physics of notations proposes a theory of design for notations based on maximising cognitive effectiveness (Moody, 2009; Van Der Linden & Hadar, 2018). To that aim, it defines nine principles to analyse and develop notations. Compared to Cognitive Dimensions, Physics of Notations ambitions to be a more complete theory, which offers to explain why notations succeed, as well as simply describe them.

More prosaically, for this work it offers two clear advantage over cognitive notations: its principles are focused on visual notations, and all of them offer actionable points to improve the effectiveness of notations. Using these principles, we propose an alternative, fig. 2 which addresses many of the weaknesses of the design identified 2.1 (next page).

| Type | Symbol and colour | Note |
|---|---|---|
| Boolean | 🟩 | Chosen to match the colour and shape of Scratch booleans |
| Number | 🟠 | Match the colour and shape of Scratch numbers |
| String | “” | A large quotation mark |
| List | [...] | Square brackets, as in JSON and arrays in many programming languages |
| JSON data | {...} | Curly brackets, as in JSON objects |
| Expression | ⊶ | Expression, formed of related blocks, are needed as inputs to some blocks, for instance lambda-extraction or conditionals |
| Block | 🟦 | To allow blocks as output of and input to other blocks (in programming terms, functions as data) |
| Any | ⬜ | Used when any data type is allowable as input, for example, in a type checking block |

*Table 3 – Data types used by open piping and their notational representation*

*Semiotic clarity* recommends that all semantic constructs find a visual expression. Boxes and wires represent input and output to functions, but data typing is an important semantic construct that should also be made clear: identical wires give no information about the data transmitted. By associating each main data type to a colour (for wires) and a symbol (for inputs and outputs), we express more of the important semantic information within the notation. Each data type used, and the proposed colour and shape to denote them is presented for reference in table 3 above.

*Perceptual discriminability* (ensuring symbols are easy to tell apart) can can be seen in table 3, where a small number of colour and symbols are used, making them highly distinguishable. Blocks have an identical shape, except for blocks processing expressions as discussed below, but input blocks and blocks carrying out operations can be distinguished by colour.

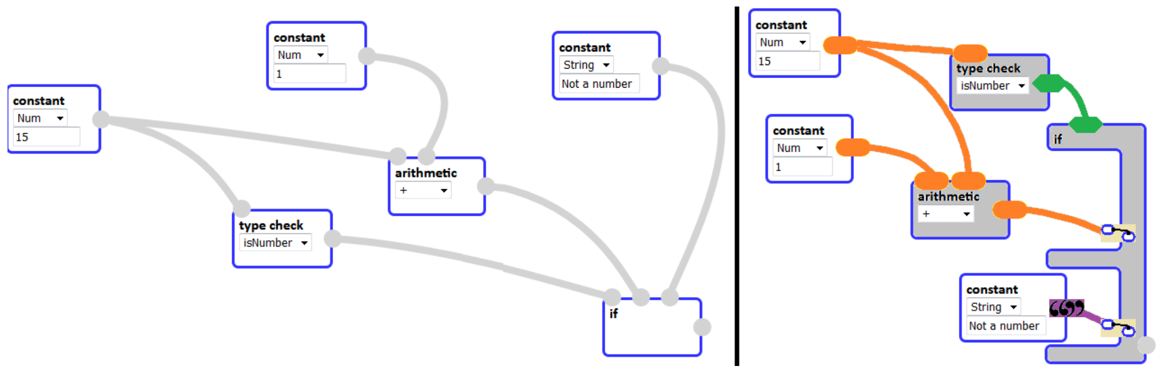


*Figure 2 – A workflow before (left) and after (right) revision for cognitive effectiveness*

*Visual expressiveness* (using the fullest range of visual variables possible) point to fully exploiting visual variables - for example, as above, with colour and shape. This can also be done by giving the blocks that receive expressions as input, a special shape, to indicate their exceptional character - an "open jaw" shape that can also visually include some of the expression.

*Semantic transparency* recommends that the appearance of notations suggests their meaning: this is done by composing function boxes with icons for input and output parameters, and by positioning inputs at the top or left and outputs at the bottom or right, following the common reading order.

*Graphic economy* is also respected as symbols remain few and easily recognised. We also propose to allow the output symbol to glide freely around the right and bottom side of a block, and inputs around its left and top sides - including not imposing an argument order, so that wires rarely cross over in complex expressions, reducing visual clutter.

*Complexity management* is supported by encoding more information in pre-attentive ways - through colour, size and shape. Allowing the argument inputs and output to "float" along the box's border also reduces complexity. The shape of blocks handling expressions as input may not reduce complexity, but it signals it to the viewer.

Fig. 2 applies the principles of Physics of Notations to one workflow. The final three principles of Physics of Notations are *Cognitive Integration*, *Dual coding*, and *Cognitive fit*, which address, respectively, the coordination of documents, the use of written annotations, and the adaptation to diverse audiences. Our proposed solution fig. 2 does not make use of these points.

## 3. Future work and reflection

A further evaluation will be needed to consider the effectiveness of the redesign proposed in section 2.2 above. But the two frameworks showed an interesting complementarity when using them simultaneously.

Cognitive Dimensions and Physics of Notations complemented each other usefully to carry out this analysis and find design options. The Cognitive Dimensions heuristics provided insights in dynamic aspects of the visualisation with its dimensions of abstraction gradient and viscosity, pointing immediately to both problem and solution. It then helped highlight many weaknesses of the design, but provided fewer actionable points to improve it. By contrast Physics of Notations focused attention on identifiable improvements to the visualisation. In particular, as (Roast & Uruchurtu, 2016) point out, Physics of Notations centrally asks the question: 'what constitutes and defines the semantic domain being visualised?' (Roast & Uruchurtu, 2016). This semantic focus brought the insight that the visualised domain must include typing, while other principles provided means to do so.

Cognitive dimensions also provides a validation of the redesign suggestions. The proposed redesign goes some way to solving several of the most egregious problems identified with Cognitive Dimensions: it is less error-prone, more tolerant of change, and gives important clues in support of hard mental operations. This shows it is valuable to exploit both frameworks in combination: Cognitive Dimensions helps consider notation abstractly and include dynamic aspects of use, while Physics of Notations focus on visual ways of expressing meaning and on identifying that meaning to be expressed.

## 4. References

Blackwell, A. F. (2001). Pictorial representation and metaphor in visual language design. *Journal of Visual Languages & Computing*, *12*(3), 223 - 252. Retrieved from `http://www.sciencedirect.com/science/article/pii/S1045926X01902071` doi: https://doi.org/10.1006/jvlc.2001.0207

Blackwell, A. F. (2006, December). The reification of metaphor as a design tool. *ACM Trans. Comput.-Hum. Interact.*, *13*(4), 490–530. Retrieved from `http://doi.acm.org/10.1145/1188816.1188820` doi: 10.1145/1188816.1188820

Boisvert, C., Roast, C., & Uruchurtu, E. (2019). Open piping: Towards an open visual workflow environment. In *Conference proceedings of 2019 international symposium on end-user development (is-eud).*

Ennals, R., & Gay, D. (2007). User-friendly functional programming for web mashups. In *Acm sigplan notices* (Vol. 42, pp. 223–234).

Green, T. R. G., & Petre, M. (1996). Usability analysis of visual programming environments: a 'cognitive dimensions' framework. *Journal of Visual Languages & Computing*, *7*(2), 131–174.

Hadhrawi, M., Blackwell, A. F., & Church, L. (2017). A systematic literature review of cognitive dimensions. In *Ppig* (p. 3).

Hull, D., Wolstencroft, K., Stevens, R., Goble, C., Pocock, M. R., Li, P., & Oinn, T. (2006). Taverna: a tool for building and running workflows of services. *Nucleic acids research*, *34*(suppl 2), W729–W732.

Instruments, N. (accessed 2019). *What is labview.* `http://www.ni.com/en-gb/shop/labview.html`. (Accessed: 2019-30-04)

Le-Phuoc, D., Polleres, A., Tummarello, G., & Morbidoni, C. (2008). Deri pipes: visual tool for wiring web data sources. *)ˆ(Eds.):'Book DERI pipes: visual tool for wiring web data sources'(2008, edn.).*

Moody, D. (2009). The "physics" of notations: toward a scientific basis for constructing visual notations in software engineering. *IEEE Transactions on software engineering*, *35*(6), 756–779.

Morbidoni, C., Polleres, A., Tummarello, G., & Le Phuoc, D. (2007). Semantic web pipes. *Rapport technique, DERI*, *71*, 108–112.

O'Reilly, T. (n.d.). *Pipes and filters for the internet.* `http://radar.oreilly.com/2007/02/pipes-and-filters-for-the-inte.html`. (Accessed: 2016-10-10)

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., ... others (2009). Scratch: programming for all. *Communications of the ACM*, *52*(11), 60–67.

Roast, C., Leitão, R., & Gunning, M. (2016). Visualising formula structures to support exploratory modelling. In *Proceedings of the 8th international conference on computer supported education* (pp. 383–390). Portugal: SCITEPRESS - Science and Technology Publications, Lda. Retrieved from `https://doi.org/10.5220/0005812303830390` doi: 10.5220/0005812303830390

Roast, C., & Uruchurtu, E. (2016). Reflecting on the physics of notations applied to a visualisation case study. In *Proceedings of the 6th mexican conference on human-computer interaction* (pp. 24–31).

Van Der Linden, D., & Hadar, I. (2018). A systematic literature review of applications of the physics of notations. *IEEE Transactions on Software Engineering*, *45*(8), 736–759.