Projecting Computer Languages for a Semantic Interaction

Camille Gobert

Projeter les Langages Informatiques pour une Interaction Sémantique

Projecting Computer Languages for a Semantic Interaction

Camille Gobert *
gobert@lri.fr
Université Paris-Saclay, CNRS, Inria
Laboratoire Interdisciplinaire des Sciences du Numérique
91400 Orsay, France

*. En troisième année de thèse sous la direction de Michel Beaudouin-Lafon.

ABSTRACT

Despite its limitations, our interaction with computer languages is still based mostly on their syntax. In my Ph.D., I study the interaction with their semantics, i.e., with the concepts expressed *using* these languages, through alternative representations I call *projections*. I critique the current design paradigms of the latter and motivate, develop and evaluate new approaches by using a user-centred design methodology that I apply to document description languages and programming languages.

CCS CONCEPTS

• **Human-centered computing** → *User studies*; *Graphical user interfaces*; • **Software and its engineering** → *General programming languages*; • **Applied computing** → *Markup languages*.

KEYWORDS

Computer language, Semantic interaction, Projection, *i*-LaTeX, Lorgnette

RÉSUMÉ

En dépit de ses défauts, notre interaction avec les langages informatiques se résume souvent à leur syntaxe. Dans le cadre de ma thèse, je m'intéresse à l'interaction avec leur sémantique, c'est-à-dire avec les concepts exprimés *avec* ces langages, à travers des représentations alternatives appelées *projections*. Je critique les paradigmes de conception actuels de ces dernières et motive, développe et évalue de nouvelles approches en appliquant la méthodologie de la conception centrée sur l'utilisateur aux langages de description de documents et aux langages de programmation.

MOTS CLÉS

Langage informatique, Interaction sémantique, Projection, *i*-LaTeX, Lorgnette

INTRODUCTION

L'avènement du numérique a rendu les *langages informatiques* omniprésents. Un langage informatique est ici défini en tant que système symbolique doté de règles de composition de symboles qui, par opposition à un langage dit *naturel*, est compréhensible par une machine dans un contexte particulier. Cette définition inclut par exemple les langages de programmation tels que JavaScript et Python, qui font partie des langages les plus utilisés par les développeurs dans le monde,[1] ainsi que les langages de description de document tels que LaTeX pour les productions scientifiques [22] et Markdown pour le web [4]. Ces langages occupent désormais une place importante dans l'industrie, sont exploités par des artistes [14], et sont en voie de devenir des connaissances courantes attendues de tous, comme en atteste l'enseignement de Python au lycée au France.[2] Afin d'interagir avec ces langages, de nombreux environnements d'édition ont été conçus. On peut distinguer deux grands paradigmes d'interaction dans les techniques qu'ils mettent en œuvre : l'interaction *syntaxique* et l'interaction *sémantique*.

Interaction syntaxique. L'interaction syntaxique implique d'interagir avec les concepts *du* langage lui-même, tels que ses symboles et les structures que ceux-ci forment. C'est le paradigme employé par les éditeurs textuels, les éditeurs structurés tels que le Cornell program synthesizer [25], ainsi que des systèmes hybrides tels que Stride [13] qui combinent l'édition textuelle et la manipulation de structures syntaxiques. Les variantes textuelles de ce paradigme le rendent générique et adapté à des technologies qui perdurent (claviers, utilitaires Unix), tandis que les variantes structurées permettent aux utilisateurs de se prémunir des erreurs de syntaxe.

Interaction sémantique. L'interaction sémantique, quant à elle, implique d'interagir avec les concepts exprimés *avec* un langage, notamment au travers de représentations alternatives adaptées à chaque concept que je nomme *projections*.[3] Dans ce paradigme, une structure contenant trois nombres pourrait ainsi être visualisée et manipulée comme une position dans l'espace ou comme une couleur selon la situation, là où l'approche syntaxique n'y voit qu'une construction générique permise par le langage. Il existe des projections pour des concepts très variés : tableaux de données [11, 17], parallélisation de boucle [27], expressions régulières [18], plateaux de jeux [1, 20], etc. En donnant forme aux concepts exprimés par les utilisateurs d'un langage, ces projections leur permettent de bénéficier de représentations plus proches de leurs modèles mentaux, réduisant ainsi les gouffres de l'évaluation et de l'exécution identifiés par Norman [16].

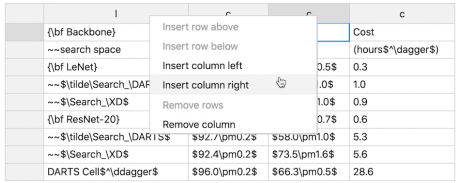
1. C'est ce que montrent deux analyses parues en 2022 : l'une des dépôts de la plateforme GitHub (https://octoverse.github.com/) ; l'autre d'un sondage de la communauté des utilisateurs du StackOverflow (https://survey.stackoverflow.co/2022/).

2. C'est officiellement le cas depuis l'introduction de l'enseignement de l'informatique au lycée en 2019 (https://www.education.gouv.fr/bo/19/Special1/MENE1901641A.htm).

3. Je propose d'emprunter ce terme aux éditeurs projectionnels [23] — dans lesquels l'utilisateur manipule directement les nœuds de l'arbre de syntaxe qui peuvent être représentés (« projetés ») de manière arbitraire — afin de désigner un concept dont la littérature en IHM fait défaut. Les concept de notation auxiliaire [9], d'augmentation du code [24] et de flexibilité code/interface graphique [11] sont trop généraux ou imprécis, tandis que ceux de « *patch* » [14], « *palette* » [18], « *codelet* » [19] et « *livelit* » [17] sont trop spécifiques.

```
\begin{table}[!t]
    \centering
    \begin{itabular}{lccc}
        \hline
        {\bf Backbone} & & Permuted & Cost \\
        ~~search space & CIFAR-10 & CIFAR-10$^\ast$ & (hours$^\dagger$) \\
        \hline
        {\bf LeNet} & $75.5\pm0.1$ & $43.7\pm0.5$ & 0.3 \\
        ~~$\tilde\Search_\DARTS$ & $75.6\pm3.4$ & $47.7\pm1.0$ & 1.0 \\
        ~~$\Search_\XD$ & $77.7\pm0.7$ & $63.0\pm1.0$ & 0.9 \\
        \hline
        {\bf ResNet-20} & $91.7\pm0.2$ & $58.6\pm0.7$ & 0.6 \\
        ~~$\tilde\Search_\DARTS$ & $92.7\pm0.2$ & $58.0\pm1.0$ & 5.3 \\
        ~~$\Search_\XD$ & $92.4\pm0.2$ & $73.5\pm1.6$ & 5.6 \\
        \hline
        DARTS Cell$^\ddagger$ & $96.0\pm0.2$ & $66.3\pm0.5$ & 28.6 \\
        \hline
    \end{itabular}
\end{table}
```

(a) Code LaTeX d'un tableau.



(b) Représentation transitionnelle du code.

Backbone search space	CIFAR-10	Permuted CIFAR-10*	Cost (hours†)
LeNet	75.5 ± 0.1	43.7 ± 0.5	0.3
$\tilde{\mathcal{S}}_{\text{DARTS}}$	75.6 ± 3.4	47.7 ± 1.0	1.0
$\mathcal{S}_{\text{XD}}$	77.7 ± 0.7	63.0 ± 1.0	0.9
ResNet-20	91.7 ± 0.2	58.6 ± 0.7	0.6
$\tilde{\mathcal{S}}_{\text{DARTS}}$	92.7 ± 0.2	58.0 ± 1.0	5.3
$\mathcal{S}_{\text{XD}}$	92.4 ± 0.2	73.5 ± 1.6	5.6
DARTS Cell‡	96.0 ± 0.2	66.3 ± 0.5	28.6

(c) Rendu du tableau dans le PDF.

Figure 1. *i*-LaTeX permet d'interagir avec le code d'un tableau à la fois (a) en l'éditant en tant que texte et (b) en manipulant la grille qu'il décrit à l'aide d'une projection sémantique affichée en cliquant sur (c) le tableau généré dans le PDF (d'où le halo bleu).

MOTIVATIONS

En dépit des avantages théoriques des projections, les environnements d'édition de langages informatiques qui en disposent sont rares, rendant leur potentiel et leurs effets réels difficiles à étudier. La littérature à leur sujet est moins fournie que celle sur l'interaction syntaxique et se concentre sur les dix dernières années. Bien que certaines évaluations suggèrent que les projections facilitent certaines tâches, telles que l'optimisation polyédrique avec les diagrammes de Clint [27], ces études sont rares et ne permettent donc pas l'émergence de méta-analyses et de conclusion claire.

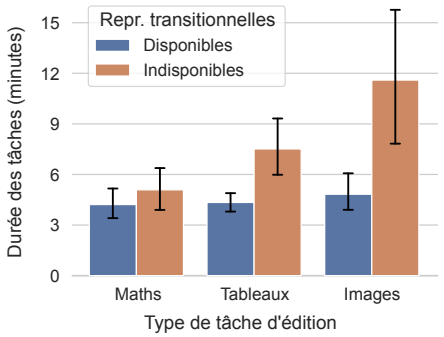
La difficulté de créer et d'étudier les projections de manière écologique (par exemple, en les appliquant à des langages couramment utilisés) tient également des paradigmes des systèmes permettant de les concevoir. Ceux-ci incluent notamment les environnements de programmation malléable tels que Smalltalk [8] et Pharo [3], les *language workbenches* tels que Barista [12] et JetBrains MPS [26], et les macros visuelles tels que celles pour Racket [1] et Hazel [17]. Or, ces paradigmes sont souvent pensés pour un langage unique, rendant la réutilisation de projections difficile et multipliant inutilement le nombre d'outils à maîtriser. En outre, ils ne permettent que rarement aux utilisateurs finaux de s'approprier les projections ou d'en concevoir de nouvelles pour leurs besoins personnels sans que cela ne requière un investissement prohibitif de leur part.

Ces limitations forment un frein à une utilisation plus répandue des projections pour interagir avec les langages informatiques. Tandis que l'utilisation de ces langages est croissante, la volatilité des besoins des utilisateurs [15], l'aspect exploratoire de la programmation [21], les utilisations inattendues des langages [5] ou encore les différences culturelles [2] requièrent une flexibilité des représentations alternatives permettant une interaction sémantique qui n'est pas atteinte par les systèmes actuels. Ma thèse s'inscrit dans le sillage de ces observations et vise à contribuer à l'étude et à la conception de systèmes dotés de projections pour des langages informatiques largement répandus, ainsi qu'à faciliter leur appropriation par les utilisateurs finaux. Mes questions de recherche sont les suivantes :

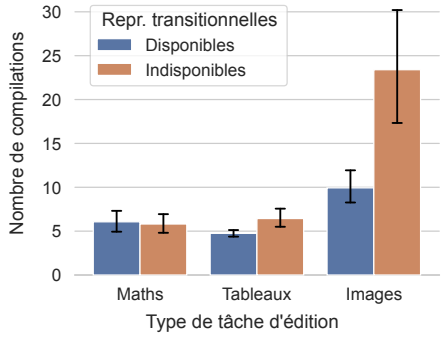
Q1 À quels besoins les projections peuvent-elles répondre et de quelles façons le peuvent-elle ?
Q2 Comment permettre aux utilisateurs d'un langage de créer et s'approprier des projections ?
Q3 Quelles sont les dimensions de l'espace de conception des projections et quel est son potentiel ?

CONTRIBUTIONS

Afin de répondre à ces questions, j'utilise la méthodologie de la conception centrée sur l'utilisateur. Je m'intéresse plus particulièrement à deux classes de langages informatiques, les langages de description de document et les langages de programmation, pour lesquels (1) j'analyse les besoins des utilisateurs de ces langages, (2) je développe des systèmes dotés de projections et (3) j'évalue les effets, fonctionnalités et limitations de ceux-ci. Ce travail s'articule autour de deux projets principaux déjà complétés, ainsi que d'un projet annexe sur lequel je souhaite encore travailler, résumés ci-dessous.



(a) Durée moyenne des tâches.



(b) Nombre moyen de compilations.

Figure 2. L'évaluation contrôlée de *i*-LaTeX montre que le fait d'avoir accès aux représentations transitionnelles a permis aux participants de compléter des tâches d'édition courantes (trois de chaque type) plus rapidement et en compilant leur document moins souvent. Les barres d'erreurs correspondent à des intervalles de confiance à 95%. Les détails sur la méthodologie et l'analyse des données sont disponibles dans l'article [7].

i-LaTeX — Un environnement d'édition de documents LaTeX doté de projections

Mon premier projet porte sur l'utilisation de projections pour le langage de description de documents LaTeX. J'ai d'abord interviewé 11 utilisateurs de LaTeX afin de comprendre leurs besoins et leurs difficultés, mis en exergue par une analyse thématique des transcrits. J'ai ensuite conçu *i*-LaTeX, un prototype d'éditeur LaTeX doté de projections pour les formules mathématiques, les tableaux, les images, et les grilles de mise en page, que j'ai également théorisées sous le terme de *représentations transitionnelles* (Figure 1). J'ai utilisé ce prototype dans une expérimentation contrôlée avec 16 participants afin d'évaluer l'effet de la disponibilité de ces représentations pour effectuer plusieurs tâches d'édition courantes. Les résultats montrent que ces représentations réduisent le temps de résolution des tâches (Figure 2a), le nombre de compilations nécessaires (Figure 2b), la charge de travail des participants ; mais aussi qu'elles affectent les stratégies employées pour résoudre les tâches. Ce projet a donné lieu à deux publications scientifiques : une première publication à la conférence IHM 2021 [6] et une version étendue à la conférence ACM CHI 2022 [7].

Lorgnette — Un système de conception de projections appropriable par ses utilisateurs

Mon second projet porte sur la création d'un paradigme permettant aux programmeurs de créer et de s'approprier plus facilement des projections. J'ai conçu Lorgnette, un prototype d'un système mettant en œuvre ce paradigme, afin de tester sa faisabilité technique et de créer plusieurs exemples de projections. En sus d'une analyse des projections de la littérature, j'ai ensuite organisé un atelier de conception participative avec 9 programmeurs afin de leur faire concevoir des projections adaptées à leurs besoins personnels dans le but d'évaluer les propriétés des projections et les contraintes techniques qui en découlent. Je souhaite comparer mon approche aux autres approches de la littérature à travers l'implémentation de multiples projections (issues de l'atelier participatif, de la littérature et d'idées personnelles) afin de mettre en avant les avantages de mon système concernant (1) la réutilisation d'une même représentation dans plusieurs langages et contextes et (2) la conception et l'appropriation de projections par les utilisateurs finaux. Je prévois de soumettre un article scientifique sur ce sujet dans la première moitié de l'année 2023.

Description et exploration critique de l'espace de conception des projections

Mon dernier projet est une synthèse du reste de mon travail, sur lequel je prévois de travailler durant la rédaction de mon manuscrit de thèse. Je souhaite proposer un ensemble de dimensions décrivant l'espace de conception des projections pour les langages informatiques, à la façon des dimensions techniques des systèmes permettant de programmer [10]. Celui-ci a pour objectif d'aider à identifier (1) le potentiel de certains sous-espaces peu explorées et (2) les pré-requis techniques et les propriétés qu'un langage doit avoir pour pouvoir y être projeté.

RÉFÉRENCES

[1] Leif Andersen, Michael Ballantyne, and Matthias Felleisen. 2020. Adding Interactive Visual Syntax to Textual Code. In *Proceedings of the ACM on Programming Languages*, Vol. 4. 1–28. https://doi.org/10.1145/3428290

[2] Ian Arawjo. 2020. To Write Code : The Cultural Fabrication of Programming Notation and Practice. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems (CHI '20)*. ACM, 1–15. https://doi.org/10.1145/3313831.3376731

[3] Alexandre Bergel, Damien Cassou, Stéphane Ducasse, and Jannik Laval. 2013. *Deep into Pharo*. Square Bracket Associates. 420 pages.

[4] Matthew Conlen and Jeffrey Heer. 2018. Idyll : A Markup Language for Authoring and Publishing Interactive Articles on the Web. In *Proceedings of the 31st Symposium on User Interface Software and Technology - UIST '18*. ACM, 977–989. https://doi.org/10.1145/3242587.3242600

[5] Alan Dix. 2007. Designing for Appropriation. In *Proceedings of HCI 2007 The 21st British HCI Group Annual Conference University of Lancaster (21)*. 1–4.

[6] Camille Gobert and Michel Beaudouin-Lafon. 2021. Représentations Intermédiaires Interactives Pour La Manipulation de Code LaTeX. In *32e Conférence Francophone Sur l'Interaction Homme-Machine (IHM '21)*. ACM, 1–11. https://doi.org/10.1145/3450522.3451325

[7] Camille Gobert and Michel Beaudouin-Lafon. 2022. *i*-LaTeX : Manipulating Transitional Representations between LaTeX Code and Generated Documents. In *CHI Conference on Human Factors in Computing Systems (CHI '22)*. ACM, 1–16. https://doi.org/10.1145/3491102.3517494

[8] Adele Goldberg and David Robson. 1983. *Smalltalk-80 : The Language and Its Implementation*. Addison-Wesley.

[9] Thomas R. G. Green. 1989. Cognitive Dimensions of Notations. In *People and Computers V*. 443–460.

[10] Joel Jakubovic, Jonathan Edwards, and Tomas Petricek. 2023. Technical Dimensions of Programming Systems. *The Art, Science, and Engineering of Programming* 7, 3 (2023), 1–59. https://doi.org/10.22152/programming-journal.org/2023/7/13

[11] Mary Beth Kery, Donghao Ren, Fred Hohman, Dominik Moritz, Kanit Wongsuphasawat, and Kayur Patel. 2020. Mage : Fluid Moves Between Code and Graphical Work in Computational Notebooks. In *Proceedings of the 33rd Symposium on User Interface Software and Technology - UIST'20*. ACM, 140–151. https://doi.org/10.1145/3379337.3415842

[12] Amy J. Ko and Brad A. Myers. 2006. Barista : An Implementation Framework for Enabling New Tools, Interaction Techniques and Views in Code Editors. In *Proceedings of the 2006 CHI Conference on Human Factors in Computing Systems - CHI '06*. ACM, 387–396. https://doi.org/10.1145/1124772.1124831

[13] Michael Kölling, Neil Brown, and Amjad Altadmri. 2017. Frame-Based Editing. *Journal of Visual Languages and Sentient Systems* 3, 1 (2017), 40–67. https://doi.org/10.18293/VLSS2017-009

[14] Alex McLean. 2011. *Artist-Programmers and Programming Languages for the Arts*. Ph. D. Dissertation. Department of Computing, Goldsmiths, University of London.

[15] Gina Neff and David C. Stark. 2002. Permanently Beta : Responsive Organization in the Internet Era. *Institute for Social and Economic Research and Policy Working Papers* (2002). https://doi.org/10.7916/D8G44X47

[16] Donald A. Norman. 2002. *The Design of Everyday Things*. Basic Books.

[17] Cyrus Omar, David Moon, Andrew Blinn, Ian Voysey, Nick Collins, and Ravi Chugh. 2021. Filling Typed Holes with Live GUIs. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. ACM, 511–525. https://doi.org/10.1145/3453483.3454059

[18] Cyrus Omar, Young Seok Yoon, Thomas D. LaToza, and Brad A. Myers. 2012. Active Code Completion. In *Proceedings of the 34th International Conference on Software Engineering - ICSE'12*. IEEE, 859–869. https://doi.org/10.1109/ICSE.2012.6227133

[19] Stephen Oney and Joel Brandt. 2012. Codelets : Linking Interactive Documentation and Example Code in the Editor. In *Proceedings of the 2012 CHI Conference on Human Factors in Computing Systems - CHI '12*. ACM, 2697. https:

//doi.org/10.1145/2207676.2208664

[20] Clément Pit-Claudel. 2020. Untangling Mechanized Proofs. In *Proceedings of the 13th ACM SIGPLAN International Conference on Software Language Engineering*. ACM, 155–174. https://doi.org/10/ghs5sn

[21] Patrick Rein, Stefan Ramson, Jens Lincke, Robert Hirschfeld, and Tobias Pape. 2018. Exploratory and Live, Programming and Coding : A Literature Study Comparing Perspectives on Liveness. *The Art, Science, and Engineering of Programming* 3, 1 (2018), 1. https://doi.org/10.22152/programming-journal.org/2019/3/1

[22] Paulo Reis, John D Lees-Miller, and Sven Laqua. 2021. Merging SaaS Products In A User-Centered Way — A Case Study of Overleaf and ShareLaTeX. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*. ACM, 1–8. https://doi.org/10.1145/3411763.3443455

[23] Charles Simonyi, Magnus Christerson, and Shane Clifford. 2006. Intentional Software. In *Proceedings of the 21st Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA '06)*. ACM, 451–464. https://doi.org/10.1145/1167473.1167511

[24] Matúš Sulír, Michaela Bačíková, Sergej Chodarev, and Jaroslav Porubän. 2018. Visual Augmentation of Source Code Editors : A Systematic Mapping Study. *Journal of Visual Languages & Computing* 49 (2018), 46–59. https://doi.org/10.1016/j.jvlc.2018.10.001

[25] Tim Teitelbaum and Thomas Reps. 1981. The Cornell Program Synthesizer : A Syntax-Directed Programming Environment. *Commun. ACM* 24, 9 (1981), 563–573. https://doi.org/10.1145/358746.358755

[26] Markus Voelter and Sascha Lisson. 2014. Supporting Diverse Notations in MPS' Projectional Editor. In *Proceedings of the 2nd International Workshop on The Globalization of Modeling Languages*. 7–16.

[27] Oleksandr Zinenko, Cédric Bastoul, and Stéphane Huot. 2015. Manipulating Visualization, Not Codes. In *International Workshop on Polyhedral Compilation Techniques (IMPACT)*. 1–8.