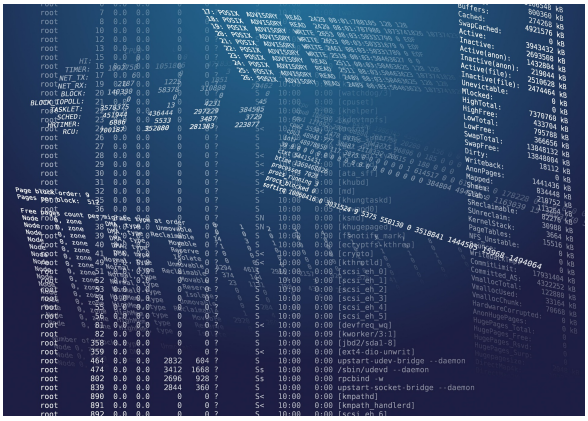
Le code comme outil : adaptabilité et désobjectivation

Jean-Baptiste Guignard, Ophir Paz, Kim Savaroche

Jean-Baptiste Guignard
Ophir Paz
Kim Savaroche

Le code comme outil : adaptabilité et désobjectivation



> #2
> Ce que le calcul fait à nos pratiques
> Communications orales enregistrées
> Séminaire PHITECO-2018
> Design

Références de citation

Guignard, Jean-Baptiste., Paz, Ophir., Savaroche, Kim. "Le code comme outil : adaptabilité et désobjectivation.", 19 décembre 2018, màj 0000, *Cahiers COSTECH* numéro 2.
http://www.costech.utc.fr/CahiersCOSTECH/spip.php?article73

Résumé et vidéo de l'intervention (séminaire PHITECO 2018)

Auteur(s)

Jean-Baptiste Guignard (Mines ParisTech) est Chercheur en Sciences Cognitives et cofondateur de Clay AIR inc. — entreprise proposant des solutions de reconnaissance gestuelle.

Ophir Paz est actuellement doctorant à l'IMS (UMR 5218, CNRS) et travaille pour Clay.

Kim Savaroche est actuellement doctorante à l'IMS (UMR 5218, CNRS) et travaille pour Clay.

Clay est une solution logicielle de reconnaissance gestuelle notamment sur smartphone. Il s'agit pour nous de produire un outil capable d'identifier en temps réel une ou plusieurs mains dans des environnements variablement bruités – il doit être robuste au travers des différents milieux auxquels il est confronté (variations radicales de lumière, mouvements de caméra, paysage défilant, etc.). Cette réalisation suppose une intégration de travaux issus de différentes disciplines scientifiques et des types de modélisation hétérogènes. Souvent issus de la recherche fondamentale, ces modèles doivent être adaptés et précisés pour être appliqués au contexte opérationnel singulier de Clay.

C'est généralement par implémentation et modification du code (*refactoring* entre autres) que se produit cette adaptation, ce qui permet à la fois d'assurer la lisibilité algorithmique (support au bon fonctionnement du produit) et d'agir directement sur son comportement. Le code informatique ainsi en position d'outil technique est constitutif de la perception des modèles par les développeurs qui les intègrent. En aval de cette phase d'intégration, le produit obtenu est testé *in situ* pour évaluer ses performances et le faire évoluer (le cas échéant). Cette évolution peut être soit une réitération du processus de modélisation précédemment décrit, soit une adaptation du code par concrétisation. Par exemple, une formule mathématique pourra être approximée tout en restant pertinente en contexte dans le but de réduire son temps d'exécution – un modèle bio-inspiré (rétine humaine pour la Computer Vision) couteux en CPU (160% de CPU sur iPhone 7) fera l'objet de diverses sapes d'optimisation jusqu'à le rendre opérant à moindre coût (0,8% de CPU).

Devenu outil technique, le code est une inscription numérique de ces modélisations, continuellement interprétée et négociée par les développeurs qui s'en saisissent. Les méthodologies de développement appliquées doivent donc privilégier sa maintenabilité par la qualité de l'activité de négociation du sens du code – le *coding*, plutôt que la qualité du code lui-même selon des conventions arbitraires issues du génie logiciel.

C'est en somme du caractère sémiotique de l'activité de *coding* que nous souhaitons rendre compte, mettant en exergue notamment les parcours d'interprétation auquel elle donne lieu. C'est, enfin, à partir du travail « fondamental » en *Computer Vision* et en IA et « appliqué » (en *general engineering* pour les stratégies d'optimisation) que les équipes de Clay

développent que nous exemplifierons cette désobjectivation du code-outil, ici également médiateur des disciplines et des pratiques.

La perception par l'outil

Saisi, l'outil ou l'objet technique joue un rôle prothétique pour nos capacités d'action, de raisonnement ou encore de perception. On considérera par extension que la création d'un modèle, qui prend ici le sens d'une représentation (au sens d'une carte ou d'un plan) d'une entité de l'environnement perçu, est dépendante des outils mis à disposition. Par exemple, si un enfant souhaite représenter une maison et qu'il a accès à des crayons de couleur, il pourra dessiner la structure de la maison comme un carré rouge, son toit comme un triangle orange (réf. slide 14). De même, le soleil peut être symbolisé par une tête jaune, souriante et rayonnante. Si l'enfant n'a pas de crayons de couleur et qu'il n'a, à proximité, qu'un crayon à papier, c'est une toute autre représentation qui adviendra. Il s'attardera par exemple sur les textures et rajoutera un motif de briques sur la cheminée. Suivant la même logique, le toit qui était un triangle de couleur unie deviendra un triangle avec des tuiles. L'enfant a façonné une représentation de la maison en fonction et au moyen des outils disponibles. À un même élément « cible » du monde, peuvent correspondre des représentations différentes selon les objets techniques employés. Le contexte est prégnant de possibles, d'outils à saisir.

À un contexte correspondent des outils, chaque variation de contexte implique une variation desdits outils, et corrélativement de la représentation obtenue *in fine*. Dans le contexte informatique au sens large, une image est modélisée par une grille de pixels, qui ont chacun une couleur associée – l'outil est ici le jeu d'instructions mis à disposition par le langage de programmation et ses possibles (corrélativement, ses restrictions). Les couleurs d'une image sont *in situ* traditionnellement décrites par le système RGB – un mélange de trois canaux : le rouge, le vert et le bleu. Chaque canal comprend une valeur entre 0 et 255 qui exprime la quantité de rouge, de vert ou de bleu à mélanger pour obtenir la couleur finale. Si un développeur analyse les codes de couleur de la « robe qui a cassé internet » (réf. slide 15), il obtient RGB(132, 145, 187). Le débat porte sur la couleur perçue sur l'image : est-elle blanche et dorée ou bleu et noire ? En passant dans le contexte informatique, RGB substantive et catégorise déjà, le désaccord ne peut plus avoir lieu – un des trois canaux est « donné à voir » comme dominant. Autrement dit, la

catégorisation en couleur n'est plus en question. Le système RGB est un modèle de description d'une couleur qui fait partie du contexte informatique. Cette représentation n'est pas adaptée à un contexte plus quotidien, sans un ordinateur à disposition. Pour le choix de la couleur d'un meuble ou d'un mur, le code de couleur RGB n'est pas transposé dans le contexte du magasin. C'est ainsi que le conseiller présentera un nuancier avec des noms associés à chaque couleur pour les différencier et désigner précisément celle souhaitée. Le système RGB, qui est une représentation de la couleur, n'a pas été transposé dans le contexte du magasin de façon linéaire, il a été adapté à la situation en une autre représentation qui est le nuancier. Un modèle ne peut donc pas être transposé d'un contexte à un autre sans une adaptation. Selon le contexte donc, la description d'une couleur va être exprimée différemment. Si deux personnes discutent de la fameuse robe, l'un peut la qualifier comme bleu marine alors que l'autre soutiendra qu'il la voit beaucoup plus claire, voire blanche. Avec un nuancier, les personnes peuvent pointer la couleur et nommer la teinte observée. La distance des deux couleurs se mesure alors par le nombre de teintes les séparant sur le nuancier. En informatique, le fait d'avoir des nombres pour exprimer une couleur encourage l'emploi d'une distance mathématique comme la distance euclidienne par exemple. Les deux représentations ont été saisies différemment, mais ont permis d'exprimer une distance qui a fait sens en contexte.

Transdisciplinarité

Étant donné cette prise de position – un modèle est pertinent en contexte et une re-modélisation est nécessaire dans tout autre – une des singularités de Clay réside dans le fait que le résultat produit s'inspire en premier lieu de travaux d'autres disciplines que l'informatique.

Bio-mimétisme – Clay inclut par exemple un pré-traitement simulant les strates d'une rétine humaine. Ceci sous-entend par exemple que modèles neurophysiologiques de rétine humaine ont inspiré le développeur (i.e. la figure du développeur) et que ce dernier a retranscrit ces modèles, puis les a adaptés pour qu'ils puissent s'appliquer sur des images en RGB (réf. slide 16). Par la suite, les couleurs corrigées ont permis une segmentation de la main plus efficace. Cette modélisation spécifique au champ informatique (et plus particulièrement au traitement de l'image) est accompagnée d'autres exigences que le système RGB. En effet, pour que Clay soit exploitable par d'autres entités, la captation de la main doit

être robuste. Étant généralement embarqué sur smartphone, Clay suppose une économie des ressources internes au *device*. En particulier, les résultats en performance ont un impact direct sur la qualité du rendu d'affichage : si les algorithmes utilisés sont trop gourmands, le traitement d'image tombe en dessous de 60 FPS, et le rendu graphique n'est plus fluide (latence et décrochage), ce qui aboutit à une désynchronisation – un décalage entre la main réelle ressentie et la main affichée dans l'environnement virtuel. La solution Clay a pour obligation de présenter un résultat « satisfaisant » (en adéquation monde perçu/monde représenté) en temps réel, et ce, peu importe l'hostilité des contextes dynamiques. Les calculs doivent être minimalisés pour économiser les ressources et prendre le moins de temps possible pour être effectués. Sans l'aboutissement de ces objectifs, Clay n'est plus cet objet saisi oublié, cet étant qui n'a d'existence que dans son rapport fonctionnel, qui prolonge les capacités cognitives de l'utilisateur. Si le système est défectueux (casse le lien virtuel/réel) en latence ou en adéquation, qu'il ne joue plus son rôle de prothèse, alors l'usager se rendra compte de son existence, le conscientisera, et par la même modifiera son rôle (d'outil saisi à outil à saisir). Par conséquent, l'utilisateur se sépare de l'objet et celui-ci n'est plus constituant pour sa perception.

Code et Coding

Le développeur est généralement considéré comme le « traducteur » en code (vers le code) de ces modélisations qui proviennent de domaines hétérogènes — le code est en effet traditionnellement perçu comme une simple série d'instructions exécutées par la machine. Le développeur va ainsi lire une modélisation biologique du fonctionnement de la rétine pour le retranscrire afin de simuler ce traitement sur une image en RGB. Le code informatique est positionné en tant que produit de cette transcription, ayant pour seule fonction d'obtenir de l'ordinateur le comportement attendu. Cependant, de la même façon que le crayon conditionne la perception de l'enfant, la modélisation informatique influe sur la façon dont le développeur se saisie de représentations provenant d'autres contextes. Par exemple, un flou habituellement obtenu par l'usage d'une lentille, ou l'éclaircissement d'une photo réalisé par des manipulations de la lumière, seront perçues par une personne aguerrie à la *computer vision* comme des algorithmes appliqués sur un ensemble de pixels. Le code existant devient à son tour constitutif de la perception du développeur. Alternativement prisme sur des représentations externes et moyen d'agir sur le comportement de la machine, le code se retrouve en

position d'outil. Dépassant le statut de produit, il est en position de médiation — constitutif de la cognition du développeur et constituant le fonctionnement de l'ordinateur. On analyse et on pense les évolutions d'un programme en fonction du code existant. Conjointement, le produit de ces évolutions (le résultat cristallisé d'un processus d'inscription codique) va se faire par une modification de ce même code. Ceci entraîne de fait un changement de la perception, qui a elle-même une incidence sur la poursuite de la modélisation : le code est ainsi le moteur de sa propre évolution.

Le développeur a pour prisme ou focale le code qu'il écrit. De ce fait, le type d'appropriation qu'il opère (une représentation mathématique, etc.) à travers ce prisme l'influence directement. Ce cycle peut être illustré par l'intégration dans le code d'éléments relatifs à l'interprétation des formes, *e.g.* les lois de la *Gestalt Theory* (théorie de la forme en psychologie). La théorie de la *Gestalt* est interprétée, ressaisie, par le biais du code. Le développeur remodélise les principes qui lui sont pertinents dans son contexte technique (ex. images, pixels, kernels, etc.). Cette remodélisation est une modification du code et, donc, de son prisme de perception. La théorie de la *Gestalt* explique que dans l'acte de perception nous ne faisons pas que juxtaposer une foule de détails, mais nous percevons des formes (*Gestalt*) globales qui rassemblent les éléments entre eux. Prenons l'exemple d'une mélodie : on se souvient d'une structure globale de musique et non d'une suite successive de notes prises isolément. Dans le contexte informatique, une forme est déterminée par ses parties (ses pixels) et son entour (l'ensemble des points qui représentent le contour). Comme le développeur voit une image comme une grille en RGB, il peut se saisir des lois de Wertheimer (réf. slide 22) pour regrouper un amas de pixels en une forme harmonieuse. Par exemple, la loi de proximité encourage à grouper les pixels qui sont proches entre eux pour créer une même unité. Cette proximité peut être modélisée par une distance euclidienne entre les coordonnées (X,Y) de deux points. Une fois modélisée, cette notion de distance comme une distance euclidienne est cristallisée puis ressaisie pour faire sens de la loi de similarité. Ainsi, deux pixels avec deux couleurs proches (distance euclidienne des composantes RGB) sont considérés comme similaires.

Réfactorisation et programmation collective

Dans la pratique, la re-modélisation sous forme de code va également s'accompagner d'une adaptation du modèle originel. En effet, chaque formalisme possède ses propres contraintes et conventions qui doivent être respectées pour en faciliter la ressaisie. Dans l'exemple d'une formule mathématique (slide 24), en plus de l'utilisation d'instructions machine correspondant aux opérateurs arithmétiques, il est incontournable de rendre le nom des variables pertinent dans leur contexte d'usage. Ici on calcule la longueur des doigts pour trouver le plus long, donc les noms abstraits issus des mathématiques sont à cet emploi : « p » devient « tip », « q » devient « base » et « d » est libellé « fingerLength ». Ce processus – le *refactoring* (réfactorisation ou réusinage)– consiste à transformer le code pour faciliter sa ressaisie sans altérer le comportement induit de l'ordinateur. Plus qu'une simple adaptation, il consiste à optimiser ou découper en modules le résultat de la modélisation. En poursuivant sur l'exemple de recherche du majeur, la version informatique se détache progressivement de la représentation mathématique en retirant la racine carrée de la formule, coûteuse en calculs et sans incidence sur les comparaisons. Si dans d'autres contextes les tailles des doigts doivent être de nouveau comparées, ce calcul parti d'une optimisation va, du fait de sa réutilisation, se cristalliser pour devenir un concept à part entière, une propriété du doigt (« squaredLength » dans l'exemple). De part cette individuation, il devient constitutif de la façon dont le développeur fait sens d'une distance, selon qu'elle soit utilisée pour une comparaison ou non.

Avec la succession de ces cycles d'intégration de nouvelles fonctionnalités et de *refactoring*, une constellation d'outils apparaît et constitue un véritable système sémiotique. Néanmoins, l'auteur d'un programme travaille rarement seul. Pour que le code écrit puisse être exploité par d'autres, le sens de cet ensemble complexe de représentations doit être partagé. Cette distribution de la signification des éléments constituant le code n'est possible que par la négociation. Les membres d'une équipe doivent pouvoir échanger pour stabiliser, cristalliser ce système sémiotique. Il existe des méthodes favorisant ces échanges. Ainsi une revue du code, où l'auteur explique ce qu'il a écrit aux autres développeurs et le refactorise si nécessaire. Le *pair programming* consiste, comme son nom l'indique, à programmer à deux, un développeur au clavier l'autre à la relecture. La négociation est ainsi réalisée en temps réel. Cette méthode peut être élargie à plus de deux et

inclure des membres de l'équipe aux compétences hétérogènes, on l'appelle dans ce cas *mob programming*.
Ainsi donc, que l'on soit développeur ou non, on interprète les représentations issues d'autres disciplines (sphères, champs, etc.) au travers du prisme (ou biais constitutif) qui modèle une perception nécessairement située. Cette « transposition » n'est pas tant une transposition qu'une remodélisation qui prend en compte l'ensemble des idiosyncrasies cible (par ex. les contraintes et spécificités du code et de son langage). Le calcul, le code, compris comme activité sémiotique, ne peut être entendu que comme une activité humaine, sujette à négociation, dont les items sont saisis et ressaisis en contexte, que celui-ci soit technique (outils) ou structurels (cadre d'attente).

Vidéo intégrale de l'intervention (séminaire Phiteco 2018)

+

https://youtu.be/3FDE6_Fhq5U