Au-delà des applications : Substrats et instruments d'interaction

Michel Beaudouin-Lafon

Au-delà des applications : Substrats et instruments d'interaction

Beyond Applications: Interaction Substrates and Instruments

MICHEL BEAUDOUIN-LAFON, Université Paris-Saclay, CNRS, Inria

Laboratoire Interdisciplinaire des Sciences du Numérique (LISN), France

This paper introduces a new interaction model based on the concepts of *interaction substrate* for organizing digital information and *interaction instruments* for manipulating these substrates. This approach makes the concept of application unnecessary. Instead, it leads to flexible and extensible environments in which users can combine content at will and choose the tools they need to manipulate it. We present Stratify, a proof-of-concept implementation that combines a data-reactive approach to specify relationships among digital objects with a functional-reactive approach to handle interaction. This combination enables the creation of rich information substrates that can be freely inspected and modified, as well as interaction instruments that are decoupled from the objects they interact with, making it possible to use instruments with objects they were not designed for. We illustrate the flexibility of the approach with several examples and present directions for future work.

CCS Concepts: • **Human-centered computing** → **Systems and tools for interaction design**; **HCI theory, concepts and models**; **Graphical user interfaces**; **User interface toolkits**.

Additional Key Words and Phrases: Interaction model, Conceptual model, Substrate, Instrument, Instrumental interaction, Reactive programming

Cet article présente un nouveau modèle d'interaction fondé sur les concepts de *substrat d'interaction* pour organiser l'information numérique et d'*instruments d'interaction* pour manipuler ces substrats. Cette approche permet de s'affranchir du concept d'application et de créer des environnements flexibles et extensibles dans lesquels les utilisateurs peuvent combiner à loisir les contenus et choisir les instruments qui leur conviennent pour les manipuler. Nous présentons Stratify, une implémentation de ces concepts qui combine une approche réactive aux données pour spécifier les relations entre les objets numériques et une approche fonctionnelle réactive pour gérer l'interaction. Cette combinaison permet de créer des substrats d'information riches qui peuvent être librement inspectés et modifiés, ainsi que des instruments d'interaction qui sont découplés des objets avec lesquels ils interagissent, ce qui permet d'utiliser les instruments avec des objets pour lesquels ils n'ont pas été conçus. Nous illustrons la flexibilité de l'approche à l'aide d'exemples et présentons des pistes de développements futurs.

Mots-clés additionnels : Modèle d'interaction, Modèle conceptuel, Substrat, Instrument, Interaction instrumentale, Programmation réactive

Reference:
This is the author's draft version of the work. It is posted here for your personal use. Not for distribution. The definitive version of record was published in IHM '23: Proceedings of the 34th Conference on l'Interaction Humain-Machine.
ISBN 978-1-4503-9825-1
https://doi.org/10.1145/3583961.3583968

1 INTRODUCTION

Les interfaces actuelles sont dans leur immense majorité fondées sur un même modèle, centré sur la notion d'*application* : une application regroupe un ensemble de commandes permettant de manipuler un type particulier de contenu, enregistré dans un format souvent propriétaire. Lorsque l'on utilise le contenu d'une application dans une autre, par exemple par

un copier-coller, celui-ci n'est pas mis à jour lorsqu'il est modifié dans l'application source. Pour éditer un contenu dans une autre application, il faut souvent convertir, voire recréer le document. Ce manque d'interopérabilité limite les utilisateurs aux fonctionnalités prévues par l'application, créant ainsi un *jardin privé* (« walled gardens ») autour de chaque application.

Le monde physique, lui, n'a pas d'« applications » : nous utilisons des objets et des outils soigneusement choisis pour chaque tâche spécifique, nous concevons et réaménageons de manière fluide nos espaces de travail pour nous adapter à l'activité en cours, et nous nous approprions les objets de notre environnement de manière parfois imprévue.

Afin de créer des interfaces plus adaptées aux besoins des utilisateurs et aux capacités humaines, nous proposons d'explorer de nouveaux modèles d'interaction qui répondent aux deux propriétés suivantes :

Flexibilité – Les utilisateurs doivent avoir le choix et ne pas être limités par les décisions des concepteurs : ils doivent pouvoir utiliser différents outils pour atteindre leurs objectifs de différentes manières, configurer leurs environnements pour qu'ils prennent en charge leurs fonctions préférées et créer plusieurs de ces configurations.

Extensibilité – Les systèmes doivent être ouverts et interopérables : les utilisateurs doivent pouvoir ajouter ou retirer des fonctionnalités selon leurs besoins, plutôt que d'être soumis aux mises à jour imposées par les éditeurs de logiciels. Des tierces parties doivent pouvoir fournir de nouvelles fonctionnalités ainsi que des interactions alternatives pour les fonctionnalités existantes. Les utilisateurs doivent pouvoir créer leurs propres extensions et reconfigurations et les partager avec d'autres.

Cet article introduit un modèle d'interaction qui répond à ces critères, fondé sur les concepts de *substrat d'interaction*, pour structurer l'information numérique, et d'*instrument interactif* [1, 4], pour interagir avec ces substrats. Nous présentons ensuite Stratify, une implémentation servant de preuve de concept de ce modèle conceptuel. Stratify combine une approche réactive aux données pour spécifier les relations entre les objets numériques avec une approche fonctionnelle réactive pour gérer l'interaction. Cette combinaison permet de créer des substrats d'interaction riches qui peuvent être librement inspectés et modifiés, ainsi que des instruments d'interaction qui sont découplés des objets avec lesquels ils interagissent, ce qui permet d'utiliser les instruments avec des objets pour lesquels ils n'ont pas été conçus. En guise de validation, nous illustrons la flexibilité et l'extensibilité de l'approche à l'aide d'exemples. Enfin nous comparons ces travaux avec l'état de l'art et concluons par des orientations pour les travaux futurs.

2 MOTIVATION

Notre objectif est de créer des systèmes interactifs qui tirent pleinement parti des compétences et capacités humaines, mais aussi des caractéristiques uniques de l'information numérique. Il s'agit de créer une sorte de « matériau numérique » qui, comme la matière physique, soit aisément compréhensible et appropriable de la même façon que nous comprenons et utilisons spontanément les lois de la « physique naïve » [21].

2.1 Des affordances aux substrats d'interaction

Selon James Gibson [14], les « *affordances* » sont les possibilités d'action d'un objet relativement au sujet (animal ou humain). Ces possibilités d'action dépendent des propriétés de l'objet. Lorsque ces affordances sont perçues, elles fournissent un moyen naturel de comprendre les possibilités de l'environnement. Par exemple, un objet solide posé sur le sol et présentant une surface horizontale à une hauteur d'une cinquantaine de centimètres permet à un adulte de s'asseoir. Eleanor Gibson a étudié le processus d'apprentissage perceptif [13] selon lequel les enfants et les adultes apprennent à reconnaître les affordances.

Dans le monde physique, nous percevons différents types de propriétés des objets afin d'en extraire leurs possibilités d'action. Nous reconnaissons leur matériau, leur forme, leur structure et leur relation avec d'autres objets, ce qui nous permet de nous les approprier pour de nouvelles utilisations. Par exemple, nous savons qu'une chaise est assez lourde pour maintenir une porte ouverte en cas de courant d'air : la chaise n'est pas vue comme un objet permettant de s'asseoir grâce à sa forme et sa résistance, mais comme un objet ayant une masse et une stabilité suffisantes. Si nous voulons tirer pleinement parti des affordances dans le monde numérique, nous devons reproduire ces différents niveaux de représentation. De plus, les possibilités d'action d'un objet numérique doivent être perceptibles, directement ou par le biais d'instruments appropriés, et ces qualités perçues doivent correspondre aux capacités réelles de l'objet. Par exemple, tout morceau de texte doit pouvoir être sélectionné pour pouvoir le copier-coller.

Le concept de *substrat d'interaction* est destiné à opérationnaliser cette notion d'affordance dans le monde numérique. Le terme de « substrat » fait référence à la notion de substrat en biologie (l'environnement qui permet à un organisme de se développer) ou en électronique (le support sur lequel un circuit électronique est imprimé)[1]. Un substrat d'interaction contient des informations numériques et définit leurs contraintes et leurs relations. Par exemple, le substrat d'un document tel que cet article contient du texte et des images, et impose des contraintes de mise en page (marges, pagination, etc.) ainsi que des relations entre des éléments de contenu telles que la numérotation des figures. Un substrat peut utiliser des informations provenant d'autres substrats et leur fournir des informations. Une collection de substrats

[1]L'une des définitions du Larousse est « Ce qui sert d'infrastructure à quelque chose ».

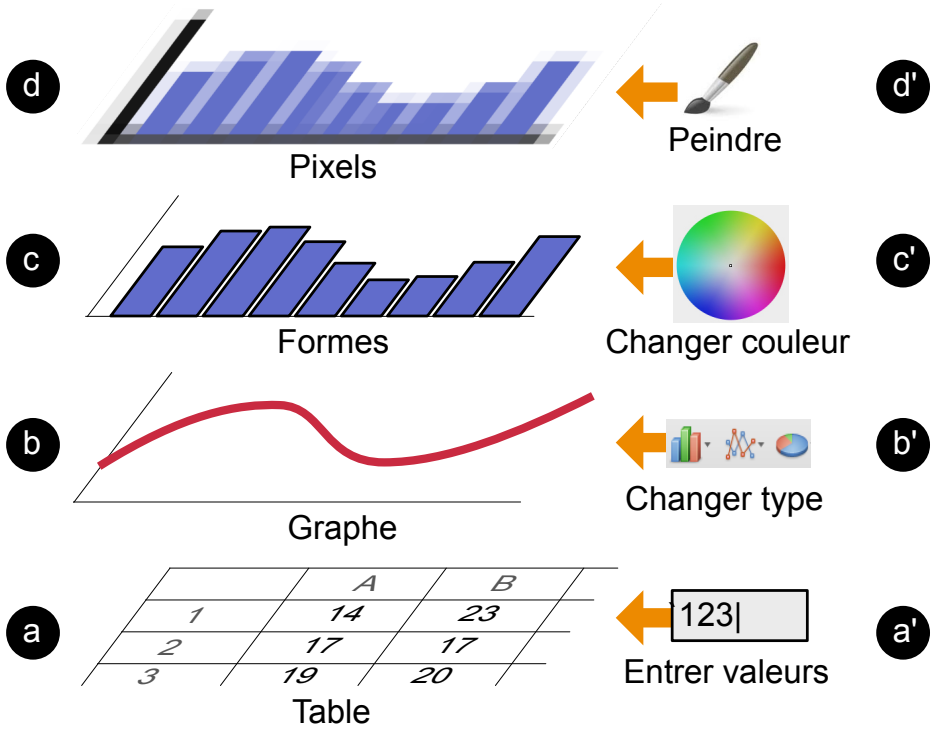

Fig. 1. Une pile de substrats et des instruments associés

crée une forme riche de matériau numérique, chaque substrat correspondant à un niveau de représentation d'un objet et à ses capacités correspondantes.

La partie gauche de la figure 1 montre un ensemble de substrats qui constituent un objet numérique. Le premier substrat (Fig. 1a) est une table de données. Ces données peuvent provenir d'un capteur (non représenté) qui ajoute régulièrement de nouvelles lignes à la table. Au-dessus (Fig. 1b), un substrat de graphe représente le contenu d'une colonne de la table sous une forme graphique. Ce graphe est ensuite représenté par un substrat contenant un ensemble de formes géométriques (des rectangles, Fig. 1c). Enfin, ces formes géométriques sont traduites en pixels affichés à l'écran par le dernier substrat (Fig. 1d). L'image que l'utilisateur voit à l'écran est donc celle d'un objet numérique formé d'une composition de quatre substrats interconnectés : la table de données, le graphe, les formes géométriques et les pixels.

Chaque substrat a son propre contenu et donc ses propres propriétés, qui forment la base des affordances que l'utilisateur peut exploiter, notamment via des instruments (voir ci-dessous). La perception de ces affordances est liée à la représentation visuelle utilisée. Par exemple, la perception du graphe évoque l'existence d'une table de données sous-jacente que l'on peut souhaiter vouloir modifier, plutôt que de simples rectangles. De même, la perception d'un texte évoque la possibilité de le sélectionner pour le copier, voire de l'éditer. Comme dans le monde physique, il peut se trouver de fausses affordances [12], par exemple lorsqu'un texte est en fait une image faite de pixels. Mais même dans ce cas, on peut imaginer un substrat qui implémente la reconnaissance de caractères pour rendre le texte de l'image sélectionnable en tant que tel.

2.2 Des outils aux instruments d'interaction

Une caractéristique remarquable de l'espèce humaine est sa capacité non seulement à utiliser des objets comme outils pour augmenter ses capacités phyisques, mais aussi à créer de nouveaux outils. Cette propension à créer et utiliser des outils a une base cognitive : lorsqu'une personne tient un outil en main, celui-ci devient une extension de son schéma corporel, comme s'il faisait partie de son corps [28]. Le fait de tenir un outil redéfinit donc les affordances de l'environnement, puisque l'outil modifie les capacités de la personne. Cette caractéristique fournit une base solide pour créer des systèmes numériques extensibles, dans lesquels de nouveaux outils numériques créent de nouvelles affordances.

François Osiurak [37, 38] suggère que l'utilisation d'outils fait appel au *raisonnement technique*, un type de raisonnement spécifique qui complète la perception directe de ce que l'outil peut faire. Le raisonnement technique repose sur des lois techniques abstraites qui, contrairement aux compétences acquises par apprentissage procédural, ne nécessitent pas de renforcement constant. Par exemple, on sait qu'un objet rigide et pointu peut percer un matériau plus tendre : un crayon peut percer une feuille de papier, un clou une planche de bois, mais un crayon ne peut pas percer une planche de bois. Le raisonnement technique permet d'expliquer l'appropriation spontanée d'objets en tant qu'outils, comme lorsque nous utilisons la pointe d'un couteau comme tournevis : au lieu de se focaliser sur la *fonction* du couteau, on se focalise sur ses *propriétés* (forme, résistance) et celles de la vis (forme de la tête) qui permettent de mettre en œuvre la loi technique abstraite « *imbriquer un objet dans un autre pour le faire tourner* ».

La plupart des logiciels interactifs exigent de nombreuses connaissances préalables, ce qui rend les outils logiciels difficiles à apprendre, à mémoriser et à maîtriser. Des fonctionnalités similaires sont souvent implémentées de façon ad hoc, différente d'une application à l'autre, et même parfois au sein d'une même application. Par exemple, le logiciel de traitement de texte Microsoft Word a des fonctions différentes pour changer la couleur de texte, de surlignage, de fond de document, de remplissage et de bord des formes graphiques. Ce manque de cohérence limite le transfert des

compétences acquises, mais aussi la possibilité d'utiliser le raisonnement technique pour résoudre des problèmes et découvrir des usages inattendus. Il existe cependant des cas d'appropriation d'objets numériques [9] et des travaux récents montrent que cette appropriation peut s'interpréter par une forme de raisonnement technique [41–43].

L'interaction instrumentale [1, 4] a introduit la notion d'outil, ou *instrument*, dans les environnements numériques. La notion de substrat introduite plus haut précise la nature des objets sur lesquels opèrent les instruments d'interaction : un instrument doit être applicable à tout substrat qui présente des propriétés appropriées. Un environnement qui combine substrats et instruments d'interaction permet alors une utilisation flexible et cohérente des outils, où ceux-ci peuvent être réutilisés dans différents domaines et où les utilisateurs peuvent raisonner sur les effets de l'application d'un outil dans différentes situations.

Ainsi, dans l'exemple de la figure 1, différents instruments peuvent intervenir à chaque niveau afin d'ajouter des données au tableau (a'), changer le type de graphique (b'), définir la couleur des barres (c') ou modifier les pixels pour dessiner des annotations (d'). En permettant à l'utilisateur d'interagir à ces différents niveaux, cette approche offre un niveau de flexibilité que les applications actuelles ne permettent pas. Par exemple, l'outil de changement de couleur peut fonctionner avec des substrats de texte, de formes graphiques, de fenêtres ou même des instruments tels que le pinceau sans que ceux-ci aient été explicitement prévus pour cela.

3 LE MODÈLE CONCEPTUEL DES SUBSTRATS

Les objets physiques sont *polyvalents* : ils sont utilisés pour leur *fonction*, mais aussi pour leurs *propriétés*. Par exemple, une feuille de papier est destinée à prendre des notes ou à faire un dessin (sa fonction). Mais elle peut aussi servir à allumer un feu parce que le papier est hautement inflammable, ou à caler une table parce qu'elle peut être pliée facilement (ses propriétés). Il en est de même pour les outils : une tasse est conçue pour boire, mais peut servir à empêcher des papiers de s'envoler par son poids ou à tracer rapidement un cercle par sa forme. Les humains apprennent ces affordances par différents moyens, de l'apprentissage perceptif [13] à la transmission culturelle, et s'approprient des objets pour les utiliser comme des outils grâce au raisonnement technique [37, 38].

Une telle polyvalence existe rarement avec les objets numériques, qui sont généralement réduits à leur rôle fonctionnel. Par exemple, les fenêtres sont des rectangles, mais on ne peut pas les tourner ou les aligner comme les formes d'un éditeur de dessins. Inversement, un rectangle dans un éditeur de dessin ne peut pas afficher le contenu d'une fenêtre. Pour rendre les objets numériques polyvalents, nous introduisons les *substrats d'interaction*.

3.1 Substrats d'interaction

Comme illustré à la figure 1, un objet numérique est constitué d'un ensemble de substrats qui correspondent à différents niveaux de représentation de l'objet, depuis les représentations proches de la machine (la table en bas de la figure) jusqu'aux représentations proches de la perception de l'utilisateur (les pixels en haut de la figure). Chaque substrat est un conteneur d'information, caractérisé par ses *propriétés*, par un ensemble de *contraintes* internes qui définissent sa structure, et par un ensemble de *relations* avec d'autres substrats qui assurent la cohérence de l'objet numérique. En particulier, le ou les substrats qui sont représentés par un substrat sont appelés ses *sources*.

Bien que la figure 1 représente une simple pile de substrats, dans le cas général, les relations entre substrats constituent un graphe sans cycle. Ainsi, un même substrat peut être la source de plusieurs autres substrats, ce qui permet de produire des représentations multiples cohérentes. D'autre part un substrat peut avoir plusieurs sources, ce qui permet de créer des objets composites.

Les objets numériques qui résultent de l'assemblage de substrats sont hétérogènes puisque les substrats qui les composent représentent des informations de nature différente, mais les relations qui les unissent assurent qu'ils forment un tout cohérent. Une autre façon de voir les choses est que les substrats d'un objet numérique sont autant de facettes de cet objet. La suite de cette section présente quelques exemples typiques de substrats, du « haut » vers le « bas » des niveaux de représentation.

Substrats de pixels – Les interfaces graphiques utilisent des représentations visuelles qui sont in fine affichées sur un écran formé de pixels. Les *substrats de pixels* sont donc la représentation visuelle « ultime » d'un objet numérique, dans le sens où ils constituent le lien entre la machine et la perception humaine. Conceptuellement, il s'agit de matrices 2D de pixels dont on peut changer la couleur individuelle par des opérations de dessin ou de filtrage. La même approche peut être appliquée à d'autres modes de perception des objets numériques, notamment tactiles, kinesthésiques et auditifs.

Substrats géométriques – Nous percevons rarement les pixels en tant que tels, mais plutôt des formes, des images, du texte et d'autres objets structurés. Par ailleurs, l'ordinateur ne manipule généralement les pixels qu'au dernier stade de son pipeline de rendu, lorsque les objets tels que le texte, les formes et les images sont transformés en pixels. Le niveau suivant de représentation d'une interface visuelle est donc constitué de graphiques structurés, où les objets géométriques sont disposés selon des contraintes de mise en page. Nous les appelons *substrats géométriques* pour souligner la nature géométrique à la fois des objets qui les peuplent et des contraintes de positionnement telles que l'alignement et la juxtaposition. Les graphes de scènes [47], le DOM[2] pour HTML[3] et SVG[4] sont des exemples de tels substrats.

Substrats structurels – Si un substrat géométrique peut être simplement une représentation de lui-même, par exemple une page Web statique ou un dessin à main levée, les formes qu'il contient sont souvent des représentations d'objets plus abstraits. Dans les interfaces graphiques, ces objets incluent les widgets tels que les boutons et les menus, ainsi que les objets du modèle de l'application. Dans le modèle classique MVC (Modèle-Vue-Contrôleur) [30], les objets du modèle sont souvent opaques et ad hoc. En revanche, nous voulons exposer leur structure et leurs propriétés internes afin qu'ils puissent être manipulés de différentes manières. Nous appelons donc les substrats de ces objets des *substrats structurels*.

Une structure très courante est la *séquence* ou liste ordonnée : un éditeur d'images gère une pile de calques, un éditeur de dessins une liste de formes, un gestionnaire de fenêtres un ensemble de fenêtres, un menu une liste d'éléments. Pourtant, chacune de ces applications représente la séquence différemment. En exposant leur structure commune, nous ouvrons la porte à d'autres représentations et manipulations. Par exemple, la table lumineuse que l'on trouve dans un éditeur de photos peut être utilisée pour présenter le contenu de n'importe quelle séquence, permettant aux utilisateurs de l'organiser comme ils le souhaitent. Les couches de l'éditeur d'images peuvent alors être visualisées séparément au lieu d'être superposées, les fenêtres du gestionnaire de fenêtres peuvent être étalées comme dans Apple Mac OS X Exposé au lieu de se chevaucher. Une séquence peut également être représentée sous la forme d'une liste linéaire de noms ou d'icônes : nous obtenons une liste de calques, une liste de titres de fenêtres et le menu linéaire familier.

D'autres structures courantes sont les arbres, les graphes et les tables, qui peuvent être mis en correspondance avec des représentations bien connues telles que des listes indentées pour les arbres, des diagrammes de liens ou des matrices pour les graphes, des grilles pour les tableaux, ou des représentations plus spécialisées si nécessaire. Les substrats

[2]https://dom.spec.whatwg.org
[3]https://www.w3.org/html/
[4]https://www.w3.org/Graphics/SVG/

structurels peuvent être imbriqués et hétérogènes : une table peut contenir un arbre dans une cellule, et une image dans une autre.

En combinant des substrats structurels liés les uns aux autres, on peut produire des représentations alternatives qui restent cohérentes entre elles. Par exemple, une séquence peut représenter les feuilles d'un arbre, ou la version triée d'une autre séquence. La combinaison de ces substrats facilite la réutilisation de représentations de niveau supérieur, comme l'utilisation de la table lumineuse décrite ci-dessus pour afficher le contenu d'un arbre, ou l'affichage d'une liste de titres de fenêtres par ordre alphabétique.

Substrats de données – Les substrats structurels doivent in fine faire référence à des données réelles. Ces données proviennent souvent de sources externes, telles que le contenu d'un fichier ou d'une base de données, un capteur ou un service réseau, et peuvent donc être dynamiques. Nous les appelons *substrats de données*. Lorsque de nouvelles données sont fournies au substrat de données, les substrats de niveau supérieur qui en dépendent sont mis à jour, jusqu'au substrat de pixels (Fig. 3a).

Le modèle conceptuel qui ressort de cette analyse est que les objets numériques sont constitués d'une combinaison de substrats, liés entre eux pour maintenir leur cohérence. Les substrats de niveau supérieur sont des *représentations* de ceux de niveau inférieur, créant ainsi une pile de représentations entièrement inspectable. Les *sources* d'un substrat sont les substrats dont il dépend, c'est-à-dire ceux qu'il représente. De nouvelles représentations peuvent être créées dynamiquement, à n'importe quel niveau, ce qui permet d'aborder un objet à différents niveaux de représentation. La possibilité de combiner les substrats existants de différentes manières favorise la *flexibilité* : les utilisateurs ne sont pas liés aux choix faits par les concepteurs et les programmeurs. Le fait que les substrats puissent être inspectés favorise l'*extensibilité* : de nouvelles représentations, de nouvelles sources de données et de nouveaux substrats peuvent être injectés dans un environnement existant.

3.2 Capacités d'action

Les substrats permettent de représenter l'information numérique d'une manière riche et polyvalente. Ils réagissent aux changements de leurs sources de données pour maintenir leur cohérence, mais le réseau de dépendances est statique. Pour modifier les substrats et leurs relations, il faut définir la façon dont les actions de l'utilisateur sont interprétées et exécutées par le système. Nous appelons *capacités d'action*[5] la façon dont les substrats *expriment* les actions possibles et *mettent en œuvre* leurs effets.

Les utilisateurs ne perçoivent que les représentations au niveau de la surface de la pile de substrats, c'est-à-dire le substrat de pixels dans une interface visuelle. Les actions de l'utilisateur exprimées à ce niveau, comme le pointage et le cliquer-tirer, doivent être interprétées et transmises aux niveaux inférieurs afin d'être traitées au niveau de représentation approprié. Par exemple, le déplacement de l'icône d'un fichier dans une fenêtre n'affecte pas le système de fichiers et peut être interprété par le substrat géométrique. Mais le déplacement de l'icône d'un fichier d'une fenêtre à l'autre nécessite de changer l'emplacement du fichier dans le système de fichiers, et doit être transmis jusqu'au substrat de données, qui l'interprétera comme une opération du système de fichiers. Bien entendu, les utilisateurs n'ont pas à connaître ces détails.

Protocoles d'interaction – Afin de découpler les deux aspects des capacités d'action, l'expression et l'exécution des actions, nous introduisons les *protocoles d'interaction* (Fig. 2). Le substrat qui émet une action est son *émetteur*, celui qui l'exécute est sa *cible*. Un protocole d'interaction décrit comment une action particulière, par exemple *glisser*, est

[5]Cette expression est notre traduction du terme anglais « agency ».

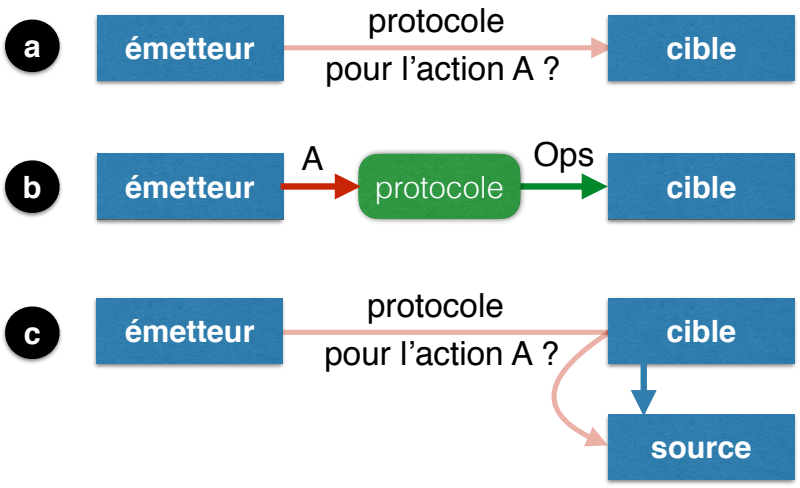

FIG. 2. Détermination d'un protocole d'interaction : (a) l'émetteur interroge la cible qui (b) retourne un protocole qui sait transformer l'action A en opérations Ops sur la cible, ou bien (c) interroge son substrat source.

transformée en une ou plusieurs opérations sur la cible, par exemple, la déplacer. Les actions peuvent être spécifiées à n'importe quel niveau d'abstraction. Par exemple, un substrat peut émettre une action de tri vers un substrat structurel pour modifier l'ordre de ses éléments.

Lorsque la cible d'une action n'a pas de protocole correspondant à cette action, l'action est transmise à sa source. C'est ainsi que la modification du nom d'un fichier est transmise jusqu'au substrat de données représentant le fichier, alors que le déplacement de l'icône d'un fichier dans une fenêtre est transmis uniquement au substrat géométrique. Lorsqu'un substrat cible exécute une action et modifie son état, les substrats de niveau supérieur se mettent à jour pour maintenir la cohérence avec le nouvel état (Fig. 3b). Ce processus est similaire à la mise à jour qui se produit lorsqu'une source de données change, sauf qu'il ne commence pas tout en bas de la pile de substrats. Un substrat peut cependant *bloquer* la propagation d'une action vers sa source (voir ci-dessous).

Les actions possibles d'une cible sont définies par un dictionnaire dont les clés sont les noms des actions et les valeurs sont les protocoles correspondants. De plus, toutes les cibles répondent à l'action consistant à changer la valeur d'une propriété si le substrat définit cette propriété et qu'elle est modifiable. Si la recherche de protocole échoue, l'émetteur peut faire d'autres essais. Par exemple, pour changer la couleur de la cible, il peut d'abord chercher un protocole pour l'action « `SetColor` » et en cas d'échec pour l'action « `SetProperty` » de la propriété « `Color` ». En cas d'échec de toutes les tentatives, l'action est simplement ignorée. La flexibilité du découplage entre expression et exécution des actions dépend donc de conventions de nommage des actions et des propriétés.

Blocage et relais des actions – La mise en œuvre d'une action à un certain niveau de représentation ne modifie pas nécessairement son substrat directement. En effet, de nombreuses propriétés d'un substrat dépendent de leur source et ne doivent pas être modifiées directement. Par exemple, considérons une forme dans un substrat géométrique dont la couleur est liée à la propriété « température » de son substrat de données source. Une action de changement de couleur de la forme devrait être bloquée non pas parce qu'il n'est pas possible de changer la couleur de la forme, mais parce qu'elle est liée par une dépendance au substrat de données sous-jacent.

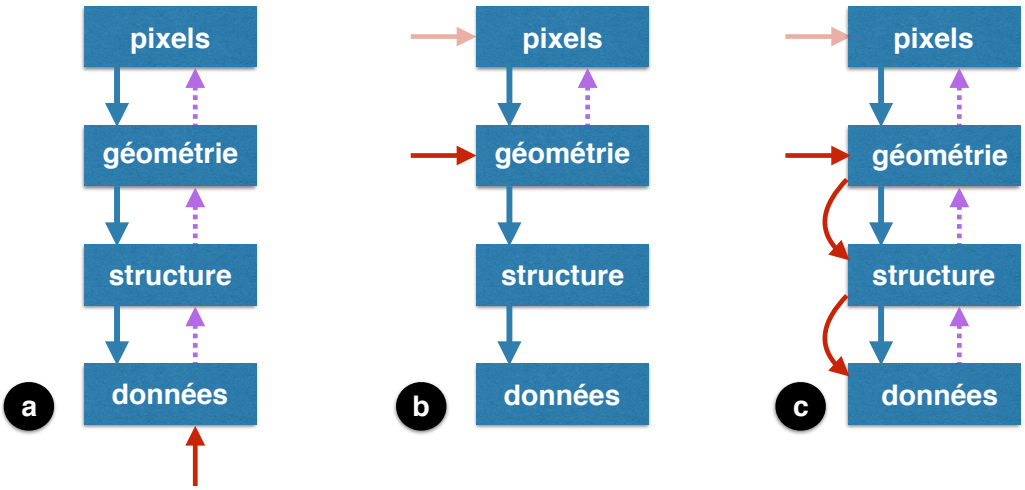

Fig. 3. Processus de mise à jour lorsque (a) un substrat de données change, (b) une action est transférée à un niveau inférieur et (c) une action est relayée. Les substrats sont en bleu, avec des flèches bleues décrivant leur source. Les actions sont en rouge, avec l'action originale qui a causé le transfert en rose. Les flèches mauve en pointillé indiquent la propagation de la mise à jour dans les substrats supérieurs.

Considérons maintenant la forme représentant l'aiguille des minutes d'une horloge analogique, dont l'angle dépend de la propriété « minutes » de la source de temps affichée par l'horloge. Pour changer l'heure en tournant l'aiguille de l'horloge, celle-ci doit fournir un protocole d'interaction pour l'action de glisser. Au lieu d'interpréter cette opération de glisser comme un changement de l'angle de la forme, elle l'interprète comme un changement du nombre de minutes de sa source, et émet l'action correspondante à la source. Nous appelons cette opération le *relais* d'une action (Fig. 3c). En supposant que la source de temps accepte l'action, elle met à jour son état, ce qui met à jour l'horloge analogique — ainsi que toute autre représentation qu'elle peut avoir. Si elle bloque l'action, faire glisser l'aiguille n'aura aucun effet.

Actions forcées – Nous pouvons également permettre aux utilisateurs d'expérimenter et de prendre des risques en *forçant* certaines dépendances, de la même façon que nous utilisons parfois un objet physique d'une manière non prévue. Pour des raisons de sécurité, ceci ne devrait être activé que dans un mode contrôlé par l'utilisateur. Dans ce mode, l'utilisateur peut contourner le blocage des actions et forcer l'attribution d'une propriété, même si elle dépend d'un substrat source. Par exemple, l'utilisateur peut forcer le changement de couleur de la forme représentant la température, rompant ainsi le lien représentationnel avec la température source. Dans l'exemple de l'horloge, l'utilisateur pourrait déplacer l'aiguille des minutes n'importe où sur l'écran, brisant ainsi la disposition de l'horloge. Comme c'est l'angle de l'aiguille qui est lié à la source de l'heure, l'aiguille déplacée continuerait néanmoins à tourner et à refléter l'heure.

En résumé, la capacité d'interagir avec différents niveaux de représentation d'un objet repose sur deux concepts :

(1) Les protocoles d'interaction, qui permettent de trouver le substrat pour lequel l'action a un sens ; et
(2) Le relais d'action, qui permet d'interpréter des actions concrètes en actions abstraites, comme le déplacement de l'aiguille de l'horloge pour la mise à jour de l'heure, en traversant les couches d'abstraction des substrats successifs.

Ces deux mécanismes garantissent que tous les niveaux de représentation d'un objet numérique sont potentiellement exposés et disponibles pour l'interaction, offrant à la fois flexibilité et extensibilité. La flexibilité résulte de la possibilité

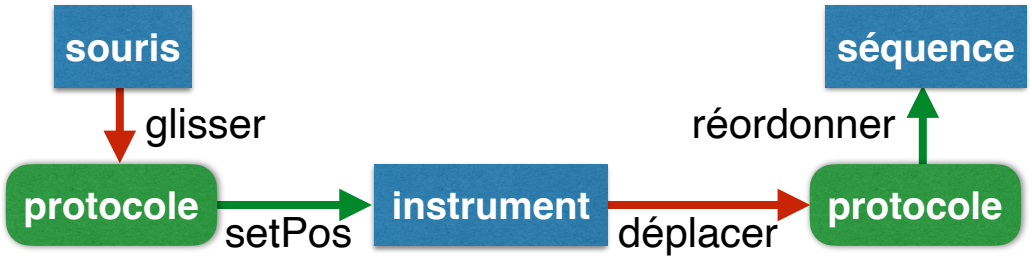

FIG. 4. L'instrument de déplacement reçoit l'action `glisser` de la souris, qui met à jour sa position, et émet une action `déplacer` qui est interprétée par la séquence comme un réordonnancement de l'élément déplacé.

de réutiliser les interactions et les protocoles existants, tandis que l'extensibilité résulte de la possibilité d'en créer de nouveaux qui peuvent interagir avec ceux qui existent déjà.

Afin de tirer pleinement parti de ces propriétés, les concepteurs doivent s'efforcer de créer des substrats aussi ouverts que possible aux actions entrantes en définissant des protocoles appropriés. Les concepteurs doivent également décomposer les substrats complexes en substrats plus simples et plus génériques, afin de faciliter la réutilisation et d'encourager les recombinaisons. Par exemple, en utilisant systématiquement des substrats textuels, tout texte sera éditable sans préjuger des instruments qui seront utilisés pour l'édition. De même, en utilisant systématiquement des substrats de type séquence lorsque l'on a une liste d'objets, on ouvre la possibilité d'insérer des substrats permettant de trier ou de filtrer la séquence.

3.3 Instruments d'interaction

Dans le monde physique, nous agissons sur les objets soit directement, par exemple en pliant une feuille de papier, soit par le biais d'outils, par exemple en écrivant sur une feuille de papier avec un crayon. L'utilisation d'un objet comme outil est le résultat d'un double processus [5, 40]. L'*instrumentation* est le processus par lequel l'utilisateur s'adapte à l'outil et l'extension du schéma corporel qui en résulte : l'outil devient une extension du corps de l'utilisateur et augmente ses capacités ; L'*instrumentalisation* est le processus par lequel un artefact est transformé en outil, éventuellement par adaptation, une fois que l'utilisateur a déterminé que l'artefact a les propriétés requises pour la tâche à accomplir, notamment par le biais du raisonnement technique [39]. Mackay [32] décrit le phénomène plus général de la *co-adaptation*, par lequel les utilisateurs adaptent et s'adaptent à la technologie.

Si la co-adaptation est courante dans le monde physique, les outils numériques peuvent rarement être utilisés autrement que ce pour quoi ils ont été conçus. Des formes limitées d'instrumentalisation existent, par exemple, lorsque l'on crée un rectangle dans un outil de dessin pour mesurer et dupliquer l'espacement entre deux objets. En exposant les différents niveaux de représentation de l'information numérique grâce aux substrats et en opérationnalisant les capacités d'action du système avec la notion d'*instrument*, nous rendons possibles des formes plus générales de co-adaptation.

Un instrument est un substrat capable d'agir sur un autre substrat en émettant des *actions* (Fig. 4). Il est contrôlé directement ou indirectement par l'utilisateur via les dispositifs d'entrée. Les dispositifs d'entrée sont des substrats de données. Par exemple, une souris fournit des données sur ses mouvements et l'état de ses boutons, et contrôle la position d'un curseur, tandis qu'un écran tactile fournit des informations sur les points de contact. Les périphériques d'entrée émettent des actions qui représentent des actions physiques de l'utilisateur telles que cliquer, glisser ou pincer. Les cibles de ces actions sont un ou plusieurs instruments, c'est-à-dire des substrats qui comportent un protocole pour

ces actions. Les instruments transforment ces actions en actions de plus haut niveau. Par exemple, la séquence formée d'un appui sur un bouton, de mouvements de la souris et du relâchement du bouton est transformée en une action de déplacement. Ils déterminent également quelle(s) cible(s) doivent recevoir ces actions. Par exemple, certaines actions de la souris s'appliquent à l'objet situé sous le curseur, d'autres à la sélection. L'instrument doit donc interroger les divers substrats pour identifier la cible de l'action de sortie de l'instrument.

Ce modèle permet en théorie à plusieurs utilisateurs d'interagir simultanément. Cependant, les substrats doivent être préparés à combiner des actions simultanées ou à les bloquer.

Les concepteurs doivent s'efforcer de créer des instruments utilisables avec différents types de substrats, par exemple pour éditer des types de données de base comme du texte, des nombres, des couleurs et des polices, ou pour déplacer et transformer des formes. Le même instrument devrait permettre de faire pivoter une forme graphique dans un éditeur de dessins, une fenêtre dans un gestionnaire de fenêtres ou un clip dans un montage vidéo. Des instruments génériques devraient permettent de manipuler des substrats structurels, tels que trier ou filtrer des séquences, ajouter des métadonnées, effectuer des recherches, ou ajouter des annotations.

Les instruments offrent à l'utilisateur flexibilité et extensibilité : les instruments bien conçus sont polymorphes [4] et peuvent être utilisés avec différents types de substrats, même s'ils n'ont pas été conçus pour eux ; Les utilisateurs peuvent remplacer les instruments par ceux de leur choix, par exemple pour utiliser leur sélecteur de couleurs préféré, et ajouter de nouveaux instruments, par exemple un outil avancé pour aligner et distribuer les objets [7].

3.4 Discussion

Le modèle conceptuel des substrats remet en cause le modèle des applications sans pour autant être incompatible avec celui-ci : une application peut très bien être construite avec des substrats et des instruments et se présenter comme une application classique. Cependant, un environnement entièrement conçu à base de substrats et d'instruments ouvre des possibilités bien plus intéressantes.

Prenons l'exemple de la gestion de fichiers, de messages de courrier électronique, de musique et de photos. Actuellement, chacune de ces activités est gérée par une application dédiée. Cependant, les données manipulées ont certaines similarités : elles gèrent des collections d'objets (fichiers, emails, morceaux de musique, photos) ayant certaines propriétés et organisées en listes. Dans les applications actuelles, l'utilisateur peut souvent faire certaines opérations dans une application mais pas dans d'autres alors qu'elles seraient pertinentes : choisir l'ordre de tri des listes, visualiser leur contenu sous forme de liste, de grille, ou de façon libre, faire des recherche, filtrer selon plusieurs critères, définir des dossiers dynamiques, etc. Dans un environnement centré sur les substrats et les instruments, ces fonctionnalités seraient fournies par des instruments opérant sur n'importe quel substrat permettant l'accès à des listes et aux propriétés de leurs objets pertinentes pour la recherche, le tri et le filtrage. Cette mise en facteur des mêmes outils rend l'interaction plus simple, car l'utilisateur n'a pas à apprendre les commandes spécifiques à chaque applications, et plus puissante, car les fonctionnalités peuvent être étendues en important de nouveaux outils et de nouveaux substrats. Ainsi, si l'utilisateur importe un nouveau substrat pour créer des présentations, qui sont des listes de diapositives, il pourra utiliser ces mêmes outils.

Cet exemple illustre la façon dont le modèle proposé peut être utile aux concepteurs d'applications interactives. Au lieu de concevoir une application comme un ensemble de fonctionnalités, il s'agit d'identifier les substrats et les instruments capables de répondre aux besoins des utilisateurs. Cette approche a déjà fait ses preuves dans plusieurs travaux antérieurs de notre équipe, qui ont soit contribué à élaborer ces concepts, soit s'en sont directement inspiré : les « paper

substrates » [11], les Webstrates [29], les substrats pour la création graphique [24, 34], les substrats narratifs [19], les dépendances entre fichiers [18], les représentations transitionnelles [15, 16], les « Textlets » [42] et les « Passages » [20].

D'autre part, nous enseignons depuis plusieurs années avec succès ce modèle en Master d'IHM[6]. En suivant l'approche de théorie générative [3], les étudiants doivent d'abord *analyser* puis *critiquer* des interfaces existantes en termes de substrats et d'instruments (ou de l'absence de ceux-ci), puis *générer* de nouvelles solutions en identifiant des substrats et instruments appropriés. Notre expérience montre que les étudiants s'approprient ces concepts et, grâce au prototypage rapide, créent des solutions qui combinent puissance d'expression et simplicité d'utilisation.

Enfin, de même que le modèle de l'interaction instrumentale a déjà démontré sa capacité à décrire l'existant, à définir des critères d'évaluation et à générer de nouvelles solutions [1], la notion de substrat permet d'analyser les contenus manipulés par les applications classiques en termes de substrats [2]. Par exemple, un tableur est un substrat qui présente une grille de cellules contenant des valeurs. Certaines de ces valeurs sont calculées par des formules qui font référence aux valeurs d'autres cellules. Ces dépendances forme un second substrat qui est la source du substrat des cellules. Dans un tableau classique, la seule propriété d'une cellule qui peut être décrite par une formule est sa valeur. Une extension consiste à autoriser l'utilisation de formules pour calculer, par exemple, la couleur d'une cellule en fonction de sa valeur ou la largeur d'une colonne en fonction de son contenu, ou pour alterner les couleurs des lignes successives. Cet exemple donne un aperçu des capacités analytiques, critiques et génératives [3] du modèle des substrats.

Il n'en reste pas moins que la validation de ce modèle passe également par la réalisation d'une implémentation informatique qui démontre sa faisabilité. C'est l'objet du prototype décrit dans la prochaine section.

4 PREUVE DE CONCEPT : STRATIFY

Stratify est un prototype destiné à démontrer et expérimenter les concepts de substrats et d'instruments d'interaction. Il s'agit d'une application Web écrite en TypeScript qui fonctionne dans n'importe quel navigateur Web moderne. Stratify utilise le DOM[7], qui est la représentation en mémoire du contenu d'une page Web, comme substrat géométrique à la fois pour le contenu HTML, pour la mise en page du document, et le contenu SVG, pour les graphiques vectoriels. Par conséquent, le substrat de pixels est géré par le navigateur web et n'est pas accessible. Nous laissons à des travaux futurs le soin de créer un substrat de pixels basé, par exemple, sur le Canvas HTML ou sur WebGL.

4.1 Substrats : Programmation réactive aux données

Un substrat d'interaction dans Stratify est un ensemble d'objets et de propriétés liés par des relations. Par exemple, une horloge analogique est composée de formes graphiques, le cadran et les aiguilles, liées à un substrat représentant le temps sous forme d'heure, minutes et secondes, lui-même lié à une source de données issue d'une horloge en millisecondes. Afin de gérer ces relations de manière déclarative, nous utilisons un modèle de programmation que nous appelons *Data-Reactive Programming*, ou programmation réactive aux données, dont le principe est proche des modèles de calcul à flot de données (dataflow) utilisés notamment dans les tableurs. Stratify utilise la bibliothèque `mobx`[8].

Les objets exposent leur état comme un ensemble de propriétés, dont certaines sont calculées à partir d'autres propriétés du même objet ou d'autres objets. Ces *propriétés calculées* peuvent être lues mais pas modifiées directement par défaut. Les propriétés non calculées sont des *propriétés internes* : elles appartiennent à l'objet et peuvent être

[6]Module « Fundamentals of Situated Interaction » de la spécialité IHM du Master d'Informatique de l'Université Paris-Saclay), co-enseigné avec Wendy Mackay.
[7]ou plutôt le DOM virtuel Inferno : https://infernojs.org/
[8]https://mobx.js.org

modifiées directement. Le modèle réactif garantit que chaque fois que la valeur d'une propriété change, les propriétés calculées qui en dépendent sont recalculées. En cas de cycles de dépendance, des itérations successives sont calculées jusqu'à ce que les valeurs se stabilisent ou qu'un nombre maximal d'itérations soit atteint[9]. En pratique, en supposant que les propriétés calculées ne dépendent que des substrats sources, les cycles ne se produisent pas car les sources forment un arbre ou un graphe sans cycle.

Dans l'exemple précédent, les aiguilles de l'horloge analogique sont des segments de droite dont l'angle de rotation est une propriété calculée à partir de l'heure stockée dans le substrat de temps. Le nombre d'heures, de minutes et de secondes de ce substrat sont eux-mêmes des propriétés calculées à partir de la source de données de l'horloge. Par conséquent, l'horloge analogique reflète constamment l'heure actuelle, sans aucune autre programmation (Listing 1)[10]. Si un rendu textuel du substrat de l'heure est géré par un autre substrat, il sera lui aussi mis à jour en temps réel.

Le réseau de dépendances créé par les propriétés calculées ressemble aux formules d'un tableur : le changement de valeur d'une cellule déclenche le calcul des formules qui en dépendent, mettant à jour leurs valeurs. Dans Stratify, les cellules ne sont pas organisées dans un tableau mais sont des propriétés d'objets arbitraires.

Bien que ce modèle de calcul ne soit pas nouveau, nous y ajoutons une particularité qui le distingue des autres modèles réactifs. Le programmeur peut définir un *setter* pour une valeur calculée, c'est-à-dire ce qu'il faut faire lorsqu'on change la valeur d'une propriété calculée. Dans l'exemple de l'horloge, pour régler l'heure en tournant l'une des aiguilles, il faut définir un *setter* pour la propriété `minutesAngle` qui calcule le nombre de minutes en fonction du nouvel angle et définir la propriété `minutes` de la source en conséquence (Listing 2, lignes 2–4). Comme la source de temps dépend d'une source de données `AtomicClock`, nous avons également besoin d'un *setter* pour sa propriété `minutes`, sinon rien ne se passera car l'action sera bloquée (Listing 2, lignes 6–8). Ce *setter* déclenche une mise à jour des propriétés calculées qui en dépendent, c'est-à-dire les substrats de l'heure et de l'horloge analogique.

Le listing 3 est une collection de substrats structurels qui représentent des séquences d'objets. Le substrat de base `Sequence` est un tableau. Le `Sorter` et le `Filter` recalculent leur séquence à la volée en la triant ou en la filtrant, tandis que le `Mapper` calcule une séquence en transformant chacun des éléments de la séquence source. Ces substrats sont utilisés dans le listing 4 pour créer une liste de tâches dont les éléments sont triés, filtrés pour éliminer ceux qui sont cochés, et mis en majuscules.

Le fait de recalculer la séquence entière chaque fois que la source change dans le `Mapper` peut poser un problème car les objets résultants sont recréés à chaque fois. Dans l'exemple de la liste des tâches, une liste de chaînes de caractères est transformée en une liste d'éléments `Todo` en créant un substrat qui contient la propriété interne `checked` pour chaque élément. Si la liste des tâches était recréée à chaque fois que la liste des chaînes de caractères change, l'état `checked` des éléments déjà dans la liste serait perdu. En utilisant un substrat `Mirror` au lieu d'un `Mapper`, seuls les éléments qui sont ajoutés ou modifiés sont recalculés. En tirant parti de l'observabilité des changements d'état, le `Mirror` peut intercepter les changements de sa séquence source et les reproduire dans sa séquence interne.

En résumé, la programmation réactive aux données est bien adaptée aux substrats car elle oblige le concepteur à identifier et à séparer les différents niveaux de représentation d'un objet, ce qui conduit à des collections de substrats hautement réutilisables. Elle fournit un modèle déclaratif pour les dépendances entre objets et les maintient à jour à tout moment, de manière transparente et optimisée. Elle permet de modifier des valeurs calculées, en définissant des *setters* qui doivent inverser le calcul de la propriété calculée afin de mettre à jour les sources dont elle dépend, qui se

[9]La bibliothèque `mobx` implémente ce fonctionnement et garantit que seules les propriétés calculées qui sont utilisées par d'autres propriétés sont recalculées, ce qui optimise grandement les mises à jour.
[10]Tous les exemples de code sont fournis en annexe.

propagent ensuite à tous les objets concernés. En pratique, chaque substrat dépend généralement d'une seule source et peut être facilement combiné avec d'autres substrats pour créer des objets complexes dont la cohérence est garantie et qui peuvent être librement inspectés et modifiés.

La programmation réactive aux données rompt avec la programmation orientée objet traditionnelle. Habituellement, les objets n'exposent pas directement leur état, mais fournissent des méthodes qui contrôlent l'accès à leur état. Stratify expose intentionnellement l'état et s'attend à ce qu'il soit modifié directement de l'extérieur. Les objets réagissent à ces changements d'état soit en les interceptant pour les bloquer ou les modifier, soit en les observant pour déclencher les mises à jour ou les effets de bord qui en découlent.

4.2 Interaction : Programmation Fonctionnelle Réactive

Afin d'implémenter les interactions et les instruments, Stratify utilise un autre paradigme réactif appelé *programmation fonctionnelle réactive*. En effet, la programmation réactive aux données n'a pas de notion de temps (les mises à jour sont—conceptuellement—instantanées) tandis que les actions se déroulent dans le temps et doivent pouvoir prendre en compte celui-ci, par exemple pour distinguer un clic d'un double-clic ou calculer une vitesse.

La programmation fonctionnelle réactive permet de décrire des calculs sur des *flux* de valeurs appelées *signaux* en les traitant et en les combinant grâce à un ensemble d'opérateurs. Par exemple, une souris contient un signal qui génère une nouvelle valeur chaque fois qu'elle est déplacée ou que l'on appuie sur ses boutons. Ce signal peut être filtré pour envoyer les mouvements uniquement entre l'appui et le relâchement d'un bouton, et peut être combiné avec un signal de clavier pour envoyer également les changements de touches de modification. Stratify utilise la bibliothèque `RxJS`[11]. Notons que ces signaux sont différents de ceux de la bibliothèque Qt[12] car ces derniers ne sont pas des flux de valeurs et ne peuvent donc pas être filtrés ni composés.

Tout substrat peut être l'émetteur et/ou la cible d'un ou plusieurs signaux. Chaque signal est typé : il porte des valeurs appelées *actions* avec une étiquette de type. Pour qu'un substrat cible puisse écouter un signal, il faut trouver un *protocole* qui corresponde au type du signal pour la cible. Stratify fournit plusieurs méthodes pour effectuer cette recherche. On peut rechercher un protocole par son nom : la recherche commence par les protocoles déclarés dans le substrat cible (chaque protocole ainsi déclaré a un nom), puis se poursuit dans sa source (si elle existe) et ainsi de suite. On peut également rechercher un protocole par le nom et/ou le type d'une propriété : on recherche alors une propriété du nom et/ou du type donnés dans le substrat cible, puis dans sa source (si elle existe) et ainsi de suite. Par exemple, si un signal comporte une action visant à modifier une couleur, on cherche d'abord un protocole par son nom, par exemple `setColor`. Si la recherche échoue, on peut faire une ou plusieurs recherches par le nom de propriété (`color`, `bgColor`, ...) et son type (`Color`). Enfin, en cas d'échec, on pourra rechercher une propriété de type `Color`, quel que soit son nom.

Ce processus rend opérationnel un aspect critique de l'interaction avec les substrats : le découplage entre les instruments et les substrats cibles, de sorte que l'on peut utiliser des instruments sur des cibles même s'ils n'ont pas été explicitement conçus pour fonctionner l'un avec l'autre. Ce découplage dépend cependant de conventions de nommage (noms des protocoles, des propriétés et des types). On peut anticiper que des ontologies et des standards de fait se développeront afin d'assurer l'interopérabilité des instruments et des substrats.

Dans Stratify, un instrument est un substrat qui est la cible d'un dispositif d'entrée tel que la souris (Fig. 5). Lorsque l'instrument identifie le début d'une interaction avec un substrat, il recherche un protocole dans ce substrat avec les méthodes de recherche décrites ci-dessus. L'instrument traite les actions envoyées par le périphérique d'entrée et émet

[11] http://reactivex.io
[12] https://doc.qt.io/qt-6/signalsandslots.html

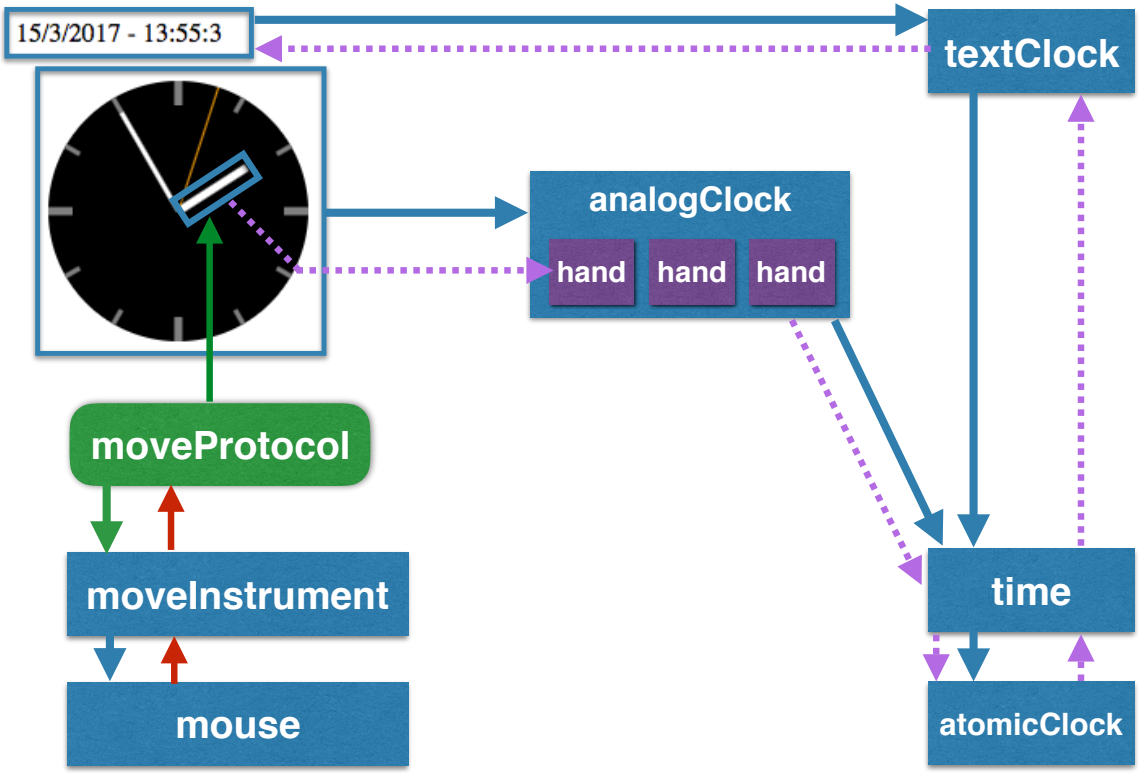

Fig. 5. Substrats et instruments dans Stratify. Les substrats sont en bleu et leur source est indiquée par une flèche bleue. Les protocoles sont en vert, et les signaux en rouge. Les flèches en pointillé montrent la mise à jour des dépendances lorsque l'instrument de déplacement fait tourner l'aiguille de l'horloge.

des actions sur un signal de sortie dont la cible est l'objet créé par le protocole. Cet objet réifie l'interaction et existe le temps de celle-ci : lorsque l'interaction est terminée, le signal se termine et l'objet d'interaction disparaît. Pendant sa durée de vie, il transforme les actions reçues de l'instrument en opérations sur le substrat cible. Cette conception offre une grande flexibilité en permettant à de nouveaux protocoles d'adapter des instruments à des substrats cibles sans modifier ni l'un ni l'autre.

4.3 Exemple : l'instrument Move

Considérons un instrument générique simple : l'instrument `Move`. Il est actionné par un substrat de périphérique d'entrée tel que la souris, et doit fonctionner avec tout substrat qui peut être déplacé ou dont le contenu peut être déplacé, comme les formes graphiques d'un canevas de dessin, les fenêtres d'un gestionnaire de fenêtres, les éléments d'une liste de tâches ou les outils d'une palette.

L'instrument de déplacement (Listings 5, 6, 7) émet des actions `start`, `move` et `stop` en réponse au signal provenant de la souris. Chaque action comporte un décalage incrémental ainsi qu'un décalage par rapport au début du déplacement. En utilisant la position initiale de la souris, l'instrument `Move` identifie le substrat à cette position et le sonde pour un protocole de nom « `move` ». La recherche de ce protocole se poursuit le long de la chaîne des sources des substrats cibles.

Si elle échoue, elle remonte vers les substrats qui l'englobent dans le substrat géométrique : par exemple si la cible originale était un texte contenu dans un rectangle, la recherche se poursuit avec le rectangle, puis ses sources. Si aucun protocole n'est trouvé, la recherche recommence par le même chemin, en recherchant les substrats dont la propriété `position` a pour valeur un point (un objet dont les propriétés `x` et `y` sont des nombres).

Déplacement d'éléments de liste – Lorsque l'utilisateur souhaite déplacer un élément d'une liste de tâches, la première cible est l'élément de liste HTML, et sa source est l'élément `Todo`. Aucun des deux ne connaît le protocole `move`. Le parent de l'élément est la liste HTML, qui met en œuvre le protocole `move`. Le protocole instancie donc l'objet qui représentera l'interaction. Cette interaction reçoit les actions de déplacement par le biais du signal de sortie de l'instrument `Move`, et traduit les mouvements verticaux de la souris en un changement de la position relative de l'élément de la liste HTML. Il en résulte que l'élément visuel se déplace verticalement en réponse aux mouvements de la souris.

À la fin de l'interaction, la liste HTML doit déplacer l'élément `Todo` dans la séquence source. Au lieu de modifier directement cette séquence, elle recherche le protocole `moveItem` dans la source, ce qui permet à la source d'implémenter le déplacement comme elle l'entend. Le substrat `Sequence` implémente ce protocole. La liste HTML l'invoque, ce qui entraîne une mise à jour de l'ordre de la séquence des tâches, immédiatement reflété dans la représentation HTML et dans toute autre représentation qu'elle peut avoir. L'interaction ci-dessus fonctionne avec toute séquence représentée par une liste HTML. Elle permet par exemple de réorganiser les éléments d'un menu ou les outils d'une palette.

Déplacement de formes géométriques – La situation est plus simple lorsqu'on applique l'instrument de déplacement à une forme dans une surface de dessin ou à une fenêtre dans un gestionnaire de fenêtres, car le déplacement ne réordonne pas les objets dans une liste. Dans ce cas, le protocole susceptible de répondre à la recherche est l'existence d'une propriété `position` : l'objet créé par le protocole pendant l'interaction met simplement à jour la position au fur et à mesure qu'il reçoit les actions `move`. Plus généralement, tout objet qui possède une propriété `position` peut ainsi être déplacé, ce qui rend l'outil de déplacement très générique.

Déplacement avec des guides magnétiques – Nous avons implémenté des guides magnétiques similaires aux StickyLines [7] : des objets graphiques peuvent être attachés à une StickyLine ; déplacer une StickyLine déplace également les objets qui lui sont attachés. Une StickyLine est un substrat qui contient la séquence des objets qui lui sont attachés. Lorsque sa position est modifiée, par exemple par l'instrument `Move`, un observateur propage la composante horizontale ou verticale du mouvement (selon le type de StickyLine) aux positions des objets de la séquence, en utilisant le même protocole `move` que l'instrument `Move`. Ceci illustre une forme d'instrumentalisation [5] : la StickyLine devient un instrument.

Les objets sont attachés et détachés d'une StickyLine en les rapprochant ou en les éloignant d'elle. Comme les StickyLines ne font pas partie des objets qui sont sondés lorsque l'instrument `Move` cherche sa cible, nous créons un protocole `move` qui déplace les objets normalement, mais vérifie également s'ils se rapprochent ou s'éloignent d'une StickyLine. Lorsque de telles transitions se produisent, il envoie des actions `attach`/`detach` à la StickyLine. Cet exemple illustre comment de nouveaux comportements peuvent être ajoutés à des substrats existants sans les modifier.

Les StickyLines ne sont pas limitées à l'alignement d'objets dans une surface de dessin. Elles fonctionnent avec tout substrat qui répond au protocole `move` ou qui possède une propriété `position`. Il est donc possible d'aligner des fenêtres dans un gestionnaire de fenêtres, des icônes dans un navigateur de fichiers, etc.

Déplacements contraints – Nous avons implémenté deux autres instruments : le `Spacer` et le `Pusher`. Le premier est basé sur nos observations de designers graphiques qui utilisent souvent des objets transitoires pour mesurer et contrôler l'espacement entre objets graphiques. Le second est inspiré du monde physique, où l'on peut pousser un objet avec un autre.

L'application de ces instruments à une forme graphique transforme celle-ci en *espaceur* ou en *pousseur*. Lorsque l'on déplace un espaceur, par exemple avec l'instrument `Move`, il se bloque s'il rencontre un autre objet, sans le chevaucher. De la même façon, un objet se bloque contre un espaceur lorsqu'on le déplace vers celui-ci. Les pousseurs poussent les objets qu'ils rencontrent lorsqu'on les déplace, mais les autres objets se bloquent contre les pousseurs. Le déplacement des objets avec la touche *Shift* enfoncée ignore ces contraintes. Les espaceurs et les pousseurs sont implémentés de façon similaire aux StickyLines et fonctionnent avec tout objet ayant une propriété `extent` qui contient son rectangle englobant, comme par exemple une fenêtre.

4.4 Autres exemples

Nous avons développé un certain nombre de prototypes avec Stratify pour expérimenter et illustrer la puissance des substrats. Nous avons déjà mentionné l'horloge et la liste de tâches, implémentées comme une collection de substrats prenant en charge les vues synchronisées et la manipulation directe à l'aide de l'instrument `Move`. Outre l'instrument `Move` détaillé ci-dessus, nous avons implémenté une collection d'instruments polymorphes, notamment :

- un instrument `TextEdit` permettant de modifier n'importe quelle propriété textuelle d'un substrat ou de l'une de ses sources, y compris les étiquettes textuelles des instruments eux-mêmes ;
- un instrument `ColorPicker` permettant de modifier la valeur de toute propriété de couleur d'un substrat ou de l'une de ses sources ;
- un instrument `Delete` permettant de supprimer des éléments dans une séquence, y compris les formes dans un canevas, les outils dans une palette et les fenêtres ;
- un instrument `Select` premettant de sélectionner un seul élément dans une séquence, tel qu'un outil dans une palette, une forme dans un canevas ou un élément dans un menu ; et
- un instrument `Load` permettant de charger un substrat ou un instrument depuis un fichier contenant son code.

Canevas – Le substrat `Canvas` contient des formes graphiques. Une série d'outils permet à l'utilisateur de créer des formes de base et des StickyLines, de les déplacer et de les supprimer. Les instruments `Spacer` et `Pusher` peuvent être appliqués à n'importe quelle forme du canevas. Un instrument `AddTool` permet aux utilisateurs de sélectionner un outil dans la palette et de l'ajouter au canevas, où il peut être sélectionné, illustrant ainsi le fait que le canevas peut contenir des objets autres que de simples formes graphiques.

Table – Le substrat `Table` contient des données tabulaires. Les cellules peuvent contenir des substrats arbitraires, à condition qu'ils aient une propriété `value`. Le tableau peut également être connecté à un signal d'entrée pour recevoir des données d'une source en direct et les ajouter au tableau.

Ce substrat devient une feuille de calcul en créant un substrat `Formula` dont la valeur est calculée selon une formule, modifiable avec l'outil `TextEdit`. Les formules peuvent faire référence aux valeurs d'autres cellules grâce à la fonction `$(row,col)`. Le modèle de programmation réactive garantit que les valeurs des formules sont automatiquement recalculées.

DataViz – Le substrat `Histogram` prend une séquence comme source et lui fait correspondre une séquence de formes rectangulaires qui peuvent être utilisées comme source pour un Canevas. Cette séquence peut être une ligne ou une colonne de la table, ce qui permet une visualisation immédiate de son contenu. La mise à jour d'une cellule de la table se reflète immédiatement dans l'histogramme.

L'outil `Resize` peut être utilisé pour modifier la taille de tout objet ayant une propriété `extent`. Les barres d'histogramme utilisent un protocole de redimensionnement spécial pour garantir que la hauteur de la barre reflète la valeur source. Si la valeur source ne peut pas être modifiée, par exemple s'il s'agit d'une formule du tableur plutôt que

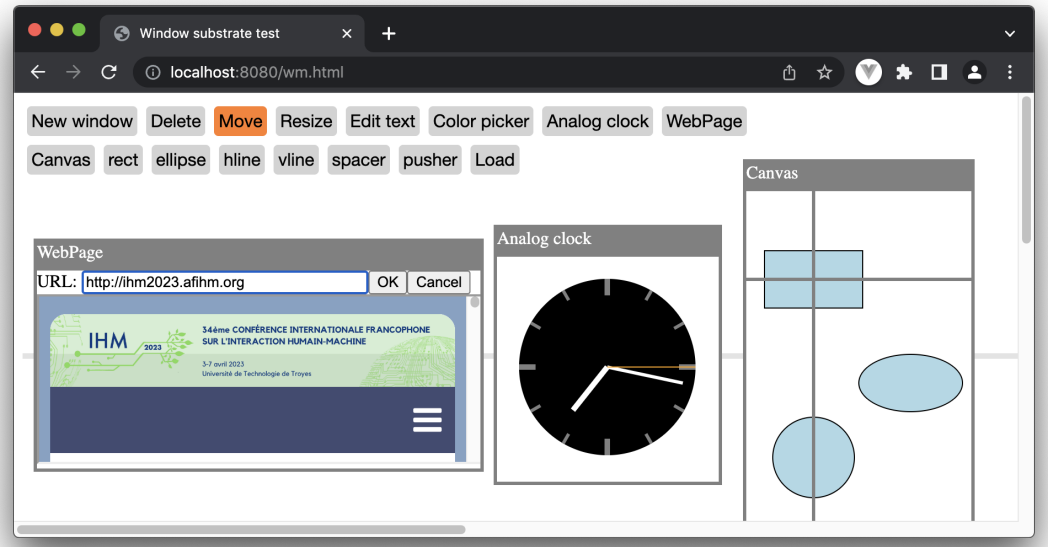

FIG. 6. Le substrat `Window Manager` avec une page web, une horloge, un canevas et une StickyLine pour aligner les fenêtres horizontalement (en gris derrière les fenêtres).

d'une simple valeur, l'action est bloquée et la hauteur de la barre ne peut pas être modifiée. Cela ne nécessite aucune modification de l'outil `Resize`.

Window Manager – Chaque fenêtre du gestionnaire de fenêtres (Fig. 6) peut contenir un substrat arbitraire, notamment une page Web, une horloge, une liste de tâches, un canevas ou un tableau. Les fenêtres peuvent être déplacées et supprimées avec les instruments `Move` et `Delete`. Les instruments `StickyLines`, `Pusher` et `Spacer` peuvent être appliqués aux fenêtres, comme dans le canevas. Enfin un instrument `Load` permet de charger dynamiquement un nouveau substrat ou un nouvel instrument.

En résumé, STRATIFY utilise une combinaison de programmation réactive aux données et de programmation fonctionnelle réactive pour mettre en œuvre le modèle conceptuel des substrats et instruments d'interaction. Les exemples illustrent la puissance du modèle et la façon dont un petit nombre de substrats et d'instruments peuvent donner lieu à une variété de scénarios d'utilisation, démontrant la flexibilité et l'extensibilité des systèmes interactifs qui en résultent. Il en résulte un modèle d'interaction qui ne fait plus appel à la notion d'application, mais à un écosystème de contenus (les substrats) et d'outils permettant de les manipuler (les instruments). Le découplage entre substrats et instruments permet d'envisager des environnements reconfigurables dans lesquels l'utilisateur pourra aisément choisir les contenus et les outils qu'il souhaite utiliser et les combiner de manière imprévue.

4.5 Limitations et perspectives

STRATIFY est une *preuve de concept* du modèle des substrats d'interaction, qui n'a pas vocation à implémenter complètement le modèle conceptuel. Ainsi, le « forçage » des actions (section 3.2) n'est pas implémenté. Surtout, l'exécution est

centralisée : Stratify s'exécute entièrement dans un onglet de navigateur Web, et n'interagit pas avec l'extérieur si ce n'est la possibilité de charger dynamiquement du code, à condition qu'il soit hébergé sur le même serveur. Les substrats et les instruments s'exécutent donc dans le même environnement (le processus de l'onglet du navigateur), ce qui ne permet pas de réaliser des scénarios multi-dispositifs ou multi-utilisateurs. D'autre part les substrats et instruments doivent être programmés en JavaScript ou dans un langage qui produit du JavaScript (tel que Typescript), ce qui limite le choix des programmeurs. Enfin il n'est pas possible d'enregistrer l'état de l'environnement pour le retrouver à la prochaine session.

Une implémentation répartie permettrait aux substrats et instruments de s'exécuter dans des processus différents, y compris sur des machines différentes, et d'être programmés dans des langages différents. Elle devrait reposer sur un protocole de communication ouvert. Une première étape pourrait consister à conserver l'environnement d'un navigateur Web mais à utiliser la communication entre onglets et avec des *webworkers*. Une implémentation réellement répartie, notamment pour un usage multi-utilisateurs, devrait gérer des copies synchronisées d'un même substrat sur plusieurs machines, comme le fait Webstrates [29] avec le DOM de ses pages Web.

Une telle implémentation permettrait d'envisager une validation en environnement réel et d'aller au-delà des exemples illustrés dans cet article. Ces exemples démontrent cependant la faisabilité technique de l'approche et couvrent une palette représentative des fonctionnalités de base d'une gamme représentatitve d'applications interactives : gestion de fenêtres, édition graphique, tableur, visualisation. Le passage à l'échelle d'un environnement complet conduira certainement à de nouvelles questions de recherche sur les façons d'organiser un tel environnement, sur les moyens à offrir aux utilisateurs pour découvrir les substrats, les outils et leurs possibilités, etc.

5 COMPARAISON AVEC L'ÉTAT DE L'ART

Nous examinons les travaux antérieurs concernant les modèles conceptuels, la notion de substrat en IHM, les environnements centrés sur les documents et les environnements interactifs flexibles.

5.1 Modèles conceptuels

Selon Norman [35], un modèle conceptuel décrit à la fois le modèle mental du concepteur du système et le modèle mental créé par les utilisateurs lorsqu'ils interagissent avec le système au fil du temps. Malheureusement, il n'existe pas à notre connaissance de formalisme pour décrire ces modèles. Le modèle conceptuel de Johnson & Henderson [25] est basé sur des objets et des opérations, mais reste de très haut niveau. Les interfaces « reality-based » de Jacob et al. [23] encouragent l'exploitation des qualités des objets physiques du monde réel dans le monde numérique, mais ne proposent pas d'ensemble clair de concepts, ni d'implémentation servant de preuve de faisabilité.

Le pipeline de visualisation de Card et al. [6] peut être analysé comme un cas particulier d'organisation de substrats allant des données brutes à l'image affichée. De même, le modèle MVC [30], bien connu en ingénierie de l'interaction, formalise le lien entre un modèle et sa représentation graphique, et peut être analysé comme la combinaison de deux substrats. Notons cependant que le pipeline de visualisation n'explicite pas le fonctionnement de l'interaction avec la visualisation produite, tandis que dans le modèle MVC, l'interaction fait partie intégrante du modèle alors que notre modèle promeut la séparation de la représentation et de l'interaction.

Le modèle conceptuel Tokens+Constraints de Ullmer et al. [48] pour l'interaction tangible est plus proche de notre travail, car il est plus précis, et peut servir de base à l'implémentation. Notre modèle s'appuie sur l'interaction instrumentale [1] et ses principes de conception [4]. Il les complète en décrivant de façon précise la notion de substrat d'interaction, déjà évoquée par Beaudouin-Lafon [2], et il ancre ces concepts dans les théories cognitives des affordances

et du raisonnement technique. En ce sens notre modèle propose une théorie générative de l'interaction [3] : il aide à imaginer de nouvelles solutions aux problèmes de conception en les analysant en termes de substrats et d'instruments.

5.2 Substrats en IHM

La notion de substrat a déjà été utilisée dans le contexte de l'interaction humain-machine. Les substrats papier [11] permettent aux compositeurs de musique de créer des pièces complexes en reliant et en superposant des morceaux de papier interactif qui représentent et interprètent des données numériques. En particulier, les substrats transparents peuvent être utilisés pour interpréter des données visibles par transparence. Ils sont, d'une certaine façon, une version littérale de nos couches de représentation.

Les substrats graphiques [34] sont des modèles mentaux créés par les graphistes pour représenter leurs idées de mise en page. Les auteurs observent que les outils numériques ne prennent pas en charge ces constructions et ils présentent plusieurs prototypes qui comblent cette lacune. Les substrats narratifs [19] permettent de collecter et rendre persistantes les traces laissées par les joueurs dans un jeu en ligne afin d'en dériver des contenus interactifs, par exemple pour rendre hommage aux exploits d'un joueur. Les auteurs les ont implémentés avec succès dans un jeu expérimental. Comme les substrats papier, ces prototypes sont de bons exemples de nouveaux types de substrats et d'instruments et montrent la fertilité de ces concepts.

Webstrates [29] utilise les technologies Web pour mettre en œuvre des médias dynamiques partageables. Le serveur Webstrates propage les modifications dans le DOM d'une page Web aux autres clients qui affichent cette page, en temps réel, et utilise la transclusion pour inclure un document dans un autre. Comme notre modèle, Webstrates utilise des instruments d'interaction et efface la distinction entre applications et documents. Cependant, le seul type de substrat pris en charge par Webstrates est le DOM, alors que notre modèle permet de gérer plusieurs niveaux de représentation de l'information numérique. Notre implémentation ne permet cependant pas de partager un substrat entre plusieurs utilisateurs.

Enfin, Repening [44] présente AgentSheets comme un « substrat de programmation » pour créer des environnements d'apprentissage interactifs. Ceci est très différent de l'utilisation que nous faisons de ce terme, bien que nous puissions certainement analyser AgentSheets avec notre modèle.

En résumé, notre modèle prolonge ces travaux et les unifie en proposant une définition plus formalisée du concept de substrat.

5.3 Environnements centrés sur les documents

La première station de travail graphique commerciale, le Xerox Star [26] n'avait pas d'applications et était centrée sur les documents. Mais depuis l'avènement de l'informatique personnelle (IBM PC et Macintosh d'Apple), nos environnements numériques sont très majoritairement centrés sur les applications. Microsoft avec OLE[13] et Apple avec OpenDoc [8] ont essayé au début des années 90 d'introduire un modèle centré sur les documents dans leurs environnements. Tous deux permettaient d'inclure dans un document des contenus de différents types, mais chaque contenu était associé à ses propres outils : dans OpenDOC, la barre de menus changeait en fonction du type de contenu édité. Une des raisons de leur échec est sans doute qu'ils ne s'affranchissaient pas vraiment des applications mais rendaient seulement leurs frontières moins visibles.

[13] https://learn.microsoft.com/en-us/cpp/mfc/ole-background?view=msvc-170

On peut également observer que beaucoup d'applications sont dédiées à la gestion de documents et ont des interfaces fondées sur des palettes d'outils. Il s'agit donc d'un mode d'interaction considéré comme naturel. De plus, les suites logicielles telles que Microsoft Office[14], Adobe Creative Cloud[15] ou Affinity[16] font de leur mieux pour gommer les frontières entre applications, avec par exemple des liens dynamiques qui se mettent à jour automatiquement lorsqu'un document inclut celui d'une autre application de la suite. Par ailleurs, un certain nombre de formats ouverts permettent aux applications d'éditer des documents créés par d'autres applications, et le copier-coller ou le cliquer-tirer de contenu entre applications est largement répandu. Dans les deux cas, la flexibilité pour l'utilisateur reste cependant limitée, en le contraignant notamment à utiliser les applications d'une même suite ou des formats ouverts, qui sont parfois plus limités que les formats propriétaires.

Il parait donc pertinent d'explorer de nouveaux modèles qui ne soient pas nécessairement centrés sur les applications. Notre modèle est différent des approches évoquées ci-dessus et plus radical : il sépare strictement les contenus des outils permettant de les manipuler. Il n'est cependant pas incompatible avec des applications qui rassembleraient un ensemble d'outils appropriés pour un type de contenu, comme lorsque l'on achète une boîte à outils au lieu d'outils individuels.

5.4 Environnements interactifs flexibles

De nombreux environnements ont été créés pour favoriser la flexibilité et l'extensibilité. La plupart sont des environnements de programmation destinés aux développeurs plutôt qu'aux utilisateurs finaux. Alan Kay parlait du logiciel comme étant de l'« argile » [27] et a créé des environnements tels que Smalltalk [17]. Plus récemment, Lively [22] propose une approche similaire, s'exécutant dans un navigateur web. Bien que nous puissions envisager un environnement de programmation pour les substrats, ce n'est pas le but premier de ce travail. Il s'agit plutôt de fournir un modèle conceptuel pour les concepteurs et les développeurs qui conduise à plus de flexibilité et d'extensibilité pour les utilisateurs.

Notre travail est plus proche de l'esprit de l'Alternate Reality Kit [46], où les objets peuvent être manipulés avec d'autres objets, sur la base d'une métaphore physique. Bien que nous ne cherchions pas le même type de littéralisme, l'objectif est similaire, à savoir faciliter la libre combinaison d'objets grâce à des outils polymorphes.

Au-delà des environnements interactifs, un certain nombre de systèmes ont démontré la puissance de la rupture avec le modèle des applications. Par exemple, Buttons [33] permet aux utilisateurs d'encapsuler un élément de comportement interactif dans un bouton configurable qui peut être partagé. Les attachements d'interfaces de Olsen et al. [36] augmentent les applications existantes, sur la base de leurs représentations de surface. Scotty [10] injecte du code dans des applications existantes pour augmenter ou remplacer leurs fonctionnalités. Notre travail vise à créer des environnements plus ouverts où de telles fonctionnalités peuvent facilement être ajoutées, plutôt que de s'appuyer sur des solutions ad hoc.

Enfin, le développement par l'utilisateur [31] fournit une flexibilité maximale en permettant aux utilisateurs de programmer eux-mêmes de nouvelles fonctionnalités. Cette approche est compatible avec le modèle proposé, et permettrait aux utilisateurs de créer leurs propres substrats et instruments.

6 CONCLUSION ET TRAVAUX FUTURS

Cet article présente un modèle d'interaction inspiré par la théorie des affordances et celle du raisonnement technique qui décrivent la façon dont les humains interagissent dans le monde physique. Il introduit un modèle conceptuel fondé sur les

[14] https://www.microsoft.com/fr-fr/microsoft-365
[15] https://www.adobe.com/creativecloud.html
[16] https://affinity.serif.com/en-us/

substrats et les instruments d'interaction. Les *substrats d'interaction* représentent un « matériau » numérique riche grâce à la combinaison de niveaux interdépendants de représentation de l'information. Les *instruments d'interaction* offrent des capacités d'interaction puissantes en découplant les sources des actions de la façon dont elles sont mises en œuvre par les substrats cibles. La combinaison de ces concepts permet d'atteindre un niveau de flexibilité et d'extensibilité supérieur à celui des environnements actuels. En particulier, il permet de s'affranchir du concept d'application comme unité fondamentale d'organisation de nos environnements numériques. Au lieu d'applications qui emprisonnent des types de contenu et les outils permettant de les manipuler dans un « jardin privé », les substrats et les instruments créent des environnements ouverts dans lesquels les utilisateurs peuvent combiner à loisir les contenus et choisir les instruments qui leur conviennent pour les manipuler.

Afin de montrer la validité et la faisabilité de ce modèle, nous avons développé Stratify, une application Web servant de preuve de concept. Stratify combine la programmation réactive aux données et la programmation fonctionnelle réactive pour fournir un modèle de calcul puissant à la fois pour les dépendances statiques et pour les changements dynamiques. Grâce au fort découplage entre les différents substrats et entre les substrats et les instruments, Stratify garantit la flexibilité et l'extensibilité mises en avant par le modèle conceptuel. La collection de prototypes que nous avons créée démontre la puissance et la généralité de cette approche.

Ce modèle et son implémentation ouvrent de nombreuses pistes pour des travaux futurs. D'une part, le modèle conceptuel peut être raffiné en poursuivant la taxonomie des substrats, en formalisant mieux la notion de protocole d'interaction, et en identifiant une gamme d'instruments génériques. Il devrait également être généralisé en l'appliquant à un plus large éventail de styles d'interaction. Au niveau de l'implémentation, il serait intéressant d'expérimenter des substrats de pixels et de prendre en charge la persistance des substrats et des instruments. Une autre piste importante est l'interaction multi-utilisateurs et multi-dispositifs, qui devra reposer sur un modèle d'implémentation réparti des substrats et des instruments, à l'aide de concepts tels que les *Conflict-Free Replicated Data Types* ou CRDTs [45].

À plus long terme, il s'agit de créer un environnement à part entière permettant de dynamiquement ajouter, modifier et retirer une large gamme de substrats et d'instruments, et de le tester dans des contextes réels afin d'observer comment les utilisateurs l'adoptent et se l'approprient. Cela permettra de vérifier si cette approche passe à l'échelle lorsque le nombre de substrats et d'instruments devient important, et de valider sa capacité à supplanter le modèle actuel à base d'applications.

REMERCIEMENTS

Ces travaux sont partiellement financés par le projet ERC № 695464 "ONE : Unified Principles of Interaction".

RÉFÉRENCES

[1] Michel Beaudouin-Lafon. 2000. Instrumental Interaction : An Interaction Model for Designing post-WIMP User Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (The Hague, The Netherlands) *(CHI '00)*. ACM, New York, NY, USA, 446–453. https://doi.org/10.1145/332040.332473

[2] Michel Beaudouin-Lafon. 2017. Towards Unified Principles of Interaction. In *Proceedings of the 12th Biannual Conference on Italian SIGCHI Chapter* (Cagliari, Italy) *(CHItaly '17)*. ACM, New York, NY, USA, Article 1, 2 pages. https://doi.org/10.1145/3125571.3125602

[3] Michel Beaudouin-Lafon, Susanne Bødker, and Wendy E. Mackay. 2021. Generative Theories of Interaction. *ACM Transactions on Computer-Human Interaction* 28, 6, Article 45 (11 2021), 54 pages. https://doi.org/10.1145/3468505

[4] Michel Beaudouin-Lafon and Wendy E. Mackay. 2000. Reification, Polymorphism and Reuse : Three Principles for Designing Visual Interfaces. In *Proceedings of the Working Conference on Advanced Visual Interfaces* (Palermo, Italy) *(AVI '00)*. ACM, New York, NY, USA, 102–109. https://doi.org/10.1145/345513.345267

[5] Pascal Béguin and Pierre Rabardel. 2000. Designing for instrument-mediated activity. *Scandinavian Journal of information Systems* 12 (2000), 173–190.

[6] Stuart K. Card, Jock D. Mackinlay, and Ben Shneiderman (Eds.). 1999. *Readings in Information Visualization : Using Vision to Think.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[7] Marianela Ciolfi Felice, Nolwenn Maudet, Wendy E. Mackay, and Michel Beaudouin-Lafon. 2016. Beyond Snapping : Persistent, Tweakable Alignment and Distribution with StickyLines. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology* (Tokyo, Japan) *(UIST '16)*. ACM, New York, NY, USA, 133–144. https://doi.org/10.1145/2984511.2984577

[8] Dave Curbow and Elizabeth Dykstra-Erickson. 1997. Designing the OpenDoc Human Interface. In *Proceedings of the 2nd Conference on Designing Interactive Systems : Processes, Practices, Methods, and Techniques* (Amsterdam, The Netherlands) *(DIS '97)*. ACM, New York, NY, USA, 83–95. https://doi.org/10.1145/263552.263581

[9] Alan Dix. 2007. Designing for Appropriation. In *Proceedings of HCI 2007 The 21st British HCI Group Annual Conference University of Lancaster, UK 21*. British Computer Society, London, UK, 1–4. https://doi.org/10.14236/ewic/HCI2007.53

[10] James R. Eagan, Michel Beaudouin-Lafon, and Wendy E. Mackay. 2011. Cracking the Cocoa Nut : User Interface Programming at Runtime. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology* (Santa Barbara, California, USA) *(UIST '11)*. ACM, New York, NY, USA, 225–234. https://doi.org/10.1145/2047196.2047226

[11] Jérémie Garcia, Theophanis Tsandilas, Carlos Agon, and Wendy Mackay. 2012. Interactive Paper Substrates to Support Musical Creation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Austin, Texas, USA) *(CHI '12)*. ACM, New York, NY, USA, 1825–1828. https://doi.org/10.1145/2207676.2208316

[12] William W. Gaver. 1991. Technology Affordances. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New Orleans, Louisiana, USA) *(CHI '91)*. ACM, New York, NY, USA, 79–84. https://doi.org/10.1145/108844.108856

[13] Eleanor Jack Gibson. 1969. *Principles of Perceptual Learning and Development.* Appleton & Co.

[14] James J Gibson. 1986. *The Ecological Approach to Visual Perception.* Lawrence Erlbaum.

[15] Camille Gobert and Michel Beaudouin-Lafon. 2022. I-LaTeX : Manipulating Transitional Representations between LaTeX Code and Generated Documents. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems* (New Orleans, LA, USA) *(CHI '22)*. ACM, New York, NY, USA, Article 554, 16 pages. https://doi.org/10.1145/3491102.3517494

[16] Camille Gobert and Michel Beaudouin-Lafon. 2022. Représentations Intermédiaires Interactives Pour La Manipulation de Code LaTeX : Interactive Intermediate Representations for LaTeX Code Manipulation. In *Proceedings of the 32nd Conference on l'Interaction Homme-Machine* (Virtual Event, France) *(IHM '21)*. ACM, New York, NY, USA, Article 10, 11 pages. https://doi.org/10.1145/3450522.3451325

[17] Adele Goldberg and David Robson. 1983. *Smalltalk-80 : The Language and Its Implementation.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[18] Julien Gori, Han L. Han, and Michel Beaudouin-Lafon. 2020. FileWeaver : Flexible File Management with Automatic Dependency Tracking. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology* (Virtual Event, USA) *(UIST '20)*. ACM, New York, NY, USA, 22–34. https://doi.org/10.1145/3379337.3415830

[19] Viktor Gustafsson, Benjamin Holme, and Wendy E. Mackay. 2020. Narrative Substrates : Reifying and Managing Emergent Narratives in Persistent Game Worlds. In *International Conference on the Foundations of Digital Games* (Bugibba, Malta) *(FDG '20)*. ACM, New York, NY, USA, Article 46, 12 pages. https://doi.org/10.1145/3402942.3403015

[20] Han L. Han, Junhang Yu, Raphael Bournet, Alexandre Ciorascu, Wendy E. Mackay, and Michel Beaudouin-Lafon. 2022. Passages : Interacting with Text Across Documents. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems* (New Orleans, LA, USA) *(CHI '22)*. ACM, New York, NY, USA, Article 338, 17 pages. https://doi.org/10.1145/3491102.3502052

[21] Patrick J. Hayes. 1979. The Naive Physics Manifesto. In *Expert Systems in the Electronic Age*, Donald Michie (Ed.). Edinburgh University Press, Edinburgh, Scotland, 242–270.

[22] Daniel Ingalls, Tim Felgentreff, Robert Hirschfeld, Robert Krahn, Jens Lincke, Marko Röder, Antero Taivalsaari, and Tommi Mikkonen. 2016. A World of Active Objects for Work and Play : The First Ten Years of Lively. *Proceedings of the 2016 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software* (2016), 238–249. https://doi.org/10.1145/2986012.2986029

[23] Robert J.K. Jacob, Audrey Girouard, Leanne M. Hirshfield, Michael S. Horn, Orit Shaer, Erin Treacy Solovey, and Jamie Zigelbaum. 2008. Reality-based Interaction : A Framework for post-WIMP Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Florence, Italy) *(CHI '08)*. ACM, New York, NY, USA, 201–210. https://doi.org/10.1145/1357054.1357089

[24] Ghita Jalal, Nolwenn Maudet, and Wendy E. Mackay. 2015. Color Portraits : From Color Picking to Interacting with Color. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (Seoul, Republic of Korea) *(CHI '15)*. ACM, New York, NY, USA, 4207–4216. https://doi.org/10.1145/2702123.2702173

[25] Jeff Johnson and Austin Henderson. 2011. *Conceptual Models : Core to Good Design.* Morgan Claypool.

[26] Jeff Johnson, Teresa L. Roberts, William Verplank, David Canfield Smith, Charles H. Irby, Marian Beard, and Kevin Mackey. 1989. The Xerox Star : A Retrospective. *Computer* 22, 9 (1989), 11–26.

[27] Alan C. Kay. 1984. Computer Software. *Scientific American* 251 (1984), 53–59.

[28] Roberta L Klatzky, Brian MacWhinney, and Marlene Behrmann. 2012. *Embodiment, Ego-Space, and Action.* Psychology Press.

[29] Clemens N. Klokmose, James R. Eagan, Siemen Baader, Wendy Mackay, and Michel Beaudouin-Lafon. 2015. Webstrates : Shareable Dynamic Media. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology* (Charlotte, NC, USA) *(UIST '15)*. ACM, New York, NY, USA, 280–290. https://doi.org/10.1145/2807442.2807446

[30] Glenn E. Krasner and Stephen T. Pope. 1988. A Cookbook for Using the Model-view Controller User Interface Paradigm in Smalltalk-80. *J. Object Oriented Program.* 1, 3 (Aug. 1988), 26–49. http://dl.acm.org/citation.cfm?id=50757.50759

[31] Henry Lieberman, Fabio Paternò, and Volker Wulf (Eds.). 2006. . Springer Netherlands, Dordrecht. https://doi.org/10.1007/1-4020-5386-X_1

[32] Wendy E. Mackay. 2000. Responding to cognitive overload : Co-adaptation between users and technology. *Intellectica* 30 (Jan. 2000), 177–193.

[33] Allan MacLean, Kathleen Carter, Lennart Lövstrand, and Thomas Moran. 1990. User-tailorable Systems : Pressing the Issues with Buttons. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Seattle, Washington, USA) *(CHI '90)*. ACM, New York, NY, USA, 175–182. https://doi.org/10.1145/97243.97271

[34] Nolwenn Maudet, Ghita Jalal, Philip Tchernavskij, Michel Beaudouin-Lafon, and Wendy E. Mackay. 2017. Beyond Grids : Interactive Graphical Substrates to Structure Digital Layout. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (Denver, Colorado, USA) *(CHI '17)*. ACM, New York, NY, USA, 5053–5064. https://doi.org/10.1145/3025453.3025718

[35] Don Norman. 1998. *The Design of Everyday Things*. Doubleday.

[36] Dan R. Olsen, Jr., Scott E. Hudson, Thom Verratti, Jeremy M. Heiner, and Matt Phelps. 1999. Implementing Interface Attachments Based on Surface Representations. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Pittsburgh, Pennsylvania, USA) *(CHI '99)*. ACM, New York, NY, USA, 191–198. https://doi.org/10.1145/302979.303038

[37] François Osiurak. 2010. What Neuropsychology Tells Us about Human Tool Use ? The Four Constraints Theory (4CT) : Mechanics, Space, Time and Effort. *Neurosychology Review* 24, 2 (2010), 88–115. https://doi.org/10.1007/s11065-014-9260-y

[38] François Osiurak and Arnaud Badets. 2016. Tool Use and Affordance : Manipulation-Based Versus Reasoning-Based Approaches. *Psychological Review* 123, 5 (2016), 35 pages. https://doi.org/10.1037/rev0000027

[39] François Osiurak, Christophe Jarry, and Didier Le Gall. 2010. Grasping the Affordances, Understanding the Reasoning : Toward a Dialectical Theory of Human Tool Use. *Psychological Review* 117, 2 (2010), 517–540. https://doi.org/10.1037/a0019004

[40] Pierre Rabardel. 1995. *Les hommes et les technologies ; approche cognitive des instruments contemporains*. Armand Colin.

[41] Miguel A. Renom. 2022. *Theoretical bases of human tool use in digital environments*. Theses. Université Paris-Saclay. https://theses.hal.science/tel-03675933

[42] Miguel A. Renom, Baptiste Caramiaux, and Michel Beaudouin-Lafon. 2022. Exploring Technical Reasoning in Digital Tool Use. In *CHI Conference on Human Factors in Computing Systems* (New Orleans, LA, USA) *(CHI '22)*. ACM, New York, NY, USA, Article 579, 17 pages. https://doi.org/10.1145/3491102.3501877

[43] Miguel A. Renom, Baptiste Caramiaux, and Michel Beaudouin-Lafon. 2023. Interaction Knowledge : Understanding the 'Mechanics' of Digital Tools. In *CHI Conference on Human Factors in Computing Systems* (Hamburg, Germany) *(CHI '23)*. ACM, New York, NY, USA. In press.

[44] Alex Repenning. 1994. Programming Substrates to Create Interactive Learning Environments. *Interactive Learning Environments* 4, 1 (1994), 45–74.

[45] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. 2011. Conflict-Free Replicated Data Types. In *Symposium on Self-Stabilizing Systems*. Springer, 386–400.

[46] Randall B. Smith. 1987. Experiences with the Alternate Reality Kit : An Example of the Tension Between Literalism and Magic. *Proceedings of the SIGCHI/GI Conference on Human Factors in Computing Systems and Graphics Interface* (1987), 61–67. https://doi.org/10.1145/29933.30861

[47] Paul S. Strauss. 1993. IRIS Inventor, a 3D Graphics Toolkit. *SIGPLAN Not.* 28, 10 (Oct. 1993), 192–200. https://doi.org/10.1145/167962.165889

[48] Brygg Ullmer, Hiroshi Ishii, and Robert J. K. Jacob. 2005. Token+Constraint Systems for Tangible Interaction with Digital Information. *ACM Trans. Comput.-Hum. Interact.* 12, 1 (March 2005), 81–118. https://doi.org/10.1145/1057237.1057242

ANNEXE

Code des substrats de l'horloge (section 4.1)

```
class AtomicClock extends DataSource {
   constructor () {
      super();
      this.ms = Date.now();
      // mettre à jour le compteur de millisecondes chaque seconde
      setTimeout(() => this.ms += 1000, 1000);
   }
}

class Time extends Substrate {
   // 'get' déclare une propriété calculée
   get date() {return new Date(this.source.ms)}
   get hours() {return this.date.getHours()}
   get minutes() {return this.date.getMinutes()}
   get seconds() {return this.date.getSeconds()}
}

class AnalogClock extends SVGSubstrate {
   get hoursAngle() {return (this.source.hours + this.source.minutes/60) * 30)}
   get minutesAngle() {return this.source.minutes*6)}
   get secondsAngle() {return this.source.seconds*6)}
   get SVG() {
      // retourne 3 segments SVG tournés des angles correspondants
   }
}

// crée une AnalogClock, dont la source est un substrat Time,
// qui a lui-même pour source une AtomicClock
new AnalogClock(new Time(new AtomicClock()));
```

Listing 1. Programmer une horloge avec Stratify

```
// dans la classe AnalogClock
   set minutesAngle(a) {
      this.source.minutes = Math.round(a/6);
   }
// dans la classe Time
   set minutes(v) {
      this.source.ms += (v - this.minutes)*60000;
   }
```

Listing 2. Déclaration des *setters* pour changer l'heure

Code des substrats de séquence (section 4.2)

```
class Sequence {
   constructor(a) {
      this.seq = []; // la séquence
      if (a) this.seq.replace(a);
   }
}
```

```
7
8  class Sorter extends Substrate {
9    get seq() { // propriété calculée
10     return this.source.seq.sort(this.compare)
11   }
12   constructor(source, compare) {
13     super(source);
14     this.compare = compare || defaultCompare;
15   }
16 }
17
18 class Filter // similaire au Sorter en utilisant Array.filter
19 class Mapper // similaire au Sorter en utilisant Array.map
20
21 class Mirror extends Substrate {
22   seqChange(change) { // change décrit le changement (ajout, ...)
23     // appliquer la modification à this.seq
24     ...
25   }
26   constructor(source, mirror) {
27     super(source);
28     this.mirror = mirror; // fonction de mapping
29     // créer la séquence résultant du mapping
30     this.seq = source.seq.map(item => this.mirror(item));
31     // intercepter les changement de la source pour les propager
32     intercept(source.seq, this.seqChange);
33   }
34 }
```

Listing 3. Substrat structurel pour les séquences

```
1  class Todo extends Substrate {
2    constructor(s) {
3      super(s);
4      this.checked = false;
5    }
6  }
7
8  // Créer une pile de substrats qui trie, filtre et mappe une liste de todos
9  let list = new Sequence(['sleep', 'eat', 'work']);
10 let todos = new Mirror(list, a => new Todo(a));
11 let sorter = new Sorter(todos);
12 let filter = new Filter(sorter, a => !a.checked);
13 let upper = new Mapper(filter, a => capitalize(a.source));
14
15 // Les changements se propagent immédiatement
16 list.seq.push('exercise');
17 todos.seq[2].checked = true;
```

Listing 4. Une séquence de *todos*

Code de l'instrument de déplacement (section 4.3)

Code de l'instrument générique `Drag`, de l'instrument `Move` qui en hérite, et du protocole `move` qui affecte la propriété `position` :

```
class DragInstrument {
  // Méthodes devant être redéfinies dans les sous-classes
  draggable() {return null;}      // retourne un protocole
  startAction() {return null;}    // retourne une action
  dragAction() {return null;}     // retourne une action
  stopAction() {return null;}     // retourne une action

  // Réponses aux actions envoyées par le dispositif d'entrée
  startDrag() {
    // Trouver la cible
    let res = this.draggable();
    if (! res) return;
    // Initialiser l'état
    this.dragging = true;
    this.delta = {x: 0, y: 0};
    this.incr = {x: 0, y: 0};
    // Instantier le protocole et y connecter le signal
    this.signal = new Signal();
    let interaction = new res.protocol(this, res.target);
    interaction.connect(this.signal);
    // Envoyer l'action start
    let action = this.startAction();
    if (action) this.signal.next(action);
  }

  drag() {
    // Mettre à jour l'état
    this.delta.x += this.source.delta.x;
    this.delta.y += this.source.delta.y;
    // Envoyer l'action drag
    let action = this.dragAction();
    if (action) this.signal.next(action);
  }

  stopDrag() {
    // Envoyer l'action stop
    let action = this.stopAction();
    if (action) this.signal.next(action);
    this.dragging = false;
    // Terminer le signal
    this.signal.complete();
    this.signal = null;
  }

  // Appelé lorsque l'outil est activé / désactivé
  activate() {
    // s'abonner au signal source, et faire en sorte que startDrag, drag et stopDrag
    // soient appelés en fonction des actions reçues
  }

  deactivate() {
```

```
    // se désabonner du signal source
  }

  constructor(input) {
    this.source = input;
    this.signal = null;
  }
}
```

Listing 5. L'instrument Drag est destiné à être sous-classé par des instruments qui répondent à une interaction de type cliquer-tirer.

```
class MoveInstrument extends DragInstrument {
  draggable()  {
    // Chercher une cible avec le protocole 'move'
    let protocol = source.findNamedProtocol('move');
    if (protocol) return protocol;
    // sinon chercher une cible avec la propriété 'position' de type 'Point'
    return this.source.findNamedPropertyProtocol('position', Point);
  }

  startAction() {
    // retourner l'action 'start'
    return { type: 'start' }
  }

  dragAction() {
    // retourner l'action 'move' avec le déplacement incrémental et total
    return {
      type: 'move',
      incr: this.source.delta,
      delta: this.delta,
    };
  }

  stopAction() {
    // retourner l'action 'stop' avec le déplacement total
    return { type: 'stop', delta: this.delta }
  }
}
```

Listing 6. L'instrument Move redéfinit les méthodes de l'instrument Drag pour trouver sa cible et envoyer des actions.

```
class MovePositionProtocol extends Protocol {
  connect(signal) {
    // Appeler move lorsque l'on reçoit des actions 'move'
    // mais seulement si le curseur a bougé d'au moins 3 pixels
    signal.filter(a => a.type === 'move')
          .skipWhile(a => norm(a.delta) < 3)
          .subscribe(a => this.move(a));
  }

  move(a) {
    // Mettre à jour la position de la cible
    this.target.position.x += a.incr.x;
    this.target.position.y += a.incr.y;
```

```
14    }
15 }
```

Listing 7. Le protocole pour déplacer la position d'un substrat filtre le signal d'entrée et met à jour la propriété `position` de la cible.