Augmenting the scope of interactions with implicit and explicit graphical structures

Raphaël Hoarau, Stéphane Conversy

▶ **To cite this version:**

Raphaël Hoarau, Stéphane Conversy. Augmenting the scope of interactions with implicit and explicit graphical structures. CHI 2012, ACM annual conference on Human Factors in Computing Systems, May 2012, Austin, United States. 10.1145/2207676.2208337 . inria-00607937

HAL Id: inria-00607937

https://inria.hal.science/inria-00607937

Submitted on 11 Jul 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Augmenting the scope of interactions with implicit and explicit graphical structures

Raphaël Hoarau
Université de Toulouse- ENAC – IRIT
Toulouse, France
Raphael.hoarau@enac.fr

Stéphane Conversy
Université de Toulouse- ENAC – IRIT
Toulouse, France
stephane.conversy@enac.fr

ABSTRACT
When using interactive graphical tools, users often have to manage a structure, i.e. the arrangement of and relations between the parts or elements of the content. However, the interaction with structures may be complex, and not well integrated with the interaction with the content. Based on contextual inquiries and past works, we have identified a number of concepts and requirements about the interaction with structure. We have explored two interactive tools: a new kind of property sheet that relies on the implicit structure of graphics; and a property delegation graph to enable users to provide an explicit graphical structure. The interactions with the tools augment the scope of interactions to multiple objects.

ACM Classification: H5.2 [Information interfaces and presentation]: User Interfaces. - Graphical user interfaces. H.5.2. User Interfaces: Interaction Styles.

General terms: Design, Human Factors

Keywords: Graphical Interaction Design, Instrumental interaction, Exploratory Design.

INTRODUCTION
When using computerized tools such as real-time editors, presentation software, GUI builders, etc. users create and manipulate graphical objects on the screen. They can edit them individually, e.g. change their color or their stroke width. Users can also consider and interact with sets of objects as opposed to individual objects. To do so, they may be required to structure the scene, by relying on concepts such as groups, styles, or masters. According to the Oxford dictionary, a *structure* is "the arrangement of and relations between the parts or elements of something complex". Using a structure may have multiple assets, such as helping users conceptualize the scene they are creating ("the background of the slide includes this drawing and this text", "this set of slides is a subpart of the presentation" etc.), and think better about the problem at hand. Here, we are interested in structures as means to interact with the content: since structuring involves sets of objects, the actions done on an element of the structure may have an effect on several objects at once.

In current interactive systems, the use and the management of structures may be complex. Users have to create and maintain them. Depending on the kind of structure, some operations may be impossible or cumbersome to do, which prevents users to explore the design space. Furthermore, systems that provide structuring do not leverage off the structures fully to provide users with new ways of interacting with the content.

Interactions with structure and multiple objects through a structure have not been studied extensively in the past. Of course, there exists works that implicitly tackle the problem, but few concepts or properties explicitly target it. For example, what are the interactions that enable users to define sets of objects? What are the available means to augment the scope of interaction i.e. apply an interaction to several targets? What are the concepts that may guide the design of interactions?

The work presented in this paper aims at improving the management of structures as means to augment the scope of interactions. Based on contextual inquires and related work, we present a number of requirements pertaining to the interactions with structures. We then present a set of interactive tools that partly fulfill those requirements. In particular, we present two new visualizations of properties and values. Together with modeless, example-based interaction and selection, they enable designers to make an opportunistic use of implicit (i.e. unplanned) sets of objects, and to structure the content explicitly.

WORK SCENARIOS
We have based our work on concrete and realistic case studies. We have conducted five contextual inquiries with "designers", the design activity being taken in its broadest sense: graphics edition (Illustrator and OmniGraffle), courses schedule (iCal), geographical map of a site (AutoCAD), lecture presentation (PowerPoint). In order to introduce the problem and concretize it, we present two of the

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
UIST'11, October 16–19, 2011, Santa Barbara, CA, USA.
Copyright © 2011 ACM 978-1-4503-0716-1/11/10... $10.00.

five case studies with work scenarios, which illustrate interactions on several objects with or without a structure. Some of the steps are annotated with words in italic to characterize them. We detail the annotations later in this section.

Software keyboard

The first scenario describes the creation of the graphical part of a software keyboard, such as the ones available on tablets. The user is a designer that employs a graphics editor (here Illustrator). The user begins by creating a fist key. She draws a round rectangle, applies a gradient to the rectangle, and adds a surrounding rectangle (stroke only). She selects the two rectangles with a selection rectangle (*designation*), and groups them with a group command in a menu (*structuring*). She then adds a soft shadow effect on the group. She overlays a label with a text 'A' on the group of rectangles, and centers the label and the group by invoking a center command on a toolbox. She then forms another group with the label and the groups of two rectangles, and names it "key", in the tree view of the graphical scene provided by Illustrator (*structuring*).

This first key serves as a model to create other keys: the user duplicates the key, and applies a horizontal translation to the copy. She proceeds with this action several times in order to form a row of keys (Figure 1). She then modifies the text of each key one by one, until she gets an 'AZERTYU' keyboard.

Figure 1: The user creates a key, and duplicates it.

Figure 2: The text of the 'I' key is not centered.

However, when she changes the letter 'A' for the letter 'I', she realizes that the 'I' text is not centered with the rectangles (Figure 2). The first object was specified incorrectly: if the three objects (label, background rectangle, stroked rectangle) are correctly aligned, the text of the label is not centered. With the first letters (AZERTYU), the problem was not existent since the widths of the texts are similar. The user could realize the problem only when tackling a thinner letter ('I').

Each label being into a heterogeneous group (containing other object types than label), the system does not provide a text center command that can be applied to a selection of objects. She has to click multiple times on an object to reach the label and apply the 'text centered' command. Therefore, she estimates that it is more efficient to start over: she deletes all copies, un-groups the first key, centers the text, groups the objects again, copies et moves the copies, and modifies each letter one by one.

This scenario illustrates several tasks that the user should accomplish.

Sets management The user relied on the ability of the system to allow sets creation, modification, and management. For example she created a single group with two rectangles, then another group with the previous one and the label. She could also name them, both because it is a way to conceptualize information, and to be able to reference it in a toolkit that displays SVG graphics. Operations on groups include destruction (on the canvas view), and reparenting of the hierarchy (on the tree view).

Designation The user designated properties, actions, and objects. For example, she changed the "alignment" property of the label to "centered".

Scope of actions Some actions allow the user to act on multiple objects at once. For example, the user grouped objects because she wanted to consider them as a single entity that keeps relative position between subparts, but also because she wanted to apply a single translation on three objects at once. Conversely, she was not able to apply the command 'set alignment' to several objects at once.

ATC strips drawing

In this scenario, the user is a visualization designer that aims at designing several enhancements of paper-strips design for Air Traffic Control. A paper strip is a tool that helps controllers verify that flights are not in conflict, i.e. their planed time of passage over beacons are not the same. He wants to design and illustrate a new digital representation with colored gradients that would show the current position of the flight between two beacons. Figure 4 shows the final result.

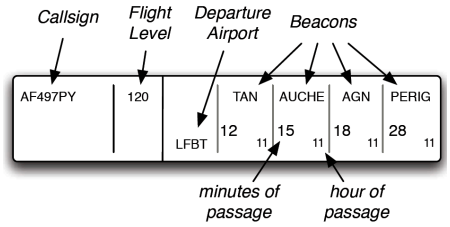

Figure 3: A paper strip.

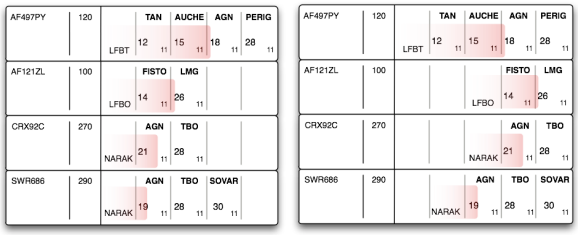

Figure 4: Two versions of a strip board.

The user starts with the drawing of a simplified strip, by imitating an actual one (see Figure 3). The strip is composed of lines, rectangles, and labels with textual informa-

tion. He duplicates the first strip multiple times, and stacks the copies in order to form a strip board (Figure 4, left). He then modifies the labels to reflect information about other flights. While thinking about his design, he elaborates a new one, based on the first version: right aligned beacons (Figure 4, right), instead of left aligned beacons (*exploratory design*). Compared to the first version, the new design would enable controllers to rank easily the flights by their order of airspace exit (the righter the sooner). Thus, he duplicates the previous board, and modifies each strip to right align beacons.

While trying to show the value of using gradients, he realizes that the design should involve two flights in conflict. To do so, he has to make the time of passage over certain beacons match. He first proceeds on the second version on the board. However, he wants stakeholders to be able to compare the two designs. In order to limit the differences between the two versions, he has to change the first board accordingly. He begins by searching the elements that must be changed (*searching*).. When he finds one, he executes the actions required to apply the modification.

Besides the tasks already evoked in the first scenario, the second scenario illustrates two new activities.

Seeking/designation The user needs to retrieve objects, based on explicit or fuzzy requirements. In the scenario, he has to search objects whose content (here a text) is similar to other ones. The search action requires scanning visually the graphical objects in the board and seeking candidate objects, at the risk of forgetting some of them. As the number of objects increase, it is more and more difficult to seek object with visual scanning. In the previous scenario, each modification becomes more costly, not only because of the number of actions to repeat, but also because of the required search effort.

Exploratory Design The user explores parts of possible solutions, and modifies existing parts of solutions. In doing so, he pursues an exploratory design activity. By combining action, visualization of intermediate results, and thinking, exploratory design generates a co-discovery of the problem and the solution. This phenomenon is important for activities in which the expected result is not known in advance: graphics edition activities, but also during slides design, or class hierarchy design [7][23].

RELATED WORK

Past works have tackled the problems of managing structures, and interacting with multiple objects, either explicitly or implicitly. We have studied the existing systems along three axes: interactions for structuring the content provided by interactive systems, design and evaluation of interactions for structuring, and structuring in programming, since many concepts from this domain are relevant.

Structuring for users

Groups In order to act on several objects at once, traditional graphical editors allows acting on a selection of objects previously defined by the user, or on groups. The only operations available for a group is 'ungroup', which removes the group entity and selects all objects that were part of the groups (no modification, addition, or subtraction). Selection can be seen as a temporary group, with 'add' and 'remove' operations e.g. by holding the shift key and selecting several elements, or holding the ctrl key and clicking on individual elements. Translation, scale and rotation can be applied on a group: all elements inside the group are transformed accordingly. Conversely, some operations like 'set color' cannot be applied to groups, supposedly because some elements inside the group do not "understand" this operation. This forces the user to un-group groups, and to apply the command on each objects. In this case, the interaction with the structure is not well integrated with the interaction with the content.

Hierarchy and groups of groups Groups can be part of a surrounding group, turning them into hierarchies. Support for management of such hierarchies range from no support at all, navigation in the parents hierarchy (clicking several times on an object enables cycling through parents in the Squeak version of Morphic [16]), to tree views in structured graphics editors such as Inkscape or Illustrator. A tree view enables user to re-parent elements with a drag and drop. However, there is no support for other operations, such as applying a color to a node in order to change all children.

Masters and DAG A Master is an element used as a "model" for other elements. For example, PowerPoint enables users to define the appearance in a master slide that other slides would inherit. Sketchpad introduced masters as shareable objects that could be used in multiple locations in the scene [21]. Changing a property of the master would modify all objects that depend on this master. This was a way to reduce the number of actions required from the user when something must be changed. Somewhat related to Masters, constraints can also be defined with a link between cells in a spreadsheet [18]. Masters can be considered as a way to structure contents with a Directed Acyclic Graph.

Properties Presto is a document management system that enables users to tag documents with properties, e.g. *year=2011* [6]. Properties provide a uniform mechanism for managing, coding, searching, retrieving and interacting with documents. For example, users can define directories (i.e. a set) of documents using properties: either by extension (by putting elements into the directory), or by intension (with a query such as *size >500k*). Conversely to purely hierarchical structures, properties enable objects to be part of several overlapping sets.

Graphical search Graphical Search & Replace [11] allows users to search for elements based on their graphical properties (*designation*). The resulting selection allows the user to change at once a particular property for all found objects (*multiple scopes*).

User-defined macros and Programming by example User-defined macros allow for automation of repetitive tasks.

Programming by example allows non-experts to automate the creation of macros [14]. The user proceeds with an example of the task to repeat, and an algorithm abstracts the actions, so as to enable application on other objects. Programming by example relies on a correct interpretation of user actions by the machine. By contrast, our work aims at providing an instrumentation that does not rely on the intelligence (often not sufficient or incorrect) of the system.

Structuring with exploratory design Some structuring techniques have been design to support exploratory design. The list of reversible actions is an important mechanism [22][12]. Side Views displays previews of interactive commands [24]. Parallel Paths is a model of interactions to support alternative exploration [25]. It lies on an arborescence of creations, instead of a linear process, and on the simultaneous views of parallel results (*comparison*). Acting on a node "before" a split of the creation path enables users to manipulate the subsequent designs at once (*scope*).

Structuring for designers

Interaction designers have already identified the need for many modifications with a low number of actions. For example, the ergonomic criteria "brevity", and more precisely the sub-criteria "minimal actions", describes the needs to limit the number of steps of an interaction [3].

Cognitive dimensions In the cognitive dimensions of notation [7], the problem described in the software keyboard scenario is identified as "viscosity". It exhibits when the structure of the information contains a lot of dependences between parts, which implies that a small change leads to numerous adjustments from the user. Viscosity is a hurdle to modification and exploratory design [8]. Since it may be costly to apply the changes, the user refrains from exploring alternatives.

In order to reduce viscosity, a solution consists in creating an "abstraction", a "power command" that would act on several objects [8]. An abstraction is a class of entities, or a grouping of elements that users will handle as a single unit. This will not only reduce viscosity, but also make the notation closer to the conceptual model. Styles in a word processing application are an example of abstraction.

Abstraction can be costly. First, abstractions must be learnt. For example, a novice user will not be able to use the "style" abstraction if he does know that it exists, or how to use it. Their creation and modification requires time and effort. The investment in abstraction management (thinking, building, modifying) should be balanced with investment in repeating a small sequence of actions to solve a small problem. Besides, abstractions can represent a hurdle to exploratory design, if they are required before any other simple actions. For example, the use of classes in object-oriented languages forces users to think about abstract descriptions, before instancing and experimenting them (also known as premature commitment). Finally, abstraction may introduce hidden dependencies: some parts of the scene may depend on others in an invisible way, which makes it hard for the user to predict the effect of a change.

Instrumental interaction and design principles Direct [22] and instrumental [4] interactions techniques are efficient at interacting with a single object: they lower the number of actions required from the user compared to other techniques, such as command lines, conversational dialogue, or modal interactions. Design principles related to instrumental interaction, such as reification (turning an object into a thing), polymorphism (applying the same change to different class of objects) and reuse (of past selection and interactions result), cope partly with actions on multiple objects [4]. Reification of concepts into instruments allows for actions on instruments that in turn will act on other objects. Reifying concepts into first-class manipulable objects extends the possibilities of actions. The degrees of temporal and spatial indirection, compatibility and integration enable users to predict the effectiveness of a given instrument. Reification can be considered like enabling direct manipulation of an abstraction (of Cognitive Dimension). Polymorphism enables designers to augment the scope of actions with heterogeneous groups. The number of actions to perform is reduced with polymorphism, since this avoids repeating for each type of objects a change with similar results. Similarly, reuse reduces the number of actions required to manage objects sets (creation, modification).

Cost of interaction techniques A particular technique is not absolutely better than another, but may be better with respect to the task to accomplish: copy, modification, or problem solving (equivalent to exploratory design) [15]. CIS is a model that helps describe an interaction technique, analyze it, and predict is efficiency in the context of use [2]. An interface is defined as a set of manipulable objects of two types: work objects (e.g. graphical shapes), and tool objects (e.g. a menu item). There are two types of action: selection, which identifies a subset of the interaction space (moving an object, such as a cursor onto a tool) and validation, such as a mouse click, that confirms the selection action.

CIS defines four properties for interaction techniques. Order and parallelism characterize the scheduling of actions. Persistence indicates whether the technique modifies attributes of tool objects, which has an effect on subsequent interactions. Fusion is the ability of a technique to modify several work objects by defining multiple manipulations at once (similar to *scope*). Development corresponds to the ability offered to the user to create copies of tools object with different attribute values.

Structuring for programmers

The problems raised so far can also occur during development activities. For example, refactoring tools in IDEs is an answer to the need for multiple scopes of action: if the user changes say the name of a method of a class, the system will apply this change on each reference of the method, possibly in other classes or files. Styles and models can be implemented in a style language (e.g. CSS), with a hierarchical structuring. Changing a parameter in an intermediate node has an effect on its children. Tags in the Tk toolkit

allows the programmer to structure objects in overlapping sets (an object can belong to multiple sets) [20]. Changes can be applied to graphical shapes or to a tag, and thus to the set of objects that hold this tag (*scope*). Tags can be defined by extension (with designated objects) or by intension (with a predicate e.g. all blue objects) [1].

Prototype-based languages offer an alternative to class-based languages for object-oriented programming [13][19]. They offer a flexible creation model that allows sharing of properties and behaviors. Such mechanisms allow users to structure a hierarchy of prototypes and to act on several clones by manipulating a prototype in the delegation hierarchy. Morphic [16], the graphical interface of Self [26], reifies prototypes and clones into graphic objects (called Morphs), and allows for their construction and edition with direct manipulation. Tools have been designed to help structure a prototype hierarchy. For example, Guru is an algorithm that automatically creates a well-organized graph of prototypes, by factorizing shared properties into new prototypes [17].

REQUIREMENTS

In this section, we synthesize the requirements for the manipulation of objects through structures. The synthesis is derived from the contextual inquiries we ran, and our analysis of the related work.

R1: Managing sets of objects

Managing sets consists in *searching* (R1.1), and *designating* (R1.2) the objects that are part of a set. It is also necessary to *modify* (R1.3) the sets (add, remove elements). Finally, users must be able to *identify* (R1.4) the objects that belong to a particular set, or determine the sets a particular object belongs to.

R2: Managing actions

Managing actions consists in *specifying their nature* (e.g. by clicking on an 'alignment" icon, or a menu) (R2.1), *specifying their parameters* ("vertical" or "horizontal") (R2.2), *perceiving their consequences* (R2.3), and specifying the *scope* (R.2.4). Perceiving the consequences with appropriate feedback enables the user to realize the effects of its action after it is triggered [22], and even before it is triggered.

R3: Fostering exploratory design

In order to support exploratory design efficiently, it is important to provide the user with tools that enable her to *try* (R3.1) and *evaluate* (R3.2) solutions during short-term exploration (R3.3), and *compare* (R3.4) different *versions* during middle-term exploration (R3.5) [23]. When satisfied with the results, the user must be able to extend the modifications to other objects. If the system does not support this task efficiently, the user will have to repeat the same actions to propagate changes (viscosity). Finally, if structuring is a solution to the viscosity problem, it is a hurdle to exploration if required *a priori*. Therefore, structuring should be made *a posteriori* (R3.6) i.e. when actions have already been done.

INTERACTIVE TOOLS

We have explored a number of interaction techniques in order to provide the user with structure-based interactions. To design them, we have used traditional participatory design process with the users we interviewed. We have conducted ideation and prototyping sessions to fix the problem illustrated in the work scenarios, and to offer new ways of interacting with multiple objects. In the following, we present the tools and their usage that we designed.

Overview

To illustrate the interactive tools, we have designed a graphical drawing application. There are four parts: the tool palette on the left side, the workspace in the middle, the sample panel on the top right corner, and a property sheet on the bottom right corner (see Figure 5).

The workspace is the main view, where users can create a new object, by clicking and resizing. Selection is performed by clicking on an object or by drawing a rubber rectangle to encompass several items at once, as done with usual graphics editors. Selected items are highlighted with a shadow, while other items are made translucent.

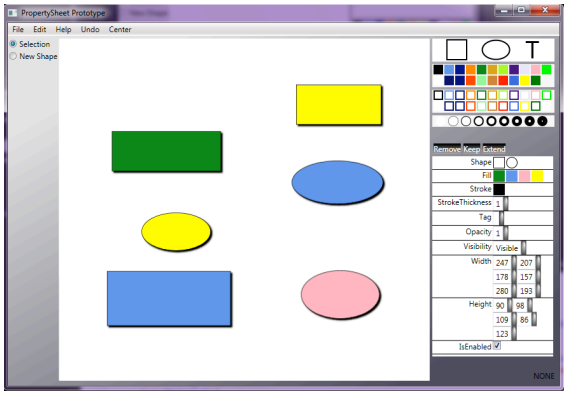

Figure 5: Overview of the application. The workspace is at the center, the samples at the top right and the property sheet at the bottom right.

The samples panel contains a set of values for shape (square, oval, T for text), fill color (represented by a colored square), stroke color (stroked-only colored square) and stroke thickness (stroked-only circle). In order to modify a property of an object in the main view, users can drag a sample and drop it onto the object. Feedback is shown as soon as the sample hovers over the object, in order for the user to understand the action, and to assess the change before effectively applying it by releasing the mouse button. This enables the user to cancel the action, by releasing the button outside of any object (*R3.1 try, R3.2 evaluate, R3.3 short term, R3.4 compare, R2.3 perceiving consequences*). Drag and dropping samples also applies to a selection with multiple objects. The interactions described so far are not new. Next section presents the property sheet with novel interactions.

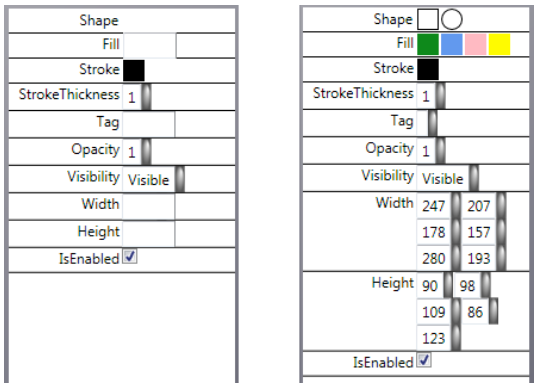

Figure 6: The user's selection contains objects with varying shapes, fill colors, width, and height. A classical property sheet (left) displays a blank fill for those properties, whereas our property sheet (at right) displays all different values.

Implicit structure: an enhanced property sheet

A property sheet (or property box) is a window containing a vertical list of pairs of property type and value (e.g. shape: rectangle, color: green, thickness: 3…). A property sheet offers two services to the user: visualizing values (with progressive disclosure [9]), and modifying them [9]. With classical property sheets, if multiple objects are selected, only "shared values" (i.e. values shared by all objects) are shown and are modifiable (see Figure 6, left). Users can change a shared value for a property type, and the system reflects changes to all selected objects (power of action). Other properties, those that are multi-valuated, still appear, but with a blank fill. Those blank fills do not inform users with the values and cannot be modified.

Our version of the property sheet differs in that it shows all values for a multi-valuated property (see Figure 6, right), instead of displaying blank. This reveals an *implicit* structure of graphics, the sets of objects that share a graphical property. Though not explicitly defined by the user, we think that such sets may be useful, since users sometimes think about objects with a graphical predicate ("all red objects"). We relied on the display of those values to design a set of interactions that offer new services for exploratory design and structure-based interaction: query and selection of objects with graphic examples, selection refinement, properties modification on multiple objects with precision, and content correction. The representation of a shared value in the property sheet actually refers to two concepts: the value in itself, and the set of selected objects that exhibits this property value. As a value per se, and similarly to the interaction with the samples panel, users can drag the representation (considered as a value) from the property sheet onto (a selection of) objects in the main view to modify a property. If the shared value is numerical, users can hover over it and rotate the mouse wheel to increment or decrement it (power and precision). Together with immediate feedback, this enables both exploration and precise adjustment of properties, thus reducing temporal offset [4] between action and reaction.

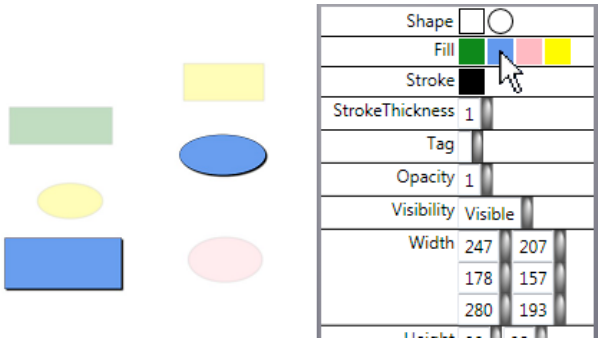

Figure 7: The user's cursor is over the blue shared value of the fill property (fill: blue). Because they don't have this shared value, the green rectangle, the pink circle and the two yellow shapes are dimmed

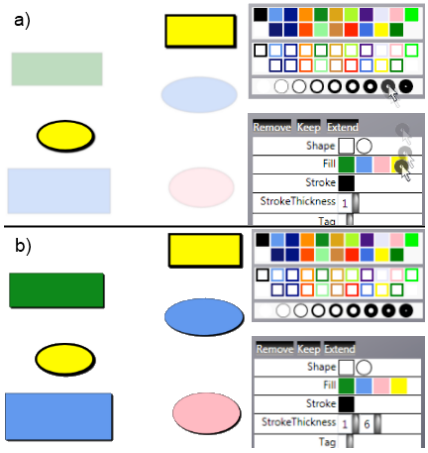

Figure 8: a) The user drags the "thickness: 6pt stroke" sample over the "fill: yellow" shared value. a) Immediate feedback turns the stroke thickness of all yellow items to 6pt. b) The user has dropped the sample, the modification is applied.

Since the representation of a shared value also reifies [5] a set of objects, hovering over a shared value highlights the concerned objects while blurring others (Figure 7). This makes it easy to figure out which set is made of what, and possibly detects outliers and correct them. In addition, users can drag a sample (hence a value) from the sample panels onto the representation of a shared value (considered as a set of objects) in the property sheet to modify at once one property of multiple objects (*R2.4 scope*) (Figure 8). Users can also drag a representation from (value) and in (set) the property sheet (Figure 9).

To select objects, users can directly click on them in the workspace, or draw a selection rectangle. In order to refine the selection, users can use three meta-instruments (i.e. instrument that control instrument, here the selection): *Remover, Keeper* and *Extender*. The interaction consists in a drag and drop of the representation of the instrument onto a shared value. *Remover* throws out of the selection all objects that have this shared value (Figure 10). *Keeper* keeps the objects that have this shared value in the selection, and

throws away the others. *Extender* adds to the selection all objects that are not selected but possess this shared value. These interactions extend the set of example-based queries introduced above (*R1.3 modify sets*).

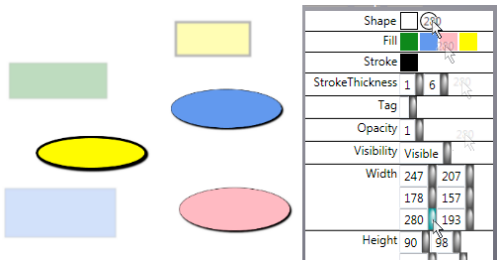

Figure 9: The user drags the "width: 280" shared value and drops it on the "shape: circle" shared value. All circles in the selection now have a width set to 280.

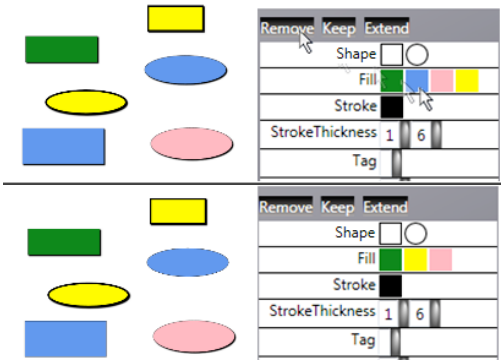

Figure 10: The user drags the Remove instrument and drops it on the blue fill shared value. All blue objects are removed from the selection.

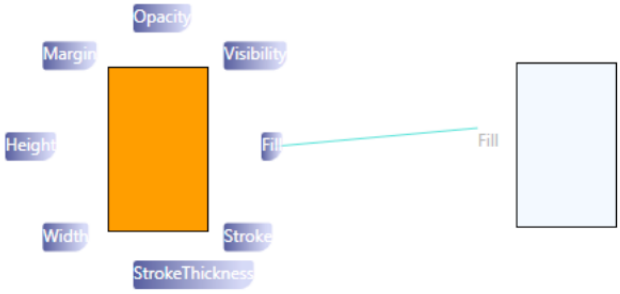

Figure 11: The user draws a link between the Fill property into an object to specify a dependency

Explicit structure: the property delegation graph

Besides the augmented property sheet, we have explored an interactive tool that enables users to structure the content explicitly. Users can specify that the property of an object (the clone) depend on the property of another object (the prototype). A prototype plays the role of the master in Sketchpad: when users change a property of a prototype by dropping a sample from the property sheet onto the prototype, all dependent clones are changed accordingly (*R2.4 scope*, *R1.3 modify sets*).

The interaction to specify a dependency is as follows (Figure 11): by clicking on an object, users can toggle the display of the properties around it. They can press on a property, draw an elastic link, and drop it onto another object. The clone object appearance reflects immediately the appearance of the clone for that property. The interaction is coherent with the interaction between a sample of the property sheet and an object.

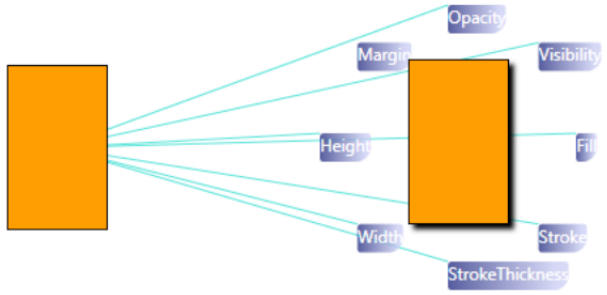

Figure 12: A prototype (left), and a clone (right)

As seen above, the management of any structure can be cumbersome. To minimize this aspect, our system proposes two ways of creating objects from others. Users can create a new object from a previous one, either by copying it, or by cloning it (*R1.3 modify sets*). Copying is the regular copy operation: properties from the copy are independent from the properties of the source. Cloning enables users to get a clone, whose properties are entirely delegated to the prototype (Figure 12). By creating a clone, users minimize the number of actions required to specify a single difference with the prototype: if they have copied instead of cloned, they would have to link all shared properties.

Choosing to clone or to copy may be premature at the moment of the creation of a new object from an existing one. To solve this problem, users can decide to change them to a copy or a clone after the creation of the object (*R1.3 modify sets*, *R3.6 a posteriori structuring*). This is made possible by tracing the history of objects, and how they were created. Toggling between copy and clone only affects the properties that were not set explicitly by the user (be it a value or a delegation).

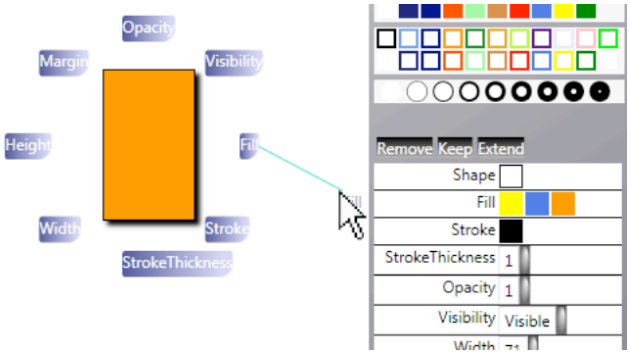

Figure 13: The fill property is dragged on a shared value to specify that the fill property of a set of objects depends on the prototype (left).

When users find out that an object could be used as a prototype, one problem is to interact with similar objects in other to make them depend on the new prototype. A viscous solution would be to interact with each object and making it a clone of the prototype. A more efficient solution consists in selecting the objects that are be clones, and drop the property of the prototypes onto an object of the selection (*R1.3 modify sets*, *R3.6 a posteriori structuring*). To increase efficiency and integration, we have leveraged off the property sheet: the interaction consists in dropping the property onto a shared value, or the name of the property (Figure 13).

DISCUSSION

Comparison with CIS

CIS proposes two dimensions of analysis related to our problem: fusion and development. Fusion is related to the modification of several objects at once, and development is related to supporting the user in creating her own tools. However, we think that those two dimensions are not sufficient to express all concepts underlying actions on multiple objects at once.

Comparison with the delegation programming model

The property delegation graph is an extension of the delegation tree found in prototype-based languages [13]. In a prototype tree, each node is linked to a prototype (a parent in the tree). A child can delegate the value of a property to its prototype. When drawing an object, the delegation algorithm checks whether a property is specified for a particular node. If not, the algorithm follows recursively the prototype branch, until the property is specified in an ancestor, and uses the value of that property for drawing.

With a tree, objects cannot have multiple parents. For example, the scene tree available in illustrator may be helpful to conceptualize the scene, but is unable to help specify cross-branches relationships. Conversely to a tree, a node in our graph of properties can have multiple parents. This enables users to be more specific about the parent that holds a particular property. Hence, a node can delegate the fill property to a prototype A, and the stroke-width property to a prototype B.

Drawbacks of the current design

The prototypes we have explored are far from being perfect. Even if we have crafted the interactions carefully so as to make them coherent and integrated as much as possible, we had to make compromise.

More work needs to done with respect to scalability. For example, the property sheet is not able to handle numerous shared values. A solution would be to provide a progressive disclosure of shared values, e.g. with scrollbars. The prototype/clone view also needs more work: if the links are numerous, the scene may result in a mess of tangled links. Again, progressive disclosure is a possible solution. We are also exploring other representations and interactions from the information visualization field.

Some interactions need to be complemented. For example, the system does not check for cycle when the user tries to link two properties. Together with forbidding, appropriate feedback is necessary, such as displaying the links to show the cycle when hovering over a property.

Furthermore, even if we designed them during participatory sessions with actual users, we did not evaluate them. In particular, we did not work on the discoverability of the interface: for example, there may be some concerns about the fact that users will not use shared values, either because shared values do not afford dropping samples on them, or because they are too complicated to understand.

CONCLUSION

In this paper, we have tackled the problem of the interaction with structures, and the interaction with content through structures. We have illustrated its importance in scenarios based on contextual inquiries, and showed that it underlies a number of past work. We have identified a set of requirements for the management of structures of interactive graphics, and the use of structures to act on the content. Based on the requirements, we have explored a set of interactions that provide partial solutions to the requirements: an enhanced property sheet, and a property-delegation graph, together with coherent and well-integrated interactions.

The examples we have shown involve drawing editors. However, this work also applies to any editor that uses graphics to display information, rather than creating them. For example, managing a calendar can benefit from the interaction we have designed: setting a particular duration or location to a set of entries, setting a location known one week after a set of meetings were scheduled, changing the title of entries that occur every Monday etc.

Our design aimed at illustrating the requirements and possible solutions, but other designs are possible. Though we think that structuring is important, we do not know yet how users actually benefit from it. During the evaluation of CIS, their authors witness that users were able to optimize their use of an interaction technique, and to adapt it to the current context [2]. We plan to design experiments that aim at assessing the usefulness of structuring, and to what extent it can support exploratory design.

ACKNOWLEDGMENTS

The authors would like to acknowledge colleagues.

REFERENCES

1. Appert, C., Beaudouin-Lafon, M. SwingStates: adding state machines to the swing toolkit. In Proc. of UIST ‘06. ACM, 2006, pp. 319-322.

2. Appert, C, Beaudoin-Lafon, M, Mackay, W. E. Context matters: Evaluating Interaction Techniques with the CIS Model. In Proc. of HCI'04, p 279-295. Springer Verlag, 2004.

3. Bastien, J.M.C, Scapin, D.L. Critères Ergonomiques pour l'Évaluation d'Interfaces Utilisateurs. Tech Re-port 156, May 1993.

4. Beaudouin-Lafon, M. Instrumental Interaction: An Interaction Model for Designing Post-WIMP User Interfaces. In Proc. CHI'00. (2000), ACM, 446-453.

5. Beaudouin-Lafon, M. and Mackay,AW. E. Reification, polymorphism and reuse: three principles for designing visual interfaces. In Proc. of AVI‘00. ACM, 2000, pp. 102-109.

6. Dourish, P., Edwards W.K., LaMarca, A. and Salisbury, M. 1999. Presto: an experimental architecture for fluid interactive document spaces. ACM Trans. Comput.-Hum. Interact. 6, 2 (June 1999), 133-161.

7. Green, T.R.G., Cognitive dimensions of notations, in People and computers v, 1989, Cambridge Uni-ersity Press, 443-460.

8. Green, T.R.G, and Blackwell, A. Cognitive dimensions of information artifacts: a tutorial. (Version 1.2 October 1998). 1998.

9. Johnson J.A., Roberts T.L., Verplank W., Smith D.C., Irby C.H., Beard M., and Mackey K. The Xerox Star: A retrospective. IEEE Computer, 22(9):11–29, 1989.

10. Kurlander, D. Reducing Repetition in Graphical Editing. Proc. of HCI International ‘93. 1993.

11. Kurlander, D, Bier, E.A. Graphical Search and Replace. In Proc. of ACM SIGGRAPH '88, 113-120.

12. Kurlander, D., Feiner, S. A History-Based Macro By Example System. In Proc. of UIST '92. ACM, 99-106.

13. Lieberman, H. 1986. Using prototypical objects to implement shared behavior in object-oriented systems. In Proc. of OOPLSA '86. ACM, 214-223.

14. Lieberman, H. Your Wish is my command: Programming by example. Morgan Kaufmann, 2001.

15. Mackay W.E. Which interaction technique works when?: floating palettes, marking menus and toolglasses support different task strategies. In Proc. of AVI '02. ACM, 203-208.

16. Maloney, J.H. and Smith, R.B. 1995. Directness and liveness in the morphic user interface construction environment. In Proc. UIST '95. ACM, 21-28.

17. Moore, I. 1996. Automatic inheritance hierarchy restructuring and method refactoring. In Proc. of OOPSLA '96. ACM, 235-250.

18. Myers, B. A. 1991. Graphical techniques in a spreadsheet for specifying user interfaces. In Proc. of CHI '91, ACM, 243-249.

19. Myers, B. A., Giuse, D. A. and Zanden, B V. 1992. Declarative programming in a prototype-instance system: object-oriented programming without writing methods. SIGPLAN Not. 27, 10 (October 1992), 184-200.

20. Ousterhout, J. K. (1994) Tcl and the Tk Toolkit. Addison-Wesley.

21. Sutherland I.E. 1963. Sketchpad: a man-machine graphical communication system. In Proc. of AFIPS '63. ACM, 329-346.

22. Shneiderman, B. (1983). Direct manipulation: a step beyond programming languages. IEEE Computer 16, 8, 57–69.

23. Terry, M. and Mynatt E.D. Recognizing creative needs in user interface design. Proc. of Creativity & Cognition. ACM, pp. 38-44, 2002.

24. Terry, M. and Mynatt, E. D. Side views: persistent, on-demand previews for open-ended tasks. In Proc. of UIST 2002.ACM, 2002, pp. 71-80.

25. Terry M, Mynatt E.D, Nakakoji K, and Yamamoto Y. 2004. Variation in element and action: supporting simultaneous development of alternative solutions. In Proc. of CHI '04. ACM, 711-718.

26. Ungar, D, Smith R, B. SELF: The Power of Simplicity. In proc. of OOPSLA '87. ACM, 227-242.