Where to go and what to show
More patterns for a pattern language of interactive information graphics

Christian Kohls
Knowledge Media Research Center
Konrad-Adenauer-Str. 40
72072 Tuebingen, Germany
+49/7071/979-103
c.kohls@iwm-kmrc.de

Tobias Windbrake
University of Applied Sciences Wedel
Feldstrasse 143
22880 Wedel, Germany
wb@fh-wedel.de

ABSTRACT

Interactive graphics are an effective way of communication and information delivery, especially for complex domains. However, domain experts are rarely aware of the potentials of interactive visual displays and which interaction principles can be in charge for communication and teaching purposes. In this paper we extend a pattern language for interactive information graphics and present four new patterns. These patterns are all based on drag operations and explain how to define flexible area restrictions and how to change the visual appearance of elements according to their positions.

Categories and Subject Descriptors

K.3 [**Computers and Education**]

General Terms

Design, Human Factors

Keywords

Design patterns, interactive graphic, educational patterns, e-learning

1. INTRODUCTION

In our pattern language we describe interaction principles for visual communication and information delivery. Interactive graphics certainly helps to analyze, understand and communicate models, concepts and data if applied appropriately [16]. Each pattern explicitly names situations where to use what kind of interaction and gives a rationale why it can support the communication process. The patterns can be used in the design of educational settings, edutainment kiosks in museums and exhibitions, materials for interactive whiteboards and Tablet-PCs.

As we observed users working with various authoring tools (Adobe Flash, Adobe Director, Matchware Mediator, Toolbook etc.) we became aware of some major problems. First, for many content applications the same interactions had to be re-implemented again and again. Second, the implementation of interactive content was only possible for programmers. Domain experts had to either learn programming or outsource the production of interactive content. Third, people without experience in multimedia did not know what forms of interaction were available, and when and why to use interactive elements. Therefore we started to write down the interaction forms we have used many times by ourselves and extracted the invariant parts.

To find more patterns we analyzed about 600 interactive information graphics and systematically extracted the patterns. We choose multimedia titles that were either very popular (best selling titles), best practice (award winners, showcases), or had a strong marketing (brands, interactive features were advertised). To consider a wide range of different visualization types we tried to find titles that differed in domain (e.g. science, language, art), target group (e.g. K-12, higher education, trainings), publishers (e.g. school book publishers, mass media) and genre (e.g. CD-ROM, web based training, interactive whiteboard content). For each title we scanned the content for interactive screens. For each interactive screen we analyzed the interaction rules and roles of visual elements.

The patterns we have identified can be implemented in any authoring environment. They should be in charge to find adequate visual interaction forms and to reason why and when to use them. While the solution part covers the roles, states and interactions of visual elements, it does not address the technical implementation. The reason is that the way software tools support the implementation of a pattern can vary a lot. The range starts with programming an application from scratch, i.e. writing a Java applet to visualize information, and ends with templates, i.e. ready-to-use solutions as they can be found in tools such as Raptivity or Articulate Engage. We have created our own tool, moowinx, which tries to balance between these two ends [8]. It

Preliminary versions of these papers were workshopped at Pattern Languages of Programming (PLoP) ’07 September 5-8, 2007, Monticello, IL, USA. ISBN: 978-1-60558-411-9.

provides wizards to choose from one of the patterns and then leads the user step by step to an interactive graphic. The user selects involved visual elements and configures all interaction. In opposition to the use of templates the user can edit each visual object on a slide individually. Editing a slide is done similar to popular presentation tools found in office software. However, the slide content can be brought to life without any programming by using the patterns of interaction [9].

The resulting pattern language for interactive information graphics relates or overlaps with languages for human computer interaction [1, 2, 7, 18], the design of websites [20, 17, 5] and patterns for e-learning. More patterns can be found in [10, 11]. An overview map of the language is given at the end of this paper.

In this paper we will focus on four patterns that provide some rules for drag & drop operations of visual elements. The first two patterns, ACTIVE AREAs and ACTIVATOR, address the visual appearance of elements that are dragged over other elements. ACTIVE AREAS change the dragged element, e.g. set a new image. On the other hand an Activator changes the areas in the background. The patterns NO-GO-AREA and SANDBOX, too, relate to each other. Both provide rules to which areas an element can be dragged. The NO-GO-AREA defines negative areas; that is areas that cannot be entered. In opposition, the SANDBOX defines positive areas that cannot be exited. From a technical point of view the two patterns are almost redundant because each pattern can be used to fulfil the goals of the other pattern – it is just a question of minimizing the areas you have to define explicitly. However, having the semantics of interactive graphics in mind, the two patterns make an important distinction between places to (not) go and places to stay in.

2. Active Areas (Hot Areas)

Display different labels, information or states of an object according to a local spatial context or background.

2.1 Examples

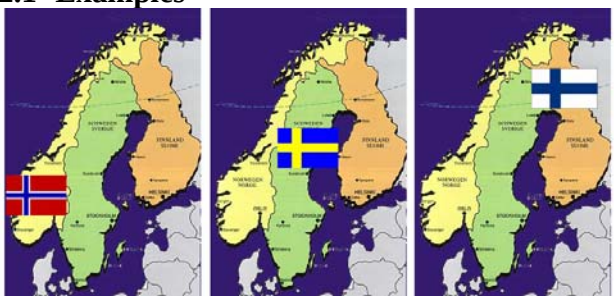

Figure 1. Map: The flag can be dragged over each country. It shows the flag of the country dynamically.

Figure 2. Dragable label: The text can be dragged over each employee and shows his/her name and job description.

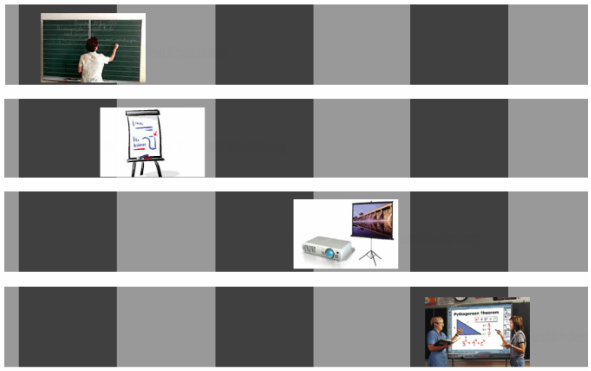

Figure 3. Timeline: The picture can be dragged along a timeline and shows several development states of blackboards and whiteboards.

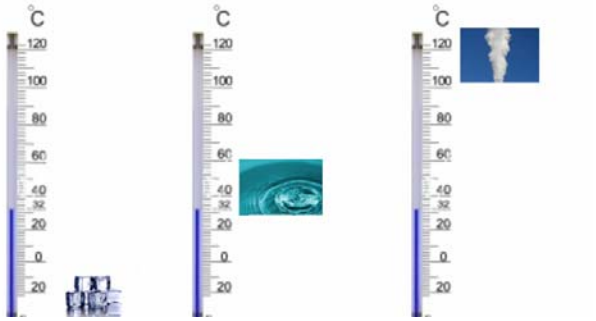

Figure 4. Water states: the state of water changes according to the temperature.

2.2 Problem

If a graphic consists of several areas where different information items should be provided, e.g. text labels, info boxes or bubbles, it becomes hard to display all information at the same time. Not only is the graphic too packed with information and thus becomes more complex, but the limited space of computer screens sets bounds to the number of information items visible at the same time. It is also true that sometimes only one area should be promoted with additional information, e.g. to control the viewer's attention. Showing all area dependent states at the same time does not show that an object can only be in one state at a time and at which time the state actually changes.

2.3 Solution

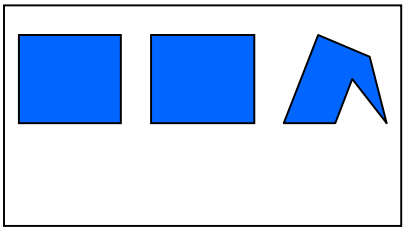

Figure 5. Define *Active Areas* by geometrical shapes.

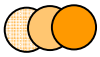

Figure 6. Define a *Morphable Object* that can be dragged.

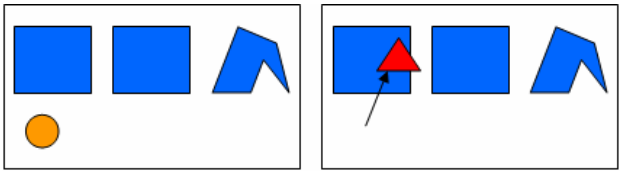

Figure 7. If the *Morphable Objects* enters an Active Area, it will change its appearance.

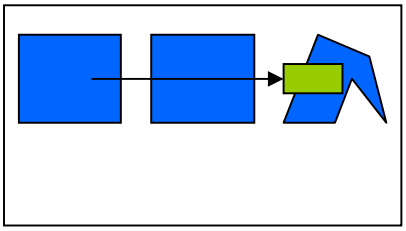

Figure 8. For each *Active Area*, the *Morphable Object* can have a different appearance.

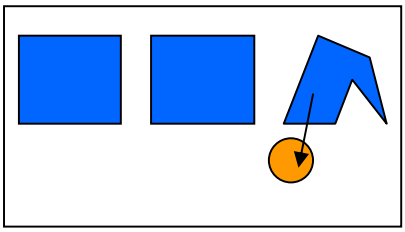

Figure 9. If the *Morphable Object* is dragged out of all *Active Areas*, it will return to its default appearance.

2.4 Details

Each *Morphable Object* relates to one or more *Active Areas*. The *Morphable Object* can change its position, for example the users drags it with a mouse pointer. *Active Areas* usually remain at fixed positions at the screens. However, this is not a requirement as the following example shows:

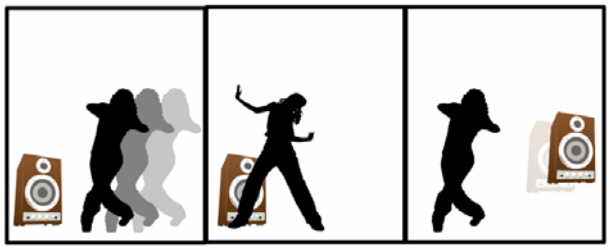

Figure 10. Moving an *Active Area*

Figure 10 shows a girl that either walks or dances (*Morphable Object*). By default the girl walks. If she is dragged over the speaker (*Active Area*) she starts dancing. However, on dragging the speaker away, the girl exits the *Active Area* and stops dancing.

Spatial properties are used to detect at which time a *Morphable Object* enters or exits an *Active Area*. There are three common strategies to distinguish whether an object is inside or outside of another object:

Hot Spot: A single point of the *Morphable Object* is defined as a hot spot. The *Morphable Object* is considered inside of an *Active Area* as soon as the hot spot point is located inside of that *Active Area*. The hot spot is usually positioned in the centre of the *Morphable Object* or in one of its corners. A well-known example for hot spots is the pixel of your mouse pointer that actually triggers mouse clicks or roll-over effects; the hot spot of your mouse pointer is located at the end of the arrow.

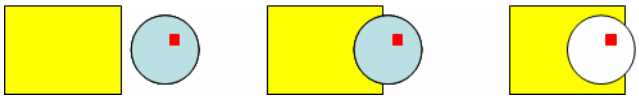

Figure 11. The red point is the Hot Spot. If the Hot Spot enters the rectangle then the circle is inside (last figure).

Intersection: The Indicator is considered to be inside of an Active Area as soon as it intersects with it.

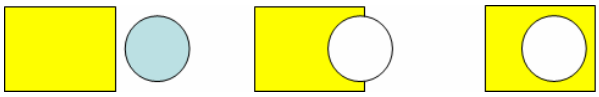

Figure 12. As soon as some points of the circle intersect the rectangle, the circle is inside (figure in the middle).

Inclusion: The *Morphable Object* is considered to be inside as soon as its bounding box is completely inside of the bounding box of the *Active Area.*

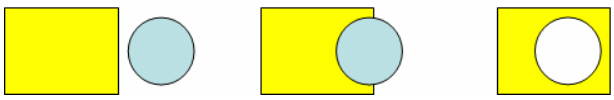

Figure 13. Only if all points of the circle are inside the rectangle, the circle is inside (last figure).

Spatially a *Morphable Object* could be inside of multiple *Active Areas* at the same time. Logically, a *Morphable Object* can only be inside of one *Active Area* because only one visual appearance can be set at a time. To resolve this conflict, one can set priorities for each area. If an element is spatially inside of two or more areas, it is considered logically inside of the area with the highest priority. In most graphical editors the z-order of visual objects (that is which objects appear in front of other objects) provides an implicit prioritization. Areas that are closer to the observer have a higher priority. If intersection is used, another option is available: to indicate that an element is inside of an *Active Area*, the *Morphable Object* can be considered inside of the area that it mostly intersects.

An *Active Area* can be defined by primitive shapes such as rectangles, ellipses, polygons or the bounding box of an image. Very often, polygons are used to define *Active Areas* on images.

Figure 14. Polygons overlay an image

Using overlay polygons you can handle different areas within one single image, e.g. a geographical map. After editing the interactive graphic, the polygons are set to transparent . If a text label (*Morphable Object)* is dragged over the invisible areas, it changes its text accordingly.

Figure 15. Dynamic text label

The example also shows that there may be other supporting objects on the screen that do not affect the behaviour of *Morphable Objects.* The image in the background is the main subject of this interactive graphic, however, it does not play any active part. Instead, the polygons drawn over the image define several *Active Areas.*

There are many ways in which the *Morphable Object* can change its appearance, most commonly are:

- Change the displayed image to show another state or information.
- Use another text to label an area.
- Change the colour, shape or opacity of the *Morphable Object* to highlight that it is over an interesting area.

In general terms, changing the appearance means some visual properties change their values, e.g. file path of an image, level of brightness, colour settings etc. However, it is critical to not change spatial properties that are being used to determine hot spot, inclusion, or intersection. Location, size and rotation are critical properties, accordingly.

2.5 What else can I do with this?

- You have a graphic consisting of multiple images and each image should be labelled individually by a dragable text or symbol that can change its state according to the background.
- You have an image with several sub-areas and each area should be labelled individually by a dragable text or symbol that can change its state according to the background.
- You want to use a single object to annotate or label some other graphic elements.
- The appearance of an object depends on its current background information, e.g. different landscapes, regions, data or objects.
- Drag an image along a timeline and show different epochal pictures depending on its position on the timeline.
- Change a displayed image according to its horizontal and vertical position.
- Show the state of an object at different stations.

2.6 Rationale

Showing diverse states according to a context provided by background graphics or spatial position makes cause and effect of object changes more obviously [15]. The visual display of the *Morphable Object* always relates to the nearest background; thus, the gestalt law of proximity is applied [3]. Labels should always be close to the subject of interest [4]. Labels at a different position could distract the user [12]. Another reason to use one but many labels is to direct the attention of the audience and to reduce the complexity a graphic. Reducing the details of a graphic can support the cognitive processing of the content. From a technical point of view the number of simultaneously displayed items has to be reduced because of a low screen resolution.

2.7 Related patterns

Activator: The ACTIVATOR pattern is the inverse of this pattern. In both patterns a pair of two states is used for each relation of a dragable element and a fixed area. In ACTIVE AREAS the dragged object changes to an alternative state. In the ACTIVATOR pattern the area in the background changes.

Roll-Over: Instead of dragging an element around, the mouse pointer directly moves over active areas and causes other objects to change, e.g. labels can switch from invisible to visible or another fixed element changes its appearance. ROLL-OVER images are easier to understand for end-users, however, it is hard to leave the mouse pointer within an area for a longer time and you cannot perform other operation at the same time.

3. Activator

Activate an image or visual object on demand.

3.1 Examples

Figure 16. X-Rays: An inspector image can be dragged over each person and the skeleton is shown.

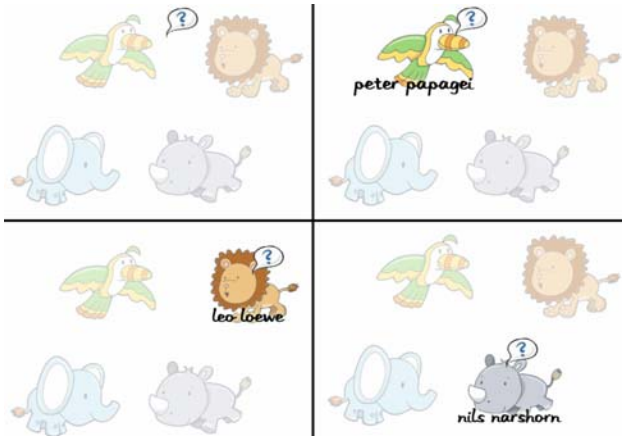

Figure 17. Labels and highlights: Dragging the bubble with the question mark over an animal shows its name.

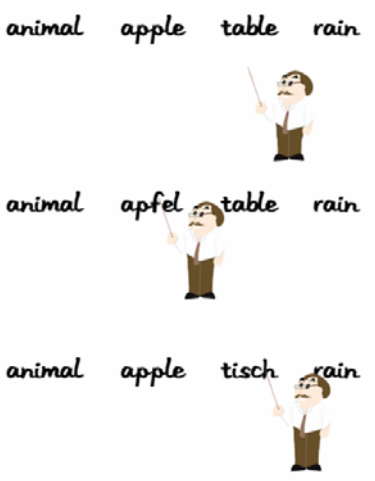

Figure 18. Translate vocabulary: Pointing at a vocabulary translates it.

Figure 19. Hidden objects: Dragging the night watchman over the images illuminates them.

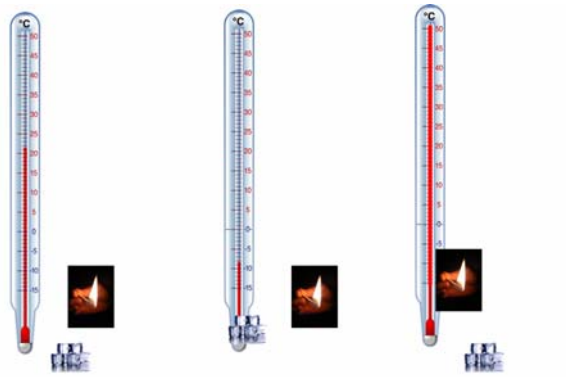

Figure 20. Show temperature: The thermometer is an image in the background that can be changed in different ways according to the object that is dragged over it. Ice cubes change the thermometer in a different way than a flame. Also, it is perceivable that the thermometer changes because a specific type of object is dragged over it.

3.2 Problem

Objects of the real world can be represented visually in more than one way, e.g. you can switch between several states, change the

perspective or enrich/reduce the image with information. For objects that can switch between two visual representations there must be an explicit trigger to activate the alternative view. If the user cannot recognize whether or not the displayed image represents the active or inactive state, there must be a visual clue to indicate that an image is activated. Sometimes the mouse pointer can be used to activate an image and switch between two different states. However, a mouse pointer can only indicate the active state for one image at a time. Also, a mouse pointer is a very abstract indicator and does not reveal how or why an object switches to an alternative state. Using the mouse pointer does not add any semantic to a graphic.

3.3 Solution

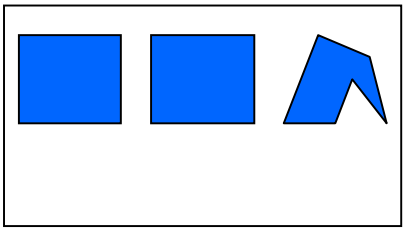

Figure 21. Define one or more *Morphable Objects* which can change their appearance on activation.

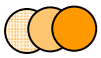

Figure 22. Define an *Activator* that can be dragged.

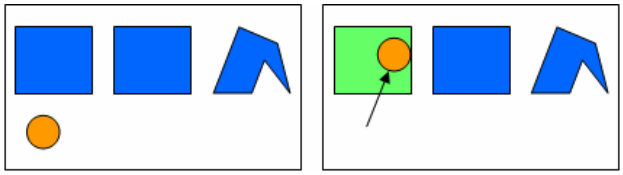

Figure 23. If the *Activator* enters a *Morphable Object,* the appearance of the *Morphable Object* changes.

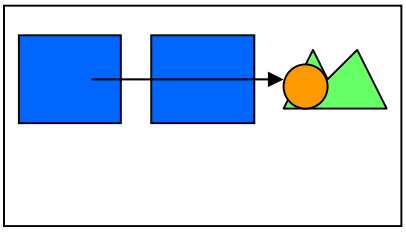

Figure 24. If the *Acvtivator* is dragged over another *Morphable Object* it activates that object.

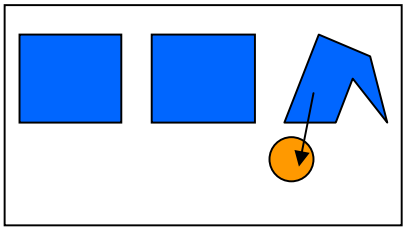

Figure 25. Each *Morphable Object* may change its appearance differently on activation. In general, if the *Activator* exits a *Morphable Object,* then the object becomes inactive and its default appearance is shown.

3.4 Details

One *Activator* can activate multiple *Morphable Objects*. Activation is triggered on intersection, inclusion or by a hot spot. On activation, visual properties of the *Morphable Object* change. Common examples for property changes are illustrated below.

Figure 26. Image file property

Change the displayed image to show an object in active or inactive state, e.g. a machine that is turned on or off.

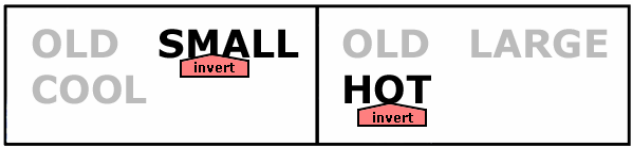

Figure 27. Text property

Set another text to change between two different perspectives, i.e. change the language of a phrase, switch from a pro argument to a contra argument, or a verbal description of before and after situations.

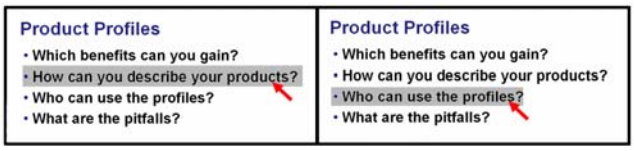

Figure 28. Colour property

Change the colour or background colour of an object to direct the viewer's attention to that object.

Figure 29. Opacity property

Change the opacity to highlight, to reveal or to hide information temporarily.

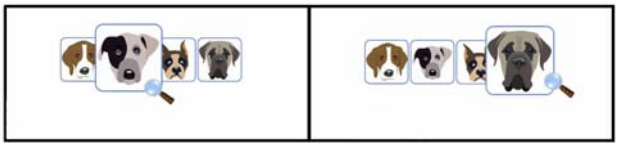

Figure 30. Size and location property

Set a larger size or relocate an object to let it pop out.

The *Activator* behaves similar to a mouse pointer but introduces some advantages. Mouse rollovers can activate an image only temporarily whereas *Activators* can activate an object permanently. *Activators* can be larger than mouse pointers and thus are better perceived in presentations. Also, an *Activator* can be of any shape and therefore provide additional information about what kind of activation or transformation is performed. In the previous examples the *Activator* "Next exercise" tells the user why the specific image is highlighted, and the magnifying glass indicates that it can be used to enlarge images.

With a mouse pointer only one object can be activated at a time. With *Activators* one can multiply the number of pointers as shown in the next example:

Figure 31: Two activators simultaneously

There are two pointers that can be dragged over an image icon to highlight it. The pointers can remain over the images while the mouse pointer can move independently to do other things. The two pointers highlight the image in different ways as well. Used in a presentation, the presenter can refer to differently highlighted images.

However, if there are two or more *Activators* that can activate the same *Morphable Object* then we are running into a conflict. Which activation wins? Or, does the activation by two *Activators* simultaneously trigger a special visual state for the *Morphable Object?* Dragging two different molecules (*Activators)* into a substance (*Morphable Object*) could cause a specific chemical reaction – meaning that the *Morphable Object* shows different results depending on the input of multiple *Activators*. In that case we would have to define special visual states for the *Morphable Object* for all possible combination of *Activators*, i.e. with n *Activators* one has to define 2^n states. To simplify the behaviour of *Activators* it is recommended to allow only one activation at a time. If multiple *Activators* point at a *Morphable Object* the latest activation should be prioritized.

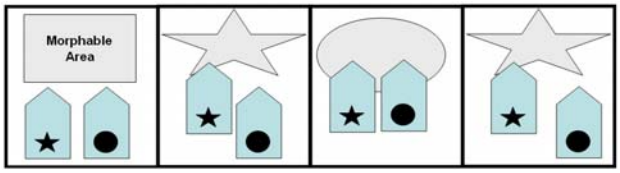

Figure 32. Multiple *Activators* for one *Morphable Object*

The star arrow (*Activator 1)* first activates the *Morphable Object*. Then, the ellipse arrow (*Activator 2*) activates the *Morphable Object* additionally. Because the latest activation is prioritized, the *Morphable Object* shows an ellipse. On dragging the ellipse arrow out of the *Morphable Object* the star arrow is still activating the *Morphable Object*.

The safest way to avoid confusion is to not use multiple *Activators* for the same *Morphable Object*. Using multiple *Activators* for different sets of *Morphable Objects* does not cause such problems. Objects that are not related to a specific *Activator* do not affect the behaviour.

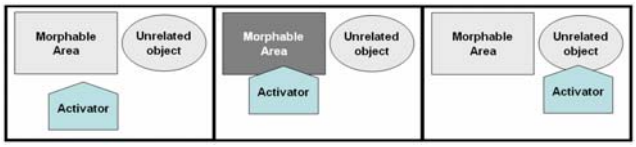

Figure 33. Unrelated objects

The *Activator* only changes the visual appearance of the *Morphable Object*. On dragging the *Activator* over the unrelated object nothing happens.

A drawback of *Activators* as compared to mouse pointers is that they introduce an additional level of indirectness. While mouse pointers may activate and deactivate images implicitly and very intuitive, an *Activator* must be dragged explicitly. Because the mouse pointer first has to be moved to the *Activator* to drag it, additional workload is involved. Since *Activators* can come in any shape, a visual affordance that indicates its function is recommended.

3.5 What else can I do with this?

- Activate or highlight one of a set of objects. Provide an explicit pointer to the highlighted object.
- Direct the viewer's attention to a specific object on the screen by using a permanent pointer.

- Change the state of an object as soon as it gets in contact with another object.
- Switch between outer and inner view of an object by dragging an “inspector” image over it.
- Hide one representation of an object unless it is explicitly uncovered.
- Map one representation A to another representation B on demand.
- Turn around playing cards or note cards.
- Show a question and reveal the solution by activating the item.
- Translate a text item by pointing on the text.
- Switch between text and image representation.
- Switch between before-after states by using a “transformer” image.

3.6 Rationale

Pointing at objects is a very important tool to direct attention [19]. Activators can point permanently at objects and provide additional semantic. Focussing single objects out of a group helps to concentrate on details. Setting and showing visual states on demand can be helpful for both discovering and demonstrating different views of one object.

3.7 Related patterns

Active Areas: The Active Areas pattern is the inverse of this pattern.
Roll-Over: (see description in the Active Areas pattern).

4. No-Go-Areas

Restrict the areas into which you can drag an object.

4.1 Examples

Figure 34 . Stay on ground: You cannot drag a person over the sea or into the sky.

Figure 35. Protect info boxes: You cannot drag the white box over the gray info box.

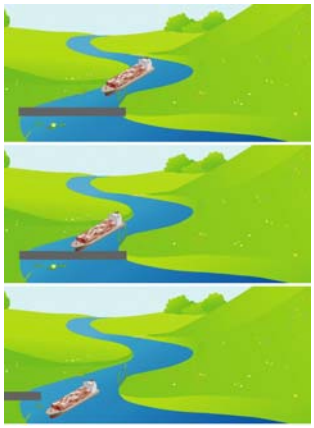

Figure 36. Channels: The ship cannot be dragged to the coastline and it cannot pass the water gate until it is opened.

4.2 Problem

If the user has all levels of freedom to drag a visual object then he can drag objects to inadequate places by intend or accident. Interactive graphics can break into invalid states, e.g. ships could fly if you drag them into the sky, or, persons can just be dragged through walls. Forbidden operations can be triggered if you can drag a graphic to all areas, e.g. adding an object to a misfitting set. Dragging an image over any area on the screen could hide important information or handicap the access to important graphic user interface elements.

4.3 Solution

Figure 37. Define *No-Go-Areas*.

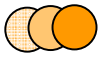

Figure 38. Define dragable objects as *Tourists*.

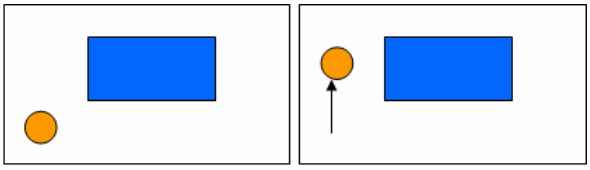

Figure 39. A *Tourist* object cannot enter the *No-Go-Area*. Outside of the *No-Go-Area* the *Tourist* can move without restrictions.

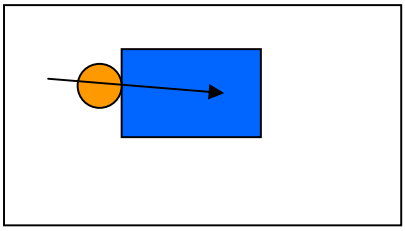

Figure 40. If the user drags the *Tourist* into a *No-Go-Area* the motion vector is truncated. The motion stops at the bounds of the *No-Go-Area*.

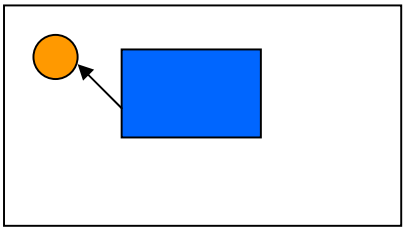

Figure 41. Still, the *Tourist* can be dragged without restrictions outside of the *No-Go-Area*.

4.4 Details

A *No-Go-Area* defines an area on the screen where other objects are not meant to be. This restriction can either apply to a defined group of dragable *Tourist* objects or to all other objects on the screen. A *No-Go-Area* is defined by a shape (rectangle, ellipse, polygons etc.) or the bounding box of an image object.

Tourist objects can be dragged by mouse or change their positions according to motion patterns (e.g. SYNCHRONIZE OBJECTS). Every change in position or size can be expressed as a vector that starts at the original location of the *Tourist*. The motion vector must not intersect with any *No-Go-Area*.

Any motion vector is truncated at the first intersection with a *No-Go-Area*. In particular, a *Tourist* cannot transit a *No-Go-Area*. For example, if the user tries to drag a *Tourist* object from the left hand side of a *No-Go-Area* to the right hand side, the applied motion differs from the requested motion.

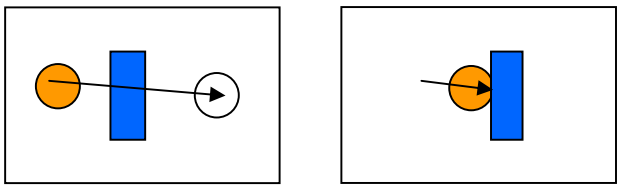

Figure 42. Left: request motion. Right: Applied motion.

Of course, the *Tourists* can move around the *No-Go-Area*:

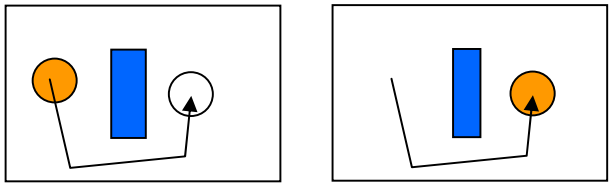

Figure 43. Requested and applied motion are the same.

Without *No-Go-Areas*, representations that are impossible in the real world could be shown and therefore could lead to misconceptions.

Figure 44. Misconception: A person moves over a castle.

Indeed, the end user could take care by himself to avoid dragging a graphic into inappropriate areas. However, learners may not know the restrictions given by nature and drag elements to the wrong places without getting negative feedback. Thus, exploring what is possible and what is not becomes unclear.

Also, avoiding wrong operations requires more concentration at presentation time and can cause additional stress. Wrong operations or invalid representations could occur by accident.

Instead of defining *No-Go-Areas*, one could define valid areas instead. This is done in the SANDBOX pattern. Which strategy to choose depends on whether the valid or invalid areas are larger, or more complex. For example, explicitly expressing all valid areas on the screen is time consuming if only some areas are denied to be entered.

4.5 What else can I do with this?

- Create barriers for dragable elements.
- Stop elements at a given point.
- Define areas which should never be covered by other elements.
- Avoid impossible intersection of images that represent solid objects.
- Build up channels which cannot be exited by a dragable element.
- Build the walls of a maze and forbid that the user can drag elements through the walls.

4.6 Rationale

Not allowing people to arrange objects in a way that either important information is hidden or the meaning of the graphic gets lost is a way of protection [14]. According to the law of

closure people tend to group objects that are in a closed container [6]. To avoid groupings that are unnatural, one can prohibit such operation in the first place.

4.7 Related patterns

Sandbox: The SANDBOX pattern is the inverse of this pattern. A NO-GO-AREA defines explicitly one area where other objects are not allowed to be. A SANDBOX defines explicitly the only area where objects are allowed to be.

Drop Limitation: Only *Accepted Objects* are allowed to stay inside a *Drop area*. Other objects can be dragged over the area, however, once they are dropped they are pushed out automatically. Also, if a maximum number of contained objects is reached further objects are pushed out as well. In opposition to NO-GO-AREAS this pattern allows other objects to enter or to pass a *Drop Area* anyway. Only on dropping an element it will be checked whether or not it is allowed to stay inside.

Drag Restriction: A dragable object is only allowed to move horizontally or vertically. This pattern is used to represent straight motion along a line

5. Sandbox

Avoid that a dragable element can exit a defined area.

5.1 Examples

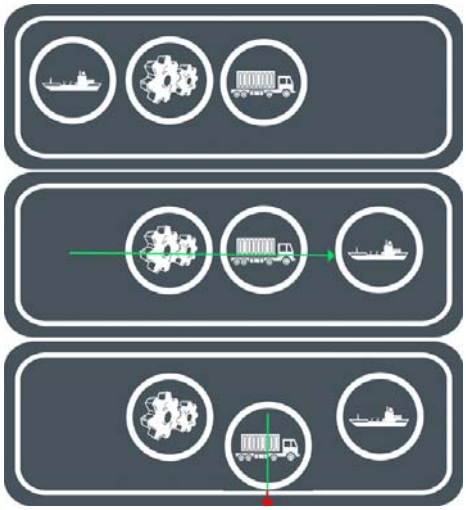

Figure 45. Keep inside the box: You drag and re-arrange all elements within the box. However, you cannot drag one element out of the box.

Figure 46. Cage: You can lock the birds into the cage but you cannot drag them out again.

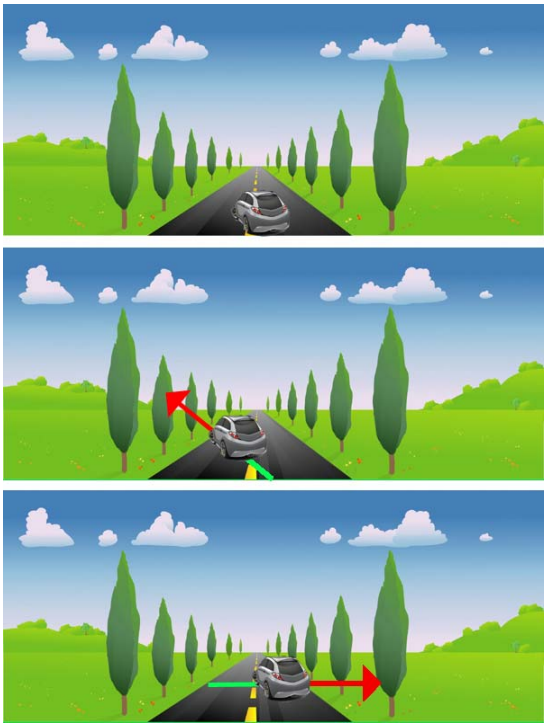

Figure 47. Stay on the road: You can drag the car on the street. The car cannot exit the street.

5.2 Problem

Some graphic elements strictly belong to one area and it must be avoided that the user can drag the graphic to other areas. If you have a graphic element that makes only sense within a defined region, e.g. a car should never exit the street it runs on, you have to provide rules that the element sticks to the area it belongs to. If objects are dragged out of their natural space, the graphic can become invalid or the semantic gets lost. Dragging elements away from their original spaces can decompose a graphic up to the point that the whole image breaks.

5.3 Solution

Figure 48. Define an area as a sandbox.

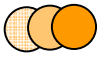

Figure 49. Define dragable objects as Child objects.

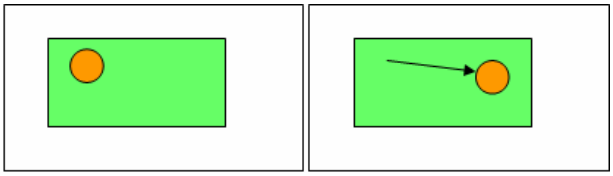

Figure 50. Child objects cannot exit the Sandbox. One can drag a Child unrestricted within the Sandbox.

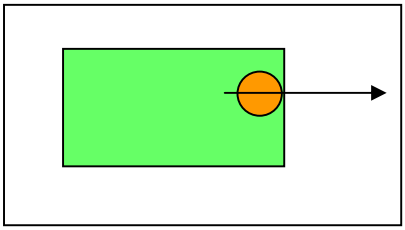

Figure 51. Dragging the Child out of the Sandbox is prohibited. The motion vector is truncated.

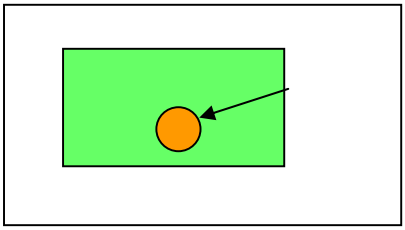

Figure 52. Still, within the Sandbox the Child can be further dragged.

5.4 Details

Many visual elements require the option to be repositioned within a local region. The most flexible way to set a new position is to drag the element with your mouse pointer. However, the user must not drag elements out of the defined areas if this would cause invalid representations.

For a *Sandbox* one can define a group of objects as *Children* that are not allowed to exit the save area of the *Sandbox*. A *Sandbox* is defined by a shape (rectangle, ellipse, polygons etc.) or the bounding box of an image object.

Child objects that are in the *Sandbox* cannot exit it any more. One can have a strong restriction in which a *Child* has to remain fully included within the sandbox.

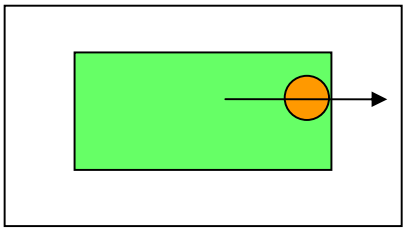

Figure 53. Strong restriction

This option is preferable if you really need to ensure that an area must not be exited partially, e.g. objects should never exit a computer screen.

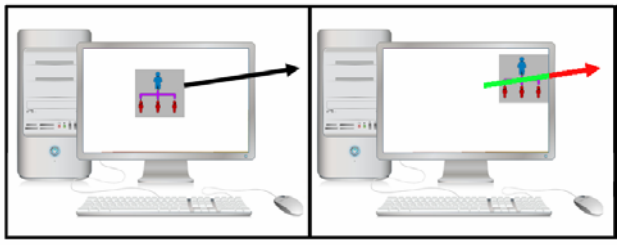

Figure 54. Objects cannot exit the screen

A more liberal option is to consider *Child* objects to be in the *Sandbox* as long as they intersect the *Sandbox*.

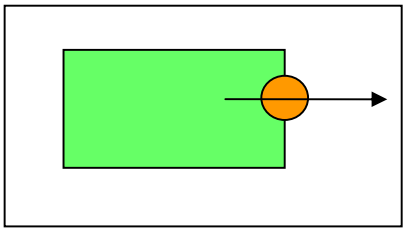

Figure 55. Liberal restriction

The critical requirement in this option is that the *Child* always has to be in touch with the *Sandbox*. For example, you want a person to remain on a platform.

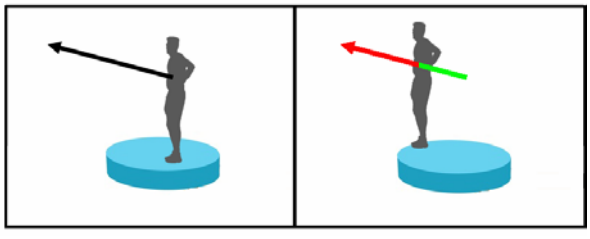

Figure 56. Objects only have to intersect

This option is also helpful to guide objects along paths. For large regions the user can manage to not drag elements out of space without explicit restrictions. But for small regions, e.g. thin channels or the space of a slider, it is impossible to exactly keep a dragable element within its valid regions.

At the beginning, usually all *Child* objects are already in the *Sandbox*. However, it is valid that *Child* objects are outside at the initial state of an interactive graphic. Thus, to drag a *Child* object from outside to inside is a valid operation even if it can only be applied once:

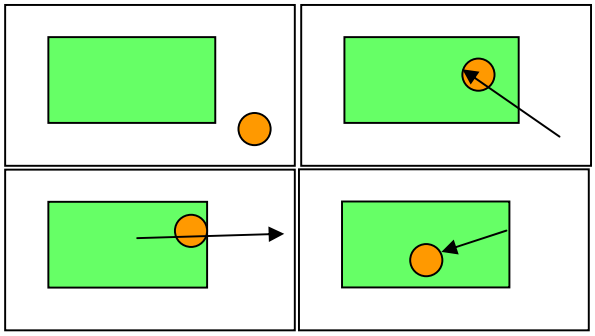

Figure 57. Child moving into the sandbox

5.5 What else can I do with this?

- Allow objects to be freely dragable within a box or container. Within the defined region any position for the object is considered as a valid place whereas all areas outside are invalid.
- Define roads, channels or paths that cannot be exited.
- Collect objects in a container and do not allow to remove the collected items.
- Let objects stick to a defined area.
- Define the space in which GUI elements can be rearranged or manipulated.
- Set an area in which you can drag a slider to different positions, e.g. to set values or navigate other objects within defined regions.
- Make sure that a moveable label sticks to an area with elements that can be labelled by it.

5.6 Rationale

Many objects belong to a specific context and must not be taken out of that context. In user interface design we find many situations in which an interface element makes only sense in specific container [18]. A sandbox prevents users from mixing mismatching elements or breaking apart elements that depend on each other.

5.7 Related patterns

No-Go-Area: The *No-Go-Area* pattern is the inverse of this pattern.

Drop Targets:

Defines multiple areas in which a *Drag Object* can be dropped. If the *Drag Object* is dropped in a *Drop Target Area*, it remains in the area. If the *Drag Object* is dropped outside a *Drop Target Area*, it immediately returns to the place where the dragging started. This pattern can be seen as a network of sandboxes. Objects can be dragged out of the sandbox but have to be dropped in another sandbox otherwise they return to their last save position in a sandbox.

Drag Restriction: (see description in the No-Go-Area pattern)

6. ACKNOWLEDGMENTS

Special thanks to Dave West who has shepherded this paper and shared his experience with the authors. Dave always encouraged us to improve the patterns further and crystallize the essence of the patterns. Also, we would like to thank the writer's workshop participants who have encouraged us to focus on visual representation rather than text descriptions. In particular, Jason Yip and, one year before, Kyle Brown have recommended to imitate a style as found in "Understanding comics" [13]. We are not that far yet, however, the high amount of visual examples and explanatory graphics is a result of their affirmation to this extraordinary form of documentation.

7. REFERENCES

[1] Bayle, Elisabeth et al.: Putting It All Together : Towards a Pattern Language for Interaction Design, Summary Report of the CHI'97 Workshop

[2] Borchers, Jan: A Pattern Approach to Interaction Design; John Wiley & Sons, 2001

[3] Chang, Dempsey; Dooley, Laurence; Touvinen, Juhani E.: Gestalt Theory in Visual Screen Design: A New look at an Old Subject. ACM International Conference Proceeding Series; Vol. 26 . Proceedings of the Seventh world conference on computers in education conference on Computers in education: Australian topics - Volume 8. 2002

[4] Clark, R.C.; Mayer, R.E.: e-Learning and the Science of Instruction, Pfeiffer, San Francisco., p67-81. 2003

[5] Duynie, Douglas K. van; Landay, James A.; Hong, Jason I.: The Design of Sites, Addison-Wesley, 2004

[6] Goldstein, E. Br.: Wahrnehmungspsychologie. Heidelberg: Spektrum Akademischer Verlag, 2002

[7] Granlund, Asa; Lafrenière, Daniel ; Carr, Daniel A.: A Pattern-Supported Approach to the User Interface Design Process, Proceedings of HCI International 2001, 9th International Conference on Human-Computer Interaction, 2001, New Orleans

[8] Kohls, C., Windbrake, T: Autorenwerkzeug Javanti. In Bode, A., Desel, J., Rathmeyer, S., Wessner, M. (Eds.): DeLFI 2003, Tagungsband der 1. e-Learning Fachtagung Informatik, p. 450-459, Garching bei München 2003.

[9] Kohls, C., Windbrake, T: Muster für interaktive Inhalte.In Engels, G., Seehusen, S. (Eds.): DeLFI 2004: Die e-Learni

[10] van Welie, M.; Veer, van der Gerrit, C.: Pattern Languages in Interaction Design: Structure and Organization, Interact 2003ng Fachtagung Informatik, Tagung der Fachgruppe e-Learning der Gesellschaft für Informatik e.V. (GI), p. 389-390. Paderborn 2004

[11] Kohls, C., Windbrake, T. : Towards a Pattern Language for Interactive Information Graphics. Pattern Languages of Programming Design 2006. Portland, Oregon: Hillside Group. URL: http://hillside.net/plop/2006/accepted_papers.htm.

[12] Kohls, C., Windbrake, T.: Moving objects. More patterns for a pattern language of interactive information graphics. EuroPloP 2007, in preparation.

[13] Moreno, R.; Mayer, R.E.: Cognitive Principles of Multimedia Learning: The Role of Modality and Contiguity, Journal of Educational Psychology, 91, p. 358-368, 1999

[14] McCloud, S.: Understanding Comics: The Invisible Art. Harper Collins Publishers, Inc., 1993

[15] Norman, D.: The Design of Everyday Things, New York: Basic Books, 1988.

[16] Rieber, L.P.: Computers, graphics, & learning. Englewood Cliffs, NJ: Prentice Hall, 1990

[17] Schumann, H.; Müller, W.: Visualisierung. Grundlagen und allgemeine Methoden, Springer, Berlin, 2000

[18] Schmitt, Silke; Schreiner, Martin; Timmesfeld, Fel; Vucica, Martina; Wallach, Dieter: PatternCube.com: Ein webbasiertes Repository für User Interface Design Patterns. In: Hassenzahl M.; Peissner, M. (Hrsg.): Usability Professionals 2005

[19] Tidwell, Jenifer: Designing Interfaces, O'Reilly, Sebastopol, 2005

[20] Ware, C: Information Visualization – Perception for Design. Morgan Kaufmann Publishers, San Francisco., p.222, 2004

Appendix: Pattern relations