

CAP3 for Interaction Design Pattern Diagrams?

Jan Van den Bergh, Karin Coninx
Hasselt University - EDM - IBBT
Wetenschapspark 2
3590 Diepenbeek, Belgium
{jan.vandenbergh, karin.coninx}@uhasselt.be

ABSTRACT

Following the success of the “Gang of four” design patterns for software engineering, the human-computer interaction community has used design patterns to document interaction design knowledge. PLML is a structured format to describe these patterns in XML. At least one actively used pattern library for interaction design adopted it to structure its knowledge.

One feature that is defined in PLML and that is also an important part of the “Gang of four” patterns for software engineering is the *diagram*. In this paper, we propose to use CAP3, a graphical abstract user interface modeling language, to specify the diagram part of interaction design patterns. We show some examples of its usage and discuss benefits and drawbacks.

Categories and Subject Descriptors

H.5.m [Information Interfaces and Presentation (e.g., HCI)]: Miscellaneous; D.2.2 [Software Engineering]: Design Tools and Techniques—*User Interfaces*

General Terms

Design, Documentation, Languages

Keywords

Pattern, Interaction design, abstract user interface, CAP3, diagram

1. INTRODUCTION

Design patterns have been investigated in the human computer interaction community for over a decade. Several libraries of patterns exist and even pattern languages have been defined up to ten years ago. One of these pattern languages is PLML [5], which resulted from a workshop at CHI 2003. This language is currently used in the design pattern library of Van Welie [1]. The entries in this library use most features of PLML.

One of these features, *diagram*, is not used in the xml-descriptions, although a limited number of wireframes and sketches is used in the library. We see a similar thing in other interaction pattern libraries;

diagrams are not or very rarely used. Borchers describes the role of the diagram as follows.

“The diagram supports the solution by summarising its main idea in a graphical way, omitting any unnecessary details. For experts, the diagram is quicker to grasp than the opening illustration.” [2]

Following this definition, we take the assumption that diagrams are especially suitable to illustrate properties about behavior and structure. This means that a major part of the interaction patterns would benefit from a diagram. Especially if it can illustrate the basic ideas of the solution proposed in the pattern in a compact, yet clear and understandable way, as class diagrams (and other diagrams) do for the patterns proposed by “the Gang of Four” in the software engineering domain [6].

We propose to use CAP3, a graphical modeling language for abstract user interfaces, to express the solutions for the diagrams. We first introduce CAP3, illustrate its use on three different design patterns, specified in the design pattern library of Van Welie [1] and one from the design pattern library from Tidwell [7]. We choose design patterns at three of the four levels of abstraction defined by Van Welie and Van der Veer [9]: posture level (how to structure a site for a specific domain?), task level (how to accomplish a task?) and action level (how to realize a specific site element?). We do not discuss experience level patterns as these do not specifically relate to structure or behavior of a specific application.

2. CAP3

CAP3 [8] is a modeling language that provides both a concrete syntax (a graphical notation) and an abstract syntax (meta-model) for abstract user interface models. Its concrete syntax is based upon the *Canonical Abstract Prototypes* notation [4] (CAP). CAP provides a number of abstract UI components to describe the structure of a (graphical) user interface in a way that is independent of any concrete toolkit or even modality.

CAP3 and CAP have three main UI components: *tool*, *container* and *active material*. A *tool* allows a user to trigger a change in the UI or in the functional core (e.g. a button), a *container* can hold data (e.g. a label, image) or any other UI component (e.g. a window), and an *active material* is a combination of both previous components and thus can hold both data and trigger a change in the UI, data or functional core (e.g. a textfield, the Microsoft Ribbon). There are many more specific UI components besides these three that have richer semantics and visuals, but are entirely optional as

CAP3		data display	data update	navigation
UI components	tool		X	X
	start/goTo			X
	stop/end/complete			X
	perform(&return)			X
	view			X
	select		X	
	create		X	
	delete/erase		X	
	modify		X	
	move		X	
	duplicate		X	
	toggle		X	
	container	X		
	element	X		
	notification			
collection	X			
active material	active material	X	X	
	input/accepter	X	X	
	editable element	X	X	
	editable collection	X	X	
	selectable collection	X		
	selectable action set			
	selectable view set			
conceptual group	x	x	x	
repeated conceptual group	x	x	x	
relations	activate			source
	update	target	source	
	mutuallyExclusive			X
	use	target		
domain object		X		

Table 1: CAP3 symbols. Symbols with a light-gray background are the core symbols. Other symbols are optional.

they are special cases of them. An overview of all visual elements that are part of CAP3 can be seen in Table 1. Each UI component is represented by a rectangle with an icon in the top-left corner. The table only depicts the icons of the UI components.

Table 1 also shows the other core parts of CAP3: conceptual groups, relations and domain objects. Conceptual groups can be used to group UI components that logically belong together and are represented by dashed rectangles. Repeated conceptual groups (RCP) are special cases of conceptual groups that are used to indicate that a group of UI components is repeated within a containing UI component. RCP are represented by a dashed rectangle accompanied by a triple down-pointing chevron.

CAP3 also contains a limited set of relations: *activate*, *update*, *mutuallyExclusive* and *use*. The relation *activate* indicates that the source activates the target UI component. The relation *Update* means that the source updates the content represented by the target UI Component or the way it is presented. *MutuallyExclusive* means that only one of the UI components can be part of the user interface at a certain moment in time. The *use* relation indicates that the source uses information from the target.

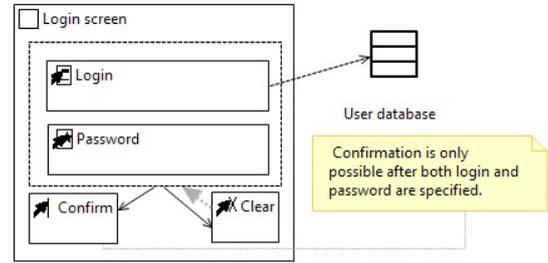


Figure 1: Login window expressed using CAP3

Figure 1 shows some of the relations and UI components applied to an example of a login dialog. It specifies that *Confirm* and *Clear* are only available after the login and password are specified using the *activate* relation. The fact that *Clear* clears both the login and password is shown using the *update* relation (thick gray dotted arrow). Fig. 1 also shows that the logins are fetched from the *user database* (domain object) and are presented in a selectable collection (*use* relation). Note that the *conceptual group* is used to reduce the number of arrows; all relations that are connected to a *conceptual group* can be replaced by relations to all its contained components.

3. CAP3 PATTERN DIAGRAMS

This section discusses how CAP3 can be used to represent diagrams using a limited set of examples spread over three of the four levels of patterns specified by Van Welie and Van der Veer [9]:

Posture level Web-based application [1]

Task level Wizard [7], collapsible panel [1]

Action level Outgoing link [1]

3.1 Web-based application

The example illustrating the posture level is the *web-based application*. As this pattern is high-level and demonstrates the common structures of a web-based application, which is rather complex, it refers to other patterns for more details. Figure 2 shows a possible diagram using CAP3. It consists of a *selectable view set*, a UI component that allows to switch between different “views”, and references to other patterns that are used in the realization of a web application: the main part of a screen in a web application is a *view* (frequently a table with all kinds of data that can be manipulated using *wizards*) or a *form*. Each UI component that refers to at least one pattern is highlighted with a colored background. The dashed line between both components / patterns means that only one of them can be used at any moment. Other common parts are *meta-navigation* (a link to a site-map, search, help, ...) and *login*. The dashed rectangle around the *View* and *Form* is a *conceptual group* and avoids the need to draw *update* relations to both the *Form* and

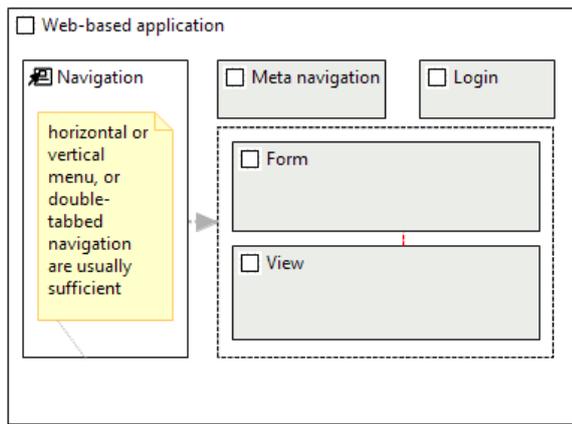


Figure 2: CAP3 diagram for web application pattern

View. A note (rectangle with one flipped corner) is used in Figure 2 to highlight that relatively simple navigation structures will usually suffice.

3.2 Collapsible panels

The *collapsible panels* pattern is an instance of a task level pattern. It is used to manage information by hiding parts of information that are only useful to a subset of the users behind descriptive titles. These titles can be expanded in place to show all hidden information. Figure 3 illustrates this pattern. It shows that an instance of the pattern will contain a set of panels (indicated by the use of a *repeated conceptual group*) that are either in a collapsed or an expanded state. In collapsed state only the title is shown. The fact that the title is represented as an *active material* shows that the title is interactive. The fact that the title serves as a mechanism to hide or show more details is frequently represented through the use of an icon that reflects the state of the panel (collapsed or expanded). This aspect is not shown in the diagram because usage of an icon is not required and is more readily explained using examples or plain text rather than using the CAP3 modeling language. Note that the type of UI component used for the Collapsible panel already gives a hint towards the situation in which the pattern is used.

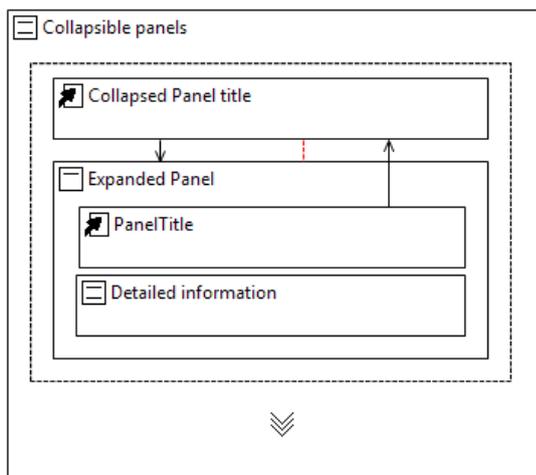


Figure 3: CAP3 diagram for collapsible panels pattern

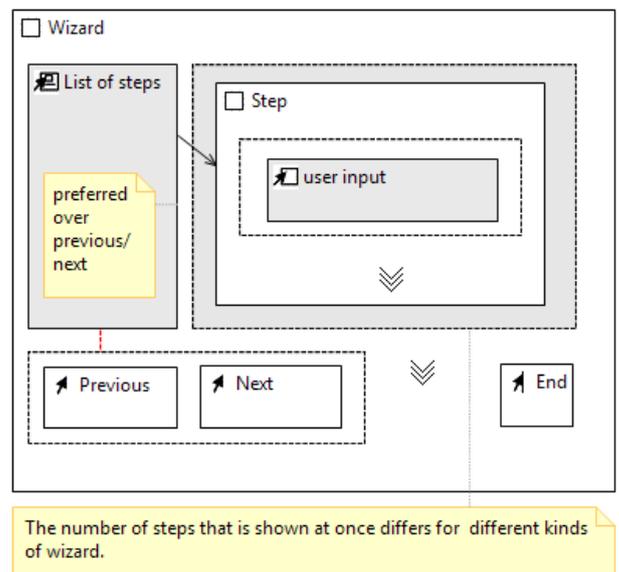


Figure 4: CAP3 diagram for wizard pattern

3.3 Wizard

The *wizard* pattern described by Tidwell [7] demonstrates how design alternatives can be represented. Most of these instantiations are summarized in Figure 4. A first category of instantiations uses multiple screens, each containing a single step of the wizard. Two different ways can be used to navigate between the different steps: through an ordered list of steps (depicted with a *selectable view collection* in Figure 4) or through *previous* and *next* buttons (each represented by a *tool* in the CAP3 diagram.) Both are represented as mutually exclusive choices, with a note indicating the preference for the ordered list of steps. A second category of wizard instantiations bundles all steps on a single page in different ways. Since the way they are combined is discussed in other patterns, a *repeated conceptual group* is used to indicate that they can be combined omitting details on how this can be accomplished. Again, a note is used to explain when multiple steps and navigation can be represented on a page.

Figure 4 illustrates that also conceptual groups can refer to patterns. In this case a *repeated conceptual group* refers to different patterns that can be used to realize the collection of steps, which is otherwise not contained in a UI component. Note that in contrast to the UI components that refer to other patterns in the *Web-based application* pattern, the *user input* UI component does not carry the name of the pattern it refers to (*Good defaults* pattern). Similarly, the names of the patterns the repeated conceptual group refers to are not mentioned in the diagram.

3.4 Outgoing link

At the action level, Figure 5 illustrates the *outgoing link* pattern. This pattern explains that when the majority of the links on a website are within the website (such as on Wikipedia), outgoing links are best accompanied by an icon that symbolizes the fact that the link points to a different site. Figure 5 makes this explicit by grouping the outgoing link in a *conceptual group* and by depicting a *link within website* without the icon. The diagram uses a note to emphasize that the icon should indicate somehow that the link goes out of the site as this is a crucial part of the pattern.

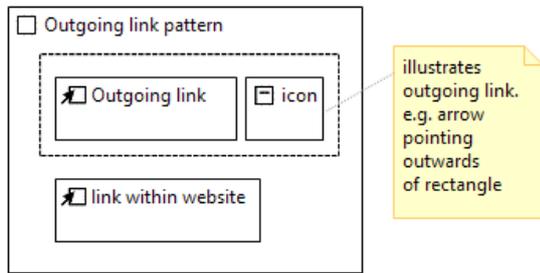


Figure 5: CAP3 diagram for outgoing link pattern

4. DISCUSSION

The examples given in the previous section indicate that it is possible to represent interaction patterns at different levels of abstraction with CAP3 and that their structure is relatively easy to understand. When using CAP3 to create diagrams for interaction patterns we intuitively added notes to illustrate design alternatives or presentation aspects that cannot be expressed in an abstract manner (see Figure 5). The alternatives are highlighted using notes when they can be represented using the same UI component (see Figure 2).

The examples above illustrate that design patterns can be linked to certain types of UI components, but also that more than one pattern can be used for a given UI component. This means it is not always practical to visually represent all alternative patterns or approaches in the diagram. Since CAP3 is a modeling language, we propose to define specific annotations for pattern definition and usage. In this way, patterns are integrated in the models and thus interlinking patterns through interactive diagrams becomes possible and model-driven development approaches can potentially be enriched.

The ability to more precisely define the type of design patterns, also creates the possibility to semi-automatically refine initial designs by applying design patterns. The compact representation of the design pattern also makes it possible to visually illustrate different options. Further combination with the domain model could lead to even better assistance in the application of design patterns in a model-driven approach. For example, the amount of data that needs to be represented could be derived from a domain model and thus limit the choices for applying the models.

CAP3 thus has the capability to be used as a diagram to compactly illustrate the solution offered by a certain design pattern and the capability to be used in a model-driven approach. Both areas however need some further investigation:

Usage as diagram. Earlier discussions with interaction designers working in industry indicated that the abstraction level of CAP3 may be a problem in its usage in projects in practice. Usage as a diagram within a design pattern requires the notation to be immediately clear to experts, since they may use it to more quickly understand or apply the design pattern. While the very concrete examples in design patterns for software engineering presented by the Gang of Four [6] may be very useful for novice users to implement the design pattern, the UML diagrams illustrating the design patterns may be more useful to experts who can fill in the details themselves. Other users should however also be able to understand the meaning of the diagram without major effort. In contrast to UML, however, knowledge of CAP3 is not as widespread among interaction designers as UML class diagrams are for software engineers.

Further investigation regarding the obviousness of the meaning of the different UI components is necessary before working on more widespread adoption.

Usage in model-driven engineering. While CAP3 was used within a model-driven engineering approach to generate concrete user interfaces, further work is needed to find out how it can support the creation of models based on design patterns through model transformations with a supporting software framework. A framework such Epsilon¹ provides useful tools to accomplish this and was already used to create the editor that was used to make all the diagrams in this paper. Fitting CAP3 into a more encompassing formal framework on interaction design, such as that proposed by Botoni et al [3], seems another requirement to create tools which integrate design-patterns in model-driven development.

5. CONCLUSION

We propose the use of a graphical abstract user interface modeling language, CAP3, to represent diagrams in interaction design patterns. Several examples of design patterns at different levels of abstraction are discussed to show that the language is able to express a diversity of interaction design patterns in a compact way. We plan to empirically investigate the clarity of diagrams expressed in CAP3 with interaction designers.

Since CAP3 is also a modeling language, the diagrams could be used in a model-driven development approach that would be accessible to interaction designers. This, however, also needs further research on supporting (model-driven) software tools. Future work includes investigation on how patterns can be used to more easily refactor or construct AUI models. We also see a need for more research on how patterns can limit the need for detailed specification at the AUI level, while still allowing the generation of more detailed concrete user interface models.

6. ACKNOWLEDGMENTS

This work is supported by the FWO project Transforming human interface designs via model driven engineering (G. 0296.08).

7. REFERENCES

- [1] Interaction design pattern library. welie.com/patterns.
- [2] J. O. Borchers. A pattern approach to interaction design. *AI Soc.*, 15(4):359–376, 2001.
- [3] P. Bottoni, E. Guerra, and J. de Lara. Towards a formal notion of interaction pattern. In C. D. Hundhausen, E. Pietriga, P. Díaz, and M. B. Rosson, editors, *VL/HCC*, pages 235–239. IEEE, 2010.
- [4] L. L. Constantine. Canonical abstract prototypes for abstract visual and interaction. In *DSV-IS*, pages 1–15, 2003.
- [5] S. Fincher. Perspectives on HCI patterns: concepts and tools (introducing PLML). *Interfaces*, (56):26–28, September 2003.
- [6] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides. *Design patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.
- [7] J. Tidwell. *Designing Interfaces*. O’Reilly Media, 2010.
- [8] J. Van den Bergh, K. Luyten, and K. Coninx. Cap3: Context-sensitive abstract user interface specification. In *EICS*, 2011. Accepted for publication.
- [9] M. van Welie and G. C. van der Veer. Pattern languages in interaction design. In *INTERACT*, 2003.

¹<http://www.eclipse.org/gmt/epsilon/>